Janis Bubenko jr
Tomas Ohlin

Introduktion till operativsystem

STUDENTLITTERATUR

DEL 2

Janis Bubenko

Tomas Ohlin

Introduktion till operativsystem

Del 2

Studentlitteratur Lund

Akademisk Forlag København

©Janis Bubenko, Tomas Ohlin 1971
Tredje tryckningen
Printed in Sweden
Studentlitteratur
Lund 1972
ISBN 91-44-06801-8

INNEHÅLL

Förord 7

3. Några datalogiska begrepp och problemställningar
3.1 Inledning och översikt 11
3.2 Operativsystemkonstruktion - ett systemeringsproblem 16
3.3 Databehandlingssystemet (dator- och programsystemet) som en resursmängd 18
3.4 Dataprocesser 23
3.5 Parallella processer 30
3.6 Samverkan mellan processer 36
3.7 Resursadministration. Låsningsproblemet 40

4. Driftsformer och möjligheter till åtkomst av datorkapacitet
Inledning 57
4.1 Jobbet, deljobbet och processen 57
4.2 Ett jobb-i-taget bearbetning 59
4.3 Satsvis bearbetning 61
4.4 Kö-vis bearbetning 64
4.5 Reelltidsbearbetning 67
4.6 Time-sharing-system (tidsdelningssystem) 72
4.7 Åtkomst till datorkapacitet 75
4.8 Blandade driftsformer 78

5. Jobbövervakning
5.1 Problemställning - jobbövervakarens uppgifter 80
5.2 Jobbövervakning och reelltidssystem 83
5.3 Tidsdelningssystem (time-sharing (TS-) system) 88
5.4 Jobbövervakarens struktur och arbetsprinciper 88
5.4.1 IBM OS/360 MFT II 89
5.4.2 General Electric GECOS III 102
5.4.3 DataSAAB D22 MK-Dirigent 108

6 Jobbprofiler och planeringsproblem 113
6.1 Jobbprofiler 114
6.1.1 Allmänt 114
6.1.2 Starkt förenklade jobbprofiler 118
6.1.3 Några karakteristiska för jobb av beräkningstyp 119
6.1.4 Jobb av administrativ typ 127

6.1.5 Reelltidsjobb och tidsdelning 131
6.2 Jobbplaneringsproblem 134
6.3 Styrspråk 143

7. Styrprogram och styrprogramfunktioner
7.1 Introduktion 162
7.2 Några grundläggande begrepp 167
7.2.1 Processer och deras egenskaper 167
7.2.2 Samverkan mellan processer och styrprogrammet 171
7.2.3 Processtillstånd 178
7.2.4 Processer - subprocesser - processinteraktion 181
7.2.5 Mekanismer för resursadministration 182
7.3 Principer för tilldelning av processortid 185
7.3.1 Problemställning 185
7.3.2 Mål för tidsplaneringen 187
7.3.3 Planeringsprinciper 189
7.3.4 Planeringsalgoritmer 191
7.4 Principer för administration av primärminne 196
7.4.1 Problemställning 196
7.4.2 Grundläggande principer för disposition och adressering av primärminne 201
7.4.3 Placeringsstrategier och minnesutnyttjande 209
7.4.4 Minnesadministration vid blockbyte (paging) 212

8 Datatransport- och filövervakningssystem
8.1 Inledning och begrepp 222
8.2 Datastrukturer 228
8.3 Dataorganisation 238
8.4 Åtkomst- och utsökningsprinciper 250
8.5 Datatransporthantering 258
8.6 Katalogisering och lagring av filer 263
8.7 Skydd av filer - säkerhetsproblem 265
8.7.1 Olika slag av "hot" mot säkerheten 267
8.7.2 Åtgärder för att öka data-säkerheten 268
8.8 Program - dataoberoende 271

9. Principer och hjälpmedel vid programkonstruktion
9.1 Introduktion 275
9.2 Programstruktur och länkning 276
9.3 Programegenskaper vid systemanvändning 287
9.4 Diagnostik, test och felsökning 288
9.5 Köravbrott och reetablering 291
9.6 Service-program 295

9.7 Systemgenerering och systeminitiering 297

10. Operativsystem - önskemål och mål
10.1 Inledning 300
10.2 Allmänt om målproblem 301
10.3 Önskemål och operativsystem 302
10.4 Några konstruktionsmål 306
10.4.1 OS/360 306
10.4.2 ICL GEORGE 315
10.4.3 UNIVAC 1108/1106 - EXECUTIVE SYSTEM (EXEC-8) 317
10.4.4 MULTICS-systemet 319
10.4.5 Några andra system 321
10.4.6 Önskemål beträffande operativsystemets prestanda och effektivitet 325
10.5 Slutstaser 327
Index 330

FÖRORD

Tillgänglig svensk såväl som utländsk litteratur avseeende operativsystem är knapphändig - om man bortser från en mängd handböcker, som beskriver speciella tillverkares produkter. Det finns många artiklar i fackpressen, som behandlar isolerade problemställningar som kan sägas inrymmas under begreppet "operativsystem". Däremot kan vi idag (1970) ej peka på något verk, där ämnesområdet behandlas på ett samlande sätt.

Inom undervisningen i informationsbehandling och även bland praktiskt ADB-verksamma har behovet av allmän, introducerande litteratur i detta område länge existerat. Med viss tvekan gav vi oss på uppgiften att försöka åstadkomma en bok om operativsystem. När vi nu framlägger resultatet av våra vedermödor, gör vi det med en viss känsla av otillfredsställelse. Visst var vi medvetna om att området var svårt att behandla på ett generellt sätt, men dock anade vi vid starttidpunkten föga de perioder av modlöshet, som vi under arbetets gång skulle uppleva. Det har för oss kommit att framstå allt klarare varför bristen på samlande litteratur om operativsystem är så påtaglig. Anledningarna är bland annat

- ämnesområdet är vagt definierat, dynamiskt och under snabb utveckling. Nya system, för att inte tala om versioner av existerande system, frisläpps tätt.
- någon enhetlig och allmänt tillämpad underliggande begreppsapparat och teoribildning kan ej anses existera.
- terminologiproblemen är dominerande - standardiseringssträvanden kan ej skönjas.
- operativsystems arbetsprinciper är, åtminstone på en mindre grov nivå, beroende av olika tillverkares datorprodukters särdrag m m.

Att någotsånär behärska ett fåtal tillverkares operativsystem är därför knappast en tillräcklig förutsättning för att skriva en uttömmande bok om operativsystem. Vi vill därför starkt understryka att vi, trots studier av olika tillverkares op-system, inte gör anspråk på att ha behandlat området generellt och uttömmande. Läsaren blir sålunda inte "os-specialist" genom att inhämta innehållet i denna bok.

Vår förhoppning är dock att boken skall anses beskriva vissa fundamentala os-principer och att den skall kunna ge den mindre erfarne databehandlaren en bakgrund, som dels underlättar specialstudier av olika speciella

---

Medförfattarens T Ohlin arbete på denna skrift är till sin merpart finansierat av Statens Naturvetenskapliga Forskningsråd.

op-system och dels gör att dessa studier kan bedrivas med en viss kritisk blick.

Av ovanstående skäl har vi valt att ge boken titeln "Introduktion till operativsystem" och ej, som vi från början avsåg, "Operativsystem". Vi vill också påpeka, att vi har strävat att belysa operativsystem ej enbart som en samling av styr-, övervaknings- och servicefunktioner utan även som en betydelsefull komponent vid konstruktion och drift av informationsbehandlingssystem.

Bokens innehåll sönderfaller vid betraktande i två delar:
1. Utvecklingen av operativsystemområdet.
2. Operativsystemprinciper och mål.

Del 1 är av översiktlig natur och belyser i någorlunda kronologisk ordning den utveckling, som har ägt rum dels på maskinvarusidan, dels på programvarusidan, och som har lett fram till operativsystem som idag anses som en fundamental komponent hos varje datorinstallation[1]. Kapitlet om maskinvaruutvecklingen kan kanske anses väl omfattande. Vi har dock valt detta omfång dels för att svensk litteratur som belyser området är knapphändig, och dels för att operativsystemen kan och bör ses som en utvidgning av de maskinella datorfunktionerna. En dos datorhistoria torde f ö kunna ses som ett värdefullt "allmänbildande" komplement till övrig existerande specialundervisning inom ADB-området.

Vi vill framhålla, att kapitlet om maskinvarans utveckling i huvudsak författats under år 1969, och därför speglas situationen fram till detta år. Utvecklingen inom detta område är mycket snabb, och publicerat material blir på kort tid inaktuellt. Vi vill därför be läsaren ha överseende med det faktum att flera intressanta datormodeller, som presenterats fr o m år 1970, inte har belysts i kapitlet.

Del 2 går mer i detalj in på olika operativsystemprinciper och mekanismer. Vi behandlar där kapitelvis
- diverse grundläggande begrepp
- olika driftsformer (ur användarens synvinkel)
- jobbövervakningsfunktionen
- jobbprofiler och inplanering av jobb
- övervakning och styrning av processer (styrprogramfunktioner)
- datatransport- och filhanteringssystem

1) Vissa avsnitt härav är med tillstånd från ACM hämtade från S. Rosens och R.F. Rosins artiklar i ACM Computing Surveys, vol. 1, nr. 1.

- programkonstruktion och serviceprogram
- mål för operativsystem

Vi har ansett det lämpligt att fysiskt separera de två delarna, och publicerar därför innehållet i två volymer. Viss innehållsmässig överlappning av innehållet kan säges existera. Bl a därför kan de läsas oberoende, dvs för att studera del 2 förutsättes ej innehållet i del 1, även om det kan vara en fördel ur överskådlighetssynvikel. För att med förståelse studera boken krävs ej några speciella "akademiska" förkunskaper. Läsaren bör dock vara bekant med elementa inom programmering, databehandling och datasystemkonstruktion. Del 1 kan exempelvis läsas efter en introducerande kurs i informationsbehandling kompletterat med en laborativ programmeringskurs. Del 2 kan beroende på studieinriktning läsas direkt efter del 1 eller sparas till kursmoment som följer efter några laborativa moment i databehandlingsteknik och systemering. I universitetsmiljö skulle detta motsvaras av att del 1 ingick på 20-poängsnivå och del 2 på 40-poängsnivå i informationsbehandling med inriktning administrativ databehandling eller numerisk analys. Den datalogiinriktade skulle kunna läsa både del 1 och del 2 på 20-poängsnivå.

Vi är medvetna om att delar av bokens innehåll ej kommer att vara okänt stoff för i synnerhet de läsare som redan besitter någon tids erfarenhet från praktisk verksamhet inom ADB-området. Vi är också medvetna om att boken ej har mycket att ge till erfarna operativsystemprogrammerare. Vi tror dock att det finns en icke ringa mängd i dag yrkesverksamma programmerare och systemerare som kan finna det intressant att se området belyst från en annan synvinkel än den som impliceras av den lokala miljön och de lokala problemställningarna.

Eftersom detta är ett första försök till en bok på svenska om det mystikomgärdade området "operativsystem" hoppas vi att läsaren har överseende med diverse inkonsekvenser vad gäller terminologi och försvenskning av engelska facktermer. För det aktuella terminologiområdet finns oss veterligt ej någon svensk standard etablerad. Vi ber också om överseende med den frekventa förekomsten av orden "ofta", "normalt" och "vanligtvis". Härigenom har vi försökt antyda att det förekommer andra lösningar, metoder och betrakelsesätt än de vi beskriver.

Med tillfredställelse skulle vi notera om denna bok skulle inspirera läsaren till specialstudier och publicering av mer detaljerade arbeten inom det omfattande område som vi berör. Det återstår åtskilligt att bearbeta inom området operativsystem, för vår del får det dock räcka för tillfället!

Janis Bubenko   Tomas Ohlin

3. NÅGRA DATALOGISKA BEGREPP OCH PROBLEMSTÄLLNINGAR

3.1 Inledning och översikt

Avsikten med bokens del 2 är att ge en översyn över i dag existerande, i vårt tycke viktiga operativsystemfunktioner och driftsprinciper. Såväl med avseende på funktioner och egenskaper som effektivitet för olika tillämpningar (jobbprofiler) varierar operativsystemen emellertid starkt från ett fabrikat till ett annat och i viss mån även från en "version" till en annan. Det är därför svårt att redogöra för dessa funktioner och principer på ett så fullständigt sätt att samtliga i dag existerande varianter blir medtagna. Vi har gjort ett intuitivt och naturligtvis subjektivt urval av sådana problemområden som vi bedömt fundamentala och någorlunda "generella".

Ett operativsystem kan definieras som "den mängd av datormekanismer (maskinella och/eller programmerade) som har till syfte att dels underlätta konstruktion och implementering av databehandlingsrutiner dels styra dessa rutiners bearbetning". Då man kan anse att program är en delmängd av komplexet "databehandlingsrutin" har vi valt det senare mer generalla begreppet. Enligt denna definition skulle operativsystemet förutom styrprogram (bearbetningsövervakande program), jobbinplaneringsprogram, språköversättare (kompilatorer) och diverse serviceprogram omfatta även vissa mer generella applikationsinriktade program för en viss installation. Detta innebär att varje installation av ett visst datorfabrikat skulle ha olika operativsystem. En del av operativsystemet är då normalt tillverkat av datorleverantören och en annan del av användaren eller av någon konsultfirma. Den senare delen är i de flesta fall olika för olika installationer eftersom den är applikationsberoende. På grund av att operativsystemets uppbyggnad normalt är modulär (dvs systemet är sammansatt av ett antal separata moduler motsvarande olika applikationsönskemål) avviker även den av leverantören tillhandahållna delen av operativsystemet vanligen från installation till installation. Dels är det sällan som två datorkonfigurationer är identiska dels är jobbprofilerna oftast olika vilket föranleder olika "konfigurationer" av den leverantörstillverkade delen av operativsystemet.

I denna bok behandlar vi i samband med operativsystem ej s k applikationsprogram av problemorienterad karaktär. Vi har således hållit oss till typer av program vars syfte är att underlätta och effektivisera till-

verkning och drift av direkt problemorienterade program. Vi behandlar alltså styrprogram och några viktigare typer av systemprogram.

I de allra flesta fall ingår dylika program (operativsystemfunktioner) i det levererade operativsystemet. Det har dock åtskilliga gånger inträffat att användaren själv behövt utöka eller modifiera ett operativsystems funktionsrepertoir för att kunna anpassa det till applikationssystemet. Detta gäller ofta fortfarande avseende reelltidssystemen för vilka levererade operativsystem i ett eller flera avseenden antingen är ofullständiga (ur användarens synvinkel) eller också ineffektiva på grund av att de är alltför generellt och flexibelt[1] uppbyggda.

Hos varje operativsystem återfinner vi en samverkande mängd av centrala systemprogram vars huvudsakliga syfte är att övervaka och fördela program- och maskinvaruresurser under exekvering av ett eller flera parallella program i datorn. Denna centrala "programmängd", som från fabrikat till fabrikat kan ha högst varierande funktionsrepertoir, kallas vanligen styrprogram (eng "supervisor" eller "executive"). Funktioner som man normalt, åtminstone vid medelstora system och uppåt, återfinner inom ett styrprogram är

- brytsignaladministration
- övervakning och styrning av processer och jobb
- administration av primärminnesutrymme
- administration av program
- övervakning och administration av datatransporter
- tidskontroll (administration av systemets "klockor")
- operatörskommunikation

Om aktuell dator arbetar med priviligierade MASTER-mode instruktioner (dvs sådana som endast kan anropas från ett program i MASTER-mode, se även kapitel 2. 2 sid 145) är det normalt att endast de program som befinner sig på styrprogramnivå får använda sig av dessa. Inom Datasaab kallar man detta för "beordringstillstånd". Program i "SLAVE-mode", dvs användarprogram och vissa systemprogram, anses då vara i "skyddstillstånd" och har då endast tillgång till en begränsad instruktionsrepertoir.[2] Under deras exekvering är bl a minnesskyddsmekanismen i funktion.

---

1) Tyvärr är det ofta fallet att priset för en generellt och flexibelt system är lägre effektivitet som dock ev kan förbättras med en kraftig utökning av maskinvaruresurserna. Värdet av generalitet och flexibilitet är svårmätbart och har ofta fått en underordnad betydelse.
2) Andra benämningar för denna uppdelning är priviligierad programnivå resp användarprogramnivå.

Den centrala delen av styrprogrammet utgörs av de program som handhar brytsignaladministrationen. Delar av dessa program exekveras i beordringstillstånd med en del av brytsignalssystemet inhiberat för att programmenej skall riskera att ständigt avbrytas av andra brytsignaler med lägre prioritet innan aktuell brytsignal är "åtgärdad".

Det är givet att ovanstående lista omfattar vissa funktioner som ej förekommer för mindre datorsystem eller hos eljest "primitiva" system, likväl som mer avancerade system kan innehålla ytterligare funktioner. Styrprogramfunktioner kommer vi att diskutera mer i detalj i kapitel 7.

Innan vi går in på styrsystem är det lämpligt att uppehålla sig något vid olika i dag existerande "driftsfilosofier" för datorer och belysa de olika åtkomstmöjligheter som existerar för att förse ett datorsystem med jobb[1] och få resultat tillbaka. Detta behandlas i kapitel 4. Vi kommer där att belysa huvuddragen hos bl a satsvis bearbetande, reelltidsbearbetande och tidsdelningssystem (time-sharing systems). Vanliga kombinationer av dessa driftsformer kommer också att diskuteras.

Kapitel 5 och 6 behandlar jobbövervakning och jobbinplanering. Vissa av de systemprogram som ombesörjer dessa funktioner arbetar normalt i skyddstillstånd, dvs på samma nivå som applikationsprogram. Vi kommer att kalla alla dessa programrutiner för en jobbövervakare (i enlighet med Datasaabs "MK-Dirigent"). Vanliga engelska beteckningar är: "job and master scheduler" (IBM OS/360) "coarse scheduler" (Univac EXEC-8) och "high-level scheduler" (ICL George III).

Jobbövervakaren samarbetar med styrprogrammet vid initiering och avslutning av arbetsuppgifter på skilda nivåer (jobb, jobbsteg (deljobb), processer). Jobbövervakarens uppgift är således att administrera och, för system där så är möjligt, planera hanteringen av jobb som tillförts systemet. Den ansvarar ofta också för att bokföring sker av resurser som brukats av ett jobb. Jobbövervakarens roll och omfattning varierar naturligtvis beroende på systemets driftsfilosofi. Vi kommer att belysa arbetsprinciper i samband med system av satsvis-, reelltids- och tidsdelningskaraktär. Även "styrspråk" kommer att behandlas i kapitel 6.

En annan av de centrala delarna (ur användarsynvinkel) hos ett operativsystem är dess datahanteringssystem. Med datahanteringssystem avses här hjälpmedel för

---

1) Tills vidare definierar vi ett jobb som en på något sätt avgränsad mängd av arbetsuppgifter som tillföres dator-operativsystemet.

- organisation, lagring och utsökning av data och program i systemets sekundärminnessystem
- administration av datatransporter till/från såväl lokala som avlägsna perifera enheter och/eller datorer

I detta sammanhang stöter vi också på problem i samband med namngivning och strukturering av datamängder samt katalogisering och skydd av datamängder mot icke auktoriserad åtkomst. Principer i samband med datahanteringssystem kommer att diskuteras i kapitel 8. Påpekas bör dock att vi i huvudsak kommer att uppehålla oss vid datahanteringssystem som arbetar på adresserbara eller namngivna datamängder utan hänsyn till deras informationsinnehåll. Vi begränsar oss till s k "utsökning efter nästa post och utsökning efter adress eller namn (dvs identifierare)". Datahanteringssystem vars sökning styrs efter det semantiska informationsinnehållet i de betraktade datamängderna hör hemma under problemkomplexet "informationssökning" och dylika funktioner ingår normalt ej som komponenter i i dag aktuella operativsystem.

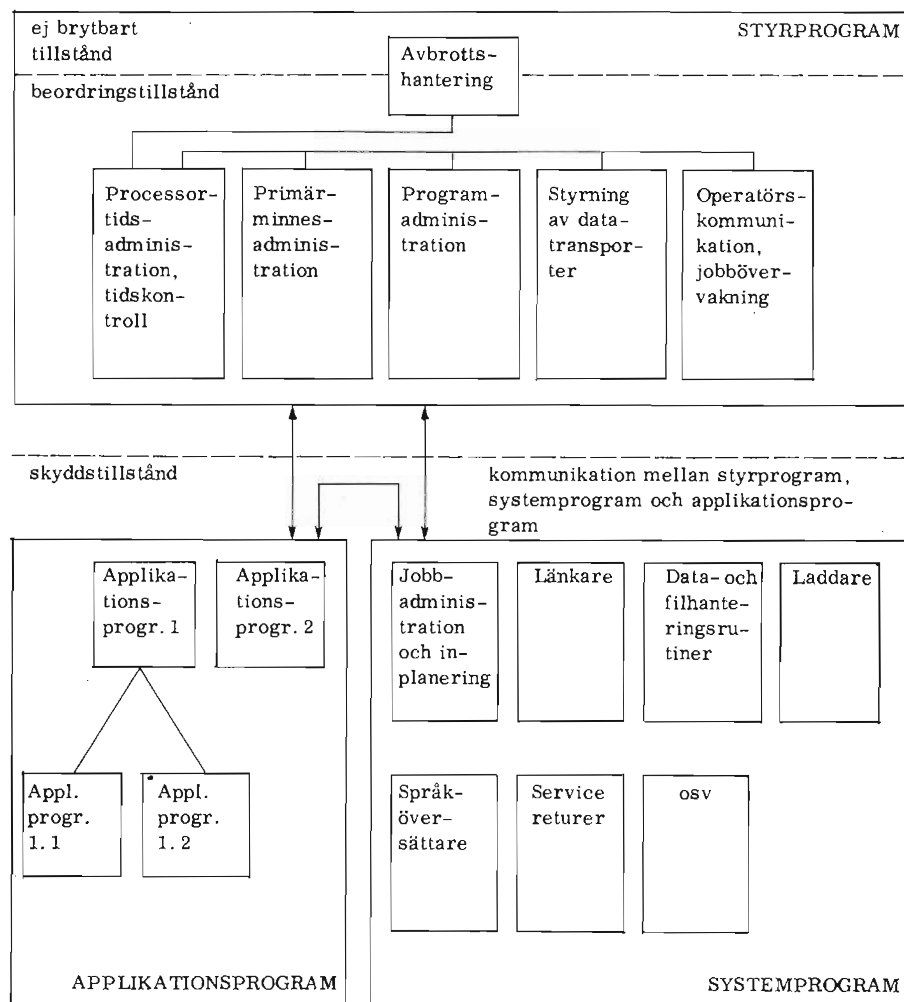
I kapitel 9 kommer vi att ta upp problem och principer som sammanhänger med programkonstruktion. För program som vid exekveringstillfället ej i sin helhet kan lagras i primärminnet utan delvis måste residera i sekundärminnet diskuteras olika principer för strukturering, lagring i primärminnet och administration av inläsning av programdelar (segment, moduler) från sekundärminne. Programmens egenskaper ur användningssynpunkt kommer att diskuteras. Visst utrymme ägnas åt länkarens arbetsprinciper. Särskilt hos reelltidssystem är problemen i samband med test och felsökning av skrivna programdelar viktiga. Även om materialet avseende dessa hjälpmedel är magert skall vi försöka att redogöra för några sådana verktyg. Diagnostik och reetableringsproblem kommer att uppmärksammas då dessa är viktiga i reelltidssammanhang.

Slutligen, i kapitel 9 kommer vi att kortfattat redogöra för några inom ett operativsystem vanligen förekommande relativt fristående servicerutiner.

Att utvärdera operativsystem och väga ett system mot ett annat är en uppgift som i dag kräver åtskillig kompetens. Trots att vissa "rangordnings- och utvärderingsmetoder" figurerat i den kolorerade datapressen kan området ännu anses vara väsentligen obearbetat.

Innan man överhuvudtaget kan föra utvärdering av ett system på tal måste en målstruktur bestämmas. Även detta arbete är långt ifrån trivialt och litet har publicerats om mål i samband med operativsystem. Vi har vågat oss på att diskutera önskemåls- och målproblematik i avslutningskapitlet.

Figur 3.1:1 vill, avslutningsvis, visa huvudelementen hos ett fiktivt operativsystem och uppdelning på tre "tillstånd" vid exekvering.



Figur 3.1:1
Styrprogrammet ombesörjer den dynamiska resurstilldelningen under exekvering av en given sammansättning av program. Samtliga systemprogram och applikationsprogram måste för sitt fortskridande löpande begära resurser hos styrprogrammet.

3.2 Operativsystemkonstruktion - ett systemeringsproblem

Resten av detta kapitel avser vi ägna åt några generella, datalogiska begrepp och problemställningar som är relevanta i operativsystemsammanhang. Dessa begrepp och problem är oberoende av speciella datorers egenskaper och torde vara av generellt intresse för flertalet typer av datorer och operativsystem. Det är ett faktum att de som sysslat med konstruktion och implementering av mer avancerade operativsystem har stött på en mängd av mer eller mindre oförutsedda svårigheter. Anledningen till detta kan bl a förmodas vara att det allmänt råder en brist på insikter om de principer som ligger till grund för de flesta av den stora mängd av olika mekanismer vars delmängder utgör operativsystem av olika slag (1).

Man kan också förmoda att flera problem har uppstått på grund av att ingen klar och konsistent målstruktur legat till grund för konstruktionsarbetet.

Systemprogrammering är ännu ett relativt nytt ämnesområde,och någon allmänt accepterad uppsättning av principer och tillhörande begreppsapparat och undervisningsmetodik att förmedla dessa insikter har ej hunnit utvecklas.

Att bland det som hittills publicerats om operativsystem och i sammanhanget relevanta problem finna en röd tråd och därur välja ut och presentera en uppsättning principer och mekanismer som kan förmodas vara allmängiltiga och accepterade överstiger ambitionsnivån för denna bok och synes kräva en omfattande forskningsinsats.

Med tanke på utvecklingstakten är det t o m tveksamt om en sådan forskningssatsning skulle kunna komma till ett logiskt slut inom rimlig tid.

Vi måste därför poängtera att det som presenteras här ej kan göra anspråk på varken logisk konsistens eller fullständighet. Framställningen är subjektiv såtillvida att vi från en mängd litteraturreferenser mer eller mindre intuitivt valt ut material som vi bedömt vara av intresse och relevans i operativsystemsammanhang. Vissa idéer och begreppsbildningar är vi själva ansvariga för.

Problemen som uppträder vid konstruktion av operativsystem kan sägas vara av generell natur och dessa systemproblem möter vi vid konstruktion av vilket komplext och svåröverblickbart system som helst (se Langefors, "Theoretical Analysis ..." ref. (14) beträffande oöverblickbarhetsproblem). Det är fråga om att med utgångspunkt från en önske-

måls- och målstruktur konstruera en delsystemstruktur och bestämma delsystemens egenskaper och relationer till andra delsystem och till omgivningen. En effektiv implementering kräver en väldefinierad och konstruktiv delsystemstruktur med en klar målutformning, relations- och egenskapsbeskrivning för varje delsystem. Vanligt är att det krävs att (se t ex (1)) denna delsystemstruktur är så definierad att "så få mekanismer som möjligt gör så mycket arbete som möjligt". I denna bok tänker vi dock ej uppehålla oss vid generella systemteoretiska problem utan väljer den mindre ambitiösa vägen att presentera ett antal avgränsade problemställningar.

Något förenklat uttryckt är uppgiften för ett operativsystems styrsystem att bistå och styra exekvering av andra program. De problem som härvid uppträder kan indelas i två klasser (1).

(a) resurstilldelning, dvs problem som sammanhänger med automatisk tilldelning av resurser till olika processer i systemet

(b) resursdelning, dvs problem som sammanhänger med simultant nyttjande av (gemensamma) resurser såsom informationsmängder (datafiler och procedurer).

Problem av typen (a) omfattar bl a

- administration och tilldelning av processortid
- administration och tilldelning av primärminne och sekundärminne
- administration och planering av datatransportoperationer
- samordning och synkronisering av processer

Problem av typen (b) är bl a aktuella för interaktiva multipel-access-system (tidsdelningssystem) där flera program simultant exekveras och där interaktion mellan programmen liksom informationsbyte mellan processerna förekommer. Sådan resursdelning kräver ett adekvat fler-nivå-system för skydd av kollektivt tillgängliga data- och procedurmängder.

För att gå in på en diskussion om resurstilldelning måste vissa grundläggande begrepp avseende processer först definieras. Resurstilldelning, och även resursdelning, är direkt förknippat med processbegreppet.

3.3 Databehandlingssystemet (dator- och programsystemet) som en resursmängd

För vårt syfte är det lämpligt att börja med resursdefinitioner även om vi riskerar hamna i en datalogiskt vanlig situation att definiera begreppet A med hjälp av ett annat begrepp B som ännu ej definierats. I vårt fall är B "dataprocessen". En dataprocess definierar vi därför tillsvidare abstrakt som "någonting som kräver "service" från resurser för att kunna utföras (exekveras)" och återkommer till en närmare diskussion om detta senare. För enkelhets skull kallar vi i fortsättningen dataprocesser kort och gott för processer.

Från ovanstående kan vi notera att "resurser förbrukas" (på något sätt) under exekvering av en process, här, liksom i andra sammanhang.

Ett databehandlingssystem kan betraktas som en mängd av resurser (med olika egenskaper) sett från en process' synvinkel. Denna process kan tillhöra en applikationsrutin eller en operativsystemfunktion eller någonting annat, t ex ett manuellt bearbetningsmoment.

Med ett databehandlingssystem avser vi den samlade mängden av

- maskinell utrustning
- programvara (inkl op system)
- applikationsprogram och användare
- datafiler
- driftspersonal och andra faciliteter för driftshantering

Alltså ett ganska omfattande begrepp. Då vårt intresse här främst rör dataprocesser och deras exekvering har utökning av systemet, genom att ta hänsyn till informationens roll i styrningssammanhang, ej varit aktuell.

Vi övergår nu till att diskutera resurser ur utnyttjandesynvinkel. För att en process skall kunna "äga rum" (exekveras) krävs utnyttjande av en, eller flera, resurser. Med hänsyn till hur olika resurser låter sig utnyttjas (förbrukas) kan vi klassificera dem som exklusivt användbara eller multianvändbara resurser.

En exklusivt användbar resurs kan betjäna endast en process åt gången. Om flera, säg N, processer kräver service från samma resurs av denna typ (så gott som) samtidigt, får N-1 processer "vänta" i en kö och endast en av processerna tilldelas service. Vi avstår här från att kommentera urvalsprinciperna (varför just den processen?) för sådan resurstilldelning då det problemet aktualiseras i samband med planering och styrning av resursfördelningen. På samma sätt, om en process söker service från

en resurs och denna resurs redan är upptagen med en annan process, måste den förstnämnda processen vänta, och placeras på något sätt i en (eventuellt redan existerande) kö av processer till resursen ifråga. Ovanstående är dock inte alltid sant. Om den nytillkomna processen har "hög prioritet" kan den för vissa (men ej alla!) resurser tänkas bryta den pågående processen och ta dess plats. Den så avbrutna processen kan återupptas senare när den högprioriterade processens servicekrav tillgodosetts. Följaktligen kan vi bland de exklusivt användbara resurserna skilja på sådana som är avbrytbara och sådana som icke är avbrytbara (eller som kan men icke önskar bli det).

Multianvändbara resurser kan simultant (eller kvasisimultant (jfr reentrant program)) betjäna två eller flera processer. Praktiskt sett finns det alltid en gräns för antalet processer som en dylik resurs kan betjäna simultant. Teoretiskt sett existerar, som vi kommer att se, resurser som simultant kan ge service till en oändligt stor mängd processer.

För våra fortsatta diskussioner är det önskvärt att införa klasser av resurser med hänsyn till deras art. Således kommer vi att tala om

- maskinella resurser
- informationsresurser
  . programvaruresurser
  . dataresurser
- manuella resurser

Maskinella resurser syftar givetvis på de olika maskinenheterna och funktionerna och har att göra med deras kapacitet. Vi anför några exempel:

> en processenhet är en maskinell, exklusiv användbar resurs, vars service till en process kan, om man så vill, avbrytas och överföras till en annan process med högre prioritet.
>
> en selektorkanal är en maskinell, exklusivt användbar resurs, vars service till en process ej kan avbrytas (i det normala fallet) då den i så fall kommer att låta data, som överförs, gå förlorade.
>
> en multiplexorkanal är en maskinell, multianvändbar resurs. Den kan, inom ramen för sin kapacitet, betjäna flera processers datatransportoperationer simultant.
>
> en programvaruresurs är en informationsresurs som kan vara antingen multi- eller exklusivt användbar. Multianvändbara programvaruresurser går ofta under benämningen reentrant. En pro-

gramvaruresurs kräver "hjälp" och service av andra resurser för att själv kunna ge service till anropande objekt.

en datapost i en fil är en informationsresurs och kan ses dels som en multi- dels som en exklusivt användbar resurs beroende på sättet som denna resurs nyttjas. Vi observerar dock att referens till en datapost kräver referens till en serie av andra resurser (maskin- och programvaruresurser).

konsoloperatören kan vi betrakta som en manuell resurs som normalt bör ses som en exklusivt användbar resurs, då det är ovanligt att denne skulle kunna betjäna flera processer simultant.

Vi kan betrakta processer och resurser ur ytterligare en aspekt, nämligen med avseende på protektion, dvs skydd mot att en process otillbörligt använder sig av en resurs. Uppenbarligen får ej vilken process som helst använda vilken resurs som helst. Det skulle bl a snabbt kunna skapa kaos i ett multiprogrammerat system. Varje process har därför i praktiken en begränsad mängd av resurser att röra sig med som dock (för en del system) kan, generellt sett, inom givna ramar dynamiskt ökas eller minskas under processens livstid.

Egentligen är situationen mer komplicerad än så. Vi kan tänka oss att en process P, initierad av en "auktoritet" A, kan förfoga över en större mängd tillåtna resurser än om den hade initierats av en auktoritet B. Auktorisationsfrågor av detta slag kommer vi dock ej att beröra här utan tar upp den i samband med en allmän diskussion om protektion i interaktiva multipelaccess och databasorienterade system.

Om databehandlingssystemet består av resursmängden R existerar det en delmängd R_{Pi}, $R_{Pi} \subset R$, som erfordras för processen Pi och som denna får (på ett eller annat sätt) utnyttja. Övriga resurser, komplementmängden R'_{Pi}, får ej på något sätt användas av processen Pi. Exempel på resurser som normalt ej får användas av en lägre auktoriserad "användarprocess" är

- privilegierade instruktioner (för manipulering av minnesskyddsmekanism, I/U-operationerna osv)
- perifera enheter som ej "tilldelats"processen osv

Bland de resurser som är tillåtna för processen Pi kan man generellt skilja mellan olika former för användning av en resurs.

Formen för användning påverkar givetvis även kravet på protektion av motsvarande resurs. Bortsett från kravet att skydda vissa informations-

resurser mot icke auktoriserad åtkomst kan vi betrakta det skydd som behövs för att i systemet vid multiprogrammering existerande parallella processer skall exekveras på ett riktigt sätt.

Antag att en multianvändbar fil F existerar med posterna $f_1, f_2 \ldots, f_N$. Två processer P_1 och P_2 exekveras parallellt och delar bl a dataresursen, filen F. Följande sekvens av händelser kan tänkas

1) P_1 läser posten f_i till sin dataarea, vid tiden t_1
2) P_2 läser posten $f_{it'}$ till sin dataarea vid tiden t_2
3) P_1 utför bearbetning och återlagrar posten f_i vid tiden t_3
4) P_2 utför bearbetning och återlagrar $f_{it'}$ vid tiden t_4

Vi antar att $t_1 < t_2 < t_3 < t_4$. Filen F kan antas ligga lagrad på ett sekundär- eller primärminne.

Om posterna f_i och $f_{it'}$ ingår i skilda fysiska "block"[1] kan ovanstående sekvens ej orsaka komplikationer oavsett vad bearbetningen i steg 3) och 4) avsåg. Förutsättningen är dock givetvis att inga andra processer existerar som refererar till filen F. Fallet blir dock komplicerat om f_i och f_{it} ligger i (tillhör) samma fysiska block. Om bearbetningarna i steg 3) och 4) ej ändrade posternas innehåll uppstår inga komplikationer. Dock vore återlagringen meningslös i ett sådant fall. Om bearbetningen 3) däremot på något sätt ändrade postens innehåll ($f_i \rightarrow f_i'$) kommer efter ovanstående sekvens ett felaktigt resultat erhållas. Efter steget 4) kommer filen att innehålla

$$F = f_1 , f_2, \ldots, f_i, f_{it}', \ldots, f_N$$

i stället för

$$F = f_i, f_2, \ldots, f_i', f_{it}', \ldots, f_N$$

Ovanstående exempel är en vanlig problemställning i interaktiva multiaccess- eller reelltidssystem där multiprogrammering, dvs existensen av flera parallella processer i systemet, tillämpas. Den protektion som krävs för att garantera olika processer (= resursanvändare) validitet i deras arbete kan graderas enligt (Murphy (2)):

---

1) Med fysiskt block avses här den datamängd som blir föremål för datatransport vid läs- resp skrivinstruktioner.

(1) om en resurs, eller del därav, ändras av en process, måste sekvensen av operationer som utför denna ändring skyddas mot all interferens som skulle kunna påverka förverkligandet av ändringen.

Vår diskussion om filen F exemplifierar (1).

(2) om en process använder en resurs men ej ändrar den, krävs det protektion som garanterar att resursen under den tid krävs av processen ej förändras så att validiteten av processens resultat påverkas.

Detta kan kanske enklast exemplifieras med ett stycke program. Antag att några satser ur två parallella processer P_1 och P_2 kan uttryckas i Algol:

```
begin   integer a; .... a:=1; ...
P1:     begin integer b, c; ... b:=2;
sats 1: a:=a+b;
sats 2: c:=a;
        .....end;
P2:     begin ...
        a:=5;
        ... end;
end
```

Här antas P_1 och P_2 kunna existera parallellt (även om detta strider mot Algol -60 konventionen att skriva program) där de delar den globala storheten "integer a" som i detta fall kan ses som en "multianvändbar" resurs. Om P_1 börjar exekveras före P_2 får variabeln a i sats 1 värdet 3. Om nu storheten a ej är skyddad och processen P_2 (som vi antar har högre prioritet till processtid) råkar starta efter denna sats, tilldelar processen P_2 a värdet 5. För processen P_1's del får variabeln c efter satsen 2 det felaktiga värdet 5 i stället för 3.

(3) om en resurs R_i användes men ej ändras av en process P, och om processen P's validitet i resultaten ej påverkas av en eventuell samtidig ändring av R_i, erfordras inget direkt skydd av detta förlopp.

Exempel på ovanstående kan vara vår filhanteringssekvens under antagandet att processen P_1 enbart läser information från F men ej återlagrar den enligt steg 3). Protektionsgraden enligt (1) kan tydligen till-

godoses om en process P_i får exklusiv tillgång till resursen så länge den utnyttjas av P_i. Protektionsgraden (2) tillgodoses genom att tilldela processerna s k "delad tillgång" (shared control) till resursen (med de restriktioner som sådan delning innebär). Någon speciell kontroll för protektion erfordras ej för fallet (3).

Så långt har vi helt allmänt diskuterat skydd av resurser för att garantera valida resultat om resurserna utnyttjas av parallella processer. När det gäller skydd av gemensamma resurser vilka kan klassificeras som tillhörande klassen "information", dvs procedurer (program) och data, är problemet av en helt annan dimension: I en s k "databasorienterad multiaccessmiljö" har användare normalt tillgång till en mängd gemensamma procedur- och datafiler. Vissa procedurer och vissa datafiler eller delar därav kan av lagliga, säkerhetstekniska eller personliga skäl ej utan urskillnad vara åtkomliga för samtliga användare.

För MULTICS systemet diskuterar Graham (4) tillämpning av koncentriska "protektions-ringar" där varje ring har en associerad mängd av programresurser. En process exekverande i ring i har access till programresurser i ringar j, $j \geq i$. Dennis et al. (3) diskuterar olika former för tillåten användning av en resurs av informationstyp:

X = exekverbar som en procedur
R = läsbar som data men ej exekverbar
XR = exekverbar som procedur och läsbar som data
RW = läs- och skrivbar som data
XRW = exekverbar som en procedur och läs/skrivbar som data

Frågor av denna art kommer vi att behandla närmare i samband med data- och filhanteringssystem.

3.4 Dataprocesser

Vi återkommer nu till problemet att närmare betrakta och definiera det abstrakta begreppet dataprocess. För enkelhets skull kallar vi det kort och gott "process" i fortsättningen. Trots att begreppet är grundläggande i databehandlingssammanhang har någon entydig definition av det ej accepterats i litteraturen.

Dennis et al. (3) anser att

"a process is that abstract entity which moves through the in-

> structions of a procedure as the procedure is executed by a processor."

Förutom att vissa processtillstånd kan urskiljas är varje process associerad med en ändlig mängd av resurser, s k resurslista, som den får utnyttja. Vidare definierar Dennis et al. (3) en beräkning (computation) vara mängden av processer med samma resurslista. Begreppet är av särskilt intresse för MULTICS-systemet men vi kommer ej att uttryckligen använda det här.

Dahm et al. (10) anser att en process är den sekvens av instruktioner, som utföres för en "användare" och att det innebär en mängd tillstånd där växlingar från ett tillstånd till ett annat bestämmes av programmet (eller programmen). IBM (11) använder begreppet "task" (uppgift) och definierar det som

> " a unit of work for the central processing unit from the standpoint of the control program; therefore the basic multiprogramming unit under the control program."

Man torde kunna säga att en process karaktäriseras av en följd av servicekrav till en begränsad delmängd av ett systems olika resurser. Denna följd är associerad med en överordnad aktivitet (användare eller styrfunktion). Under sin "livstid" genomgår processen en serie tillståndsförändringar.

Som framgår av ovanstående anser vi att en process ej enbart har att göra med processenheters arbete. Vi betraktar processenheten som en av en mängd olika resurser, som krävs för att processen skall kunna äga rum.

För att diskutera och beskriva processer i fortsättningen skall vi ta till hjälp ett Algol-liknande språk. Vi kommer även att diskutera parallella processer. Då Algol 60 enbart avser att beskriva sekvensiella processer kommer vi att utöka språket med ett antal begrepp som gör det möjligt för oss att diskutera parallella processer och samverkan mellan processer. Dessa begrepp har vi lånat dels från språken SIMULA I (12) och SIMULA 67 (13), dels från Dijkstras (5) arbete avseende "samverkande sekvensiella processer". Vi vill särskilt betona att det som vi lånat från SIMULA-språken på ett delvis felaktigt och kanske orättvist sätt speglar dessa språks syntax, verkliga omfång och syfte, men väl tjänar vårt syfte här att beskriva den komplexa mängd av händelser som inträffar när en mängd processer exekveras och samarbetar.

Vi skisserar nedan ett program (ofullständigt) i Algol (figur 3.4:1). Programmet skall tjäna som underlag för våra diskussioner om processer.

```
(1)   P: begin
(2)            integer totalpris, inkl moms;
               real momsen;
(3)            integer array antal, pris [1:10];
               ...
(4)      P1:   begin integer acksumma, i;
(5)                  acksumma: = 0;
(6)                  for  i: = 1 step 1 until 10 do
(7)                  acksumma: = acksumma + antal [i] * pris [i];
(8)                  totalpris: = totalpris + acksumma;
(9)            end;
               ...
(10)     P11:  begin
(11)                 inkl moms: = totalpris * (1 + momsen);
                     .........
(12)           end;
               ...
(13)  end
```

Figur 3.4:1

Vi kommenterar programmet kortfattat för att underlätta den fortsatta läsningen för personer med begränsade Algol-kunskaper.

(1) och (13) anger programblockets början och slut samt programmets namn P.

(2) deklarerar[1] heltalsvariablerna "totalpris" och "inklmoms", samt den "flytande" variabeln "momsen". Dessa är globala variabler och åtkomliga från alla inre begin .... end block i programmet. "Totalpris" skall ackumulera den totala kostnaden för en mängd artiklar. "Inklmoms" är "totalpris" multiplicerat med en faktor beroende på rådande omsättningsskattesats.

(3) här deklareras två vektorer (fält), vardera med tio element.

(4) här börjar ett programblock P1 och vi deklarerar de för blocket lokala variablerna "acksumma" och "i".

(5) acksumma nollställs.

(6) - (7) vi beräknar

$$\text{acksumma} = \sum_{i=1}^{10} \text{antal}[i] * \text{pris}[i]$$

(8) "totalpris" uppdateras med "acksumma".

(9) slut på P1-blocket.

(10) programblock P11 börjar.

(11) "inklmoms" beräknas.

(12) slut på programblock P1.

(13) slut på programblock P.

Även om hela programmet P. rad (1) - rad (13) kan betraktas som en helhet har vi här särskilt markerat två "inre" programblock P1 och P11. Resten av programmet kan betraktas som en (global) omgivning till P1 och P11. Normalt kommer satserna och underblocken i P exekveras strikt i sekvens, dvs först P1 och sedan P11. Låt oss betrakta programblocket P1. Vid exekvering av P1-instruktionerna säger vi att en process äger rum. Vi säger dessutom att processen, som tillsvidare ej har ett namn, tillhör klassen P1, där P1 är ett programblock. Enligt tidigare diskussioner har en process en följd av resurskrav. I detta fall har processen P1 följande primära resurskrav för att kunna utföras:

---

1) I praktiken innebär detta att när programmet exekveras, sker en utrymmesreservation i primärminnet för variablerna i fråga.

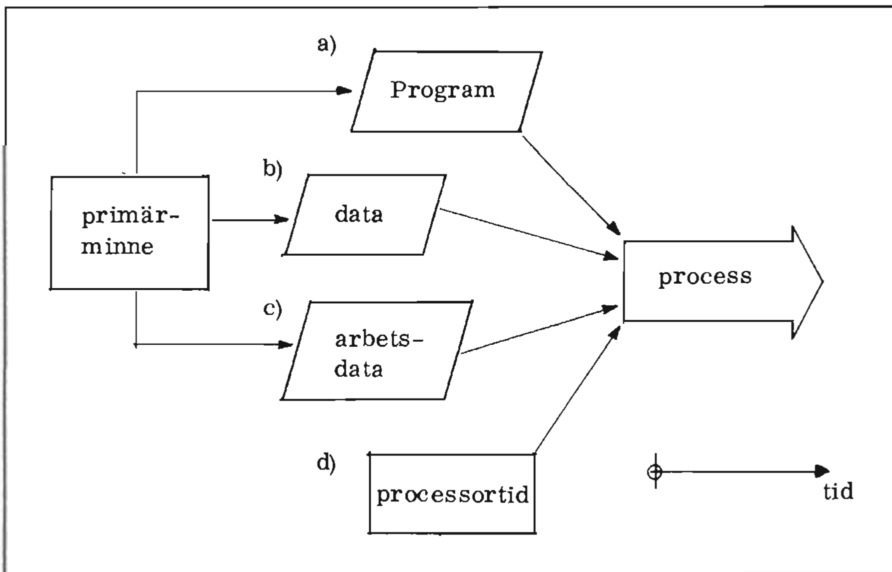
(a) program, i detta fall de instruktioner som representeras av satserna (4) - (9).

(b) data, i vårt fall i form av de globala storheterna totalpris, inklmoms, momsen, antal [1] - antal [10] och pris [1] - pris [10].

(c) arbetsdata för mellanresultat (arbetsutrymme) t ex för ackumulering av summan, "varvräkning" osv.

(d) processtid för exekvering av instruktionerna.

Om vi betraktar resursen (a) enligt ovan (program), är det uppenbart att härtill krävs primärminnesutrymme. Programmet är ju att betrakta som datamängd ur lagringssynpunkt. Samma är förhållandet för resursen " "data" (b) och "arbetsdata" (c). Resursbehovet kan därför schematiskt åskådliggöras såsom i figur 3.4:2.



Figur 3.4:2
Primärt resursbehov för att en process skall kunna äga rum (exekveras).

Processtillstånd

I figuren 3.4:2 har vi markerat rent maskinella resurser som rektanglar. Om samtliga resurser (a) - (d) är tillgängliga utföres processen och den

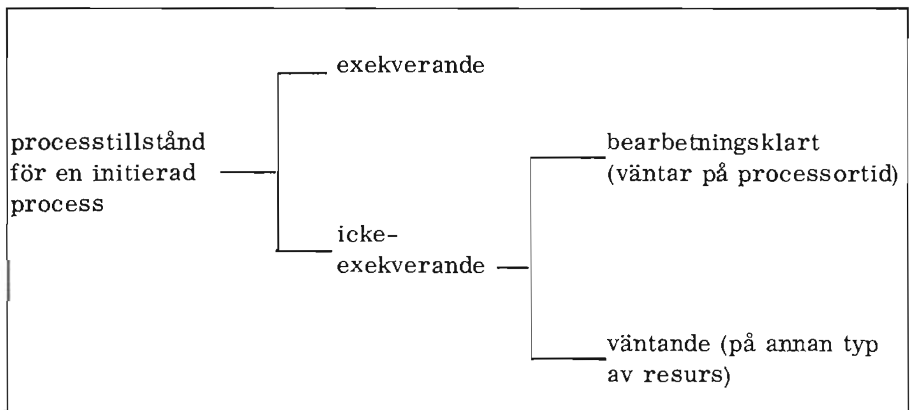
kan därvid sägas befinna sig i exekverande tillstånd. Det exekverande tillståndet har en tidsutsträckning som bestämmes av datorsystemets kapacitet och de operationer som skall utföras, i vårt fall en add-multiplikation av 10 st antalsvärden och prisuppgifter.

Påståendet att samtliga resurser krävs för att starta processen är dock ej helt riktigt. Processen som styres av programdelen P1 utför instruktionerna sekvensiellt och kräver vid varje tidpunkt endast en delmängd av de angivna resurserna. T ex i det femte steget utförs satsen

acksumma: = acksumma + antal [5] * pris [5];

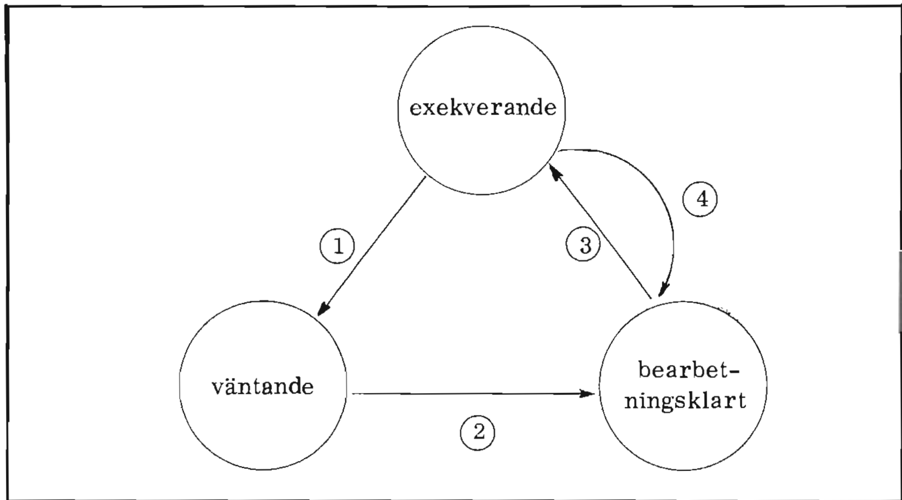
och om den satsen betraktas som instruktion för en "delprocess" behöver denna ej åtkomst till storheterna antal [i] och pris [i], $i \neq 5$. Var gränsen rörande resursbehovet skall dras för en process är en praktisk fråga som beror på syftet med analysen. Vi gör inget avsteg från ett generellt betraktelsesätt om t ex hela blocket P1 betraktas som instruktioner för styrning av en process.

Om någon av resurserna (a) - (d) saknas kan processen ej fortgå och säges då befinna sig i ett icke-exekverande tillstånd. Vanligen brukar man här skilja mellan om enbart processortid krävs eller om processen väntar på övriga resurser av typen program, data och primärminne eller om processen väntar på att bli aktiverad (ev av någon annan process eller extern händelse). I det första fallet är processen bearbetningsklar (eng ready) och i det senare fallet är processen i ett väntande (eng wait) tillstånd. Man kan här skilja mellan om processen väntar på att uteliggande resurskrav skall tillgodoses eller om processen är i väntan på någon extern signal t ex från en operatör eller användare.



Figur 3.4:3
Möjliga tillstånd för en process.

Växlingen mellan dessa olika tillstånd kan åskådliggöras enligt fig 3.4:4.



Figur 3.4:4
Växling mellan olika tillstånd hos en initierad process.

Växlingen (1) sker när en process passiveras i väntan på en eller flera uteliggande krav på resurser. När dessa (eller ett tillräckligt antal av dem) tillgodosetts sker en växling (2) till bearbetningsklart tillstånd och processen förblir där i väntan på resursen processortid. När processortid tilldelas sker växlingen till exekverande tillstånd (3). Växlingen (4) vill indikera möjligheten att processens exekveringsfas kan avbrytas av en annan process som har högre prioritet till resursen processortid.

Resurstillgång

Betrakta åter figur 3.4:1. Om samtliga resurser föreligger, kommer processen, styrd av instruktionerna i blocket P1, omedelbart att utföras. Om inga andra processer parallellt pågår i systemet kommer processen att utföras utan maskinavbrott.

I det generella fallet, där vi kan tänka om flera parallella processer, kan vid initiering av processen av klassen P1 en mängd saker inträffa och olika förutsättningar föreligga.

Till exempel:

(a) programmet P1 befinner sig ej i primärminnet (P kan tänkas vara

segmenterat). Detta förhållande kräver att P1 lokaliseras och inläses till lämpligt primärminnesutrymme, som då krävs för ändamålet. Inläsning av P1 görs vanligen av en speciell "systemprocess", säg av klassen X, som därför måste initieras. Här uppstår givetvis samma problem som vid initiering av processen av klassen P1. Medan inläsning av program för P1 ombesörjes befinner sig P1 i ett väntande tillstånd på resursen "program". [1]

(b) erforderligt primärminnesutrymme finns ej tillgängligt vid initiering. Processen får anmäla behovet av primärminne och sedan passivt avvakta till dess att minnesutrymme kan tilldelas. Administration och tilldelning av minne sker givetvis av en eller flera andra processer (systemprocesser) som själva också behöver initieras och kräver resurser.

(c) data befinner sig ej i primärminnet utan måste läsas in med en datatransportoperation. Datatransporten ombesörjes och initieras av en annan process (systemprocess) som själv krävs initierad.

3.5 Parallella processer

Vi observerar i programmet i figur 3.4:1 att addmultiplikationen där utföres sekvensiellt trots att den teoretiskt hade kunnat utföras parallellt. Praktiskt är detta till viss grad möjligt om flera processenheter finns att tillgå i datorsystemet. Mer specifikt menar vi att de processer som beskrivs av Algol-blocket

(3.5-1)
```
begin integer produkt;
      produkt:=antal [i] * pris [i];
      totalpris:=totalpris + produkt.
end;
```

skall kunna existera parallellt för i = 1, 2, ..., 10 (operationen är uppdelad på (onödiga) två satser enbart av illustrationsskäl). Om vi hade 10 processenheter till vårt förfogande skulle tidsåtgången för denna beräkning bli åtskilligt lägre än för den sekvensiella behandlingen enligt pro-

1) Man skulle även kunna säga att P1 ej är initierad innan erforderlig programresurs föreligger.

gram i figur 3.4:1. Eftersom alla processenheter i detta fall antas arbeta på samma primärminne reduceras tidsåtgången med en faktor 10 endast om multiplikationen tar, jämfört med access till primärminnet och addition, oändligt lång tid.

Innan vi går vidare i diskussionen om parallella processer är det lämpligt att föreslå ytterligare några begrepp som gör det möjligt att beskriva och manipulera processer.

Från SIMULA I (12) lånar vi notationen att deklarera en klass av processer

```
(3.5-2)  activity< identifierare > (< formell parameterlista >);
         < specifikationsdel> ;
         begin < satser>
         .....
         end;
```

Tillämpat på vårt exempel fås

```
(3.5-3)  activity P1 (i); integer i;
         begin
             integer produkt;
             produkt: = antal [i] * pris [i];
             totalpris: = totalpris + produkt
         end;
```

Parametern "i" tjänar i detta fall som en pekare (länk) till de data som skall bearbetas. Vi antar att totalpris, antal och pris är globala variabler resp fält vars namn är "kända" av processerna av klassen P1.

Vi antar vidare att en process etableras men ej initieras genon "process anropet"

```
< identifierare > (< aktuell parameterlista >)
```

eller i vårt fall

P1 (1), P1 (2), osv ... P1 (10)

Vi kommer också att ha behov av att referera till processer. En ny "typ-

deklaration" element införes därför (12). En elementvariabel kan ej tilldelas ett värde i vanlig Algolmening utan tjänstgör enbart som en pekare till en process. Om ingen process existerar har elementvariabeln värdet none. Antag att vi har deklarerat två elementvariabler "element A, B;". Följande två satser introducerar då två processer i systemet med namnen A och B

```
A: = P1 (1);

B: = P1 (2);
```

Vi behöver också en instruktion att initiera processer. Vi inför

```
activate X
```

där X antingen är namnet (elementvariabel) på en existerande process (som kan vara passiv) eller ett processanrop. Våra tio processer enligt (3. 5-3) skulle nu kunna initieras för parallell bearbetning genom

```
(3. 5-4)  for k: = 1 step 1 until 10 do
          activate P1 (k);
```

Programmet enligt fig 3.4:1 fortsätter därefter omedelbart med att utföra blocket P11 dvs

```
P11: begin
          inklmoms: = totalpris * (1 + momsen); ....
     end;
```

Givetvis medför detta ett felaktigt resultat då blocket P11 ej får utföras förrän alla tio processerna P1 (i), I = 1, 2, ..., 10, är avslutade. Problemet kan lösas på olika sätt. Utan att införa ytterligare mekanismer att manipulera processer kan följande lösning tänkas

```
(3. 5-5)  P: begin  integer totalpris, inklmoms, p, k;
                    real momsen;
                    integer array antal, pris [1:10];
                    activity P1 (i); integer i;
                    begin integer produkt;
                          produkt: = antal [i] * pris [i];
                          totalpris: = totalpris + produkt;
                          p: = p - 1;
```

```
        end P1; .......................
        p: = 10;
K:      for k: = 1 step 1 until 10 do activate P1 (k);
L:      if p > 0 then goto L;
P11:    begin inklmoms: = totalpris * (1 + momsen); ....
        end; .....
        .............................
end P
```

Huruvida de aktiverade processerna P1(1) - P1(10) verkligen utförs parallellt beror på antal processenheter och diverse styrprogramegenskaper hos datorsystemet. Om bara en processenhet finnes, utförs de givetvis i sekvens. Om t ex två processenheter finns kan utförandet tillgå så att processenhet 1 utför processerna P1(1) - P1(5) och processenhet 2 utför P1(6) - P1(10) i serie. Exakt hur processtiden fördelas mellan processer är en planerings- och prioritetsfråga som vi kommer att diskutera senare. Här konstaterar vi bara att ett antal parallella processer startats vilkas utförandetid ej är bekant, dvs de fortskrider med okänd hastighet. I det generella fallet kan sådana processer befinna sig långa perioder i ett inaktivt tillstånd i väntan på program, data, minnesutrymme eller annan service från systemets resurser. Icke utnyttjad processtid tilldelas andra parallellt existerande processer som befinner sig i ett bearbetningsklart tillstånd.

Programmet P (3.5-5) bildar under sitt utförande själv en process som vid satsen K initierar 10 nya processer - delprocesser till P. Vi antar att dessa delprocesser initieras med prioritet så att de kan utföras före nästa sats (L) i huvudprocessen. I annat fall kommer satsen L att utföras i det oändliga. Vi har antagit att satsen L kan komma att utföras antingen när någon av P1(1) - P1(10) befinner sig i ett väntande tillstånd eller givetvis när de är avslutade. Varje process P1(i) minskar den globala storheten p med 1 och satsen P11 kan därför initieras endast nät p nedräknats till 0.

Med hjälp av programmet P (3.5-5) kan vi belysa ytterligare en komplikation som kan uppträda vid parallella processer när processer med olika prioritet exekveras eller när flera processenheter arbetar på ett gemensamt primärminne. De globala storheterna "totalpris" och "p" kan här betraktas som exklusivt användbara resurser. Om mer än en process vid en viss tidpunkt tillåtes manipulera dessa finns risk för att ett felaktigt resultat erhålles.

Satsen "totalpris: = totalpris + produkt" för processen P1(i) kan tänkas representerad av en-adress-instruktionssekvensen:

(a) tag innehållet i minnescellen "totalpris" till ackumulatorregistret (AR)

(b) addera innehållet i minnescellen "produkt" (för process P1(i)) till AR

(c) lagra AR's innehåll i minnescellen "totalpris".

Om utförandet av denna sekvens av någon (främmande) anledning brytes mellan (b) och (c) och kontrollen (processenheten) överföres till en parallell, säg P1(j), process där operationerna (a), (b) och (c) utföres, kommer "totalpris" att få ett felaktigt värde när processen P1(i) efter en tid återfår kontrollen och utför instruktionen (c). Ett motsvarande fenomen kan uppträda om två processer exekveras parallellt på skilda processenheter. Accesskonflikt kan då uppstå till den gemensamma storheten "totalpris".

Dijkstras "semaforer"

Vi har närmast ovan exemplifierat s k "kritiska faser" vid parallell exekvering av processer. Under en process' kritiska fas av detta slag får endast denna process vara under exekvering, dvs under denna fas utesluts utförandet av andra processer. Givetvis kan sådan inbördes uteslutning vara nödvändig av en mängd ytterligare orsaker än vad som exemplifierats ovan.

Problemet har studerats av Dijkstra ((5) och (9)) som introducerat det teoretiska begreppet "semafor" som ett hjälpverktyg vid synkronisering av processer.

Semaforer är heltalsvariabler (integer) med ett speciellt användningssyfte. Semaforer kan vara globala eller lokala till en process. Operationer på semaforer kan endast utföres genom två primitiva instruktioner, de s k P- och V-operationerna. När semaforer används vid "inbördes uteslutning av processers exekveringstillstånd" kan de endast anta värdena 0 och 1. Generellt kan de dock betraktas som icke-negativa heltal.

P- och V-operationerna definieras enligt (Dijkstra (5) sid 68)

> V-operation: en V-operation är av formen V (< semafor-identifierare >);. Dess funktion är att öka värdet för motsvarande semafor med 1. Operationen betraktas som icke-avbrytbar av andra processer.

P-operation: en P-operation är av formen P (<semafor-identifierare>);. Dess funktion är att, om semaforens värde är större än 0, minska det med ett. Om värdet är ≤ 0 inordnas denna operation (dvs process) i en kö till samma semafor och subtraktion med ett utföres så snart värdet blivit > 0. Operationen är, liksom för V-operationen, icke avbrytbar och introducerar dessutom en potentiell fördröjning av processen.

Programmet 3.5-5 kan nu förbättras med hjälp av P- och V-operationerna:

```
(3.5-6)  P: begin  integer totalpris, inklmoms, p, k, uteslut;
                   real momsen;
                   integer array antal, pris [1:10];
                   activity P1(i) integer i;
                   begin integer produkt;
                   produkt: = antal [i] * pris [i];
                   P (uteslut);
                   totalpris: = totalpris + produkt;
                   V (uteslut);
                   V (p)
                   end; .....
                   uteslut: = 1; p:= -9;
         K:        for k: = 1 step 1 until 10 do activate P1(k);
         P11:      begin
         L:        P (p);
                   inklmoms: = totalpris * (1 + momsen);
                   V (p);
                   .....
                   end
```

I detta program har vi infört två semaforer - de globalt deklarerade variablerna "p" och "uteslut". Processens P1(i) kritiska fas "totalpris: =...;" görs exklusivt utförbar genom att den föregås av en P-operation på semaforen "uteslut". Variablen "uteslut" initieras med värdet 1. Vi inser att om samma variabel initierats med värdet N, $N > 0$, hade vi tillåtit N processer simultant utföra denna kritiska fas. Semaforen "p" användes för att förhindra att satsen P11 utföres innan samtliga processer av klassen

P1 avslutats. Den initieras med värdet [1-(antal P1-processer)] dvs -9. När en P1 process avslutas utför den en V-operation på denna semafor. När samtliga processer P1 avslutats har p värdet 1 och P-operationen på rad L kan utföras.

Den praktiska implementeringen av dessa semaforer förtjänar att kommenteras. Problemet kan lösas genom att varje semaforoperation innebär ett en av anrop till en överordnad "styrprocess" (dvs operativsystemets styrfunktioner) som är ensam auktoriserad myndighet att manipulera semaforer.

Problem av ovan beskrivet slag har existerat långt före lanseringen av semaforer och lösningar starkt påminnande om användning av semaforer har varit kända sedan början av 60-talet då multiprogrammeringstekniken blev allmänt känd. Dijkstra och hans medarbetare måste dock krediteras för en pedagogiskt väl genomtänkt presentation av problemställningen kring parallellt samarbetande processer. Även om vi ej helt kommer att följa Dijkstras metodik vid tillämpning av semaforer kommer vi även i fortsättningen använda dessa begrepp (dock uppblandade med några SIMULA-begrepp).

3.6 Samverkan mellan processer

I föregående avsnitt använde vi oss av två slag av semaforer. Vid inbördes uteslutning m a p exekvering användes semaforen "uteslut" som där kunde anta värdena 0 och 1. Vid synkronisering av de olika processernas aktiva faser användes semaforen "p", vars initialvärde bestämdes av hur många händelser av ett visst slag behövde inträffa för att en annan, väntande, process skulle få fortsätta. Man kan därför betrakta semaforen "p" som en "privat semafor" till en process - i vårt fall processen P11. Varje process i systemet kan på samma sätt ha en eller flera privata semaforer med vars hjälp dess förlopp och samverkan med andra processer kan styras. På grund av att det i Algol icke är tillåtet att från ett yttre block ha åtkomst till datastorheter deklarerade i ett inre block måste, åtminstone framställningsmässigt, semaforerna deklareras som globala storheter till berörda processer. Semaforteorien säger heller ingenting om planering och köadministration av processer. P- och V-operationerna antas (från processens synvinkel) som momentana och två sådana operationer kan ej utföras simultant. Två P-operationer på samma semafor t ex

```
process 1: .......... P(resurs); ......
process 2: .......... P(resurs); ......
```

skapar, om de ges samtidigt, en kö till semaforen "resurs" där en av processerna placeras före den andra, Manipulation av denna kö, t ex en omsortering av ordningsföljden, är "oåtkomlig" för processerna. Enligt semaformetodiken betraktas processer som cykliska och ständigt existerande i systemet. Övergången mellan aktiva och inaktiva faser styres av deras P-operationer på lämpliga semaforer. En "död" process är givetvis inaktiv och motsvaras av att processen ifråga har utfört en P-operation på en privat semafor, säg, "go", vars värde vid tillfället var 0. När processen skall sättas igång utför en annan, initierande, process en V-operation på semaforen "go".

Vi skall nu med ett enkelt exempel illustrera användning av semaforbegreppet vid samverkan mellan två processer. Antag en process X som på något sätt framställer (mottar eller genererar) meddelanden av fix längd och sedan överlåter dem till process Y för bearbetning. Antag vidare att tiden för framställning liksom tiden för bearbetning av ett meddelande varierar på så sätt att genomsnittstiden för framställning av ett meddelande är längre än den genomsnittliga behandlingstiden. Vi har här den i datorsammanhang vanliga situationen att en buffert, rymmande ett antal meddelanden, mellan bägge dessa processer erfordras för att de så litet som möjligt skall uppehålla varandras arbete. Bufferten verkar sålunda utjämnande på variationerna hos processernas framställnings- resp bearbetningstider[1]. Vi inser att om bufferten görs mycket stor (dvs rymmer ett stort antal meddelanden) så är risken liten för att X-processen skall bli fördröjd, på grund av brist på ledig plats att lagra ett meddelande i. Omvänt, om bufferten endast rymmer ett meddelande kan dessa processer överhuvudtaget ej arbeta parallellt om vi förutsätter att:

- X behöver en buffertplats för att börja framställning av ett meddelande
- Y bearbetar meddelandet i bufferten och "frisläpper" motsvarande utrymme först när meddelandet färdigbearbetats

Situationen illustreras i figur 3.6:1, som visar hur en buffert, förutom plats för lagring av meddelanden, kräver en mängd administrativa data och normalt även plats för länkadresser. Vi tänker oss att länkadresserna i detta fall pekar på nästa upptagna resp lediga plats i bufferten. Vi har således två "kedjor" av platser - en kedja för upptagna platser som innehåller meddelanden vilka ännu ej bearbetats eller befinner sig under bearbetning och en kedja för platser som är lediga.

Vi inser att när processen X eller Y skall söka efter ledig plats resp söka efter nästa meddelande att bearbeta eller infoga ett meddelande i bufferten

1) Detta problem behandlas analytiskt bl a i (15), kap 2.

Administrativa data om buffert

Antal buffertplatser (N)
Antal upptagna buffertplatser (NU)
Antal lediga buffertplatser (NL)
Adress till första lediga plats (AFL)
Adress till första upptagna plats (AFU)
Adress till sista upptagna plats (ASU)

Buffertplatser och länkadresser

Plats för meddelande	Länkadress
- " -	"
- " -	"
- " -	"
- " -	"

Figur 3.6:1

resp friställa en plats i densamma, kommer processerna att behöva manipulera dels buffertens administrativa data och dels länkadresserna. Eftersom X och Y här anses som parallellt exekverande processer kan dylik manipulation av administrativa data och länkadresser leda till felaktigheter om ej hänsyn tas till att skydda dessa globala data mot "samtidig" åtkomst och ändring från både X och Y. Därför införes en semafor "buffadm", som kan anta värdena 0 och 1, och som medger (enligt tidi-

gare resonemang) en ömsesidig uteslutning av X och Y processernas ex exekvering under vissa kritiska faser. Vi skisserar nu ett program som endast utvisar sådana data och operationer som är av relevans för samverkan mellan dessa våra två processer.

```
(3.6-1) begin  integer NU, NL, buffadm;
        comment NU anger antal upptagna platser;
        activity X;
        begin
        X1:    P(NL);
               P (buffadm);
               < sök efter ledig plats, ändra länkadress m m>;
               V (buffadm);
               <framställ meddelande >;
               P (buffadm);
               <infoga meddelandet i kön för bearbetning, uppdatera
                 administrativa data>;
               V (buffadm);
               V (NU);
               goto X1
        end;
        activity Y;
        begin
        Y1:    P (NU);
               <sök efter nästa meddelande att bearbeta och utför be-
                 arbetning>;
               P (buffadm);
               <avlägsna bearbetat meddelande, uppdatera administra-
                 tiva data och länkadresser>;
               V (buffadm);
               V (NL);
               goto Y1
        end;
```

```
buffadm: = 1;
NU = 0; NL = N; <initialisera länkadresser m m>;
activate X;
activate Y;
.......
end
```

Vi ser att processen X styrs av semaforen NL. Om NL = 0, dvs bufferten är full, får X vänta till dess Y-processen utfört en V (NL) operation. På samma sätt styrs Y-processen av semaforen NU. Om NU = 0 är inget meddelande färdigt att bearbetas och Y ligger i "vänteloop". Observera att Y-processen ej behöver utföra en P (buffadm) operation för att bearbeta nästa meddelande i bufferten eftersom bearbetningen ej påverkar länkadresser och övriga administrativa data. Endast när meddelandet avlägsnas och buffertplatsen frigöres (av Y) behövs en samtidig uteslutning av X-processens eventuella möjlighet till buffertadministration.

Programmet (3.6-1) exeplifierar användningen av dels binära semaforer (buffadm) dels andra variabler (NL och NU) för styrning av samarbetet mellan två processer. Vi kan också betrakta NL och NU som tillståndsvariabler vilka kan manipuleras med P- och V-operationerna.

Det är lätt att inse att resonemanget kan utvidgas till mer än två parallella, samverkande processer. Därvid måste givetvis ett motsvarande antal nya semaforer (tillståndsvariabler m m) införas. Om vi vill arbeta med meddelanden av variabel längd kompliceras programmeringen något dock utan att principerna ändras. Dijkstra redovisar i (5) även ett exempel där semaforbegreppet används vid "konversation" mellan en operatör (manuell process) och ett antal parallella processer, via en semi-duplex datakanal.

3.7 Resursadministration. Låsningsproblemet

Allmänt

Vi har tidigare noterat att ett databehandlingssystem kan betraktas som en resursmängd, och att resurser förbrukas vid exekvering av en process. Vi gjorde också skillnad mellan multianvändbara och exklusivt användbara resurser. Exklusivt användbara resurser kallas ofta även för

seriellt användbara resurser då processer som vill använda dem måste göra det i serie. I föregående avsnitt har vi demonstrerat hur ett sådant seriellt utnyttjande av resurser kan styras genom användning av semaforer, P- och V-operationer. Om samtliga exklusivt användbara resurser som en process kan komma att kräva skulle tilldelas vid processens initiering och ej avbokas förrän processen avslutats, skulle resursutnyttjandet (dvs datorsystemets utnyttjande och effektivitet) bli dåligt.

Då processer ofta nyttjar samma resurser, fast vid olika tidpunkter, kan en sådan fast tilldelning också medföra en låsning av hela systemet, varvid inget nyttigt arbete skulle bli utfört.

Resursadministrationen kan sägas omfatta tre huvudproblem:

(1) att planera och initiera olika resursförbrukande processer så att, med hänsyn till processernas precedensrelationer och diverse tidskrav (avseende startordning), så mycket arbete[1] som möjligt utföres av systemet.

(2) att administrera den löpande tilldelningen av resurser till de olika, normalt parallellt pågående, processerna på ett sådant sätt att det totala resursutnyttjandet[1] blir så högt som möjligt.

(3) att utföra planeringen och administrationen under beaktande av att "låsning" undviks och dels avhjälps om sådan inträffat.

Problem (1) och (2) kommer närmare att diskuteras dels i samband med jobbövervakning och jobbinplanering (kap 5 och 6), dels i samband med olika styrprogramfunktioner (kapitel 7). Detta avsnitt tänker vi huvudsakligen ägna åt "låsningsproblemet" som kan uppträda vid parallella processers utnyttjande av exklusivt användbara resurser.

Resurser

Vi antar här att ett systems resurser kan beskrivas genom vektorn R

(3.7-1)

$$R = \begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ \vdots \\ r_m \end{bmatrix}$$

1) Vi är medvetna att här krävs en ytterligare specificering vad som avses. Från fall till fall kan här olika kriteria tillämpas.

där r_i anger "antalet resurser" av typen i som systemet förfogar över. Man skulle också kunna säga att systemet omfattar resurserna R_1, R_2, ..., R_m och att r_i anger antalet "tilldelningsenheter" som resursen utgörs av. Till exempel kan ett 32 Kbytes primärminne betraktas (beroende på tilldelningsprinciper) som en resurs med 32 tilldelningsenheter av storleken 1 Kbytes, eller 128 enheter om 256 bytes osv. En radskrivare, om systemet bara omfattar en dylik, måste givetvis betraktas som en tilldelningsenhet.

En process' resurskrav

En process' P_i krav på systemets resurser vid en tidpunkt t betecknar vi med $p_i(t)$.

(3.7-2) $$p_i(t) = \begin{bmatrix} p_{1i}(t) \\ p_{2i}(t) \\ p_{3i}(t) \\ \vdots \\ p_{mi}(t) \end{bmatrix}$$

där $p_{ji}(t)$ anger resursbehovet av typen j för processen P_i vid tidpunkten t. Givetvis måste resursbehovet för varje process vara mindre än eller lika med systemets resurser av motsvarande typ, dvs

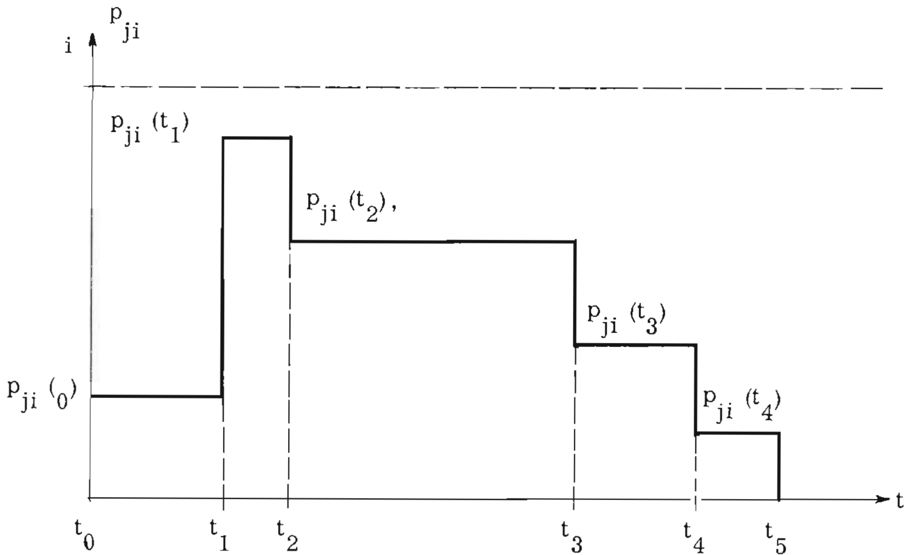
(3.7-3) $$p_{ji}(t) \leq r_i \quad \text{för alla processer i och resurser j}$$

När en process P_i exekveras kan den sägas genomgå en serie tillståndsförändringar avseende i dess resursbehov

$$p_i(t_0) \rightarrow p_i(t_1) \rightarrow p_i(t_2) \rightarrow \text{osv}$$

$$t_i < t_{i+1} \qquad (i > 0)$$

Till exempel kan krav på resurser av typen j tänkas variera enligt figur 3.7:1.



Figur 3.7:1
En process P_i resursbehov av typen j som funktion av tiden.

Motsvarande förlopp kan skisseras för processens P_i behov av andra resurser $k \neq j$. Det bör poängteras att tidpunkterna t_1, t_2, ... osv ej är kända om processen exekveras parallellt med andra processer och därvid konkurrerar om systemets resurser[1].

Om vi för att förenkla diskussionen bortser från att processen P_i även nyttjar andra resurser kan enligt 3.7:1 processens livslängd uttryckas som $(t_5 - t_0)$. Under denna tid utnyttjar processen P_i resursen j i genomsnitt.

$$\sum_{k=0}^{4} (p_{ji}(t_k) \cdot (t_{k+1} - t_k))/(t_5 - t_0)$$

Utnyttjandet varierar mellan minimivärdet $p_{ji}(t_0)$ och maximivärdet $p_{ji}(t_1)$.

---

1) Vissa fördröjningar kommer att inträffa på grund därav. Se även (15), där detta problem behandlas.

Ett enkelt tilldelningsförfarande hade varit att vid t_0 tilldela P_i dess maximala behov av resursen i fråga, dvs här $p_{ji}(t_1)$. Man inser dock att detta ej är särskilt ekonomiskt (även om principen fortfarande är vanlig när det gäller operativsystems resurstilldening till applikationsprogram). Om resursen j kan tilldelas på ett sådant sätt som svarar mot processens behov vid olika tidpunkter kan icke utnyttjade delresurser tilldelas andra processer. Från figur 3.7:1 framgår att P_i begär utökade resurser vid t_0, t_1 och sedan successivt återlämnas dessa vid t_2, t_3, t_4 och t_5. Vid t_5 är samtliga resurser av typen j återlämnade.

Låsning

En låsningssituation kan inträffa om man tillåter processer att begära ytterligare resurser utan att redan tilldelade resurser återlämnas. Standardexemplet kan skissas med hjälp av två skolelever A och B som bägge arbetar på var sin teckning och därvid behöver en svart och en röd penna för att färdigställa teckningen. Om A lägger beslag på bägge pennorna (vi antar att endast två dylika pennor finns i klassen) kommer B att få sitta och vänta medan A gör sin teckning färdig. Om, däremot, A målar med den svara pennan och B med röda och sedan A behöver den röda utan att lämna ifrån sig den svarta och på samma sätt B behöver den svarta utan att lämna ifrån sig den röda inträffar ett "låst tillstånd". Ingen av de två eleverna kan nu få sin teckning färdig.

Ett annat sätt[1)] att exemplifiera låsning är att betrakta processers med tiden varierande belastning på en resurs (vi inskränker oss till en resurs, för enkelhets skull). Antag att resursen är primärminnesutrymme och att vi har 64 Kbytes att disponera för parallell bearbetning av två processer P_a och P_b. Vi antar att det enda vi från början vet är att P_a-processens maximibehov är 48 Kbytes och P_b-processens 36 Kbytes. Vi kan sedan givetvis vid varje tidpunkt följa upp hur mycket som tilldelats resp process.

Antag att vid en viss tidpunkt följande situation råder:

Situation 1:

Process	Aktuell tilldelning	Maximibehov	Aktuellt ytterligare behov
P_a	32	48	16
P_b	16	32	16
Summa	48		
Ledigt utrymme	16		

1) Dijkstra betraktar i (5) en liknande situation (the deadly embrace)

Denna situation anses "säker" eftersom bägge processerna har möjlighet att avsluta sin bearbetning (deras ytterligare eventuella resursbehov kan täckas - dock ej samtidigt). Skulle vi tilldela processen P_a ytterligare 1 Kbytes är situationen fortfarande "säker" även om endast P_a nu kan fortsätta.

Om vi däremot tilldelar både P_a och P_b 1 Kbytes ytterligare primärminne fås följande

Situation 2

Process	Aktuell tilldelning	Maximibehov	Aktuellt ytterligare behov
P_a	33	48	15
P_b	17	32	15
Summa	50		
Ledigt utrymme	14		

Det inses att detta är ett "osäkert tillstånd". Om ingen av processerna återlämnar några resurser (minnesblock om 1 Kbytes) innan de kräver sitt maximibehov kommer en låsning att inträffa. Bägge behöver ytterligare 15 Kbytes men endast 14 Kbytes finns att tillgå (såvida ej åtgärder såsom "swapping" eller avslutning av någon process tillgripes).

Att undvika låsningsrisken

När det gäller låsning vid parallell exekvering av processer är de primära frågorna (5):

(a) vilka förutsättningar krävs för att en ny process skall kunna initieras?

(b) vilka förutsättningar krävs för att man utan risk för låsning skall kunna tilldela resurser till en process?

Svaret på (a) är givetvis att processens maximibehov ej får överskrida datorsystemets resurser R för någon resurstyp.

Svaret på fråga (b) är att tilldelning av resurser till olika processer kan fortgå utan direkta komplikationer så länge som ett "säkert tillstånd" existerar. Om ett resurskrav resulterar i ett osäkert tillstånd måste tilldelning av motsvarande resurs uppskjutas.

Att konstatera om ett tillstånd är säkert eller ej innebär en undersökning

huruvida samtliga initierade processer kan garanteras att exekvera färdigt. En enkel algoritm anges härför i (5). Algoritmen börjar med att undersöka om det finns minst en process vars ytterligare behov är mindre än ledigt resursutrymme. Om så är fallet undersöks de övriga parallella processerna som om den nyssnämnde hade exekverat färdigt och återlämnat samtliga resurser som den haft tilldelade. Algoritmen kan skrivas som

```
(3.7-1) begin
        integer array ytterligare krav, aktuell tilldelning [1:N];
        integer ledigt utrymme, minneskapacitet; Boolean säkert tillstånd;
        Boolean array tveksam [1:N];
        comment antag att vid denna tidpunkt N parallella processer
        existerar och att "ledigt utrymme" anger antalet disponibla
        minnesblock; ....
        <initiering av variabler>;
        for i: = 1 step 1 until N do tveksam [i]: = true;
     L: for i: = 1 step 1 until N do
                 if tveksam [i] ∧ ytterligare krav [i] ≤ ledigt utrymme then
        begin tveksam [i]: = false;
                ledigt utrymme: = ledigt utrymme + aktuell tilldelning [i];
                goto L
        end;
        säkert tillstånd: = ledigt utrymme = minneskapacitet;
        ..........
        end
```

Algoritmen ovan undersöker förutsättningslöst varje situation. Om vi kan utgå ifrån att ett säkert tillstånd existerar och uppgiften är att undersöka huruvida ett krav på ytterligare resurser från en process "i" kan leda till ett osäkert tillstånd behöver bara konstateras om processen "i" är med bland de processer som med säkerhet kan avslutas. Efter det har vi återvänt till ett "säkert tillstånd".

Andra metoder

När systemet väl hamnat i en "låst situation" kan en generellt tillämpbar

lösning av konflikten ej föreslås. Från fall till fall måste olika åtgärder studeras beroende inte minst på aktuellt operativsystems funktionsrepertoir.

Havender (8) diskuterar olika metoder att undvika låsning:

Metod 1: Om en process "håller" en resurs R_1 och sedan begär en reresurs R_2, tilldela R_2 endast om i) inga andra processer i väntande tillstånd existerar som tidigare tilldelats R_2 och som ii) senare kommer att kräva resursen R_1. Innebörden av denna metod är att alla resurser skall begäras i samma ordning. Det är dock mindre ofta som denna metod kan tillämpas i praktiken.

Metod 2: Begär samtliga resurser för en process samtidigt. Processen förblir i "vila" till dess samtliga resurser erhållits. Metoden är givetvis enkel att tillämpa men föga attraktiv ur effektivitetssynvinkel.

Metod 3: Vid begäran om nya eller utökade resurser återlämna samtliga resurser först och begär sedan en "ny uppsättning". Även denna metod är enkel ur tillämpningssynvinkel men kan orsaka mycket administrationsarbete hos operativsystemet.

Metod 4: Om begäran om en resurs refuseras och om metod 1 ej är tilllämpbar undersök andra handlingsalternativ och vänta ej på resursen i fråga samtidigt som andra resurser uppehålles. Metoden ställer höga krav på programmerare och torde endast i undantagsfall kunna tillämpas.

Havender (8) diskuterar tillämpning av dessa metoder vid tilldelning av

- datamängder (data-sets)
- perifera enheter (devices)
- primärminne (regioner)

för jobb och jobbsteg (med tanke på IBM OS/360). Det föreslås att

- metod 2 tillämpas vid tilldelning av datamängder. Anledningen är att vissa datamängder är globala för jobbet I och används (läsning/skrivning) av de olika jobbstegen. Skulle ytterligare datamängder begäras av ett jobbsteg föreligger låsningsrisk eftersom ett annat jobb kan "hålla" dessa datamängder och samtidigt efterfråga någon global datamängd hos jobbet I.

- metod 3 föreslås för tilldelning av perifera enheter vilket innebär att vid varje jobbstegs slut samtliga perifera enheter återlämnas och sedan en ny uppsättning krävs. Metoden skapar dock problem då onödigt monteringsarbete riskeras. Genom lämplig planering kan dock risken minimeras.

- metod 3 föreslås för tilldelning av primärminne för jobbstegen, vilket lätt inses är att föredra.

Normalt sker tilldelning av de olika resurserna vid bestämda tidpunkter, t ex vid jobbets initiering och vid initiering av de olika jobbstegen. I (8) diskuteras därför en s k "kombinerad tilldelning".[1] Frågan är: skall dessa tre slag av resurser tilldelas i någon särskild ordning för att undvika låsningsrisk, och i så fall vilken? Eftersom vissa datamängder tilldelas för hela jobbet vid dess början måste de även för varje jobbsteg tilldelas först. Vid val av turordning mellan primärminne och perifera enheter påverkas valet av att jobbinitiering och tilldelning av perifera enheter ombesörjes av systemprogram som i detta fall arbetar i det primärminnesutrymme som sedan skall nyttjas för applikationsprogrammet. Detta avgör valet, dvs primärminne bör här tilldelas före perifera enheter. Poängteras bör att detta endast gäller för operativsystem som arbetar efter principer liknande OS/360 MVT.

Habermann's "låsningsteorem"

Baserat på Dijkstras arbeten (5) har Habermann (7) utvecklat tre teorem som berör låsningsproblematiken. Vi skall här kortfattat redovisa dem. Ett system av n parallella processer $P_1, \ldots, P_n$ och m resurstyper $R_1, \ldots, R_m$ antages. Systemets totala resurser beskrives genom vektorn R (se 3.7-1). En process' P_i i tiden varierande krav på systemets resurser anges genom $p_i(t)$ (se 3.7-2. P_i's maximala krav på systemets resurser anges med vektorn b_i

(3.7-4) $$b_i = \begin{bmatrix} b_{1i} \\ b_{2i} \\ \vdots \\ b_{mi} \end{bmatrix}$$

där $b_{ji} = \max_t p_{ji}(t)$. En matris B definieras nu som

---

1) Speciellt med tanke på IBM's operativsystem OS/360 MVT.

$$(3.7\text{-}5) \quad B = [b_1, b_2, \ldots, b_n] = \begin{bmatrix} b_{11} & \ldots & b_{1n} \\ b_{21} & & \\ \cdot & & \\ \cdot & & \\ \cdot & & \\ b_{n1} & \ldots & b_{nm} \end{bmatrix}.$$

och utvisar maximikraven på systemets resurser för samtliga n parallella processer.

Vi antar att systemet vid tiden t har tilldelat processerna resurser som beskrivs av P(t)-matrisen

$$(3.7\text{-}6) \quad P(t) = [p_1(t), \ldots, p_n(t)] = \begin{bmatrix} p_{11}(t) & \ldots & p_{1n}(t) \\ \cdot & & \cdot \\ \cdot & & \cdot \\ \cdot & & \cdot \\ \cdot & & \cdot \\ p_{m1}(t) & \ldots & p_{mn}(t) \end{bmatrix}$$

R-vektorn är konstant, såvida ej systemets resurser på något sätt ändras under det betraktade tidsintervallet. För en given mängd processer kan B-matrisen antas konstant i tiden. P(t) ändras givetvis i tiden genom att processer tilldelas och återlämnar resurser. Om en ny process infogas i mängden av parallella processer utökas B- och P(t)-matriserna med var sin kolumn.

Ett allokeringstillstånd definieras nu som trippeln

$$(3.7\text{-}7) \qquad (R, B, P(t))$$

Problemet är att avgöra huruvida ett sådant tillstånd är "säkert" dvs saknar låsningsrisk. Först definieras innebörden av relationsoperatorer tillämpade på vektorer och matriser[1]. Låt u, v vara två m-dimensionella vektorer och F, G två matriser (m x n). $u \leq v$ innebär att $\forall_i u_i \leq v_i$ och $u < v$ innebär $(u \leq v) \wedge (\exists_i u_i < v_i)$. På samma sätt om $F = [f_1, f_2, \ldots, f_n]$, $G = [g_1, g_2, \ldots, g_n]$ där f_i och g_i är m-dimensionella vektorer så innebär $F \leq G$ att $\forall_k f_k \leq g_k$ och $F < G$ att $(F \leq G) \wedge (\exists_k f_k < g_k)$.

---

1) Betydelsen för vissa symboler: $\forall$ = för alla. $\exists$ = det existerar. $\nexists$ = det existerar ej.

På motsvarande sätt definieras relationsoperatorerna $\geq$ och $>$.

Ett realiserbart (allokerings-) tillstånd är ett tillstånd där följande tre villkor är uppfyllda

V1: $\forall_k b_k \leq R$

V2: $P(t) \leq B$ (för alla t)

V3: $\rho(t) \geq 0$ (för alla t)

där $\rho(t) = R - \sum_{k=1}^{n} p_k(t)$ = systemets outnyttjade resurser vid tiden t. Vi observerar också att V1, V2 och V3 är oberoende av kolumnordningen i matriserna B och P(t).

Hur en situation kan inträffa som förorsakar risk för låsning har vi beskrivit tidigare i samband med exemplet om minnesutrymmet 64K i och två processer P_a, P_b. Villkoret för att ett tillstånd skall kallas "säkert" är här analogt.

Vi söker en process, säg P_{k_1}, sådan att $\rho(t)$ tillåter den avslutas, dvs

$$b_{k_1} - p_{k_1}(t) \leq \rho(t).$$

Sedan söks en process P_{k_2}, sådan att

$$b_{k_2} - p_{k_2}(t) \leq \rho(t) + p_{k_1}(t)$$

osv. Låt S vara en ordnad sekvens av processer och låt s(k) vara ordningsnumret för process P_k i denna sekvens. S kallas en fullständig sekvens om den innehåller samtliga processer P_i, $i = 1, \ldots, n$.

Ett säkert tillstånd definieras enligt

(3.7-8) Ett realiserbart tillstånd (R, B, P(t)) är säkert om det existerar en fullständig sekvens sådan att

V4: $(\forall P_k \in S)\ b_k \leq \rho(t) + \sum_{s(l) \leq s(k)} p_l(t)$

för alla t i tidsintervallet.

En fullständig sekvens enligt V4 kallas också för en "säker sekvens". Det

mest ogynnsamma fall som kan inträffa är att $p_i(t)$ för varje process är en ständigt icke minskande funktion av t ända till dess b_i uppnåtts. Ett säkert tillstånd kräver att även detta fall kan lösas.

(3.7-9) Teorem 1: Om ingen process P_i återlämnar resurser förrän den har tilldelats sitt maximibehov b_i, kan en låsning undvikas om och endast om allokeringstillståndet är säkert.

Vid ett säkert tillstånd är det ej nödvändigt att tilldela processerna resurser enligt deras ordning i en fullständig sekvens. Normalt existerar många möjliga, fullständiga sekvenser. Totala antalet sekvenser som kan definieras på n processer är $n!$ Dock behöver samtliga fall ej undersökas för att konstatera om ett tillstånd är säkert eller ej. Med hjälp av teorem 2 kan övre gränsen sättas till $n(n+1)/2$.

(3.7-10) Teorem 2: Om allokeringstillståndet är säkert och en delsekvens S uppfyller villkoret V4, kan S utvidgas till en säker sekvens.

Detta teorem minskar arbetet att finna en fullständig sekvens. Innebörden är också att om en delsekvens S ej kan utvidgas till en fullständig sekvens, är fortsatt sökning (av de $n!$ olika möjliga sekvenserna) meningslös.

En algoritm som undersöker om ett givet realiserbart tillstånd är säkert eller ej är i grova drag följande

0. Starta med en mängd T av parallella processer
1. En tom sekvens S uppfyller V4
2. Försök att utvidga S med en process $P_k \in T$ så att V4 uppfylls.
3. Om S ej kan utvidgas med P_k, försök med någon annan process i T. Om S kan utvidgas överför processen från T till S. Återgå till 2. Avsluta när antingen S ej går att utvidga eller när T är tom. När T är tom är tillståndet säkert.

I praktiken är det dock mer intressant att konstatera huruvida övergången från ett tillstånd till ett annat kan förorsaka risk för låsning. Vanligen föreligger situationen att en process P_k vid ett säkert tillstånd $(R, B, P(t))$ begär en eller flera nya resurser. Dessa resurser kan tilldelas P_k om det resulterande tillståndet kan konstateras vara säkert. Konstaterandet förenklas avsevärt genom

(3.7-11) Teorem 3: Om ett säkert tillstånd ändras genom att tilldela P_k ytterligare resurser och om man kan finna en sekvens S, ej nödvändigtvis fullständig, som innehåller P_k och uppfyller V4, så är även det ändrade tillståndet säkert.

Om systemets resurser ej är "hårt ansträngda" är sannolikheten stor att bland mängden av möjliga sekvenser finna en där P_k ingår. Algoritmen kan modifieras på så sätt att man alltid försöker att först utvidga sekvensen med P_k. Lyckas detta är det ändrade tillståndet säkert och sökandet kan avbrytas.

Den intresserade läsaren hänvisas beträffande dessa tre teorem till ref (7) där bevis för teoremen presenteras.

Att under drift minska systemets resurser R kan innebära risk för låsning. Utökning av antalet parallella processer i systemet innebär att B- och P(t)- matrisens kolumnantal ökas.

Att implementera dessa algoritmer i ett "verkligt system" förefaller ej vara alltför komplicerat. I regel har systemet några få särskilt ansträngda resurskategorier vilka då troligen är de enda som löpande behöver undersökas ur låsningssynvinkel. Ett praktiskt problem kan vara att varje process' maximikrav förutsättes vara känt. I vissa "statiska" system krävs att användaren anger b_i i inledande styrsatsen för P_i. Detta är alltså en god bas för ovanstående analys. Andra mer flexibla system, där detta ej krävs, komplicerar därmed förutsättningarna för låsningsriskanalysen.

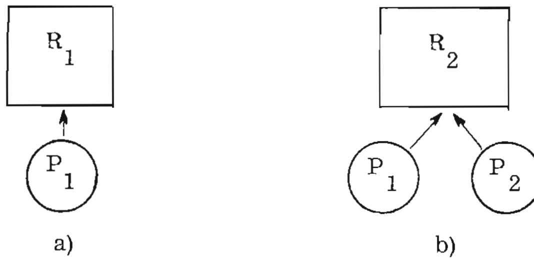
Precedensmatrisen som ett hjälpmedel för att upptäcka låsning

Precedensmatrisens användbarhet i systemsammanhang har utförligt demonstrerats i (14). Vissa låsningstillstånd har karaktären av en "sluten slinga" som kan upptäckas genom manipulation av en precedensmatris som beskriver processers inbördes "vänte-förhållanden" avseende resurser[1].

Låt $R_1, R_2, \ldots, R_m$ vara resurser, sådana att tilldelningsenheten för varje resurs är hela resursen. Låt $P_1, \ldots, P_n$ vara parallellt existerande processer sådana att en process kan exekvera endast om samtliga dess krav på resurser (vid en viss tidpunkt) tillfredsställts. Om en process P_i tilldelats en resurs R_j säger vi att P_i är en omedelbar precedent (eller 1:a precedent) till R_j. Detta visas grafiskt i figur 3.7:2.

Om en process P_i kräver en resurs R_j och R_j redan exklusivt tilldelats en annan process P_k, placeras P_i i kö till resursen R_j efter processen P_k. I detta fall blir P_j en omedelbar precedent till P_k.

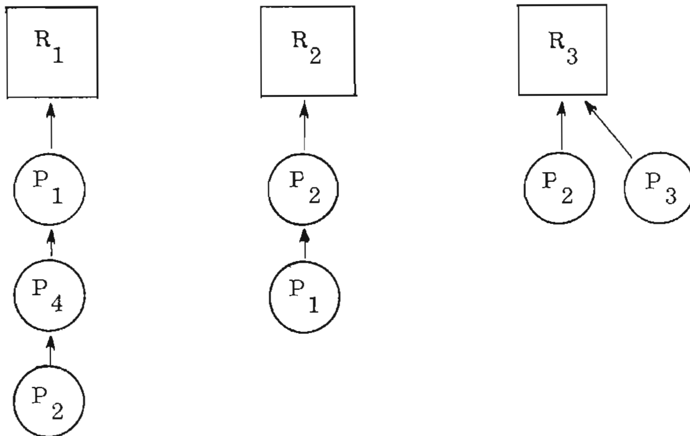
1) P-matrisen har även använts i ref (2).



Figur 3.7:2
a) visar en resurs R_1 som för tillfället tilldelats en process P_1.
b) visar en resurs som är multianvändbar och tilldelats både P_1 och P_2.

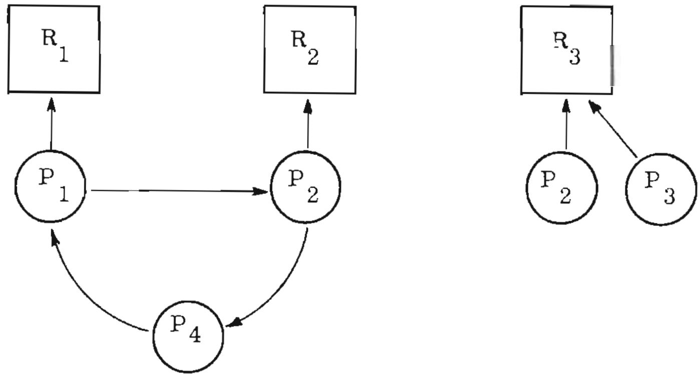
P_j är i detta fall en 2:a precedent till resursen R_j. Ett låsningstillstånd existerar om en process som tilldelats minst en resurs har sig själv som j:te precedent ($j \geq 2$).

Betrakta figur 3.7:3.



Figur 3.7:2
Ett allokeringstillstånd där R_1 tilldelats P_1, R_2 tilldelats P_2 och R_3 delas mellan P_2 och P_3. P_4 och P_2 väntar på R_1 och P_1 på R_2.

Systemet enligt figur 3.7:2 kan även ritas enligt 3.7:3.



Figur 3.7:3

En precedensmatris för ett tillstånd enligt 3.7:3 kan nu uppställas enligt

$X = [X_R, X_P]$

X	R_1	R_2	R_3	P_1	P_2	P_3	P_4
P_1	1				1		
P_2		1	1				1
P_3			1				
P_4				1			

där $X_{ij} = 1$ anger att process i är en omedelbar precedent till resurs eller process j. X har uppdelats i två submatriser X_R och X_P. Om $X_P = 0$ är ingen process i väntan på någon annan process, dvs alla resurser har endast första precedenter eller inga precedenter alls. Ett låsningstillstånd kan konstateras genom att beräkna X_P^2, X_P^3 osv t o m max X_P^n. Om $X_{ii} \in X_P^j$, dvs om det för någon potens av X_P det finns minst ett diagonalelement = 1 så existerar, om motsvarande process redan exklusivt håller en resurs, risk för låsningstillstånd.

I vårt fall blir

$X_P^2 =$

	1	2	3	4
1				1
2	1			
3				
4		1		

;

$X_P^3 =$

	1	2	3	4
1	1			
2		1		
3				
4				1

av vilket framgår att P_1 och P_2 orsakar låsningstillståndet under förutsättning att P_1 ej släpper R_1 förrän R_2 erhållits. P_4 har ingen resurs sig tilldelad (P_4-raden är tom i X_R, dvs den har ingen omedelbar succedent i form av en resurs).

Litteratur

1. Corbató, F.J. & Saltzer, J.H., Some Considerations of Supervisor Program Design for Multiplexed Computer Systems. Proc. IFIP Congress 1968, New Holland Publ. Co., 1968, p. 66.
2. Murphy, J.E., Resource Allocation with Interlock Detection in a Multi-Task System. AFIP's 1968 FJCC Proceedings, Thompson, 1969, p. 1169.
3. Dennis, J.B. & van Horn, E.C., Programming Semantics for Multiprogrammed Computations. CACM, Vol. 9, No. 3, March 1966, p. 143.
4. Graham, R.M., Protection in an Information Processing Utility. CACM, Vol. 11, No. 5, May 1968, p. 365.
5. Dijkstra, E.W., Cooperating Sequential Processes, in Genuys F. (editor), Programming Languages. Academic Press, 1968.
6. Daley, R.C. & Dennis, J.B., Virtual Memory, Processes and Sharing in MULTICS. CACM, Vol. 11, No. 5, May 1968, p. 306.
7. Habermann, A.N., Prevention of System Deadlocks. CACM, Vol. 12, No. 7, July 1969, p. 373.
8. Havender, J.W., Avoiding Deadlock in Multitasking Systems. IBM Systems Journal, No. 2, 1968.
9. Dijkstra, E.W., The Structure of "THE"-Multiprogramming System. CACM, Vol. 11, No. 5, May 1968, p. 341.
10. Dahm, D.M., Gerbstadt, F.H. & Pacelli, M.M., A System Organization for Resource Allocation. CACM, Vol. 10, No. 12, Dec. 1967, p. 772.
11. IBM Operating System/360, Concepts and Facilities, File No. S 360-36, Form C28-6535-0, 1965.
12. Dahl, O.J. & Nygaard, K., SIMULA I - a Language for Programming and Description of Discrete Event Systems. Introduction and Users Manual, Norwegian Comp. Center, Oslo, 1965.
13. Dahl, O.J., Myrhaug, B. & Nygaard, K., SIMULA 67 - Common Base Language. Publ. No. S-2, Norwegian Comp. Center, Oslo, May 1968.

14. Langefors, B., Theoretical Analysis of Information Systems. Studentlitteratur, Lund, 1967.

15. Bubenko Jr, J., Databehandlingsteknik, del I. Studentlitteratur, Lund 1968.

4. DRIFTSFORMER OCH MÖJLIGHETER TILL ÅTKOMST AV DATORKAPACITET

Inledning

I detta kapitel skall vi först diskutera olika driftsfilosofier såsom de torde framstå för en förbrukare av datorkapacitet. I samband därmed skall vi närmare penetrera begrepp såsom jobb, deljobb (kan sägas motsvara "jobbsteg") och processer. Efter det kommer vi att belysa några för en programmerare typiska möjligheter till åtkomst av datorkapacitet.

Arbete som belastar "produktionsapparaten" datorcentralen blir idag föremål för behandling enligt någon av följande driftsfilosofier (givetvis beroende på arbetets art och/eller operativsystemets möjligheter):

- ett jobb-i-taget bearbetning (operatörsstyrd)
- satsvis bearbetning
- kö-vis bearbetning
- reelltidsbearbetning
- tidsdelning (time-sharing)

Det bör påpekas att uppdelning i dessa kategorier ej är betingad av några strikta definitioner. Vi har valt att hålla oss till dessa kategorier då de, åtminstone idag, kan anses vara "etablerade" ur användarsynvinkel. Skillnaderna mellan dessa kategorier betingas av om aktuellt operativsystem innehåller vissa speciella funktioner eller ej avseende multiprogrammering, jobbtillförsel, jobbinplanering m m. Dessutom, som vi tidigare påpekat, existerar det idag system som samtidigt tillämpar två eller flera av ovanstående driftsfilosofier. Till exempel kan ett operativsystem samtidigt styra dels ett reelltidsarbete dels satsvis eller kövis pågående "bakgrundsarbete".

4.1 Jobbet, deljobbet och processen

Ett jobb kan allmänt sägas utgöra "en given, avgränsad arbetsuppgift som tillföres datorsystemet". Denna definition är emellertid ej tillräcklig då varje arbete kan (teoretiskt) delas upp i delarbeten osv ner till den enskilda instruktionen i programmet. Ett jobb innebär exekvering av ett eller flera program. Det kräver också alltid insats från olika funktioner (=program)

inom operativsystemet. Exempel på sådana funktioner som nästan alltid aktiveras är jobbövervakning och inplanering, programladdning och givetvis styrprogrammet som styr resurstilldelningen under exekvering av ett program. Kompilatorer och länkaren är ytterligare exempel där applikationsprogrammet betraktas som en datamängd och blir föremål för någon form av bearbetning. Jobbet uppdelas ur användarens synvinkel i ett antal deljobb eller som det vanligen kallas - jobbsteg. Detta kommer närmare att diskuteras i kapitel 5 och 6. Vad som här bör påpekas är att uppdelningen av ett jobb i deljobb kan göras hierarkiskt på en mängd olika nivåer dvs ett deljobb kan uppdelas i egna deljobb osv. Jobbövervakaren betraktar normalt jobbstegen som deljobb och ombesörjer att dessa initieras för att sedan exekveras under kontroll (styrning) av styrprogrammet. Ett deljobb kan under exekvering "avsöndra" ytterligare egna lokala deljobb (utan att blanda in jobbövervakaren), initiera dessa genom styrprogrammets försorg och eventuellt invänta deras avslutande innan det självt fortsätter. Man kan också se saken så att deljobbet nu fungerar som övervakare åt egna deljobb - en hierarkisk styrningsstruktur.

Typiskt för ett deljobb är att det tar en viss tid att utföra och att det därvid förbrukar en del av datorsystemets resurser. När ett deljobb utföres säger vi att en process äger rum. Beroende på den nivå som vi betraktar deljobben på kan processen förefalla mer eller mindre komplicerad eller sammansatt. Vissa operativsystemtillverkare talar om "interna processer" och "datatransportprocesser" som den lägsta nivå som operativsystemet betraktar deljobben på dvs dessa är de minsta planerings- och styrningsbara enheterna ur styrsystemets synvinkel. Interna processer kännetecknas av att processortid och primärminnesåtkomst erfordras för processens framåtskridande. Transportprocesser kännetecknas av att de främst erfordrar kanalkapacitet och åtkomst till primärminne samt avsedda perifera enheter eller datorer.

Figur 4.1:1 visar en tänkbar logisk (precedens-) struktur hos ett jobb som överlämnats till systemet för bearbetning. På jobbövervakarnivå har jobbet uppdelats i fem deljobb som i detta fall måste behandlas i den sekvens som angivits i figuren. Om systemet tillåter multiprogrammering inom ett jobb kan deljobbet 2 behandlas parallellt med deljobben 3 och 4. Deljobb 5 kan påbörjas endast om 2 och 4 avslutats felfritt. Debitering kan ske för resurser som förbrukats av deljobben 1 t o m 5. "var för sig" eller för jobbet i sin helhet. Det senare är kanske det vanligast förekommande idag.

Ett deljobb kan som antytts normalt uppdelas i två eller flera del-deljobb. I figuren är en hypotetisk uppdelning av deljobbet 3 gjord i deljobb på närmast lägre nivå. Grafen med deljobben 3.1 t o m 3.7 visar dessa deljobbs precedens struktur. Även här kan viss parallell bearbetning förekomma om styrprogrammet tillåter ett varierande antal parallella processer inom ett deljobb.

På den i figuren lägsta nivån har vi visat delprocesserna 3.4.1 t o m 3.4.8 som för illustrativa ändamål kan betraktas som interna processer resp transportprocesser. Simultanitet kan även här förekomma om styrsystemet, antalet buffertutrymmen och maskinvaran så medger.

Vi kan här tänka oss att 3.4.1 är en transportprocess som efter avslutning "startar up" en ny transportprocess 3.4.2 och initierar internprocessen 3.4.3. Först när både internprocessen 3.4.3 och datatransportprocessen 3.4.2 avslutats kan en ny internprocess 3.4.5 initieras. Detta kan exempelvis motsvaras av en vanlig 2-buffrad inläsning och bearbetning av blockvis lagrade data på magnetband. Även andra former för val av efterföljande deljobb är tänkbara - valet av succedent kan bero på avslutningsstatus (t ex normal eller "abnormal" avslutning eller annat villkor) hos aktuellt deljobb. Om deljobb 3.3 avslutas på ett visst sätt väljes 3.5 som efterföljande deljobb, i annat fall väljes deljobb 3.4.

Det är givet att operativsystemets administrerande arbete på motsvarande sätt indelas i deljobb, dock med ett normalt betydligt mer komplex logisk beroendestruktur.

Nedbrytningsprincipen enligt figur 4.1:1 kan vi tillämpa vid samtliga driftsfilosofikategorier. Skillnaden utgöres av

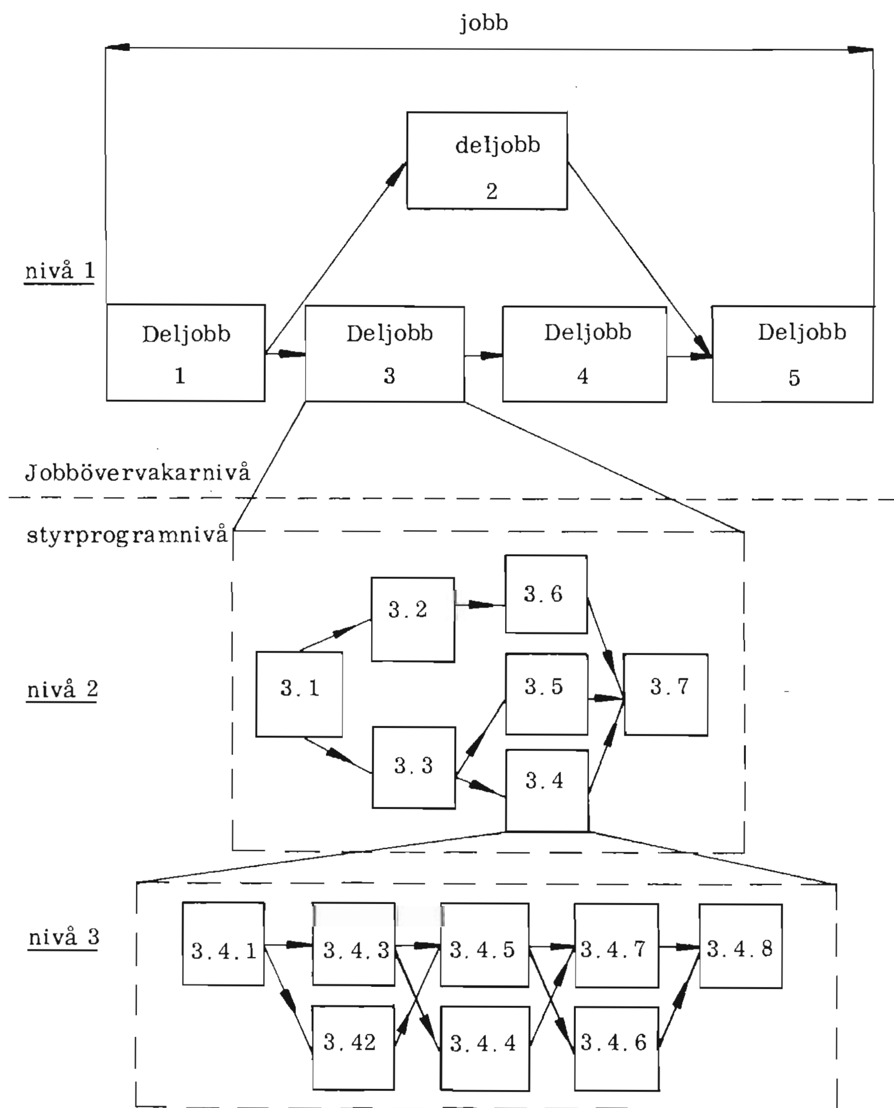
- huruvida övergången från ett deljobb till ett annat på nivå 1 ombesörjes av operatören eller "automatiskt" av operativsystemet
- hur "stora" jobb som vi åt gången tillför systemet
- vilken omloppstid som gäller resp krävs och
- hur systemet "internt" planerar och styr dessa jobbs och deljobbs bearbetning och hur styrningen utföres av parallella processer.

Styrning av deljobben på nivå 1 kommer att belysas i kapitel 5 och 6. Styrning av deljobben på nivå 2 och uppåt kommer att beröras i samband med kapitel 7 som handlar om styrprogram och styrprogrammekanismer.

Vi övergår nu till en kort beskrivning om olika driftsfilosofier. I samband med det kommer fig 4.1:1 att åberopas.

4.2 Ett jobb-i-taget bearbetning

Denna form av bearbetning är idag ej särskilt vanlig och förekommer då endast på de allra minsta datorerna där den kan vara ekonomiskt försvarbar. Alla deljobb på nivå 1 (fig 4.1:1) initieras av operatör eller av ansvarig pro-



Figur 4.1:1

Tänkbar logisk nedbrytning av ett jobb. Exekveringsframskridandet genom de olika deljobben styrs här dels av deljobbens avslutningsstatus och dels av de för deljobben erforderliga resurserna. För att bättre illustrera figuren kan vi i detta fall anta att deljobben på

nivå 1 består av självständiga (self-contained) program som skall exekveras
nivå 2 består av dessa programs delprogram
nivå 3 består av delprogrammens interna processer och transportprocesser

grammerare. Parallell bearbetning av jobb eller deljobb, förutom överlappning av internbearbetning med in/utmatning, brukar ej förekomma hos dessa system. Styrprogrammet är här relativt primitivt, belägger endast en ringa del av primärminnet (storleksordning några tusen tecken) och innehåller, förutom rutiner för brytsignalshantering, rutiner för styrning av datatransporter till/från anslutna perifera enheter. Administrationstid orsakad av operativsystemet är föga relevant i detta sammanhang. Användaren debiteras här för den tid som utförandet av samtliga deljobb, inklusive uppsättning och programbyten, tar. Hänsyn till hur mycket och hur effektivt systemets resurser utnyttjas tages normalt ej i detta fall.

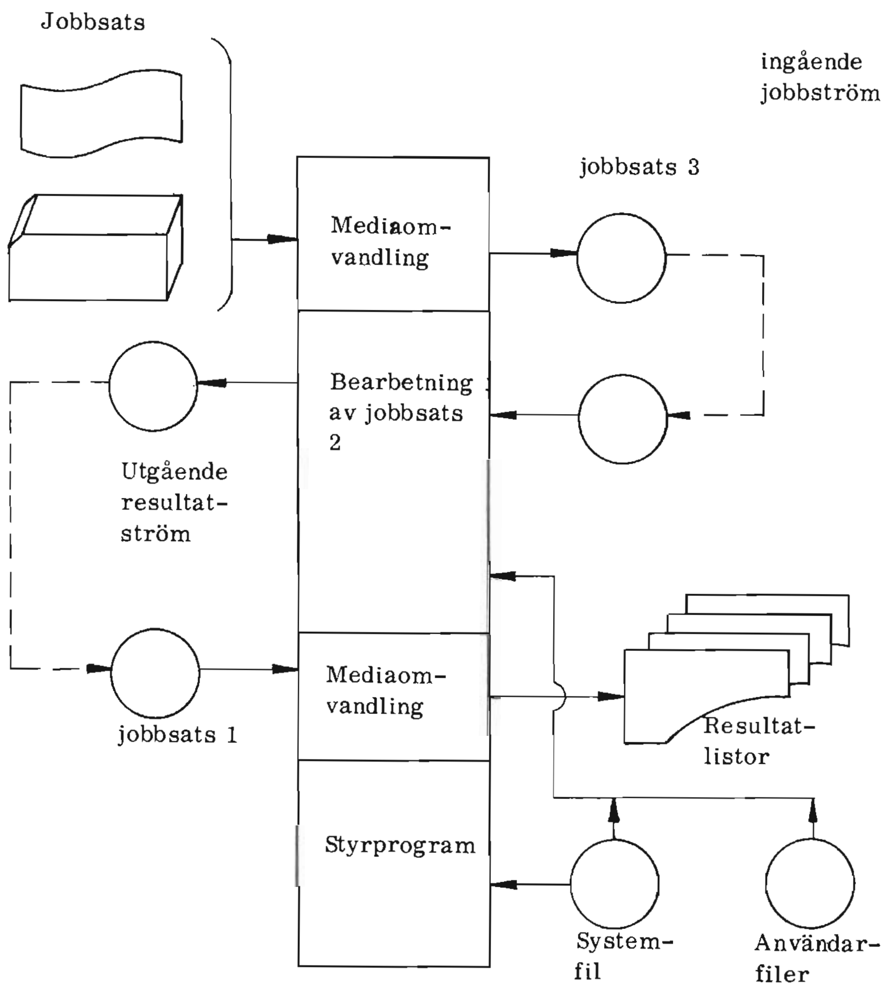
4.3 Satsvis bearbetning

Ovanstående innebär att jobb hopsamlas till en "sats" (s k "batch") som sedan "kollektivt" tillföres datorsystemet. Tillförseln kan tillgå genom att jobben, bestående av styrinstruktioner, program och data stansade på hålkort eller hålremsa, överförs till ett snabbare inmatningsmedium av sekundärminnestyp (magnetband, direktminne) och bildar där en sekvens av avgränsade arbetsuppgifter vanligen kallad för "jobbström". När överföringen avslutats, initieras en sekvensiell utsökning och bearbetning av jobbströmmen. Om styrsystemet så tillåter kan parallell bearbetning av flera jobb förekomma.

Då jobben ligger i sekvens-organiserad fil är någon planering eller prioritetsstyrd bearbetning av dessa normalt ej möjlig. I vissa system kan dock jobb med hög prioritet (s k "rush-jobs") bearbetas i förtur om de tillföres datorn över en annan, separat inmatningsström och normalt också en annan inmatningsenhet. Påpekas bör att det existerar system som arbetar med flera jobbströmmar och där under uppbyggnad av en jobbström sker bearbetning av en annan jobbström. På de tidigare systemen, i början av 60-talet, skedde denna uppbyggnad av jobbströmmar ofta med hjälp av en fristående satellitdator. Idag är det vanligt att dessa mediaomvandlingar utföres med multiprogrammeringsteknik parallellt med bearbetning av tidigare uppbyggd jobbström på samma datorsystem.

Vidstående bild av bearbetningen är möjligen typisk för installationer där arbetsprofilen präglas av många relativt korta jobb - t ex tekniska och vetenskapliga beräkningar. Men även hos installationer vars arbetsprofil präglas av mer tidskrävande administrativa rutiner kan ovanstående driftssituation uppkomma i samband med programmering och test.

När jobbprofilen kännetecknas av längre körningar med stora datamängder på sekundärminnen som kräver montering osv (den "administrativa" driftsprofilen), är ovanstående princip för mediaomvandling av jobb-satsen ej

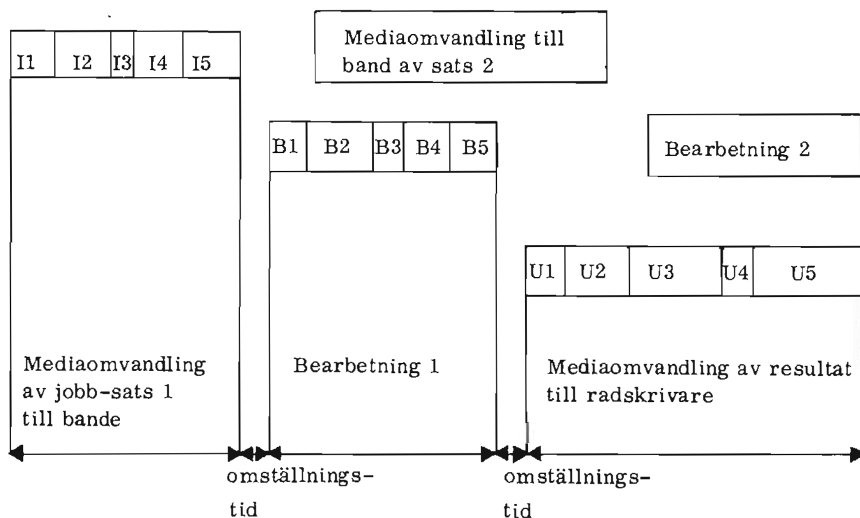


Figur 4.3:1

Satsvis bearbetning med två parallellt pågående mediaomvandlingar.

alltid lönsam. Man inmatar då jobb-satsens styrkort och parametrar manuellt över operatörskonsolen och utnyttjar i stället kortläsare. remsläsare och radskrivare till parallellt pågående mediaomvandling för att överföra indatamängder från hålkort eller remsa till magnetband eller , på utmatningssidan, resultatmängder till radskrivare[1]. Själva "huvudrutinerna" görs på så sätt sekundärminnesorienterade vilket torde vara driftsekonomikt riktigt under förutsättning att tillräckligt sekundär- och primärminnesutrymme finnes. Dessa rutiners primärminneskrav är normalt stora och genom sekundärminnesorientering kan summan av produkten (utrymmeskrav) • (residenstid) över samtliga program hållas så låg som möjligt. Att minimera denna summaprodukt är normalt väsentligt för driftsekonomin hos samtliga system.

Figurerna 4.3:2 och 4.3:3 illustrerar tänkbara tidsprofiler för satsvis bearbetning med ingående resp utgående jobb resp resultatström på sekundärminne.

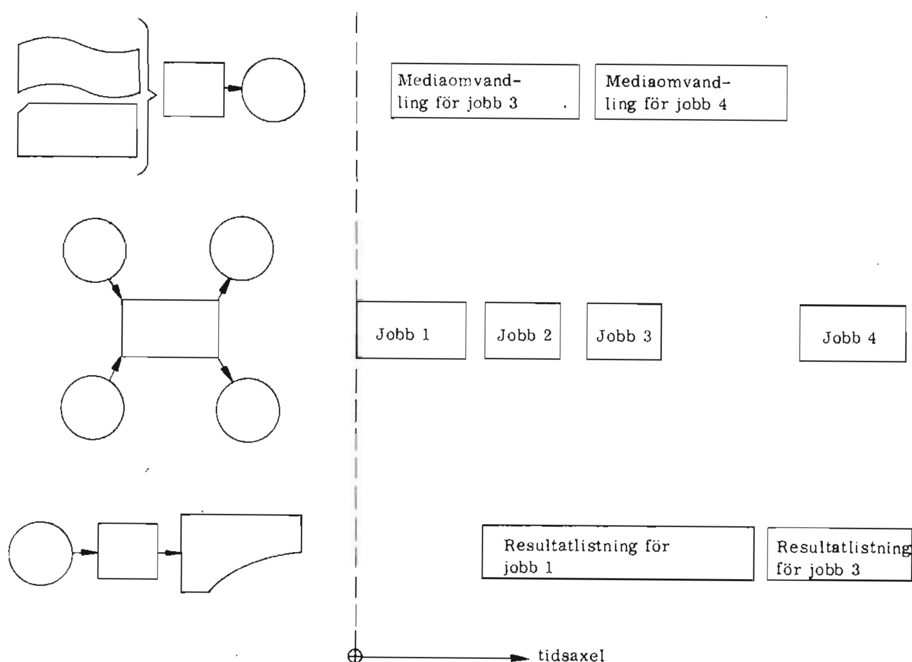


Figur 4.3:2

Hypotetiskt tidsförlopp för satsvis bearbetning av relativt korta jobb (1-10 minuter) med ringa datamängder. Teckenförklaring:

I1 inmatning av program och data för jobb 1
B2 bearbetning av jobb 2
U3 utmatning av resultat för jobb 3.

1) Detta innebär att driftsprincipen närmar sig jobb-för-jobb bearbetning med undantaget att parallell bearbetning av jobb kan förekomma.



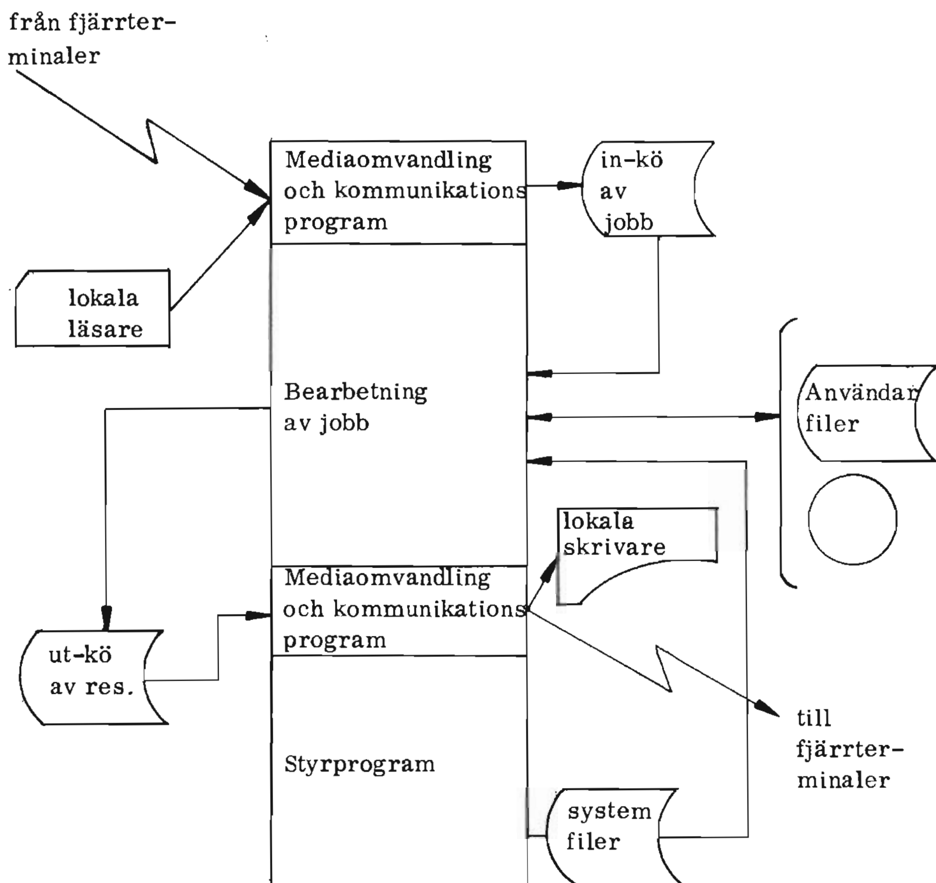
Figur 4.3:3

Typisk tidsprofil för "administrativa" bearbetningar satsvis med parallellt pågående mediaomvandling av data- och resultatmängder. Observera att jobb 2 ej har sådan utmatningsmängd att listning lönar sig att göra separat och parallellt med andra jobb. I sådana fall kan indata och utdata läsas resp skrivas över någon prisbillig perifer enhet för att ej blockera de relativt dyrbara snabba hålkortsläsarna och radskrivarna. Mediaomvandling av indata för jobb 1 och 2 ej visad i figuren.

4.4 Kö-vis bearbetning

Denna typ av bearbetning innebär att jobben ej uppsamlas till en "sats" som sedan mediaomvandlas, utan att till datacentralen ankommande jobb omedelbart inmatas (och mediaomvandlas) till en "kö" som normalt residerar på ett direktminne (figur 4.4:1). Samtidigt med denna inmatning sker bearbetning av jobbkön och utmatning av resultatet till en utgående kö, även denna vanligtvis placerad på ett direktminne. Innehållet i den utgående kön mediaomvandlas till radskrivare (eller motsv) av en simultant pågående process. Denna bearbetningsprincip skiljer sig från satsvis bearbetning framför allt genom att

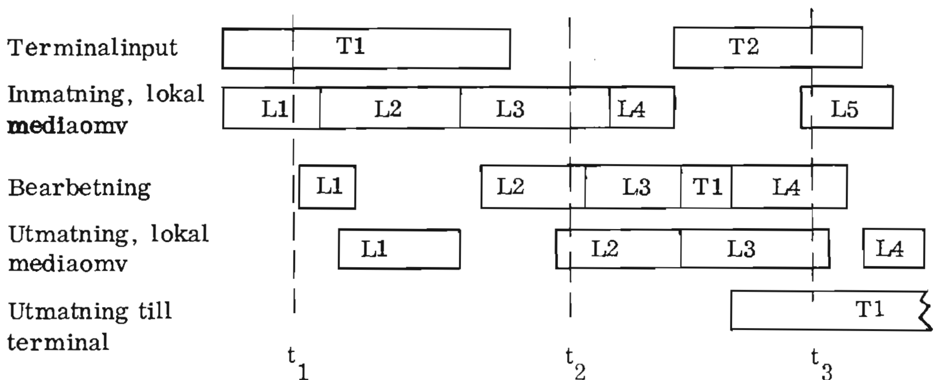
- prioritetsstyrd selektering av jobb ur inkön och viss planering av multiprogrammeringen är möjlig då ofta mer än ett jobb befinner sig i inkön och direktminneslagringen medger andra former av utsökning än den rent sekvensiella.



Figur 4.4:1

Principskiss över kö-vis jobb-bearbetning

- den "teoretiska" omloppstiden för ett jobb, dvs den tid från den det inlämnats till datacentralen till den att resultatet föreligger, kan härmed avsevärt förkortas[1)] i förhållande till satsvis bearbetning.
- genom direktminneslagring av köerna utnyttjas systemets sekundärminnen på ett effektivare sätt och medger också att flera lokala eller avlägsna inmatningsenheter fyller på den ingående jobbkön. På samma sätt kan den utgående resultatkön simultant "tömmas" av flera lokala eller avlägsna utmatningsenheter. En "balansering" av antalet av systemets in/utmatningsenheter är möjlig för att anpassa in-utmatningskapaciteten till jobb-bearbetningskapaciteten.



Figur 4.4:2

Tänkbar tidsprofil för kö-vis bearbetning med dels inmatning av jobb lokalt (Li) och från terminal (Ti). Det antas i denna figur att endast ett jobb bearbetas åt gången.

Figur 4.4:2 visar en tänkbar tidsprofil för denna driftsfilosofi. Observera att tiden för inmatning av jobb resp utmatning av resultat från resp till terminal i figuren är visad längre än tiden för lokala läsare och skrivare. Detta kan bero på att överföringshastigheten över telekommunikationslinjerna är lägre än motsvarande hastighet till lokala enheter. Detta indikerar också

1) Att det ej alltid så är fallet i praktiken torde bero på viss byråkratisering och på bristande planering och anpassning av arbetet för datacentralspersonalen, som så att säga "fortsätter att arbeta satsvis" även om dator systemet arbetar "kö-vis". Att programmerare skulle få snabbare och mindre komplicerad tillgång till programkörning skulle av många driftschefer kanske ses som ett intrång på ett område som de betraktar som sitt.

att utrymmeskravet på direktminne för in- resp utgående köer ej enbart beror på datavolymen utan även på resp läs-, skriv- och överföringshastigheter. Ju långsammare dessa operationer sker desto mer "bufringsutrymme" för köer erfordras. Ej sällan har detta utrymmesbehov underskattats och en "flaskhals" inom systemet har därmed uppkommit. Den enklaste nödåtgärden är då att för utrymmeskrävande jobb ersätta direktminneslagring med magnetband och för dessa jobb återgå till satsvis bearbetning med ökad omloppstid som följd.

I figur 4.4:2 observerar vi också att i uppstartningsfasen (t_1-t_2) ej alla av systemets funktioner är aktiva. Vid tiden t_1 pågår endast inläsning av terminaljobb 1 och det lokala jobbet 1. Vid t_2 pågår bearbetning av det lokala jobbet L2, samtidigt som L3 inmatas och L2's resultat listas lokalt[1].

Av denna driftsprincip kan givetvis olika varianter förekomma. Den kan kombineras med satsvis bearbetning, multibearbetning av jobb kan förekomma (i mån av tillgång till primärminnesutrymme), jobb kan tillföras från terminal men resultatet lisatas centralt och återsändas med t ex post eller mopedbud, resultatet kan också begäras överförd till annan terminal (destination) osv. Driftsprincipen finns också realiserad genom att utnyttja två hopkopplade datorer (t ex ASP-systemet, se avsn 2.2 sid 142) där den ena ombesörjer in/utmatning och viss filhantering och den andra ombesörjer bearbetning av jobben. Denna driftsprincip med alla dess varianter får anses som den idag mest förekommande. Minnesutrymmesbehovet för den residenta delen av styrprogrammet varierar starkt dels beroende på tillverkare dels beroende på förekomsten av olika, mer eller mindre nyttiga, specialfunktioner, möjlighet till jobbplanering, antal perifera enheter osv. Tyvärr kan vi ej ange någon "normal" storleksordning avseende utrymmesbehovet. För system som vi kommit i kontakt med och som kan sägas tillämpa denna driftsfilosofi (med större eller mindre inskränkningar) har primärminnesbehovet för styrprogrammen varierat mellan 20 K-tecken och 200 K-tecken eller mer. På samma sätt kan operativsystemets behov av sekundärminne (exklusive kö-utrymme) variera mellan 100 K-tecken och 1000 K-tecken eller mer. Till detta kommer utrymme för programbibliotek, arbetsfiler osv.

4.5 Reelltidsbearbetning

Vi skall här endast diskutera reelltidsfunktionen hos ett operativsystem och motsvarande driftsfilosofi. Hos de flesta reelltidssystemen är det, åt-

---

1) Den lokala utmatningskön är här tom och L2's resultat kan börja listas i samma stund som de uppstår.

minstone teoretiskt (om än ännu så länge sällan i praktiken), möjligt att med lägre prioritet "bakgrundsbehandla icke reelltidsjobb" satsvis eller kövis.

Reelltidsbearbetning innebär normalt att systemet interaktivt[1] samspelar med en eller flera användare och/eller med ett eller flera fysiska system vars förlopp det avser att observera och/eller styra. Olika s k terminaltyper och möjligheter att via kommunikationslinjer och datorer ansluta dessa behandlas ej här.[2]

Typiskt för reelltidssystem är att aktiviteter hos dessa, dvs processer i systemet, utlöses av meddelanden som inkommer från terminalerna. På sätt och vis kan vi här tala om kö-vis bearbetning där jobben utgöres av meddelanden dock med skillnaden att

- meddelanden utlöser processer vars sammanlagda totala tidsåtgång är mycket kort (0.1 - 1 sekund) jämfört med ett normalt jobb vid kö-vis bearbetning
- meddelandena, normalt komprimerade och kodifierade, tillför ej systemet algoritmer (dvs program) om hur data skall bearbetas. Genom tolkning av meddelandet avgör systemet vilka processer det skall genomgå och vilket svar, om något, skall sändas tillbaka. Meddelandena har ingen möjlighet att explicit starta exekvering av systemprogram o dyl.

Ett reelltidssystem är således helt dedikerat ett avgränsat applikationsområde såsom t ex platsbokning (flygbolag), orderbehandling (handelsföretag) osv. Givetvis kan till innehåll och struktur olika meddelanden förekomma vilka då helt eller delvis kräver olika bearbetningsförlopp. Program för samtliga tänkbara meddelanden och dessas bearbetningar måste därför förutses och finnas i systemet. Vi kan generellt säga att bearbetning av ett meddelande kräver att en serie (med ev parallella grenar) av processer aktiveras i en viss följd som så småningom producerar ett "svar" vilket sändes tillbaka till aktuell terminal.

Interaktionen med en terminal innebär att terminalen ständigt producerar nya meddelanden vars innehåll och struktur i många fall är beroende på tidigare erhållet svar från datorsystemet. På samma sätt kan datorsystemets bearbetning av ett meddelande M(i, X) från terminal X vara beroende på dess svar S(i-1, X) på meddelande M(i-1, X) och annan statusinformation som i datorn uppsamlats för terminal X under bearbetning av ytterligare tidigare meddelanden M(j, X), $j < i-1$.

---

1) Ett meddelande sänt från en källa till datorsystemet kräver svar innan källan kan avge nästa meddelande.
2) Den intresserade läsaren hänvisas till ex vis Martin, J: Telecommunications and the computer, Prentice-Hall 1970.

Figur 4.5:1 visar ett förenklat, hypotetiskt förlopp för bearbetning av ett meddelande där 6 (klasser av) processer antas involverade. Beroende på meddelandets innehåll, struktur och terminalens X tidigare "historia" kan följande alternativa vägar tänkas för bearbetningen:

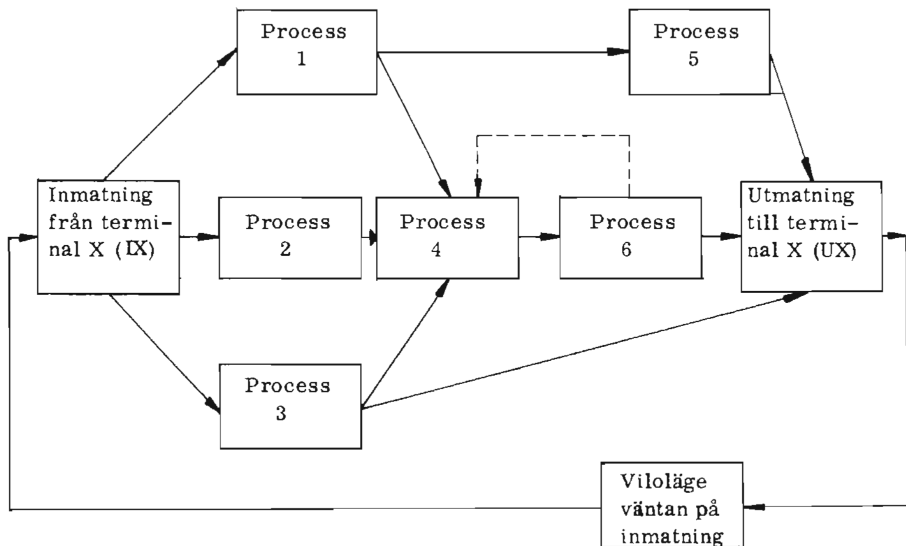
1. IX - 1 - 5 - UX

2. IX - 1 - 4 - 6 - UX

3. IX - 2 - 4 - 6 - UX

4. IX - 3 - 4 - 6 - UX

5. IX - 3 - UX



Figur 4.5:1

Bearbetning av ett meddelande från terminal X kan följa olika vägar i ett reelltidssystem beroende på meddelandets typ, informationsinnehåll och/eller informationsinnehållet i systemets datafiler.

Iterativa förlopp, ett antal "slingor" t ex 4-6-4 osv kan givetvis förekomma men vi har avstått från att komplicera figuren med dylikt. För att utföra någon av ovanstående processer krävs resurser i form av

a) program
b) minnesutrymme
c) data och access till filsystemet
d) kanaltid
e) linjetid (för IX och UX processerna)
f) processortid och
g) arbete av olika operativsystemfunktioner, som i sin tur kan kräva resurser av typen a), b), c), d), f) och g) rekursivt till lägsta funktionsnivå.

Då i varje ögonblick ett obestämt antal terminaler kräver service från systemet i form av överförda meddelanden (eller önskar överföra dylika) uppstår normalt konflikter till resurser och därmed köbildningar.

En väsentlig del hos mer avancerade reelltidssystems styrsystem arbetar just med administration av olika slag av resursköer[1]. System som behandlar ett meddelande i taget har normalt inga direkta interna köproblem och kallas vanligtvis för "single-thread"-system. Hos system där bearbetning av meddelanden kan ske parallellt (dvs multiprogrammering) erfordras mekanismer för hantering av interna resursköer. Sådana system brukar kallas för "multithread"-system.

Figur 4.5:2 visar ett tidsdiagram för interaktion mellan 3 terminaler och ett datorsystem där vi för enkelhets skull har antagit att bearbetning av ett meddelande tar 1 tidsenhet om dessa behandlas i serie (dvs "single-thread"). Om fler meddelanden bearbetas parallellt antas den "individuella" tiden öka på grund av resurskonflikter. Figur 4.5:2 visar en tänkt situation för "single-thread"- och figur 4.5:3 en tänkt situation för "multi-thread"- bearbetning.

Av figur 4.5:2 och 4.5:3 förefaller det som "multi-threading" ur responstidssynpunkt vore att föredraga. Detta är dock ej alltid fallet då dels för styrning av sådan bearbetning krävs extra styrprogramutrymme och styrprogramtid dels systemets konfigurering kan vara sådan att "multi-threading" ej ger någon direkt tidsvinst då det existerar en dominerande och tidskritisk facilitet[2] i systemet.

---

1) För övrigt en normal företeelse hos alla system som tillämpar multiprogrammeringsteknik.
2) Till exempel om systemets samtliga filer residerar på ett massminne med en accessmekanism kan obetydliga tidsvinster göras genom "multithreading" teknik om varje transaktionsbearbetning kräver mycket accesser till filsystemet.

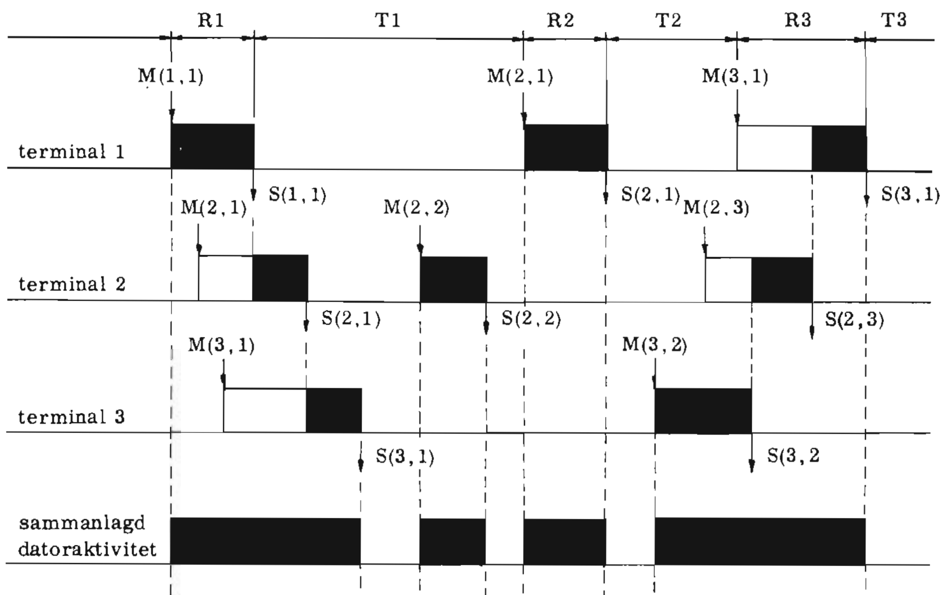
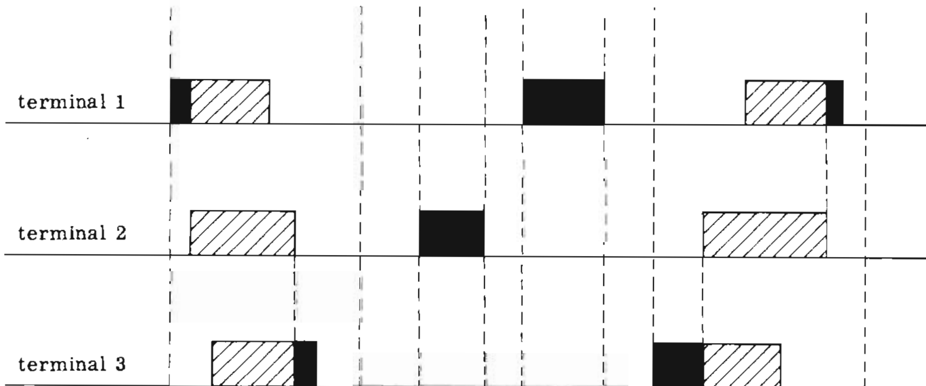


Fig 4.5:2



Figur 4.5:3

I figur 4.5:2 visas "single-thread" bearbetning med tre terminaler. Skuggade fält markerar datoraktivitet. Vita fält markerar väntetid (Ri = responstid, Ti = terminaltid). I figur 4.5:3 är motsvarande "multi-thread" bearbetning visad. Streckade fält markerar att meddelandet bearbetas parallellt med ett eller flera andra meddelanden. Observera att de individuella behandlingstiderna är längre i det senare fallet men att fler meddelanden per tidsenhet medhinnes på detta sätt.

4.6 Time-sharing-system (tidsdelningssystem)

Relativt mycket utrymme för en allmän diskussion om det idag aktuella begreppet "tidsdelningssystem" har anslagits i kapitel 2 i samband med operativsystemutvecklingen. Därför skall vi här fatta oss kort. Tidsdelningssystem (time-sharing-system, även kallade "multipelaccessystem") har ur driftssynpunkt många beröringspunkter med reelltidssystem. Bägge karaktäriseras avseende datainmatning av kommunikationslinjeanslutna fjärrterminaler och meddelanden från dessa som utlöser kedjor av processer av begränsad tidsåtgång. Det finns emellertid två fundamentala skillnader. För time-sharing gäller:

(1) Programmeraren (eller användaren) vid terminalen, som fn vanligen är av skrivmaskinstyp, är ej enbart hänvisad till en begränsad mängd i förväg "inbyggda" funktioner som systemet kan utföra utan kan själv

(a) tillföra systemet egna procedurer i något av systemet accepterat programmeringsspråk och t ex få omedelbar syntaxkontroll av den senast tillförda språksatsen

(b) kan namnge procedurer och lagra dem i användarens eget "förhyrda" utrymme på systemets direktminnen

(c) i princip kombinera egna program med andra användares lagrade program eller systemprogram av allmän servicekaraktär, under förutsättning att vederbörande användare är "auktoriserad" därtill.

(d) användaren kan på motsvarande sätt lagra och manipulera datamängder (egna, "allmänna" eller tillhörande andra).

Med andra ord har användaren tillgång till en mängd av systemets resurser som vederbörande kan efter eget behov och under beaktande av vissa säkerhetsföreskrifter och sin egen budget förbruka.

(2) Den interna planeringen och fördelningen av systemets resurser under exekvering av terminalöverförda meddelanden eller arbeten är något annorlunda än hos reelltidssystem i allmänhet. Man arbetar i tidsdelningssystem normalt med att inordna deljobben i en cirkulär kö (ev flera) och sedan tilldela det första jobbet i kön en bestämd mängd, ett "kvantum" processortid (storleksordning 0.1 - 1 sekund). Om deljobbet efter denna tid ej är slutfört placeras det sist i kön och nästa deljobb i kön utväljes för bearbetning.

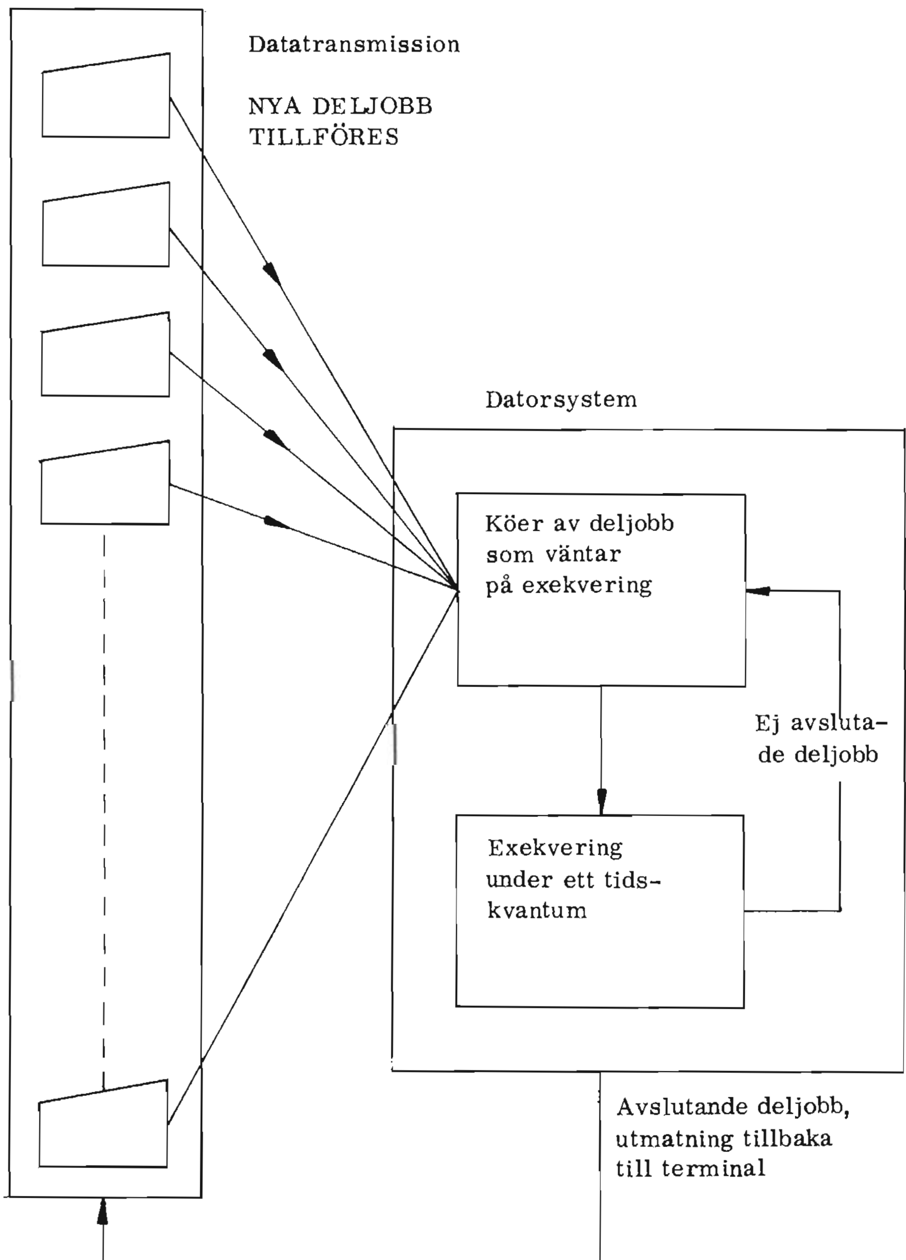
Var deljobben lagras under sin väntetid i kön beror på systemets interna planeringsfilosofi. Vissa system använder ett direktminne för att lagra alla data och program som tillhör deljobben. I primärminnet befinner sig vid

varje tillfälle endast ett jobb. Det finns system där primärminnesutrymmet är så stort att samtliga jobb kan "ligga kvar" på samma utrymme och aktiveras av styrprogrammet när deras tur kommer att erhålla ett kvantum processortid. Andra system tillämpar någon form av blockutbytesteknik (paging) och håller i primärminnet endast sådana programblock som är aktuella för motsvarande program. Som påpekats i avsnitt 2.2 kan även varierande köprinciper tillämpas t ex en högprioritetskö för "korta" deljobb (t ex syntaxkontroll av en programsats) dels en lågprioritetskö för längre produktionsjobb.

Vi noterar alltså att i tidsdelningssystem, i motsats till i reelltidssystem, användaren har möjlighet att få systemet att arbeta som svarar mot varje användares behov. Det må vara kortare eller längre tekniska beräkningar, eller administrativa rutiner. Reelltidssystem, konstruerade för en bestämd applikation kan endast utföra det arbete som man beslutat sig för under det s k systemarbetet och informationsbehovsanalysen. Systemet är med andra ord, liksom de flesta administrativa databehandlingssystem, stelbent och ställer sig dyrbart[1] om ändringar och/eller utökningar av systemet skall göras.

Teoretiskt sett finns det möjligheter att implementera en reelltidsapplikation under styrning av ett operativsystem för tidsdelning. Praktiskt sett har svårigheter förelegat - främst avseende effektiviteten av en sådan lösning och även på grund av att tidsdelningssystem normalt ej kan arbeta med terminaler av den mängd skiftande slag som förekommer vid reelltidssystem såsom s k speciella "agent sets" (vid flygbokning), kassaapparater, speciella terminaler vid processstyrning osv. Det är ett känt och ekonomiskt kännbart faktum att administrativa databehandlingssystem snabbt föråldras dvs att de "behov" de ursprungligen konstruerats för väsentligen ändras och/eller utökas under systemets driftstid. Trots att de konstrueras med hopp om ökad flexibilitet och rationalitet har det med få undantag visat sig att de automatiserade rutinerna blivit ur flexibilitetssynvinkel underlägsna tidigare manuella eller hålkortsbaserade system. Tidsdelningstekniken med möjlighet till enkel modifikation och/eller utökning av systemets funktioner öppnar här en teoretisk möjlighet till mer flexibla applikationssystem. För att lösa detta praktiskt krävs dock arbete och utveckling dels på maskinsidan (större o billigare sekundärminnen) dels på informationssystem- teori- och programvarusidan (konventioner och verktyg för människa- maskinsamspel, metoder för programoberoende datastrukturering och datadefinition samt enklare beskrivning av de processer som önskas utförda).

1) Med nuvarande metodik för system- och programkonstruktion.



Figur 4.6:1

Principskiss över ett tidsdelningssystems arbetsätt.

4.7 Åtkomst till datorkapacitet

Vi avser nedan belysa åtkomst till datorkapacitet med tonvikt på programinkörning och produktionskörning av användarprogram.

Från denna diskussion undantar vi reelltidsdrift då denna filosofi endast medger utförande av en förutbestämd och definierad mängd av arbetsuppgifter.

Olika driftsfilosofier medger givetvis olika möjligheter för en användare att "komma åt" en dator för att få sina program körda. Dessa olikheter påverkar också en programmerares sätt att arbeta och dessutom effektiviteten i detta arbete. Några betydelsefulla faktorer[1] i detta sammanhang är

(1) tiden från det att ett fel i programmet konstaterats (och felet korrigerats), till dess att en ny testkörning företagits och resultat erhållits. Denna punkt avser även produktionskörning, dvs tiden från den tid en komplett uppsättning indata för en körning föreligger till dess att körningen utförts. Låt oss kalla detta för "<u>åtkomsttid</u>",

(2) möjlighet att exekvera ett program <u>partiellt</u> och därvid när som helst kunna avbryta en exekvering för att t ex ändra programmet eller initialisera/byta värden på programparametrar o dyl

(3) tid på dygnet då program kan exekveras eller lämnas för exekvering. Härmed avser vi eventuella körningspass för programtestning, produktionskörning osv.

Följande möjligheter till åtkomst av datorkapacitet torde kunna anges

- centraliserad s k "open-shop"-drift
- centraliserad s k "closed-shop"-drift
- decentraliserad "open-shop"-drift (från medel- eller höghastighetsterminaler)
- decentraliserad "closed-shop"-drift (från medel- eller höghastighetsterminaler)
- tidsdelningsdrift från lokala eller avlägsna långsamma terminaler (skrivmaskin, bildskärm)

---

1) Vi måste här bortse från kvalitativa faktorer såsom en kompilators diagnostik, op systemets styrspråk m m då inga teoretiska eller empiriska utredningar om dessa föreligger. Man måste trots det vara medveten om dessa faktorers betydelse.

Med central drift avses att in-utmatningsenheter till datorn är centralt belägna dvs i omedelbar anslutning till datacentralen.

Decentralisering innebär här att in-utmatningsenheterna utgöres av från datacentralen avlägset belägna terminaler av olika kapacitet och utformning. T ex kan en terminal vara en liten dator, dvs en filial-datacentral, försedd med läsare och läsare och skrivare och som över en telefonlinje överför jobb och mottar resultat från den centralt belägna datorn. Alternativt kan en terminal utgöras av en skrivmaskin eller bildskärm placerad hos någon användare.

"Open-shop" innebär att ansvaret för jobbinmatning och omhändertagande av resultat, dvs viss operatörsverksamhet, vilar på användaren som då är närvarande under körning (m a o "självbetjäning").

"Closed-shop" innebär att datorcentralen har en utbyggd serviceorganisation för mottagning och körning av jobben samt distribution av resultaten. I detta fall har programmeraren dock ofta en möjlighet att beskåda körning av sitt jobb (men ej ingripa) t ex genom en ljudtät glasruta.

Från programmerarnas sida är givetvis open-shop-driften eller tidsdelningsdriften normalt att föredra ur effektiv programkonstruktionssynvinkel. Datacentralchefer, å andra sidan, visar ofta en förkärlek för den "slutna driftsformen" och söker omgärda sin datacentral med en hierarki av mellanhänder, kontroller och föreskrifter. En utredning om totaleffektiviteten hos olika drifts- och åtkomstprinciper torde vara påkallad och av största intresse för landets förbrukare av datorer och datorkapacitet.

Tabell 4.7:1 visar en uppställning av olika åtkomstprinciper och möjlighet att tillämpa dessa beroende på den driftsfilosofi som tillämpas för datorsystemet. Tabell 4.7:2, slutligen, anger storleksordningen för åtkomsttid beroende på driftsfilosofi och åtkomstprincip. De angivna siffrorna markerar rimliga minimivärden och maximivärden. Maximivärdet avsers gälla för en relativt tungt belastad datoranläggning. Faktorer såsom avstånd till terminal och/eller datorcentral liksom datorcentralens serviceorganisation påverkar givetvis angivna värden. En annan väsentlig faktor är givetvis jobbets storlek. I tabellen 4.7:2 har vi antagit att jobben normalt utgörs av kortare beräkningsarbeten (t ex 100 satser i FORTRAN) och att datorsystemet har en hög kapacitet så att den direkta exekveringstiden inom datorsystemet ej är av avgörande betydelse.

Åtkomst-princip	Driftsfilosofi			
	Ett-jobb-i taget bearbetning	Satsvis bearbetning	Kö-vis bearbetning	Tidsdelning
Centraliserad open-shop	Möjlig, ev med bokning av tid	Ej tillämplig	Möjlig	Möjlig, t ex centralt datorlaboratorium
Centraliserad closed-shop	Möjlig	Möjlig	Möjlig	-
Decentraliserad open-shop	Ej direkt tillämplig	Ej direkt tillämplig	Möjlig	Möjlig, t ex lokala datorlaboratorier
Decentraliserad closed-shop	Ej direkt tillämplig	Möjlig	Möjlig	-
Tidsdelningsdrift med fjärrterminaler	-	-	-	Ja

Tabell 4.7:1

Möjlighet till åtkomst av datorkapacitet beroende på driftsfilosofi.

Åtkomst-princip	Driftsfilosofi			
	Ett-jobb-i taget	Satsvis	Kö-vis	Tidsdelning
Centraliserad open-shop	1-48 tim	-	5 min-1 tim	1 min-10 min
Centraliserad closed-shop	1-48 tim	2-24 tim	10 min-24 tim	-
Decentraliserad open-shop	-	-	5 min-1 tim	1 min-10 min
Decentraliserad closed-shop	-	2-24 tim	10 min-24 tim	-
Tidsdelning med fjärr-terminaler	-	-	-	1 min-10 min

Tabell 4.7:2

Ungefärliga min-maxvärden för åtkomsttider till datorkapacitet beroende på åtkomstprincip och driftsfilosofi.

4.8 Blandade driftsformer

Driftskombinationen som i allt större utsträckning eftersträvas vid planering av nya databehandlingssystem är sats- eller kö-vis bearbetning plus en eller flera på samma dator samtidigt pågående reelltidsrutiner. Denna kombination kan givetvis realiseras på ett mer eller mindre effektivt ur resursutnyttjandesynvinkel. Om reelltidsapplikationerna ej är kritisk karaktär, dvs en tids absolut driftsstopp ej förorsakar ett allvarligt avbräck i företagets verksamhet, har implementering av dessa normalt ej förorsakat större problem. Om, däremot, reelltidsapplikationerna varit av vital betydelse för företagets operativa verksamhet har man vanligen ej riskerat en störning av denna och därför installerat en eller flera tvillingdatorer med möjlighet till snabb automatisk omkoppling mellan dessa i händelse av driftsstopp på "reelltidssystemets" dator. Erfarenheten har visat att driftsstopp

normalt orsakas i lika hög grad av programvarufel som maskinvarufel trots inbyggda skyddsmekanismer. Det torde också höra till ovanligheten att vitala reelltidsprogram bearbetas samtidigt med t ex uttestning av nya programdelar på samma dator.

Ytterst få system existerar idag där driftsfilosofierna sats- eller kövis, reelltids- och tidsdelningsbearbetning tillämpas simultant. Den administrativa tid som åtgår i operativsystemet i sådana fall har visat sig vara av en icke acceptabel storleksordning (åtminstone för dagens maskinvarukapacitet). Även behovet av utrymmesresurser för operativsystemets residenta delar har i dessa fall vanligen visat sig överstiga vad som ursprungligen (vid anskaffningen)planerats.

Alltsedan de första datorerna har många "dataexperter" inom sig hyst drömmen om jättedatorn som med oöverträffad ekonomi skulle kunna utföra alla slags bearbetningar och tillämpa en mängd olika driftsfilosofier simultant. Som påpekats i kapitel 1 pågår ständiga försök till realisering av denna dröm och ett fåtal "giganter" har i ett lika fåtal exemplar skådat dagens ljus. Att tillverkarna överlevt dessa jättedatorers ekonomiska bakslag torde bero på framgången för deras övriga utrustning av mer måttlig kapacitet. Trots att det vid det här laget måste anses som styrkt att multiprogrammerade jättedatorers ekonomi jämfört med ekonomin för mindre datorer (vilka inriktats mot bestämda applikationer) eller nätverk av mindre datorer, ej alltid utfaller till de förstnämndas fördel tycks ordet "jättedator" besitta en magisk klang som gör politiker och andra beslutsfattare attraherade av dylika projekt.

Litteratur

1. Datortillverkares manualer och beskrivningar över resp operativsystem.

Böcker, som helt eller delvis behandlar området "operativsystem".

2. Cuttle G och Robinson P.B., (editors) "Executive programs and operating systems", Mac Donald, London, 1970. (Boken är huvudsakligen inriktad mot ICL-1900 seriens GEORGE-operativsystem)

3. Katzan Jr. H., "ADVANCED PROGRAMMING. Programming and operating systems", Van Nostrand Reinhold Computer Sceince Series, New York, 1970. (Boken handlar mest om programmeringsspråk och kompilatorterknik. Dock belyses operativsystem i ca 30 sidor av totalt 280 sidor)

4. Rosen S. (editor), "Programming Systems and Languages", Mc Graw Hill, New York, 1967 (730 sid) (Ca 210 sidor ägnas åt operativsystemprinciper och begrepp och består av 11 artiklar skrivna av skilda författare. Vid det här laget kan innehållet delvis anses något föråldrat)

5. JOBBÖVERVAKNING

5.1 Problemställning - jobbövervakarens uppgifter

Jobbövervakning är ett begrepp som, beroende på vilken typ av operativsystem som avses, kan omspänna en mängd olika funktioner. Vi sammanslår här dessa till ett system kallat "jobbövervakare" (JÖ). Bortsett från reelltidssystem och tidsdelningssystem, där begreppet "jobb" inte direkt kan urskiljas på samma sätt som för sats- eller kö-vis arbetande system, kan man säga att jobbövervakarens uppgifter normalt omfattar

- inläsning av jobb till inkö (jobb, omfattande styrinformation, program och ofta även data)
- interpretering (tolkning) av styrinformation
- analysering av behovet av resurser, vilka erfordras för att jobbet skall kunna utföras
- tilldelning och bokföring av resurser
- laddning av initiala programsegment för deljobben och initiering av bearbetningen
- ombesörjande av kommunikation mellan deljobb inom samma jobb
- ombesörjande av kommunikation med operatören (dvs sändning och mottagning av meddelanden avseende jobbövervakningen, monteringsanvisningar för perifera enheter osv
- avslutning av deljobb och avbokning av resurser som ej erfordras för nästa deljobb
- avslutning av jobb och i samband därmed avbokning av samtliga resurser som tilldelats jobbet
- framställning av ett resursförbrukningsbesked som dels kan användas som som debiteringsunderlag dels för senare grov analys av den belastning som olika jobb orsakat på datorsystemet. (Det senare kan vara till hjälp för planering av datordriften i syfte att höja dess effektivitet)

Vi noterar att JÖ här säges ha ansvar för vissa funktioner som mer i detalj utförs av OS-rutiner med andra namn. Vi syftar på initieraren, länkaren, laddaren, avslutningsrutinen m fl. Detta innebär ingen motsägelse

eftersom vi kan se JÖ som "överordnad" dessa specialrutiner. Om vi så vill kan vi t o m betrakta dessa rutiner som delar av JÖ. Konventionerna härvidlag skiljer sig mellan olika system.

Om operativsystemet arbetar med den ingående jobbströmmen residerande på ett direktminne kan varje där inläst jobb kommas åt direkt utan sekvensiell utsökning av filen. Detta ger jobbövervakaren en möjlighet att välja ut jobb ur jobbkön efter andra principer än FIFO (First In, First Out), dvs en möjlighet till planering och/eller prioritetsstyrd selektering av jobb.

Planeringsproblematiken är av olika orsaker komplicerad och kommer översiktsmässigt att diskuteras senare (kapitel 6).

De resurser som en jobbövervakare har ansvar för, dels för sina egna deljobb (dvs funktioner inom jobbövervakaren) och dels för till systemet inmatade jobb, är av kategorin "utrymmesberoende"[1] faciliteter såsom

- primärminnesutrymme
- sekundärminnesutrymme
- perifera enheter (även kommunikationslinjer).

Vilka olika faciliteter (samt konfigurering och kvantiteten av dessa) som en jobbövervakare har att administrera, bestäms vid initiering av densamma eller hos vissa andra system vid s k systemgenerering. JÖ administrerar "tabeller" som utvisar status (t ex upptagen, ledig, ej i drift, eller ur funktion) för samtliga systemets utrymmesberoende faciliteter.

JÖ administrerar däremot ej systemets "tidsberoende faciliteter", dvs vars utnyttjande mätes i använd tid. Exempel på sådana faciliteter är

- processor (-tid)
- kanaler (-tid)
- accessmekanismer till sekundärminne (-tid).

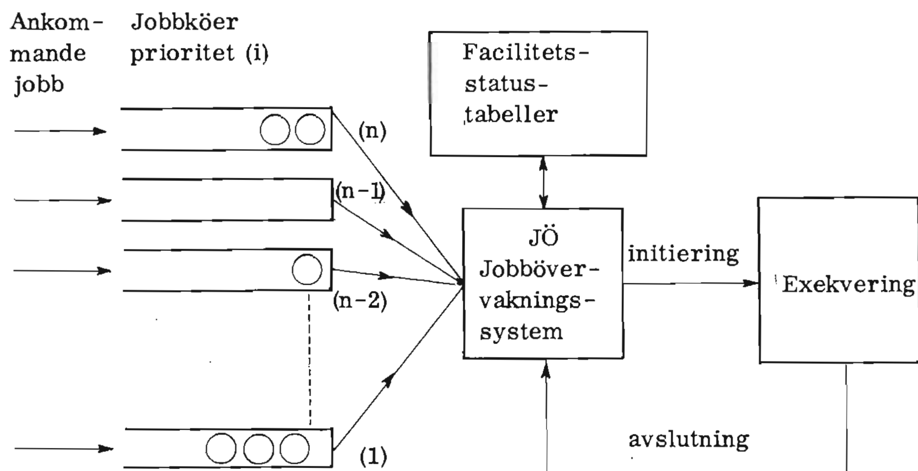
Administration av närmast ovanstående faciliteter (resurser) blir aktuell när ett deljobb exekveras (kan även vara ett JÖ-deljobb) och ombesörjes av styrprogrammet (se kapitel 7).

Prioritetsstyrd selektering av jobb innebär bl a att brådskande jobb kan tilldelas hög prioritet och att de därigenom får förtur i kön av jobb som

---

1) Deras utnyttjande mätes i använt antal (av någon lämplig sort t ex enheter, block, celler el dyl) och deras tilldelning till ett program gäller normalt under en tidsperiod motsvarande ett jobbsteg (deljobb).

väntar på bearbetning. Man kan skissera situationen som en jobbström fiktivt uppdelad på n separata köer figur 5.1:1.



Figur 5.1:1.
Prioritetsstyrd selektering av jobb. (n) antas ha högsta prioritet om dess resurskrav kan uppfyllas.

JÖ kan i enklare fall därvid välja det jobb

- vars resurskrav kan uppfyllas och
- som har högsta prioritet bland dylika jobb.

Mer "ambitiösa" (sådana som säges utföra inplanering) jobbövervakare tar även hänsyn till andra faktorer såsom

- andra jobb som befinner sig under exekvering och deras karaktäristika
- önskemålet att utnyttja vissa (eller alla) faciliteter så mycket som möjligt för att därigenom öka systemets totala produktionsvolym
- monteringstiden för perifera faciliteter, begärda av jobben
- möjligheten av avbryta ett eller flera pågående jobb (el deljobb), upprätta en återstartpunkt och lagra undan dem på ett sekundärminne för att därefter initiera särskilt brådskande högprioriterade jobb ur inkön

m m.

Möjligheterna att utforma olika slag av "planeringsalgoritmer" är ej sällan stora. Jobbövervakarens arbetsprinciper påverkas av de resursutnyttjande-principer som tillämpas hos operativsystemet. Till exempel, hos system som tillåter multiprogramkörning måste JÖ kunna initiera simultanbearbetning av mer än ett jobb och därvid ombesörja facilitetsbokföring och övrig administration för parallellt pågående jobb. Parallellt utnyttjande av resurser mellan olika jobb skapar risk för s k låsning av resurser (deadlock, "deadly embrace") med "stillestånd" som följd om inte särskilda hänsyn härtill tas vid planering och tilldelning av resurser. Detta problem har diskuterats i kapitel 3.

Minnestilldelnings- och administrationsprinciper påverkar också principerna för JÖ:s arbete. Till exempel,om primärminne tilldelas i form av

- regioner av fixerad storlek
- regioner av variabel storlek
- ej sammanhängande fragment

tillämpas ofta olika inplaneringsprinciper

5.2 Jobbövervakning och reelltidssystem

Något annorlunda är situationen beträffande jobbövervakning hos reelltids-system. Det existerar idag datorsystem vilka, enligt tillverkarna, arbetar under s k "real-time operating systems". Därvid uppstår främst två frågor:

1. på vilket sätt skiljer de sig från operativsystem utformade för sats- eller kövis bearbetning?

2. måste man ha ett "reelltidsoperativsystem" för att kunna implementera en reelltidsapplikation?

För att tillfredsställande kunna svara på dessa frågor måste ett "typiskt" reelltidsarbetes karaktär först belysas. Vissa karaktäristiska drag hos reelltidsbearbetning har redan behandlats i kapitel 4. Av vad där sagts framgår att vi främst kan skilja mellan två "ambitionsnivåer" hos ett reelltidssystem, beroende på om en ensam eller flera transaktioner kan hanteras simultant av detsamma. Denna simultanbearbetning (multithreading) ställer främst krav på operativsystemets styrprogram dvs de program som övervakar deljobb sedan de initierats. Detta kommer att beröras i kapitel 7.

Karaktäristiskt för RT-programmet (rättare sagt: RT-programsystemet, då det normalt består av en mängd delprogram av vilka endast ett fåtal samtidigt kan befinna sig i primärminnet) är att det ständigt[1] befinner sig i ett operativt tillstånd, dvs det är antingen transaktionsbearbetande eller väntande på nya transaktioner (meddelanden) från terminalerna. Då belastningen, orsakad av bearbetningen av ankommande meddelanden och karaktäriserad av meddelandetyper och ankomstintervallernas fördelning, varierar på ett normalt icke förutsägbart sätt[2], varierar också RT-programmets krav på datorresurser dels ur utrymmes- och dels ur tidssynpunkt. En annan viktig skillnad mellan RT-programmet och "konventionella"[3] program rör det förras interaktion med ett, ofta stort, antal terminaler av varierande typ, samt kravet på korta responstider.

De i tiden starkt varierande resurskraven och kommunikationen med terminalerna är sålunda de egenskaper som mest markant skiljer RT-jobb från övriga "konventionella" jobb.

Kravet avseende responstiden innebär att ett RT-jobb (meddelandebearbetning) ej kan startas genom JÖ:s försorg varje gång ett meddelande ankommer till datorsystemet. Tidsförlusten härvid skulle orsaka icke acceptabla dröjsmål. Tidsåtgången för JÖ att initiera ett jobb är normalt av storleksordningen 2-4 sekunder på en dator av storleksordningen IBM 360/50 mest beroende på åtkomsttider till skivminneslagrade kataloger, tabeller m m. De flesta reelltidssystemen har kravet att kunna behandla mer än 1 transaktioner per sekund. Vissa större system räknar med 20 transaktioner/sekund eller mer.

De flesta operativsystemen löser detta problem genom att RT-jobbet är operativt hela tiden som RT-systemet är "uppkopplat". Det innebär att inplaneringsproblem på jobbnivå ej är relevanta i detta sammanhang. RT-programsystemet kan därför i princip betraktas som ett jobb, som startas när RT-systemet avses börja sin serviceverksamhet och som avslutas först när systemets servicetid är över.

Även om praktiken i de flesta fall uppvisar RT-system som är helt dedikerade åt en viss applikation (eller en viss typ av RT-applikationer) indikerar de existerande RT-operativsystemens uppbyggnad önskemålet att i

---

1) Dvs så länge RT-systemet erfordras vara i drift.
2) Problemställningen diskuteras i samband med jobbprofiler under kapitel 6.
3) Vi använder här begreppet "konventionella" program för att beteckna sådana program som icke är av interaktiv karaktär och lämpar sig bäst för sats- eller kövis bearbetning.

"bakgrunden" kunna behandla ett eller flera parallella jobb av satsvis eller kövis karaktär. Anledningen är strävan att i möjligaste mån utnyttja datorsystemets samtliga resurser. Normalt är en RT-applikations belastning på ett datorsystem starkt varierande under dagen eller dygnet med perioder av mycket låg eller extremt hög aktivitet. Exempel på dylika applikationer ses t ex inom banker där toppbelastning kan förväntas, säg, mellan kl 8.15-9.00 och kl 11.00-14.00. Platsbokningssystem för flygbolag o d har ofta kravet att vara i ett operativt tillstånd dygnet runt med extremt låg aktivitet under natten som följd. För att med nöjaktig responstid klara av en toppbelastning under dagtid måste dylika system ha en kapacitet som vida överstiger behovet under perioder med låg belastning. Denna "överskottskapacitet" kan användas för satsvis eller kövis bearbetning, som därvid alltid går med lägre prioritet än RT-systemet. Under dagtid äger då oftast blott begränsad satsvis eller kövis bearbetning rum.

Det finns även andra orsaker till önskemålet att på ett och samma system mer eller mindre samtidigt kunna behandla både RT-program och bakgrundsjobb. RT-applikationen kan i och för sig belasta datorsystemet relativt jämnt under sitt operativa tillstånd, men dels kan belastningen vara låg eller huvudsakligen beröra en viss facilitet hos systemet (t ex en viss direktminnesaccessmekanism), och dels kan RT-systemet behöva åtkomst till filer vilka samtidigt används av övriga applikationer som bearbetas satsvis.

Vi återkommer nu till frågan om skillnader mellan ett "konventionellt" operativsystem och ett reelltidsorienterat operativsystem. Generella uttalanden kan här ej göras. Det förefaller dock rimligt att säga att ett RT-operativsystem innehåller vissa funktioner på styrprogramnivå som normalt ej återfinnes hos de konventionella systemen. Detta gäller främst

1. administration och hantering av kommunikation med avlägsna terminaler

2. administration av ett RT-programsystems i tiden starkt varierande krav på datorresurser främst avseende primärminne och ofta även arbetsutrymme på sekundärminnen

3. administration av (eller tillhandahållande av hjälpmedel för) programhanteringen för RT-systemet, dvs ombesörjande av inläsning av programmoduler från sekundärminne, överföring av kontroll mellan olika processer, samordning av inbördes beroende processers arbete osv

4. administration eller tillhandahållande av hjälpmedel för administration av

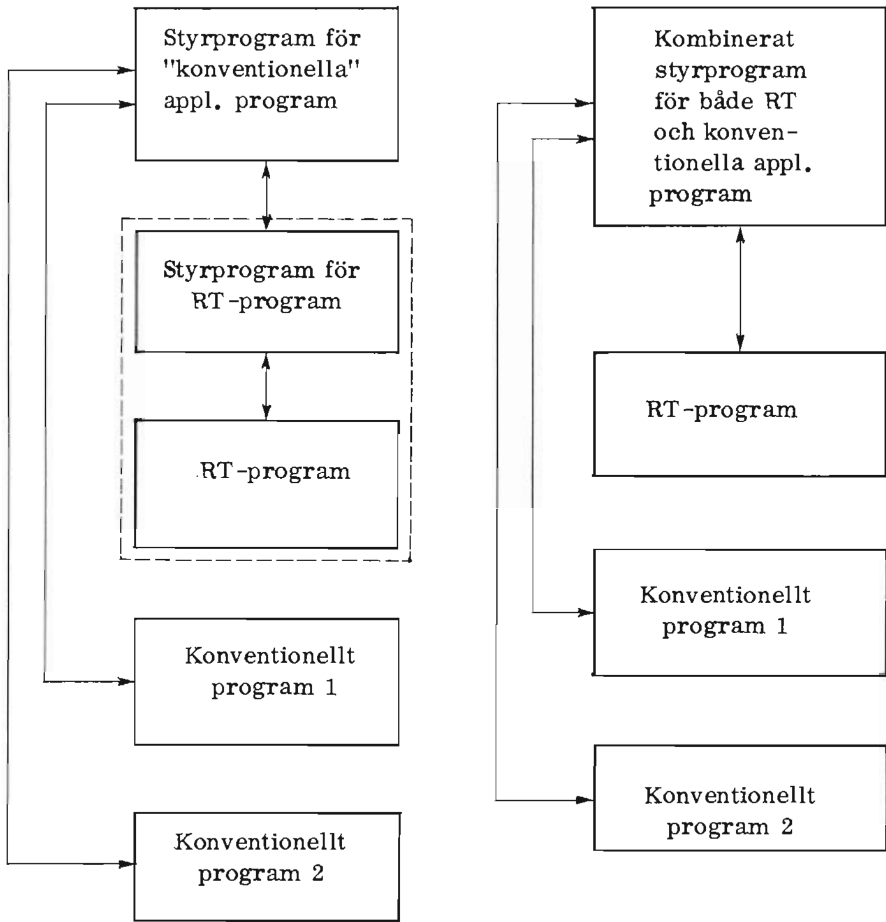
- katastrofsituationer (driftsavbrott)
- överbelastning
- programutprovning m m.

Flera mer avancerade "konventionella" system innehåller funktioner som på en grov betraktelsenivå förefaller motsvara önskemålen avs ovannämnda funktioner.

I praktiken har det dock visat sig att dessa system har behövt kompletteras med diverse specialrutiner för att kunna administrera RT-programs exekvering. Ej sällan har detta klarats på det sättet att man konstruerat ett speciellt styrsystem, som ur det egentliga styrprogrammets synvinkel arbetat som ett högprioriterat, ständigt i operativt tillstånd varande användarjobb, och som under sig administrerat bearbetningen av RT-applikationsprogrammen (fig 5. 2:1).

Därmed kan också den andra frågan besvaras. Det är sålunda i de flesta fall möjligt att komplettera ett konventionellt operativsystem med funktioner som gör det användbart i RT-sammanhang. Detta kräver dock en ofta icke obetydlig arbetsinsats. På senare tid har dock datortillverkare kunnat erbjuda en möjlighet att komplettera de konventionella styrsystemen med mer eller mindre specialsydda underordnade styrsystem för reelltidsapplikationer.

Frågan om det ena eller andra systemets för- och nackdelar i dessa avseenden är för komplicerad (och outforskad) för att för närvarande generellt kunna behandlas i en bok av denna typ. Potentiella användare kan endast manas till stor försiktighet vid utvärdering, då många system, betraktade på en grov nivå, förefaller besitta alla önskvärda egenskaper. Bristerna upptäcks ofta först när systemeringen och programmeringen har passerat "the point of no return".



Figur 5. 2:1
Två principer för grovstruktur hos ett operativsystem för såväl satsvis som reelltidsbearbetning.

5.3 Tidsdelningssystem (time-sharing (TS-) system)

Driftsegenskaperna hos TS-system medför att "jobb" i många fall ej kan urskiljas på samma sätt som hos sats- och kövis bearbetande system. Vid TS-system samspelar användaren interaktivt med systemet och överför därvid dels styr- och programinstruktioner och dels data till systemet. Man skulle kunna säga att jobbet består av den arbetsmängd som åstadkommes under den sammanhängande tid som användaren är inkopplad till "systemet". Under denna tid utför systemet en mängd olika "deljobb" åt användaren som kan variera i omfattning från syntaktisk kontroll av en inmatad instruktion till exekvering av ett eller flera användarprogram.

Man skulle således under rubriken "jobbövervakning" kunna sammanföra de funktioner hos TS-systemet som ombesörjer

- inbokning av användaren för ett körningspass
- tilldelning av sekundärminnesutrymme och övriga oftast "fasta" resursbehov under körningen
- bokföring av använda resurser, framställning av debiteringsunderlag och uppdatering av användarens "konto" avs tillgängliga medel (tid) för fortsatt körning
- avbokning av användaren efter dennes avslutade användning av TS-systemet.

Som framgår av ovanstående är den främsta skillnaden mellan JÖ för ett konventionellt system och JÖ ett TS-system kravet att den senare när som helst (så när som på interna kötider) skall kunna reagera på styrkommandon från samtliga terminaler och att ett deljobb initieras utan dröjsmål.

TS-system är huvudsakligen intressanta att betrakta ur jobbinplanerings- och styrprogramsynvinkel. Typiska styrinstruktioner för TS-system betraktas i kapitel 6. Styrning och administration av jobb (meddelanden) som redan initierats för bearbetning i ett TS-system betraktas i kapitel 7.

5.4 Jobbövervakarens struktur och arbetsprinciper

Som framfår av föregående avsnitt avses med JÖ ett begrepp som omfattar en mängd olika funktioner. Även om olika jobbövervakares struktur varierar starkt beroende på tillverkare,kännetecknas de grovt bl a av att vara uppbyggda av ett flertal program på olika nivåer. Därvid kan man

säga att ett JÖ-program på en viss nivå administrerar eller styr JÖ-program på lägre nivåer. JÖ:s arbete innebär ur styrprogramsynvinkel exekvering av en mängd olika processer (tasks). Den process som administrerar JÖ:s arbete befinner sig på den högsta nivån och genomlöps normalt i beordringstillstånd - dvs på samma skyddsnivå som de flesta styrprogramprocesser (se kapitel 7 betr styrprogram).

Vid beskrivning av JÖ:s struktur väljer vi att ej försöka finna "föreningsmängden av en mängd olika jobbövervakares funktioner". Ej heller har vi ansett det vara lämpligt att skissera en fiktiv jobbövervakare under arbetsnamnet "Anund" (Fred. den 10 juli, 1970) och framhålla den som en slags förebild.

Istället har vi valt att kortfattat betrakta huvuddragen hos några existerande jobbövervakare med vederbörligt angivande av fabrikat och källpublikation. Eftersom vi inledningsvis allmänt har talat om JÖ:s uppgifter kan dylikt till stor del undvikas nedan.

Anledningen till att vi har valt att beskriva just dessa jobbövervakare och inga andra, eller tvärtom, är ett intressant problem som vi skall be att få återkomma till i något annat sammanhang.

5.4.1 IBM OS/360 MFT II

Som torde ha framgått av de inledande kapitlen 1 och22 har IBM för sin maskinserie 360 ett flertal olika operativsystem som skulle kunna indelas i följande grupper.

1. TOS, DOS (tape resp disk operating system)

2. OS i olika versioner

3. övriga såsom HASP, EMFT osv.

TOS och DOS är mindre resurskrävande system som i huvudsak används på de mindre datormodellerna i 360-serien. Inom OS-gruppen finns operativsystem med varierande "ambitionsgrad" (och åtföljande krav på datorresurser) från PCP (Primary Control Program), som endast tillåter sekvensiell bearbetning av ett jobb i taget, till MVT (Multiprogramming with a Variable number of Tasks), som medger prioritetsstyrd inplanering/selektering av jobb och parallell bearbetning av ett variabelt antal processer (deljobb, tasks). MFT II (Multiprogramming with a Fixed number of Tasks, Version II) ligger mellan dessa ambitionsnivåer (dock betydligt närmare MVT) och innebär att prioritetsstyrd jobbinplanering/

selektering och parallell bearbetning, med vissa restriktioner, är möjlig. Vissa utmärkande egenskaper hos MFT II kommer att belysas nedan.[1]

Ur användarens synvinkel torde de mest framträdande egenskaperna hos MFT II vara följande:

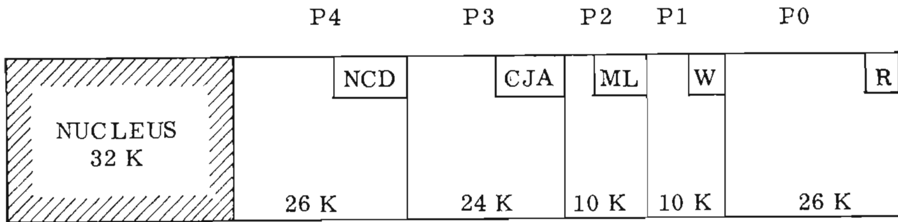
- systemet kan hantera ett vid systemgenereringen bestämt antal parallella processer. Antalet är maximerat till 52, dock är 3-6 vanligt i praktiken.

- systemet arbetar med en (i praktiken) fixerad indelning av minnet i ett antal partitioner (regioner) av fast längd. Maximala antalet partitioner som systemet kan hantera är 52. Detta utsäger också att multibearbetning av flera processer inom en partition ej är tillämpbar. Gränsen för aktuellt antal partitioner bestäms vid systemgenerering. Upp till denna gräns kan indelningen varieras under pågående drift (med vissa restriktioner). Exempel på en partitionering är visad i figur 5.4:1.

- bl a den fixerade minnesdispositionen har gjort det önskvärt att kunna "dirigera" jobb med olika krav på primärminnesresurser till olika partitioner av lämpligt avpassad storlek. Begreppet jobbklass har därför införts. Ett jobb kan, via JOB-styrsatsen, placeras in en av 15 olika klasser A-O. För var och en av dessa klasser kan en egen in-kö sägas existera (flera av dessa kan dock vara tomma beroende på om jobb av motsvarande klass tillförts systemet eller ej). Varje partition som definierats kan åläggas att handha jobb ur upptill tre olika inköer (dvs jobb-klasser). Man anger här också i vilken ordning dessa tre olika köer skall utsökas. Figur 5.4:2 visar en uppställning, som motsvarar partitioneringen enligt figur 5.4:1 och ur vilken kan utläsas att t ex avseende partitionen P3 skall först C-kön, sedan J-kön (om C-kön är tom) och sist, om J-kön också är tom, skall A-kön utsökas. Varje gång en ny jobbselekteringsfas börjar skall dock C-kön utsökas först. Om flera jobb existerar in en viss inkö valjes det jobb som har högsta selekteringsprioriteten eller det jobb som först ankom till systemet om de har samma selekteringsprioritet. Man måste skilja på selekteringsprioritet och processortidsprioritet (eng "dispatching priority"). I MFT II bestäms processortidsprioriteten av den partition som jobbet/deljobbet styrts till. Sålunda har PO den högsta

---

1) Av utrymmesskäl är framställningen kortfattad och delvis ofullständig. Den intresserade läsaren hänvisas till IBM publikationer, exempelvis IBM System/360 Operating System, Planning for Multiprogramming with a Fixed Number of Tasks, Version II (MFT II), File No. S360-36, Form C27-6939-0.



Figur 5.4:1
Exempel på partitionering (indelning i fixa minnesregioner) av ett 128K primärminne för MFT II (källa: IBM Form C27-6939-0). Fem partitioner P0-P4 för användar- och systemprogram har definierats. Den residenta delen av operativsystemet (NUCLEUS) är här visad uppta 32K[1] (den kan dock kräva mer utrymme beroende på konfigurering av systemet). NUCLEUS innehåller vitala program i styrprogrammet samt för jobbövervakningssystemet. Grupper av bokstäverna A-O som angivits för resp partition anger de jobbklasser som partitionen avses bearbeta. R innebär att partitionen reserverats för en resident läsare (systemprogram för läsning av det ingående jobbflödet). W innebär att partitionen reserverats för en resident skrivare (systemprogram för utskrift av det utgående resultatflödet). En partition kan bearbeta högst 3 jobbklasser. Jobben selekteras i den klassordning som bokstavsföljden anger och inom en klass på basis av den selekteringsprioritet som angivits för jobbet.

Jobbklasser:

Partitioner:	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	W	R
P0																	1
P1																1	
P2												2	1				
P3	3		1							2							
P4			2	3										1			

Figur 5.4:2
Fördelning av jobbklasser på olika partitioner. Siffrorna i matrisen anger selekteringsprioritet mellan jobbklasser där 1 är den högsta prioriteten.

---

1) K står här för Kb, dvs 1K = 1024 bytes.

prioriteten och P52, om sådan definierats, den lägsta prioriteten. Processortidsfördelningen mellan de olika partitionerna administreras av styrprogrammet. Vissa partitioner kan reserveras för speciella systemprocesser, s k läsare ("readers"-R) och skrivare ("writers"-W) och får ej användas för bearbetning av jobb av klasserna A-O. Läsare och skrivare behandlas mer i detalj nedan.

De olika funktionella komponenterna hos MFT II-systemet samt deras inbördes relationer avseende informationsflöde är visade i figur 5.4:3.

Komponenterna är[1]

- jobbdisponent (JD) - "master scheduler"
- jobbadministratör (JA) - "initiator/terminator" el "scheduler"
- läsare - (system input-) "readers"
- skrivare - (system output-) "writers"

Vi beskriver dessa komponenter kortfattat.

JD-jobbdisponenten är en del av NUCLEUS (se fig 5.4:1) och befinner sig således ständigt i primärminnet. Dess funktion är att ombesörja mottagning av diverse kommandon från operatören, eller från ingående jobbflöde, och se till att dessa verkställs, samt att ombesörja överföring av diverse meddelanden från de exekverande programmen eller systemet till operatören och vice versa. Exempel på operatörskommandon som mottas av JD är

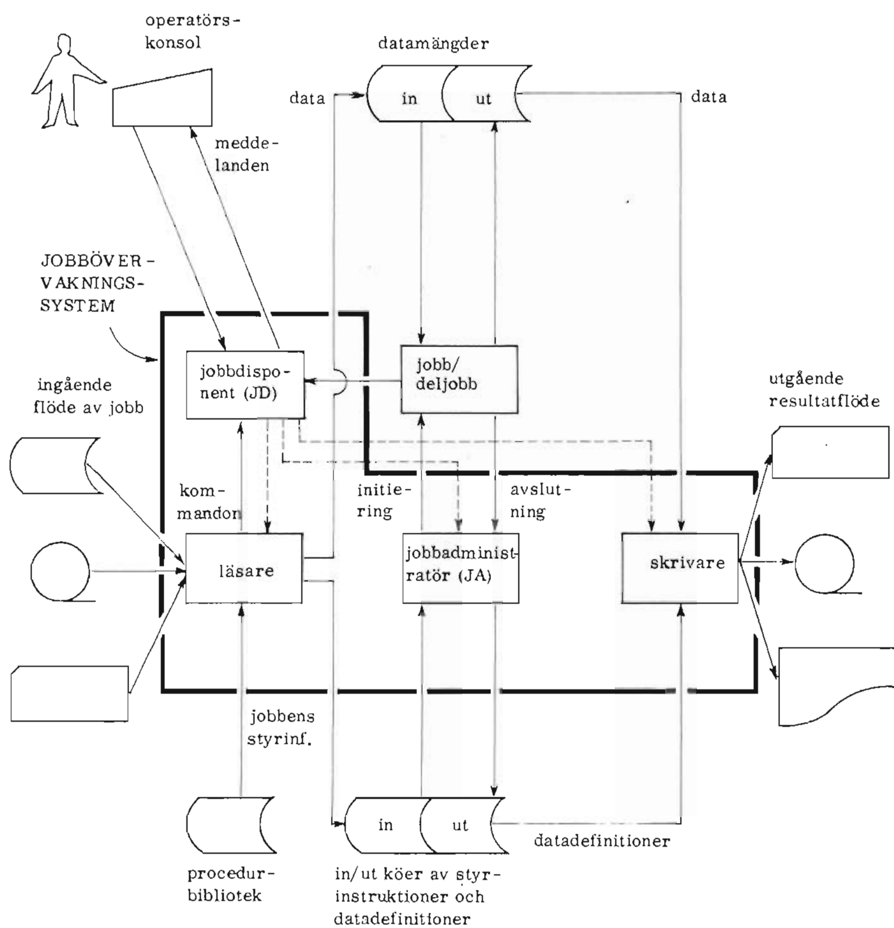
START (följt av lista av operander (parametrar)) startar läsare resp skrivare i angivna partitioner och läser resp skriver från resp till avsedd enhet

START INIT (operander)
startar en eller flera jobbadministratörer (JA)

STOP (operander)
används för att avsluta arbetet hos en läsare, skrivare eller JA.

---

1) Det vanskliga att till svenska översätta diverse programvarubegrepp torde vid det här laget vara konstaterat.



Figur 5.4:3
Skiss av jobbövervakarens struktur och arbetsprinciper.
(källa: IBM C 27-6939-0)

CANCEL (operander)
används för att omedelbart avsluta ett köande, under inplanering varande eller bearbetande jobb.

DEFINE LIST
används för att få besked om existerande partitionsindelning av primärminnet samt för att eventuellt ändra partitionsindelningen[1])

HOLD (operander)
används för att förhindra initiering av ett eller samtliga jobb i inköerna

Exempel på systemmeddelanden till operatör är

IEE607A DEFINITION PARAMETER ERROR, REPLY AGAIN

IEE610A PROBLEM PROGRAM PARTITIONS EXCEED 15, RESPECIFY

IEE611A CHANGE PARTITIONS NOT ADJACENT, RESPECIFY
osv

Pågående jobb kan sända meddelanden till operatör via JD, som då tillfogar en angivelse från vilken partition meddelandet härrör t ex

P3 HÖR DU PELLE, SKIVPACKE NR 2 PÅ 2314 HAR VISST FASTNAT I ETT SPÅR, SPÅR, SPÅR, ...

Något senare:

P3 NEJ, INTE DÄR, INTE DÄR, INTE DÄR, ...

Ytterligare något senare:

P3 TACK, NU KÄNNS DET BÄTTRE

Ur arbetssynpunkt kan JD sägas vara uppdelad på två processer, en som ombesörjer kommunikationerna mellan operatören och systemet och en som utför aktuella kommandon.

---

1) Ett krav för en omdefinition av partitioneringen är att jobb i motsvarande partitioner är avslutade. Två partitioner kan "slås ihop" till en, endast om de gränsar till varandra. Partitioners storlek definieras i moduler om 1K bytes. Den minsta tillåtna partitionsstorleken är 8K bytes.

JA-jobbadministratören är ett icke-resident systemprogram som arbetar i en av de s k "problem-program"-partitioner som definierats. Vissa regioner används alltså omväxlande för systemprogram och användarprogram. JA är aktiv endast under kortare tidsperioder vid jobb-byten resp jobbstegsbyten (dvs mellan deljobb) och behöver därför ej uppta permanent primärminnesutrymme. Det finns två typer av JA hos MFT II. En som kräver 26 Kbytes primärminne och en mer avancerad version som kräver 44 Kbytes. Beroende på vilken JA som systemet tillämpar krävs det att minst en av partitionerna är av storleken minst 26 Kbytes resp 44 Kbytes. Detta har följden att jobb- eller jobstegsbyte för partitioner med storleken mindre än 26 Kbytes (resp 44 Kbytes) endast kan ske när en partition av storleken 26 Kbytes (resp 44 Kbytes) eller större blivit ledig.

Funktionellt kan en jobbadministratör uppdelas i en initieringsfunktion (IF) och en avslutningsfunktion (AF).

För att selektera ett eller flera jobb placerar systemet JA i en tillgänglig partition. IF undersöker då först om någon "liten" [1] partiton behöver inplanering och ombesörjer tillsammans med AF avslutning av färdigbehandlat jobb och ev initiering av ett nytt. När samtliga små partitioner fått små JA-servicekrav tillgodosedda initierar JA ett jobb i den partition som den själv nyss arbetade i. Som tidigare nämnts sker selektering av jobb för initiering på basis av

- jobbklass
- prioritet inom en jobbklass
- tillgänglig partition, motsvarande aktuell jobbklass.

IF tilldelar också jobb erforderliga perifera enheter och ombesörjer via andra systemrutiner att erforderligt sekundärminnesutrymme tilldelas jobben. Krav på dylika resurser måste uppfyllas innan ett jobb skall kunna initieras.

JA ger dessutom, via JD, meddelanden till operatör angående eventuell montering/nedmontering av magnetband, skivpackar m m. JA gör också erforderliga kontroller för att förvissa sig om att de enheter som monterats motsvarar de krav som framställts i motsvarande jobbs styrsatser.

---

1) Med "små" partitioner avses sådana partitioner som ej rymmer en jobbadministratör dvs < 26 Kbytes resp < 44 Kbytes.

Vi kan ej här gå in på en detaljerad diskussion om tilldelning av perifera enheter (device allocation)[1]. Sägas bör dock att detta är ett område där stora effektivitetsvinster kan göras med hjälp av planering.

Av kostnadsskäl är antalet perifera enheter hos en datorinstallation begränsat. Vid system som tillämpar multiprogrammering är konflikter avseende perifera enheter oftast orsaken till att två jobb ej kan köras parallellt. Risken för konflikter kan minskas om kravet på perifera enheter framställs med så få restriktioner som möjligt.

Till exempel, om kravet på en enhet enbart är att man där skall kunna lagra en sekvensiellt åtkomlig datamängd kan detta krav uppfyllas dels av direktminnen dels av bandstationer. Exempel på ett icke flexibelt krav är behovet av en viss bandstation på en viss kanalenhet.

Andra JA-funktioner, som kan öka utnyttjandegraden hos datorsystemet, och som kan sägas tillhöra den mer avancerade JA-versionen är

- monteringsbesked för band, skivpackar o d till operatör avges i god tid före kraven blir aktuella. Montering för ett jobb kan på så sätt helt eller delvis överlappas med bearbetning av i tiden närmast framförliggande jobb.
- automatisk igenkänning av förmonterade magnetband eller skivminnespackar. Operatören kan härigenom i god tid montera band och skivminnen som kommer att erfordras senare. JA noterar vilka enheter som monterats och behöver sedan, när dessa enheter krävs, ej ge monteringsanvisningar (med tidsvinst som följd).
- igenkänning av jobb som ej erfordrar montering ("non-setup" jobb). När initiering av ett jobb eller deljobb fördröjs på grund av långa monteringstider kan JA söka av inkön och initiera det "nonsetup" jobb som har högsta prioritet. När detta jobbs första deljobb avslutats kan en kontroll göras om monteringen för det fördröjda jobbet är klar osv.

Avslutningsfunktionen (AF) har först att bestämma huruvida ett jobb eller ett deljobb skall avslutas. Det arbete som här skall utföras har beskrivits i avsnitt 3.2.1. Utdata från ett jobb består dels av datamängder från problemprogrammet dels av datamängder som alstrats av operativsystemet (JÖ, styrprogrammet m fl). Utdata kan indelas i ett antal <u>klasser</u> som närmare beskrivs i samband med skrivaren. Data-definitioner avseende

---

1) Läsaren hänvisar till "IBM /S/360, Concepts and Facilities", Form C28-6535-0, p. 56.

ett jobbs olika utdatamängder placeras i köordning i en utkö. Element i denna kö "pekar på" läget av de aktuella datamängderna. Kön avsökes och åtgärdas av en eller flera skrivare.

Läsare (system input readers) är systemprogram, vars exekvering ombesörjer inläsning av det ingående jobbflödet, omfattande styrinstruktioner, program och data. Så länge det finns något att läsa arbetar dessa parallellt med systemets skrivare och bearbetning av jobb (användar- eller problem-program). Systemet tillåter parallellt arbete av högst tre läsare som då kan arbeta från "problempartitioner" eller speciella R-partitioner. Det finns två typer av läsare med minneskraven 26 Kbytes resp 44 Kbytes. Valet av lämplig typ beror på vilken JA som valts. Man skiljer också mellan

- residenta läsare
- transienta läsare

En resident läsare, som startas i t ex P0 genom operatörskommandot:

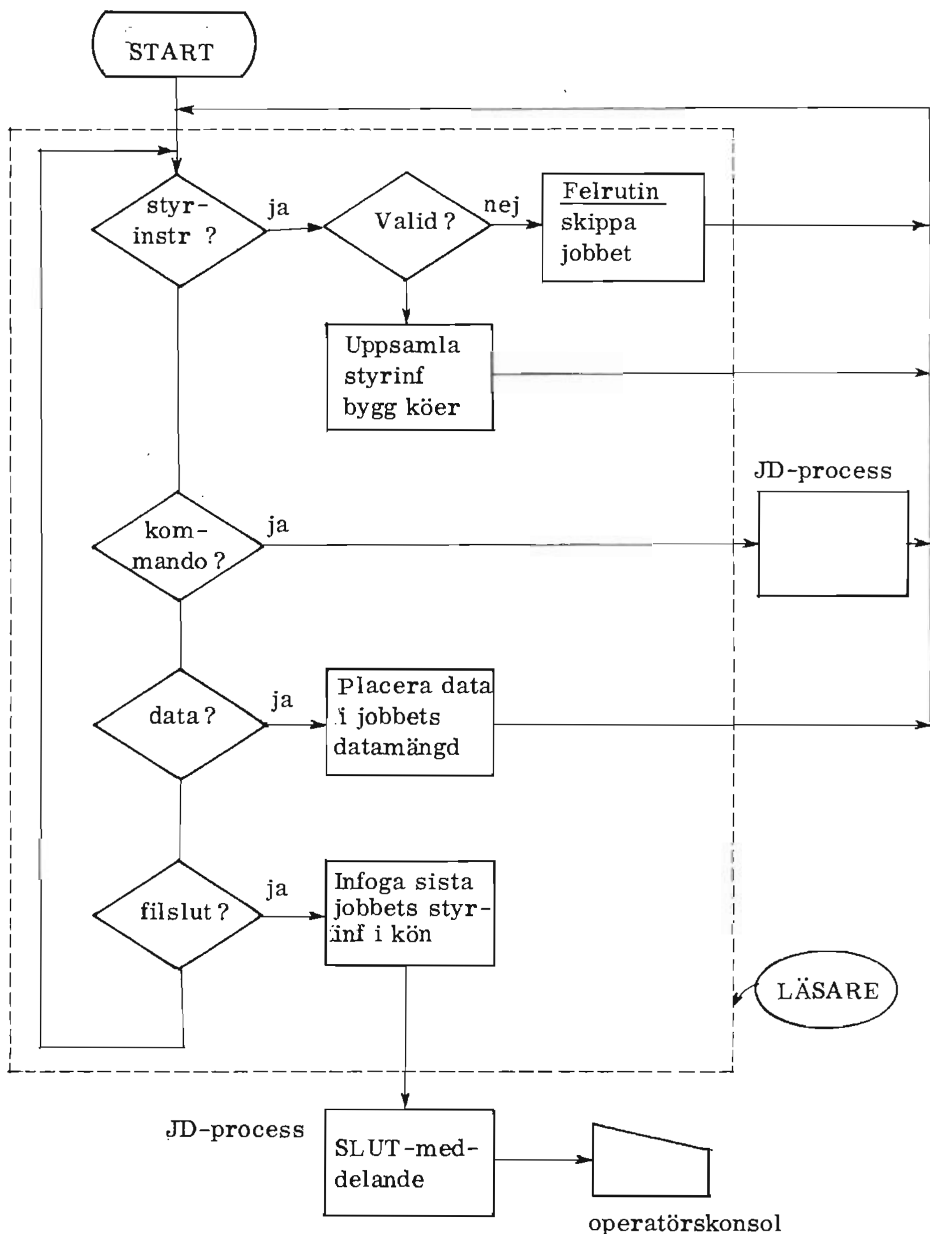
> START READER (P0) (namn på periferenhet som innehåller inflödet).

läser inflödet (så länge det finns något eller till dess att den STOPpas) till samtliga inköer.

En transient läsare, som arbetar "tillfälligt" i en problem-partition, läser inflödet för just den partitionen eller för en "liten" partitions jobbklasser. När ett jobb inlästs överlämnas kontrollen till JA, som initierar jobbet. När jobbet avslutats och inget annat "in-köat" jobb behöver partitionen inläses den transienta läsaren dit och kan återuppta sitt arbete. En transient läsare kan antingen åläggas att arbeta i en viss bestämd partition eller också kan dess placering från fall till fall avgöras av systemet (dvs i första tillgängliga partition av tillräcklig storlek).

Valet mellan transienta resp residenta läsare är en effektivitetsavvägningsfråga och beror på volymen hos det ingående jobbflödet och tillgängligt primärminne.

En läsares arbete består kortfattat av att inläsa och analysera inlästa data och placera dem i lämpliga datamängder resp köer och initiera viss annan aktivitet som framgår av figur 5.4:4.



Figur 5.4:4
Läsarens arbetsförlopp (källa: IBM, Form C27-6939-0).

Skrivarens (system output writer) uppgift är att ombesörja utmatning av dels problem-data, och dels systemmeddelanden från sekundärminnesutrymmen till avsedd utmatningsenhet. Därvid avsöker skrivaren den kö av "datadefinitioner" som successivt byggs upp av JA. Skrivarens primärminnesbehov är av storleksordningen 10 Kbytes. På samma sätt som för läsare kan residenta resp transienta skrivare urskiljas. De arbetar i en R-partition resp i någon användar-partition. Skillnaden avseende transienta läsare är att de startas och avslutas av operatör.

Utdatamängder härrörande från bearbetning av ett jobb kan av användaren klassificeras i olika utdataklasser (s k SYSOUT-klasser, obs dessa har inget direkt samband med jobbklasser). Upp till 36 olika klasser (A-Z, 0-9) kan specificeras. En skrivare kan specificeras att ombesörja utmatning från max 8 utdataklasser. Om man t ex anger att en skrivare skall ansvara för utmatning av utdata i klasserna A, B och C sker utmatningen i samma prioritetsordning. Dvs A avsökes och sedan B endast om A är tom. Efter det att en datamängd utmatats sker avsökning åter med start från A. Möjligheten att kunna ange olika klasser för utdata kan utnyttjas på olika sätt. Genom att upp till 36 skrivare simultant kan vara i arbete och kan kopplas till olika utdataklasser kan klassbegreppet användas dels för att styra utmatningen till olika (ev på geografiskt skilda platser belägna) enheter dels för att separera olika slag av utdata. Det kan t ex vara önskvärt att separera systemmeddelanden från problem-utdata i det fall man använder sig av förtryckta blanketter o dyl.

Allmänt om läsare och skrivare kan sägas att, i erforderliga fall, specialversioner givetvis kan tillverkas av användaren själv. Då läsaren även ombesörjer viss tolkning av indata är dessa svårare att tillverka på egen hand än skrivare. Härigenom kan t ex inläsning från och skrivning till avlägsna terminaler av annat fabrikat men med kompatibel datarepresentation ordnas.

Systemets MFT II användning för reelltidsjobb organiseras vanligen så att ett motsvarande antal partitioner reserveras för dessa, ur systemets synvinkel, "oändligt" långa jobb. Då dessa jobb är tidskritiska ges de normalt högprioriterade partitioner (P0, P1, osv). Som tidigare (avsnitt 5.2) påpekats kan bearbetningen av transaktioner inom en RT-partition vara mer eller mindre komplicerad ur multiprogrammeringssynvinkel. RT-jobben har normalt inget behov för ovan nämnda läsare och skrivare, utan ombesörjer in/utmatning till terminaler via speciella "telekommunikationsprogram" som arbetar inom motsvarande partition (inlänkade hos programmet).

På i princip samma sätt kan andra partitioner nyttjas av jobb med t ex grafisk in/utmatning eller mediaomvandlingsprogram.

Allmänt kan sägas om MFT II's (och liknande systems) jobbövervakningsprinciper att partitions- och klassbegreppen lämnar merparten av ansvaret för ett effektivt utnyttjande av systemet på resp användare. Val av oläm - lig partitionering eller olämplig klassindelning av jobb kan ha följden av att endast en ringa del av systemets "maximala prestanda" kan utvinnas för en viss installation. Av bl a vad som redogjorts om MFT II bör med önskvärd tydlighet framgå att "datorjobb-produktionsplanering" är ett utomordentligt intrikat problem med ett stort antal variabler och faktorer. Författarna veterligt existerar ännu ingen tillräckligt realistisk, matematisk modell för jobbinplanering, t ex för drift enligt MFT II principen. Ansatser till diskret simulering av jobbearbetningen görs dock på olika håll.

Det bör, för undvikande av missförstånd, särskilt betonas att planeringsproblem ej enbart uppträder hos system av MFT II-typ. Dessa problem återfinnes hos samtliga system med ett någorlunda "avancerat" operativsystem. Planeringsproblemen ökar normalt i vikt ju större datorsystem som kommer i fråga. I allmänhet torde tillfredsställande kapacitet kunna utvinnas ur datorsystem om deras drift ställdes under databehandlingstekniskt sakkunig ledning. Med detta vill vi påstå att det i allmänhet ej räcker med en operatörsutbildning för att framgångsrikt leda driften på ett datorsystem med ett någorlunda komplicerat operativsystem.

För att utvidga belysningen kan påpekas att kravet på högt utnyttjande av datorn måste balanseras mot kravet på rimliga omloppstider för jobb. Det går t ex normalt inte att inplanera jobb (teoretiskt) flera dagar fram i tiden bara för att det skulle vara lämpligt ur resursutnyttjandesynvinkel.

I figur 5.4:5 visas ett fingerat exempel på bearbetning av nio jobb i en dator med ett 128 Kbytes primärminne med partitionsindelning enligt figur 5.4:1. Figuren kan ses som ett koordinatsystem med primärminnesutrymme som vertikal axel och tid som horisontell axel. [1)]

Vi anger nedan några karakteristiska tidpunkter i diagrammet:

T0 Systemet initialiseras och NUCLEUS inläses

T1 Operatör startar läsaren i P0 som börjar läsa inflödet av jobb

---

1) Tidsdimensionen är av överskådlighetsskäl skalmässigt förvanskad. T ex tar jobbinplanering (JA) en tid av storleksordningen 1-5 sekunder. Ett jobb kan givetvis hålla på i flera minuter.

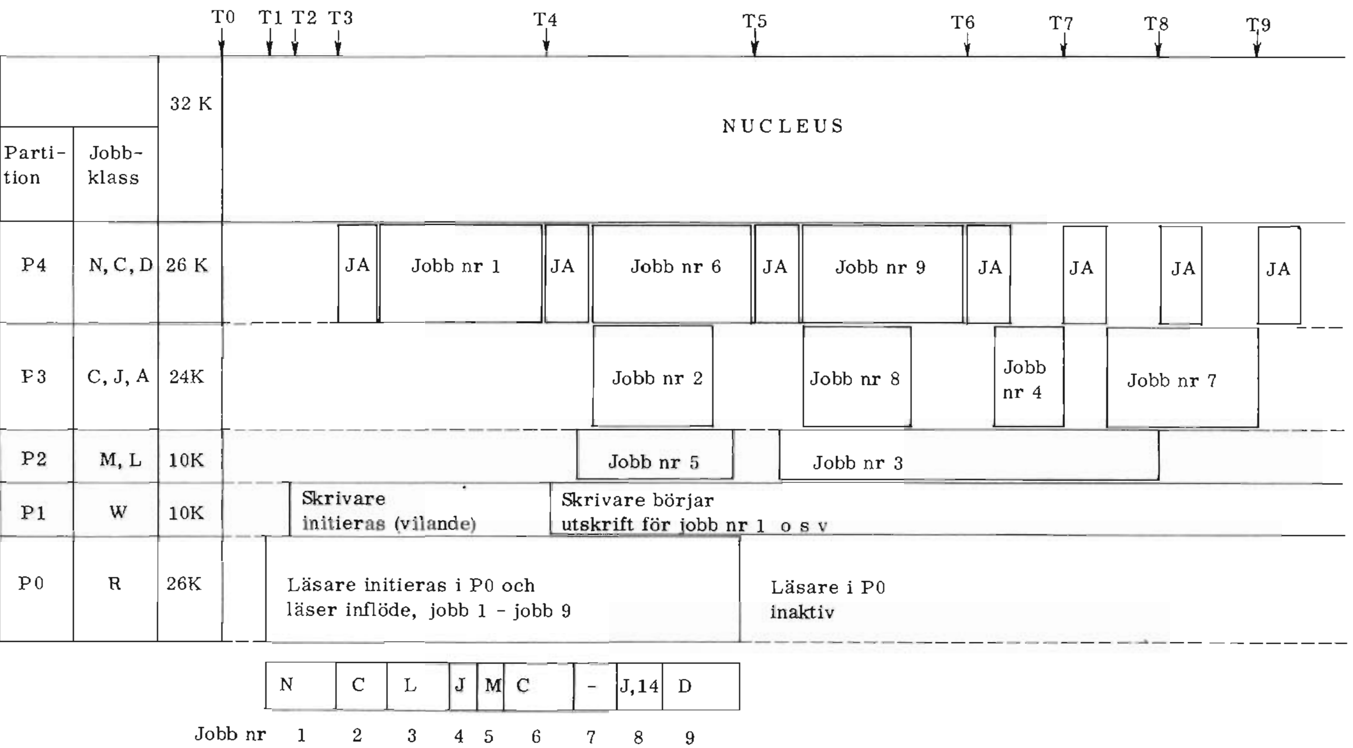


Fig 5.4:5

T2 Operatör startar skrivaren i P1 som dock till en början är inaktiv i brist på data att mata ut

T3 Jobbadministratören (JA) startar upp jobb nr 1 som har klass N och sålunda skall in i P4

T4 JA avslutar jobb 1 och initierar jobb 2 i P3 och jobb 5 i P2

T5 JA avslutar jobb 5, 2 och 6 samt startar jobb 3 i P2, jobb 8 i P3 och jobb 9 i P4

T6 JA avslutar jobb 8 och 9 samt startar jobb nr 4 i P3

T7 JA avslutar jobb nr 4 och startar jobb nr 7 i P3

T8 JA avslutar jobb nr 3

T9 JA avslutar jobb nr 7

Som framgår av figuren kräver JA att utrymme av 26 K och måste därför alltid ha tillgång till P4. Alternativt hade P0 vid T5 kunnat omdefinieras till en problempartition med klasserna J och A och vi hade då kunnat minska totaltiden för bearbetning. Men sådana omdefinieringar är givetvis vanskliga och kräver viss planering. Till slut kan observeras att, om systemet arbetar med en resident läsare och en resident skrivare, åtgår 68/128 av primärminnet till residenta systemprogram. Mindre än 50 % återstår för användarprogrammen. Dylik minnesaptit är typisk för många moderna operativsystem.

5.4.2 General Electric GECOS III

GECOS III (General Electric Comprehensive Operating System III) är det operativsystem som tillhandahålls kunder som använder någon av datorsystemen i General Electrics 600-serie. Denna serie omfattar flera datormodeller, vilka samtliga kan sägas tillhöra den "stora" datorklassen. För denna serie tillhandahålls GECOS III i form av ett "take it or leave it"-system, dvs kunderna har ej möjlighet att välja mellan olika system, på skilda ambitionsnivåer. Vissa möjligheter att påverka systemets funktion finns. Dock avses "normala" användare ej ge sig in härpå annat än i begränsad omfattning. Den information som nedan lämnas härrör från publikationen GE 600 Line GECOS III, General Electric CPB-1488B.

GECOS III är ett system för multiprogrammering, med vars hjälp simultant kan utföras:

- satsvis och/eller kövis lokal bearbetning
- satsvis och/eller kövis bearbetning från avlägsna terminaler
- time-sharing bearbetning

Systemet skiljer sig från ovan belysta IBM OS MFT II bl a genom att inga fasta primärminnesregioner kommer till användning, utan användarprogram (processer) tilldelas så mycket primärminne de behöver, och laddas till från början till adress ej fixerade områden i primärminnet. Dynamisk relokering tillämpas. Tilldelade primärminnesområden skall dock vara sammanhängande för varje process, fragmentering tillämpas ej. I detta avseende liknar GECOS III snarare IBM OS MVT, liksom flera andra på marknaden existerande system.

Primärminnesbehovet för den residenta delen av systemet, kallad HCM (Hard Core Monitor), ligger mellan 10 och 16 K 36-bitsord, beroende på maskinkonfiguration. HCM's primärminnesbehov varierar ej under bearbetning.

Systemet kan styra flera processorer. Inom HCM befinner sig bl a

- tabeller för referens till aktuella periferienheter
- utrymme för kommunikation mellan användarprogram och operativsystem
- vissa avbrottsrutiner
- vissa felrutiner
- styrfunktion för processortidsfördelning (the dispatcher)
- styrfunktion för tilldelning av primärminne (core allocator)

m m

I GECOS III har filosofin med program i priviligierat resp användartillstånd lämnats, och alla systemfunktioner utförs via vanliga anrop från användarprogram till systemrutiner. Förutom för själva programmet erforderligt primärminne tilldelas varje användarprogram vid laddning ett till programmet associerat systemområde, kallat Slave Service Area (SSA). I detta utrymme befinner sig all information som erfordras för kommunikation mellan användarprogrammet och styrsystemet. I SSA befinner sig bl a

- I/U-tabeller (köutrymme, tilldelningstabeller för periferienheter m m)
- systemanroplänkar
- länkar till för programmet aktuella filer

m m.

En fördel med denna princip kan sägas vara att användarprogran enkelt kan rullas ut resp rullas in (se nedan). Därvid åtföljs varje användarprogram ständigt av sin SSA, där bl a all erforderlig statusinformation befinner sig.

Jobbövervakaren (JÖ) hantering av jobbflödet sker i GECOS III på följande sätt:

1. Ett jobb inträder i systemet via en direktansluten inläsningsenhet eller avlägsen terminal, under kontroll av ett inläsningsprogram (Input Program, motsvarar tidigare nämnd Reader). Jobbet placeras i en in-kö på sekundärminne. I denna kö, där upp till 50 jobb kan befinna sig, kvarhålls jobbet även sedan det laddats, och ända till dess det producerat all sin utmatning. Härigenom möjliggörs återstart från in-kön i händelse av systemfel (redan inlästa jobb behöver sålunda därvid ej återinläsas, återinläsning krävs emellertid för jobb som vid systemfeltidpunkten befann sig under inläsning).

2. Inläsningsprogrammet bygger upp en fil med styrinformation avseende jobbet, bl a hämtad ur jobbets styrsatser, och levererar denna fils identifikation till en kö hörande till den del av JÖ som har ansvar för tilldelning av yttre resurser (peripheral allocator, PA).

3. HCM uppmanas att ladda PA till minnet, och detta program avsöker den nämnda kön sekvensiellt, noterar om för jobbet erforderliga yttre resurser finns tillgängliga och tilldelar i så fall dessa till jobbet. I annat fall betraktas nästa jobbs behov i kön. Därefter informeras JÖ's primärminnestilldelare (core allocator, benämnd GEPOP, vilken befinner sig resident i minnet) om att jobbet är redo för exekvering.

4. När tillräckligt primärminne finns tillgängligt, tilldelar GEPOP erforderligt dylikt till jobbet. Laddaren kallas in till minnet, och efter laddning av jobbet påbörjas exekveringen.

5. Utmatning från jobbet uppsamlas i en utmatningsfil på sekundärminne (en systemfil), som efter jobbets avslutning läses av ett utmatningsprogram (jfr Writer) och sänds till avsedda periferienheter (lokalt eller avlägset belägna).

Kommunikation med operatören sköts i GECOS III av samma del av JÖ som ansvarar för primärminnestilldelning (GEPOP). Om operatören önskar kommunicera med systemet genererar denne på konsolen ett avbrott, varpå GEPOP där skriver ut "???". De meddelanden som operatören därefter ger är alltid av formen:

Ett verb (6 eller färre karaktärer), följt av
ett argument (12 karaktärer)

Meddelanden från systemet till operatören kan emellertid vara längre och mer beskrivande än så.

Låt oss nu något närmare betrakta primärminnestilldelningen i GECOS III och speciellt dess samröre med prioritetsmekanismen. Varje jobbs slutliga prioritet är en funktion av

- den externa prioritet som angivits i jobbets inledande styrsats
- jobbets erforderliga systemresurser
- den tid jobbet hittills fått vänta på bearbetning

Primärminnestilldelaren inom JÖ arbetar enligt regler som nedan beskrivs:

- Processer med prioritet noll anses över huvud taget ej vara "kandidater" för primärminnestilldelning. Ett jobb med prioritet noll kan endast ges högre prioritet (och därmed komma i fråga för minnestilldelning) genom manuellt ingripande från operatören på konsolen. Härigenom möjliggörs att jobb förvaras ständigt väntande inom systemet, för att tas fram till bearbetning då operatören noterar att systemet är speciellt lågt belastat.

- Processer med prioritet större än noll tilldelas primärminne i ordning efter prioritet. För processer med samma prioritet sker tilldelningen i ordning efter väntetid i systemet.

- En tröskelprioritet kan fixeras inom systemet (en systemparameter). Om prioriteten för en process är större än tröskelvärdet kommer processer med prioritet under tröskelvärdet att rullas ut till sekundärminne, för att bereda plats åt den högprioriterade processen.

- Ingen process med prioritet under tröskelvärdet tilldelas minne innan en process med prioritet över tröskelvärdet, som väntar på att tillräckligt minnesutrymme skall bli ledigt, tilldelats sitt minnesutrymme.

Den nämnda principen med tröskelprioritet är viktigt i time-sharing sammanhang (och även i reelltidssammanhang). Därmed kan undvikas att tunga pågående satsvisa processer oacceptabelt försenar processer som kräver snabb behandling.

För att förtydligat ge den intresserade en uppfattning om principerna för primärminnestilldelning i GECOS III rekommenderas studium av figur 5.4:6.

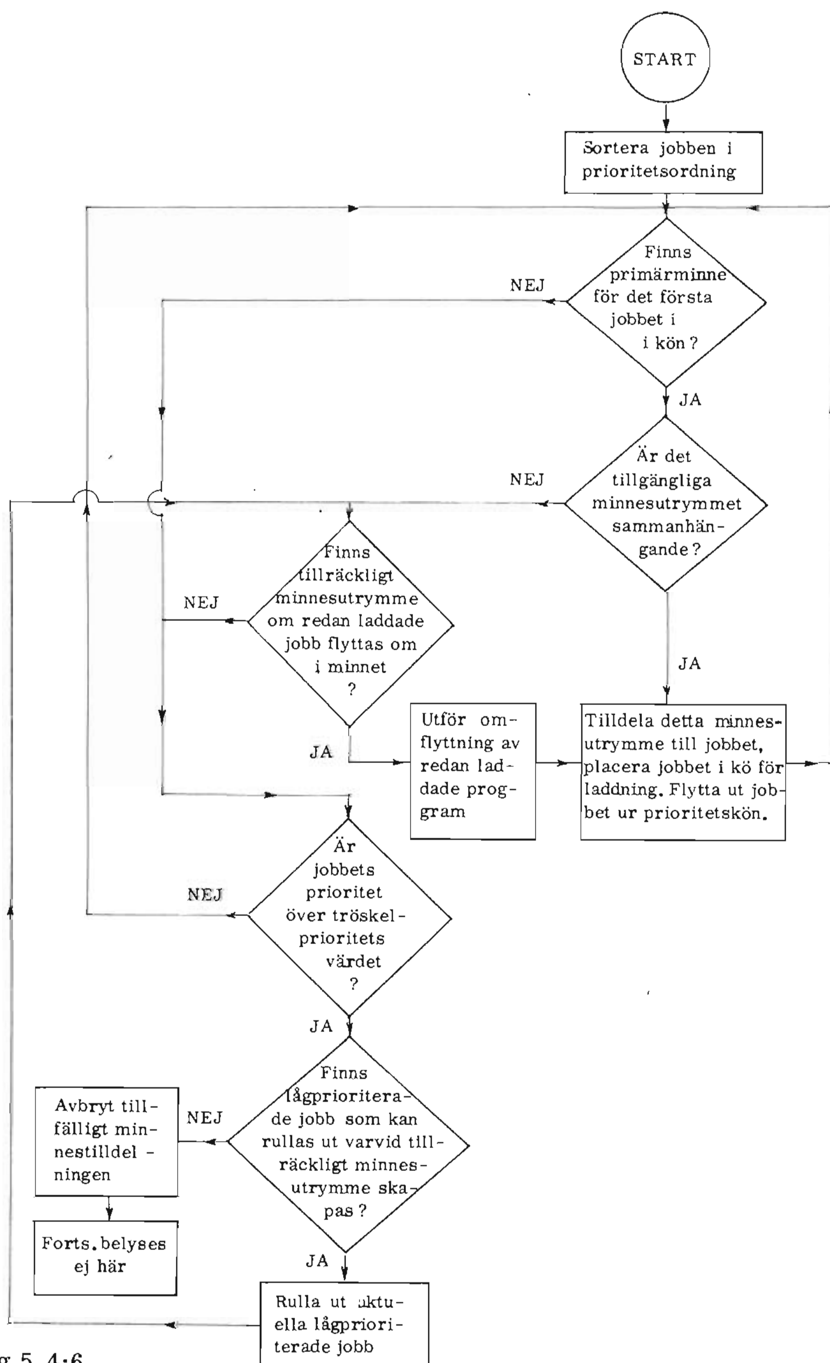


Fig 5.4:6
Del av primärminnestilldelningsalgoritmen för GECOS III.

Vi ger nedan, och avslutningsvis, en belysning av hur jobbövervakning i time-sharing tillgår i GECOS III. Härvid kommer även fördelning av processortid att något beröras, då dessa funktioner intimt samspelar.

I time-sharing kan, som nämnts, knappast samma typ av "jobb" urskiljas som i satsvis miljö. "Time-sharing jobb"motsvaras snarast av enskilda dialoginteraktioner. I GECOS III framräknar primärminnestilldelningsprogrammet minnesprioriteten för varje dialoginteraktion. Denna prioritet baseras på förhållandet mellan den tid terminalanvändarens kommando fått vänta och summan av den processor- och in/utmatningstid som dittills använts för kommandot (ett kommandos behandling i systemet blir normalt frekvent avbruten innan den är komplett). Med jämnda tidsmellanrum gör systemet försök att tilldela primärminne till jobb med högsta prioritet.

Åtskilliga andra synpunkter har emellertid inflytande över primärminnestilldelningen, och därpå följande laddning och exekvering:

1. Efter det att ett jobb har laddats ges det åtminstone ett kvantum processortid, eller blir ovillkorligt kvar i minnet en viss specificerad tid (en systemparameter, väsentligt längre än ett kvantum) innan det kan rullas ut p g a andra väntande högprioriterade program. Därmed undviks situationen att ladda ett jobb för att omedelbart därefter tvinga det att rullas ut igen. Om ett jobb befunnit sig i minnet den nämnda,på förhand specificerade,tiden betraktas jobbet som en "utrullningskandidat", dvs må rullas ut av högre prioriterade program. Därmed undviks att jobb med lågt behov av processortid men med omfattande behov av in/utmatning kan uppta minne för lång tid. (Parametrarna för ett kvantum samt max-tid-i-minnet kan justeras alltefter vunnen systemerfarenhet).

2. Vid tilldelning av primärminne till aktuella jobb görs försök av systemet att fördela jobben så att bästa minnesutnyttjande uppnås, dvs så att summan av outnyttjade "hål" i minnet minimeras (varje jobb tilldelas, som nämnts, sammanhängande minnesutrymme). Om ett jobb är för stort för att rymmas i tillgängliga "hål" lämnas det tillfälligt åsido, och nästa högprioriterade jobb betraktas.

3. Ett jobb som är vilande i primärminnet p g a avvaktan på slutförd in/utmatning, må rullas ut.

4. Samtliga jobb i minnet tillåts inte rullas ut "samtidigt" (eller nästan samtidigt) för att bereda plats åt nya högprioriterade jobb, emedan därvid under utrullningen processorn (eller processorerna) inte skulle ha något jobb att bearbeta, bortsett från "cycle stealing" under utrullningen. Minst ett jobb som kan förbruka processorkapacitet behålls sålunda i primärminnet.

5. Processortid fördelas, i huvudsak, mellan laddade jobb till det jobb som dittills konsumerat, eller tilldelats, minst processortid.

Vi har härmed belyst några av egenskaperna hos dels styrprogram- dels jobbövervakningsfunktionen i GECOS III, och lämnar därmed detta specifika operativsystem.

5.4.3 DataSAAB D22 MK-Dirigent

MK-Dirigenten (MK står för Multi-Körning) är ett av två olika operativsystem som tillhandahålls användare av datorsystemet DataSAAB D22. Detta system arbetar med multiprogrammering, till skillnad från SK-Dirigenten (Single-Körning), under vilken endast ett användarjobb i taget kan bearbetas.

MK-Dirigenten arbetar, liksom GECOS III, med variabla primärminnesregioner för bearbetande jobb. Varje jobbs minnesutrymme är sammanhängande, jobb kan emellertid dynamiskt flyttas i minnet. Inklusive terminalövervakning kräver MK-Dirigenten cirka 25 K 24-bits ords primärminne. Nedanstående information om jobbövervakningsfunktionen i MK-Dirigenten är hämtad ur "Presentation av D22/D220 Operativsystem", utgåva 3, A6373.05, ZN-69:99, 690606.

1. <u>Inläsning av jobb</u>. Inläsningsdelen av JÖ (jobbövervakaren) ansvarar för inläsning av jobb till en <u>inläst kö</u> på skivminne. Denna kö kan rymma upp till cirka 20 jobb. Från inläsningskön läses jobben in till den <u>aktuella kön</u>, som kan rymma högst 10 jobb. Överföringen från inläst kö till aktuell kö sker i <u>startprioritetsordning</u>, vilket är detsamma som den ordning i vilken jobben lästs in.

2. <u>Resursundersökning</u>. Initiering av ett jobbsteg föregås alltid direkt av resursundersökning. Jobbstegets behov av primärminne och yttre enheter fastställs därvid. Detta kan inte göras för hela jobbet (samtliga jobbsteg) på en gång, eftersom ett jobbsteg kan styra efterföljande jobbstegs resursbehov.

3. <u>Initiering av jobbsteg</u>. Initiering av jobbsteg från olika jobb i <u>aktuella kön</u> sker med hänsyn till startprioritet och resursbehov, och kan belysas med ett enkelt exempel:

Antag att 3 jobb befinner sig i aktuella kön, A, B och C. Startprioritetsordningen är A - B - C.

- Jobbet A består av jobbstegen A1, A2, A3 och A4
- Jobbet B består av jobbstegen B1, B2 och B3
- Jobbet C består av jobbstegen C1, C2 och C3

Antag att jobbstegen A1 och B1 färdigbearbetats, samt att vid den aktuella tidpunkten jobbstegen A2 och C1 är under bearbetning. Jobbsteget B2 har ej kunnat startas på grund av bristande resurser. Då A2 avslutats försöker JÖ starta A3 eftersom jobb A har högsta startprioritet. Räcker inte resurserna, försöker JÖ med B2. Om även B2 kräver mer än som finns tillgängligt, får C1 fortsätta ensamt. C2 tillåts dock ej startas förrän C1 har avslutats p g a sannolik koppling dem emellan.

Två möjligheter finns att påverka bearbetningsordningen, som annars alltså sker efter startprioritet. Den ena är användning av en speciell styrsats i styrkoden (kallad DIVIDER), och den andra är att ett jobb görs till URGENT-jobb, och därmed går före alla andra i kön. Användning av DIVIDER avses hindra att jobb med mycket stora resursbehov fastnar i jobbkön, och aldrig startas upp. Denna styrsats, som kan inplaceras manuellt i jobbstyrkoden i samband med ett jobbsteg med stort resursbehov, eller tillfogas från konsolen (av operatören) om jobbet redan lästs in, medför att nya jobbsteg ej initieras förrän DIVIDER-jobbsteget fått sina resursbehov tillfredsställda och startats. DIVIDER får användas med sparsamhet, för att effektivt systemutnyttjande skall uppnås.

Den andra möjligheten att påverka bearbetningsordningen är genom att använda benämningen URGENT för ett jobb. Detta kan endast utföras av operatören, som via konsolen meddelar JÖ att ett visst jobb är URGENT. Därmed går detta före alla andra jobb i kön. Endast ett av de inlästa jobben får vara URGENT.

4. Skydd mot jobblåsning. Jobblåsning kan inträffa i multiprogrammeringssammanhang som följd av att två jobbsteg "väntar på varandra". Detta har samband med filer som sparas från ett jobbsteg till ett efterföljande jobbsteg i samma jobb. Två simultana jobb med sparade filer kan låsa varandra så att inget av dem kan fullföljas. Före initiering av varje jobbsteg genomlöper därför JÖ en jobblåsningsalgoritm, som fastställer om initiering av det aktuella jobbsteget kan ge upphov till jobblåsning, och vidtager i aktuella fall åtgärder därmot. Låsningsproblematiken har berörts i kapitel 3.

5. Resurstilldelning. Primärminnestilldelningen tillgår på det viset att ett jobbsteg tilldelas minne i moduler om 512 ord (24 bits). Tilldelat område är sammanhängande, och för ett jobbsteg "främmande" områden i minnet

bevakas med hjälp av maskinvarumässigt minnesskydd. Dynamisk relokering i minnet tillämpas, i syfte att på bästa sätt utnyttja detsamma. Till exempel, antag att vid en viss tidpunkt tre jobbsteg P1, P2 och P3 befinner sig i minnet, fas 1 i fig 5. 4:7.

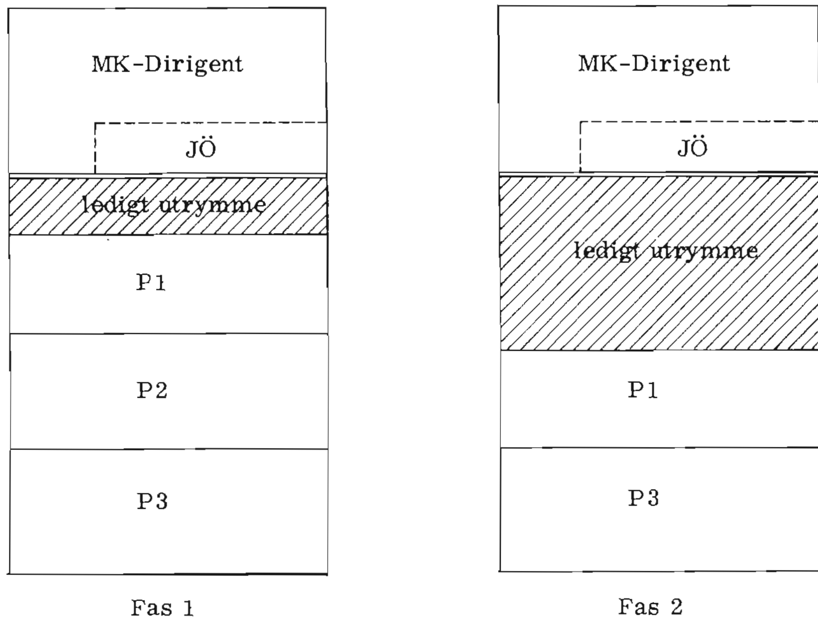


Fig 5.4:7.
Dynamisk primärminnesrelokering för DataSAAB D22 MK-Dirigent.

Antag vidare att P2 är det jobbsteg som först i tiden avslutas (på normalt eller onormalt sätt). Därvid ledigförklaras det utrymme P2 upptagit under fas 1. För att bereda maximal plats i minnet för väntande jobbsteg flyttas därefter P1 (det relokeras dynamiskt) så nära P3 som möjligt, så att hela utrymmet mellan P1 (i dess nya läge) och MK-Dirigenten kan disponeras för nya jobbsteg (fas 2 i fig).

Tilldelning av yttre enheter till ett jobbsteg utförs på basis av det behov som angivits i jobbstegets styrkod. JÖ medger dock vissa modifikationer därav, i och med att operatören har möjlighet utbyta vissa långsamma yttre enheter mot snabbare (såvitt detta inte strider mot programmets avsikter). Typiska byten av denna typ är från kortläsare till skivminne, från radskrivare till skivminne osv. Detta kan utföras i syfte att öka systemets totala bearbetningseffektivitet vid multiprogrammering. Uppen-

bart är att för att dylikt byte skall kunna äga rum utan att påverka jobbstegens avsedda bearbetning, så måste mediaomvandlingar föregå resp efterfölja exekveringen. Om operatören bytt radskrivare mot skivminne för ett visst jobbsteg, måste efter jobbstegets avslutning en mediaomvandling från skivminne till radskrivare äga rum. Denna körs då som ett separat jobbsteg.

6. Kommunikation mellan system och operatör. De typer av meddelanden mellan systemet och operatören som utväxlas är av typ som i huvudsak liknar de båda tidigare beskrivna operativsystemen, och berörs därför här ej närmare.

7. Resultatutmatning. Aktuell jobbutmatning uppsamlas i systemutrymme på skivminne, för att senare konverteras till,enligt styrkoden för jobbet, avsedd yttre enhet. Användaren har emellertid möjlighet att med hjälp av speciella styrsatser dirigera utmatningsfiler, avsedda för olika yttre enheter, till samma yttre medium. DataSAAB kallar detta för generering av en mosaikfil. Namnet härrör från det faktum att de utmatade posterna placeras på den gemensamma yttre enhetern i den ordning de matas ut, och alltså kommer att ligga blandade med varandra. För varje fil som blandats måste sedan en mediaomvandling ske till ursprungligen avsedd yttre enhet. För att generering av en mosaikfil skall löna sig, bör den givetvis placeras på ett snabbt yttre medium, t ex band.

8. JÖ's avslutande uppgifter. Under genomlöpande av ett jobb skriver JÖ successivt ut en s k jobblogg på systemminnet (skivminne). Då jobbet avslutats matas jobbloggen ut på radskrivare. Jobbloggen innehåller uppgifter om vilka jobbsteg som genomlöpts, filer som använts, antal poster som behandlats per fil m m. JÖ lagrar även upp ett debiteringsunderlag på systemminnet. Detta skrivs emellertid inte ut omedelbart, utan fylls på successivt allteftersom andra jobb färdigbearbetats. Då det för debiteringsinformation reserverade utrymmet på systemminnet tenderar att bli fyllt, sänds en uppmaning till operatören att överföra debiteringsunderlaget till magnetband. Från detta band kan sedan med jämna tidsmellanrum (dag, vecka) med hjälp av standardprogram eller egna program, debiteringsinformation skrivas ut på radskrivare. Därvid kan t ex fakturor direkt produceras.

Ytterligare data som JÖ lagrar på systemminnet rör prestandastatistik för datorsystemet som helhet. Även denna information konverteras vid lämpliga tillfällen till radskrivare, och utgör underlag för studier och eventuella modifikationer av t ex systemparametrars inverkan på systemets prestanda.

Litteratur

Beträffande "allmän" operativsystemlitteratur, se litteraturreferenser i slutet av kapitel 4. Som underlag för detta kapitel ligger olika tillverkares handböcker avseende resp operativsystem. I den mån något system beskrivits har referens till motsvarande publikation infogats direkt i texten.

6. JOBBPROFILER OCH PLANERINGSPROBLEM

Ett datorsystem konfigureras normalt i hög grad med hänsyn tagen till den profil av jobb, som förväntas eller kan förutsägas i den aktuella miljön. Såväl antal och typer av in/utenheter och dessas prestanda som minnesstorlekar och processorkapacitet i en lämplig konfiguration är beroende på vilka typer av jobb som blir aktuella om man eftersträvar en låg bearbetningskostnad. Det låter sig emellertid inte göras att på förhand med säkerhet förutsäga de kommande jobbtyperna. Ett datorsystem skall ju vara i funktion under ett icke ringa antal år, avskrivningsmässigt ofta 5 eller 6, men i praktiken ej sällan längre. Under systemets livstid är det oundvikligt att det aktuella företaget undergår förändringar, som påverkar jobbtyperna som föreläggs datorn för bearbetning. Bland skäl för detta kan vi nämna att

- kunnandet avseende rationalisering ökar inom tillämpningsmiljön
- den tekniska och administrativa utvecklingen går framåt
- själva existensen av ett datorbaserat informationssystem föder alltid nya önskemål och behov. Företaget stimuleras att automatisera nya rutiner osv.

Vi noterar alltså att en osäkerhet före installationsdags alltid råder avseende de verkliga typer av jobb, som datorn skall bearbeta. Icke desto mindre kan vi i många fall urskilja grupperingar av jobbtyper som inom ett företag håller sig någorlunda konstanta under datorns livstid. Planeringen av datordriften eftersträvar normalt att hålla den aktuella jobbprofilen tämligen oföränderlig, i syfte att successivt anpassa datorsystemet till allt effektivare bearbetning av denna jobbprofil. Företagets utveckling, den ökade automatiseringen osv motverkar emellertid detta statiska tänkande genom att, som nämnts, kontinuerligt aktualisera tillförsel av nya jobbtyper. Sålunda kommer de för jobbinplanering inom datordriftsavdelningen ansvariga ständigt att arbeta med avvägningsproblem. Hur hårt skall användarna styras avseende jobbtyper (för att tillfredsställa datorsystemet), och hur intensivt skall datorsystemet anpassas till den kontinuerliga jobbprofiländringen (för att tillfredsställa användarna)?

6.1 Jobbprofiler

6.1.1 Allmänt

Vad är en "jobbprofil"? Frågan är ej enkel att besvara och ordet jobbprofil lär ej gå att hitta i något uppslagsverk avseende datatermer och deras definitioner. Man skulle kunna dels tala om ett jobbs "yttre egenskaper" dels om dess "inre egenskaper". De yttre egenskaperna kunde främst karaktärisera kontaktytan människa-dator medan de inre egenskaperna skulle avspegla jobbets beteende under exekvering och dess krav på olika slag av resurser. Det förefaller rimligt att dela in jobb efter deras yttre egenskaper i klasserna

(a) sats- eller kövis bearbetning

(b) reelltidsbearbetning

(c) tidsdelning

För var och en av klasserna kan sedan de yttre och inre egenskaperna anges. Vi gör nedan ett försök att ange några sådana enligt vår uppfattning utmärkande egenskaper (listan av egenskaper är ej angiven i någon speciell ordning)

(a) sats- och kövis bearbetning

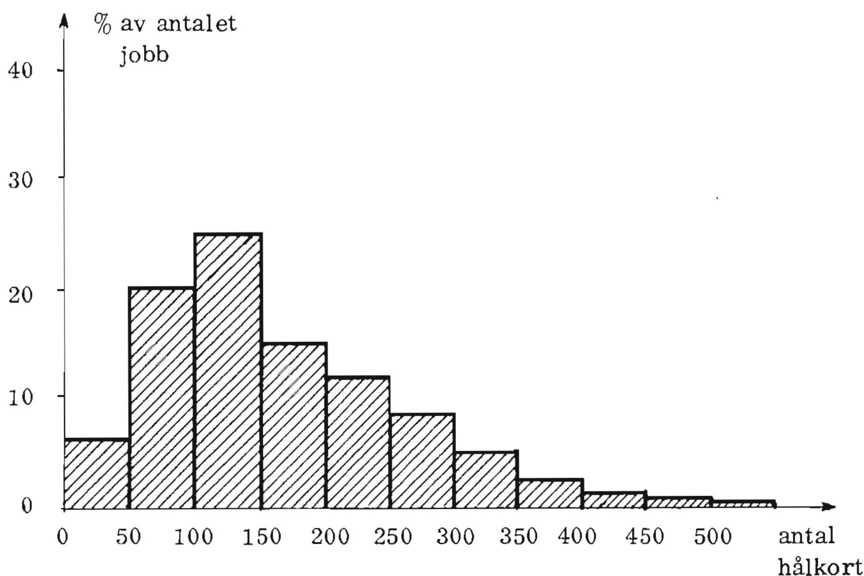
(a1) yttre egenskaper:
- användarkategori
- ankomsttid på dagen
- krav på omloppstid, prioritet o dyl
- produktionsjobb eller kompilering/test
- använt programmeringsspråk (krav på åtkomst till motsvarande kompilator)
- antal jobbsteg
- programstrukturen (overlay-strukturen t ex)
- krav på speciella systemprogram, minnes-"dumps" osv

(a2) egenskaper vid exekvering som belastar systemresurser (för varje jobbsteg)
- indatavolym och indatamedia (tecken, kort eller motsvarande)
- utdatavolym och utdatamedia (tecken, rader, sidor)
- behov av primärminnesutrymme, ev uppdelat i program och datautrymme
- behov av sekundärminnesutrymme (direktminne) eller speciella, dedikerade sekundärminnesenheter (skivor, trummor, magnetband, celler m m)

- transporterade massdatatecken (in/ut), dvs antal lästa/skrivna tecken (poster) på direktminnen eller andra sekundärminnen
- antal accesser till sekundärminnen
- behov av processortid (eller hellre: antal genomförda "maskininstruktioner", för att medge approximativ omräkning till annat datorsystem)
- fel i styrsatser
- syntax-fel i programmet (vid kompilering)
- fel i objektprogrammet (t ex onormal avslutning)

osv

Listan kan på intet sätt anses komplett eller allmänt tillämpbar men tjänar syftet att illustrera den mängd av faktorer som en jobbprofil är beroende av. För var och en av dessa faktorer kan nu för en stor mängd jobb ett frekvenssamband, efter omfattande faktainsamling, anges (t ex enligt figur 6.1:1).



Figur 6.1:1
Exempel på histogram för indatavolymen för jobb (uttryckt i antal hålkort). Klassbredd: 50 hålkort.

(b) reelltidsbearbetning[1)]

(b1) yttre egenskaper

- terminalsystemets struktur (antal och typer av terminaler)
- människa-datorinteraktionens struktur (olika typer av transaktioner och deras sannolika förlopp)
- transaktionsfrekvensen under olika tider på dygnet, toppbelastningar under året osv

(b2) egenskaper vid exekvering som belastar systemresurser

- indatavolym för en interaktion (fördelningssamband)
- utdatavolym för en interaktion (")
- en interaktions behov av olika applikationsprogram och deras inläsning till primärminnet (delvis en systemkonstruktionsfråga)
- behov av processortid
- behov av accesser till olika sekundärminneslagrade filer
- behov av primär- och sekundärminnesutrymme för program och datalagring under bearbetning av en transaktion (bestående normalt av flera interaktioner människa-dator) osv

(c) tidsdelning

I många avseenden har ett tidsdelningssystems jobbprofil (trots att man knappast kan tala om jobb här på samma sätt som för sats/kövis bearbetning) karaktäristiska egenskaper liknande ett reelltidssystems. Även här möter vi människa-datorinteraktion via terminaler av olika slag. Typiska yttre egenskaper för tidsdelningssystem skulle kunna vara

(c 1) yttre egenskaper

- total tid vid terminal för användare (per gång)
- "tänketidsfördelningen" (vid terminal)
- använt programmeringsspråk
- använda biblioteksrutiner
- typ av jobb (procentuell fördelning mellan programuppbyggnad, test, prod. körning)

m m

(c2) egenskaper vid exekvering

- behov av processortid totalt och per interaktion (fördelning)
- indata - utdata - och massdatatransportstorlekar
- accesser till sekundärminne
- programstorlek (t ex antal "pages")

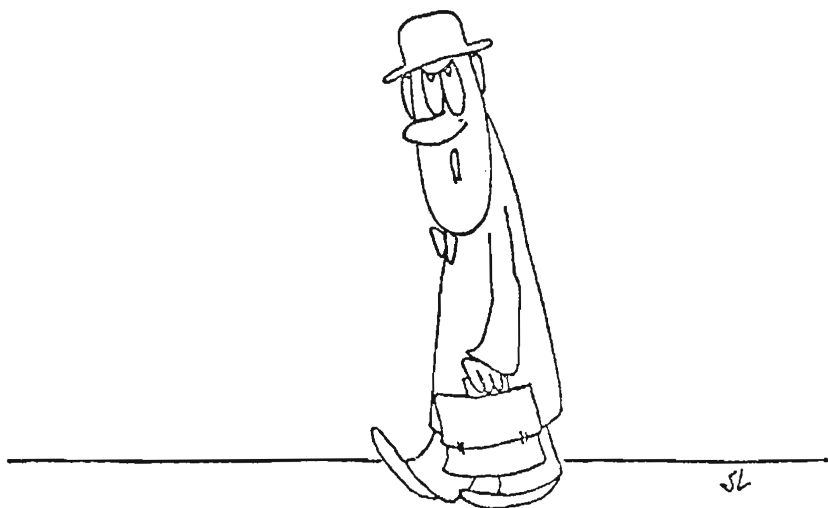
---

1) Som vi redan diskuterat i kap 4 och 5 kan ett "jobb-struktur" - resonemang i samband med R-T bearbetning knappast tillämpas.

- primärminnesbehov (om tillämpligt)
- behov av sekundärminnesutrymme för program, data och temporära arbetsfiler
- behov av processortid och direktminnesåtkomst vid produktionskörningar

m m

Att diskutera jobbprofiler med hänsyn tagen till samtliga ovan angivna faktorer är en svår uppgift inte minst beroende på att "typiska data" saknas. Vi skall dock ge några verklighetsanknutna exempel längre fram. Inledningsvis diskuteras två till overklighet förenklade profiler genom att endast fyra faktorer medtagits.



6.1.2 Starkt förenklade jobbprofiler

För att få överblick medtar vi först endast följande faktorer:

1. behov av processortid

2. behov av primärminnesutrymme

3. behov av I/U-enheter

4. behov av kort omloppstid

För att på enklaste sätt få problemet gripbart kan vi nu anta en blott tvågradig skala som variationsområde för de ovan angivna fyra faktorerna. För var och en av de fyra faktorerna kan vi då tänka oss "nivåerna"

a) stort behov

b) ringa behov

Om behovet av var och en av faktorerna kan anta två värden (stort resp ringa) finner vi totala antalet jobbtyper vara 2^4, alltså 16. Vi har då att betrakta 16 olika jobbtyper, varav de första är

- stort behov av processortid
- stort behov av primärminnesutrymme
- stort behov av I/U-enheter
- stort behov av kort omloppstid

- stort behov av processortid
- stort behov av primärminnesutrymme
- stort behov av I/U-enheter
- ringa behov av kort omloppstid

- stort behov av processortid
- stort behov av primärminnesutrymme
- ringa behov av I/U-enheter
- stort behov av kort omloppstid

osv

Situationen i praktiken är emellertid aldrig sådan att alla jobb inom en användarmiljö är av samma typ. Blandningar förekommer alltid, så att t ex en viss jobbsammansättning kan utmärkas av att i medeltal:

5 % av jobben är av jobbtyp 3
10 % av jobben är av jobbtyp 7
30 % av jobben är av jobbtyp 9
55 % av jobben är av jobbtyp 16

I de flesta situationer känner vi inte exakt denna fördelnings utseende. Den ändras dessutom oftast med tiden datorsystemet varit installerat.

Till och med detta starkt förenklade betraktelsesätt visar att antalet tänkbara profiler och sammansättningar är stort.

Vi har nu möjlighet till intuitiv förståelse för problemet att "konfigurera" ett datorsystem. Man söker ju att anpassa varje konfiguration till den aktuella "miljöns jobbprofil". När man i praktiken diskuterar jobbprofiler begränsar man sig vanligen till vissa schablonprofiler, som kan anses typiska för förekommande miljöer. Figur 6.1:2 antyder utseendet av ett par dylika profiler baserat på ovan redovisade betraktelsesätt.

6.1.3 Några karaktäristiska för jobb av beräkningstyp

I detta avsnitt kommer vi att redogöra för några karaktäristiska egenskaper för jobbsammansättningar som förekommer vid universitetsdatacentraler, forskningsinstitutioners datacentraler och andra datacentraler vars huvudsakliga inriktning är tekniska-vetenskapliga beräkningar. Driftsprincipen här antas kö-vis eller satsvis bearbetning. Fler och fler av datacentralerna, vars jobbsammansättning är av ovan nämnda inriktnint, kan även erbjuda time-sharing service. Time-sharing betraktas dock separat i avsnitt 6.1.5.

Inledningsvis, i avsnitt 6.1, diskuterade vi några yttre och inre egenskaper hos jobb. Endast för ett fåtal av dessa är statistiska sammanställningar publicerade eller på annat sätt redovisade.

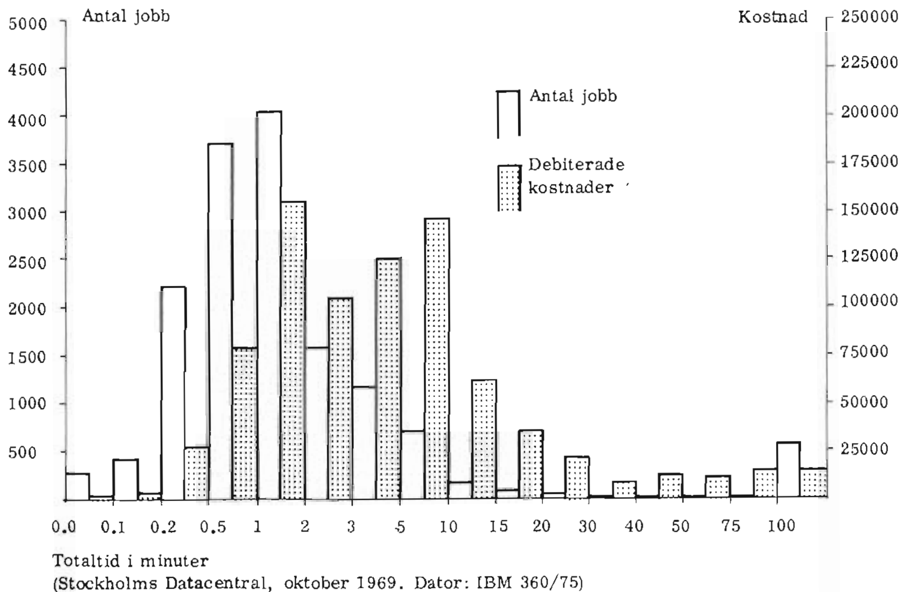
Tidsbehov

Figur 6.1:3 och 6.1:4 visar jobb- och kostnadsfördelningen hos Stockholms Datacentral (universitets- och forskningsmiljö) för oktober 1969. Figuren 6.1:3 betraktar jobbens totaltider (tiden från det att jobbets första jobbsteg initieras till dess att dess sista jobbsteg avslutas) och figur 6.1:4 betraktar jobbens processortidsutnyttjande. Normalt debiteras kunden även för hela eller del av den processortid som förbrukas av operativsystemet i samband med administration av jobbet i fråga.

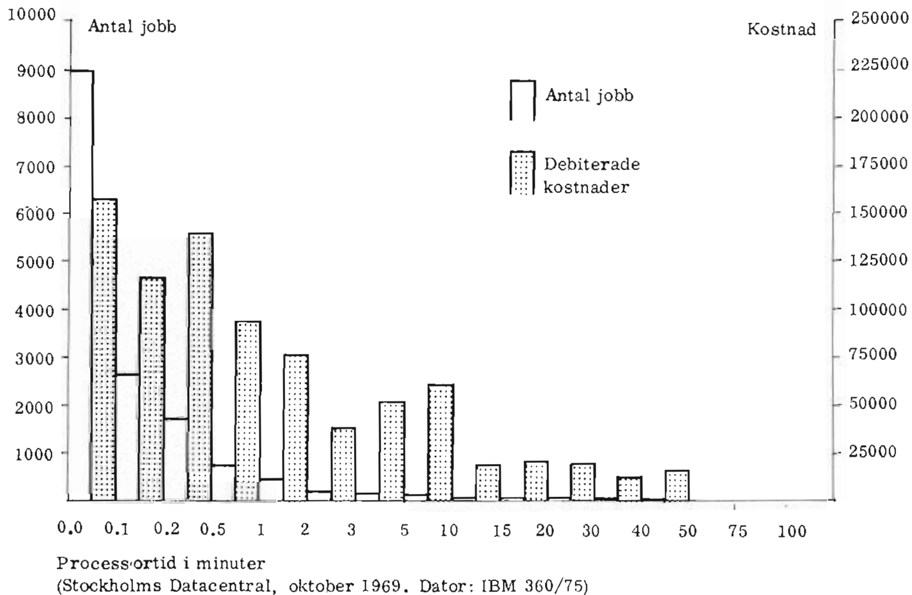
S = Stort, R = Ringa } Behov

	Jobbtyper																Faktorer
	S	S	S	S	S	S	S	S	R	R	R	R	R	R	R	R	Behov av processortid
	S	S	S	S	R	R	R	R	S	S	S	S	R	R	R	R	Behov av primärminne
	S	S	R	R	S	S	R	R	S	S	R	R	S	S	R	R	Behov av I/U-enheter
	S	R	S	R	S	R	S	S	S	R	S	R	S	R	S	R	Krav på kort omloppstid
Administrativ Jobbsammansättning		5				20			5			15		15		40	
Tekniskt-vetenskaplig jobbsammansättning (även universitet)		2	10	15				5			13			5	50		

Figur 6.1:2
Tänkbara jobbsammansättningar (som en vägd "summa" av jobbtyper) för två tillämpningsområden. Siffrorna i tabellen anger procent.



Figur 6.1:3



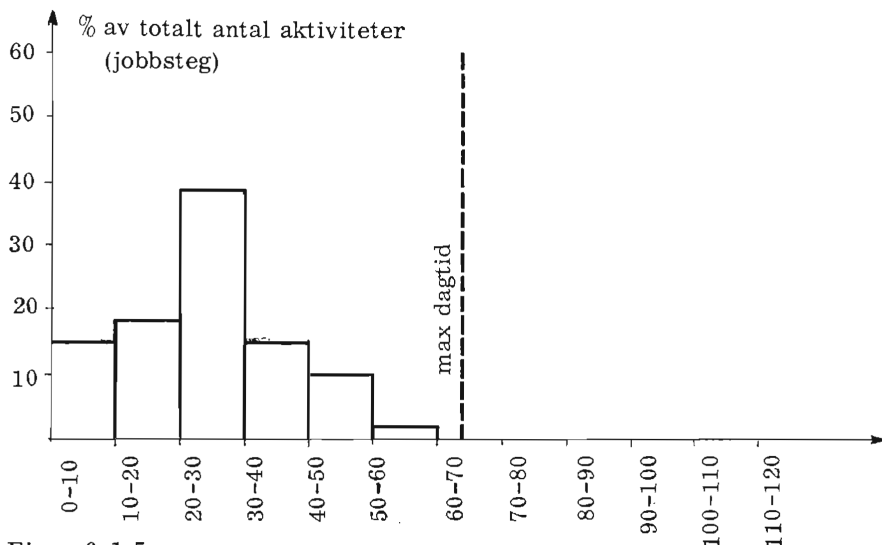
Figur 6.1:4

Figurerna indikerar att totaltiden för ett jobb är avsevärt längre än motsvarande processortid. Bidragande faktorer till detta förhållande är dels multiprogramkörningen, dels den för denna jobbsammansättning typiska, stora andelen mycket korta jobb. För dessa mycket korta jobb[1] åtgår en proportionellt stor tidsandel (ca 40-50 % är ej ovanligt) för administration i operativsystemet (lokalisering, inläsning av systemprogram, kompilatorer, datamängder, jobbstegsbyten osv). Observera också att de "kortare" jobben (totaltid $\leq$ 10 min) svarar för den största delen av datacentralens intäkter. En enkel analys av värdena i figuren ger dock det tillfredsställande resultatet att debiteringen är någotsånär rättvis. De korta jobbens kostnad per totaltidsenhet är ungefär på samma nivå som de längre jobbens. Därmed har dock inget sagts om "timtaxan" (för processortid) för datacentralen.

Det bör påpekas att fig 6.1:3 och 6.1:4 enbart anger två tidsegenskaper hos en jobbsammansättning som applicerats på ett bestämt datorsystem med ett bestämt operativsystem. Ändringar i datorkonfigurationen och/eller operativsystemet kommer givetvis att förändra bilden. Till exempel kan utökning av primärminnet och anskaffning av snabbare direktminnen tänkas minska totaltiden för korta jobb.

Primärminnesbehov

Figur 6.1:5 visar frekvenssamband för olika jobbstegs primärminnesbehov (hämtat från ett större svenskt företags tekniska beräkningscentral).



Figur 6.1:5
Frekvensdiagram för primärminnesbehov i K ord (1 ord = 36 bitar). Teknisk-vetenskaplig jobbsammansättning.

1) Totaltid mindre än ca 2 minuter.

I detta fall används ett operativsystem där tilldelat primärminne kan variera från jobbsteg till jobbsteg. Jobbsteg vars behov överstiger 64 Kord är här ej tillåtna under dagtid. Även här bör påpekas att figurens data är helt bundna till ett visst dator/operativsystem (GE 625-GECOS III) och användarmiljö.

Användarkategorier

I en undersökning (1) vid University of Michigan redovisas olika karaktäristika för jobb bl a beroende på användarkategori. Följande indelning av användare tillämpades

<table>
<tr><td colspan="7">samtliga användare (S)</td></tr>
<tr><td colspan="3">programmeringskurser (PK)</td><td colspan="3">forskning (F)</td><td rowspan="2">data-
centralens
personal
(DC)</td></tr>
<tr><td>avance-
rade
kurser
(PK1)</td><td>grund-
kurser
(PK2)</td><td>andra
kurser[1]
(PK3)</td><td>institu-
tionsper-
sonal,
"diverse
forskning"
(F1)</td><td>dokto-
rander
(F2)</td><td>forsk-
nings-
projekt
(F3)</td></tr>
</table>

1) där datorn användes som ett hjälpmedel.

Ett jobbs avslutning definierades som "felaktigt" om någon av följande händelser inträffade under dess bearbetning

(a) minnesdump (efter onormal avslutning)

(b) in-utmatningsfel

(c) överskriden tids- eller utdatavolymgräns

(d) andra fel som orsakar icke normal avslutning

Jobbet definierades som "ej exekvering" om det innehöll fel som omöjliggjorde laddning eller om exekvering ej var begärd. Jobb som icke tillhörde någon av ovanstående kategorier definierades som "riktiga".

För varje användarkategori och varje jobbkategori insamlades bl a följande uppgifter

- antal använda biblioteksrutiner
- programlängd
- administrationstid hos op. systemet
- laddtid
- exekveringstid (avs. användarprogram)
- totaltid för jobbet
- utmatning (utmatningsvolym: kort och/eller rader) under exekvering
- utmatningsvolym för "dumpar"
- utmatningsvolym under översättning (kompilering)
- total utmatningsvolym
- total inmatningsvolym

m m

För varje användare och jobbkategori redovisas medelvärde, standardavvikelse samt 10-, 50- och 90-percentilerna avseende ovannämnda faktorer. Av dessa resultat[1)] framgår bl a att de mindre sofistikerade användarna, i detta fall PK och F, ofta använde mer resurser vid felaktiga än vid riktiga jobb. Medelvärdet för exekveringstiden för felaktiga jobb var signifikant större än exekveringstiden för riktiga jobb för dessa två användarkategorier. Inom F-kategorien uppvisade dock även F1-gruppen dåliga värden - dvs längre exekveringstid för felaktiga jobb.

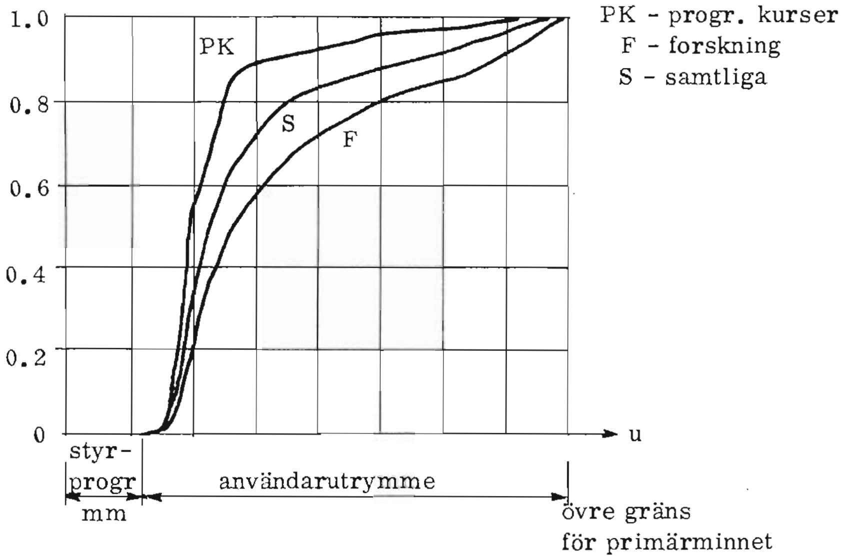
Figur 6.1:6 visar programlängdens ungefärliga fördelning för användarkategorierna PK, F och S (vi har ej redovisat kurvorna exakt enligt ref (1), eftersom resultaten är starkt maskinbundna).

Figur 6.1:7 visar motsvarande fördelningar för exekveringstiden. De små "sprången" vid 0,5, 1 och 2 minuter indikerar att dessa värden är populära som maximaltidsangivelser i styrsatserna.

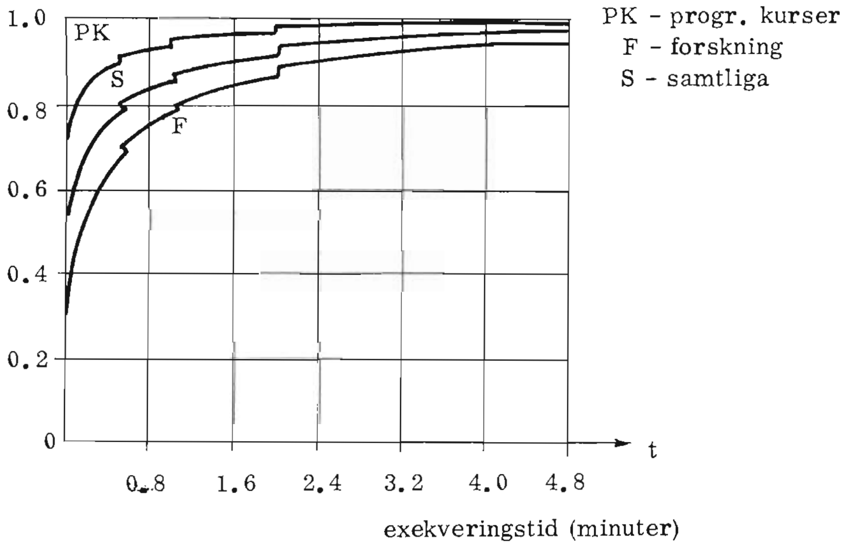
Undersökningen (1) har bl a också intresserat sig för att söka finna eventuellt samband mellan programlängd och totaltiden för bearbetning. Figur 6.1:8 indikerar att, för denna undersökning, en uttalad positiv korrelation mellan programlängd och totaltid tycks föreligga för program av längden 16 K eller mindre.

Samtidigt som författarna (1) varnar för några generella slutsatser av denna undersökning (det gäller ju här enbart ett bestämt dator- och operativsystem) är en av deras intressantare slutsatser att de flesta empiriska fördelningssambanden förefaller vara väl approximerbara med sammansatta exponentialfördelningar. Konstaterandet är av intresse för utveckling av kvantitativa, analytiska modeller för studier av dator- och operativsystem.

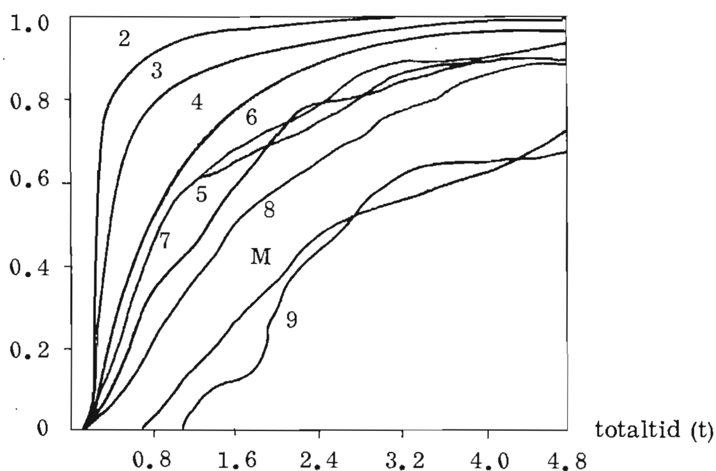
1) Läsaren hänvisas till ref (1)



Figur 6.1:6
Andelen jobb vars utrymmeskrav är mindre än eller lika med ett visst (u-) värde (källa: ref. (1)).



Figur 6.1:7
Andelen jobb vars exekveringstid är mindre än eller lika med ett visst tidsvärde (t) (källa: ref. (1)).



Figur 6.1:8

Andelen av jobb vars totaltid är mindre än eller lika med ett visst tidsvärde (t). Jobben är klassindelade 1-9,M efter deras primärminnesbehov (P), där K = 1000.

Klass	2:	0K < P ≤ 8K
"	3:	8K < P ≤ 12K
"	4:	12K < P ≤ 16K
"	5:	16K < P ≤ 20K
"	6:	20K < P ≤ 24K
"	7:	24K < P ≤ 28K
"	8:	28K < P ≤ 32K
"	9:	32K < P ≤ 32768
"	M:	program med överlagringsstruktur

(källa: ref (1))

6.1.4 Jobb av administrativ typ

Förutom många av de egenskaper som diskuterats för jobb av beräkningstyp karaktäriseras administrativa jobb i stor utsträckning av

- speciella behov av sekundärminnen och olika in-utmatningsenheter
- krav på vissa bestämda precedensrelationer mellan jobb (t ex ett jobb som producerar datamängden Q får ej bearbetas efter ett annat jobb som kräver Q som indatamängd)
- krav på körning av speciella jobb vid vissa bestämda tidpunkter (tid på dagen, viss veckodag, viss dag i månaden, osv)

In-utmatningstransporter och sekundärminnestransporter av data är normalt de faktorer som tämligen starkt präglar ett administrativt jobbs "profil". Någon allmänt accepterad uppsättning av faktorer varmed ett dylikt jobb skulle kunna beskrivas existerar oss veterligt ej.

Med hänsyn till krav på datorresurser skulle ett administrativt jobb delvis kunna beskrivas med de faktorer som vi diskuterat under (a2) i avsnitt 6.1.1. Släpper man kravet på att föreslå en generellt tillämpbar klassificering av administrativa jobb torde följande indelning i många fall kunna accepteras (tabell 6.1-1).

Tabell 6.1-1

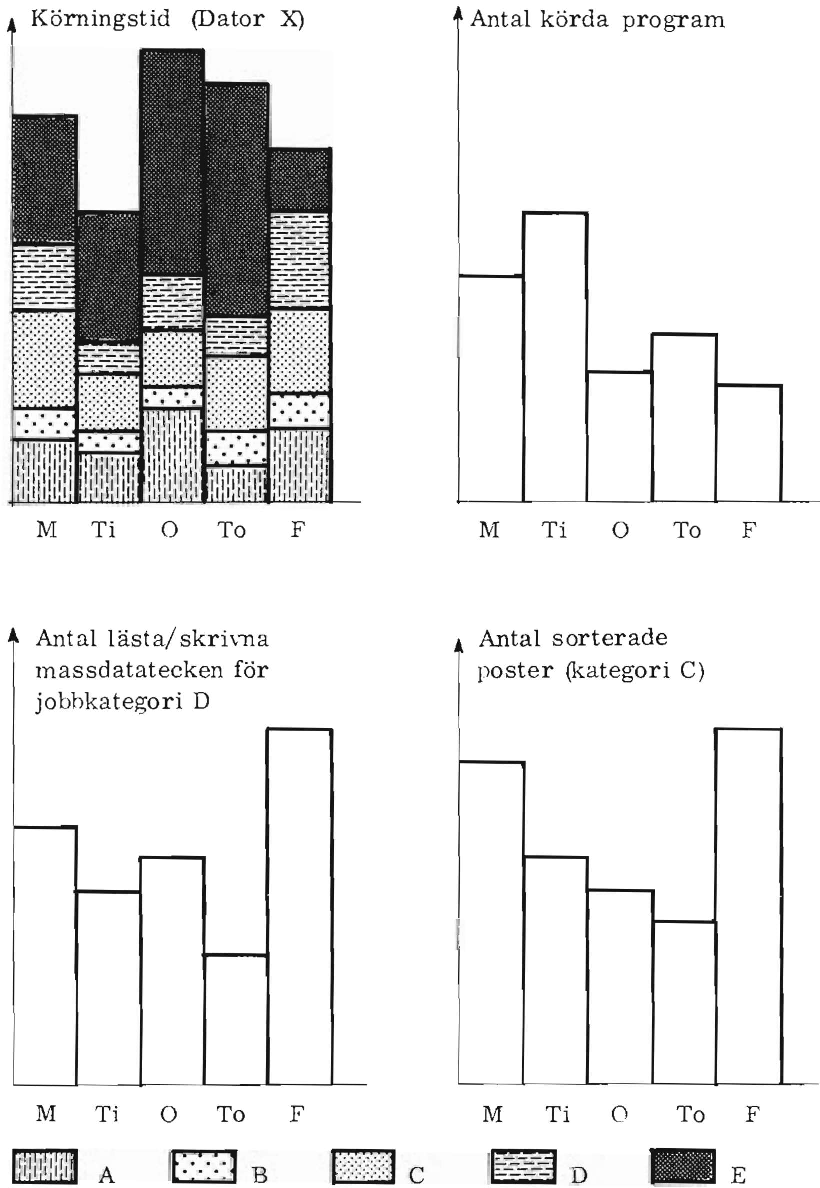
Jobb-kategori	Karaktäristisk beskrivning
A	Inläsning: mediaomvandling av indata från hålkort eller hålremsa till sekvensorganiserad fil på magnetband eller direktminne.
B	Inläsning och samtidig listning (eller kortstansning): mediaomvandling enligt A dock med viss kontroll av indata och utskrift (eller stansning) av felmeddelanden o dyl.
C	Sortering: sortering av sekvensorganiserade filer på direkt- eller bandminne. För sortering krävs även s k "arbetsfiler".
D	Bearbetning: sekundärminnesorienterad bearbetning där man ej tillämpar "långsamma" perifera enheter såsom kortläsare, radskrivare o dyl.
E	Listning: mediaomvandling från (normalt) sekvensorienterad fil på magnetband eller direktminne till radskrivare.

I praktiken förekommer ofta jobb vars "profil" är en blandning av några av ovanstående kategorier. Vi avstår dock från att här diskutera sådana ytterligare kategorier. Tabellen 6.1-2 skisserar typiskt utrustningsbehov och kvantitativa faktorer för approximativ tidsanalys av jobb i dessa kategorier. För tidsanalys av ett jobb krävs givetvis ytterligare data avseende datorsystemet. Studier i tidsanalys av processer är dock ej syftet med denna bok och därför betraktar vi ej problemet djupare här.

Tabell 6.1-2

Jobb-kategori	Karaktäristiskt utrustningsbehov	Typiska kvantitativa faktorer för approximativ beräkning[1] av tidsbehov för jobbet
A	Läsare (kort el. hålremsa), en bandstation eller utrymme på ett direktminne	Antal hålkort eller tecken på hålremsa, antal skrivna tecken på sekvensfil, blockningsfaktor
B	Som för A, plus en radskrivare eller kortstans	Som för A, plus antal skrivna rader eller stansade hålkort
C	Minst 3 sekvensfiler dvs tre bandstationer eller motsv. utrymme på direktminne(n)	Filens storlek (uttryckt i antal poster och postlängd), antal sekvensfiler, tilldelat primärminnesutrymme (använd sorteringsmetod påverkar givetvis tidsberäkningen)
D	Något karaktäristiskt behov kan ej anges. Behovet måste anges från fall till fall uttryckt i antal magnetbandsstationer, antal icke-sekvensfiler och deras sekundärminnesbehov	Om bearbetningen är starkt transportbegränsad, kan tiden approximativt beräknas på basis av antal transporterade massdatatecken och antal accesser till resp icke-sekvensfiler. Normalt är dock processortidsbehovet för denna jobbkategori så stort att hänsyn till det måste tas
E	Sekvensfil (magnetband eller utrymme på direktminne) samt en skrivare	Antal skrivna rader och lästa tecken från sekvensfil, blockningsfaktor

1) Övriga data som erfordras rör det aktuella datorsystemet.



Figur 6.1:9
Diverse karaktäristiska data för administrativa jobb vilka är bearbetningsbundna till viss veckodag.

Figur6.1:9 visar ett möjligt sätt[1] att grafiskt representera några karaktäristika för till veckodag bundna körningar. Uppgifterna avseende körningstid är givetvis bundna till aktuellt datorsystem (X). De flesta datorer som används för administrativ databehandling tillämpas multiprogramkörning i större eller mindre grad. Detta skapar problem när det gäller redovisning av ett jobbs tidsåtgång då denna varierar beroende på vilka andra jobb som bearbetas parallellt. Det förefaller lämpligt att vid tidsredovisning ange den tid som jobbet tar när det bearbetas ensamt i datorsystemet. Vid "extrapolering" till ett nytt dator/operativsystem kommer dettas möjligheter till multiprogramkörning, jobbens karaktäristika och andra förutsättningar för inplanering, av jobb sedan att avgöra total tidsåtgång för en mängd av jobb. Någon enkel formel att, på basis av de enskilda jobbens tidsbehov, beräkna totaltidsbehovet kan vi dock ej presentera.

Figur 6.1:10 visar slutligen exempel på ett möjligt sätt att åskådliggöra bearbetningsjobbs behov av två typer av sekundärminne. Alternativt hade man kunnat ange tidsbehovet för jobb av typen (i, j) (dvs bearbetningsjobb som kräver "i" direktminnesenheter och "j" bandstationer).

Antal direktminnesenheter \ Antal bandstationer	0	1	2	3	4	5	6	7
0	5	3	2	10	20			
1		5						
2			15	5				
3		10	5					
4		5						

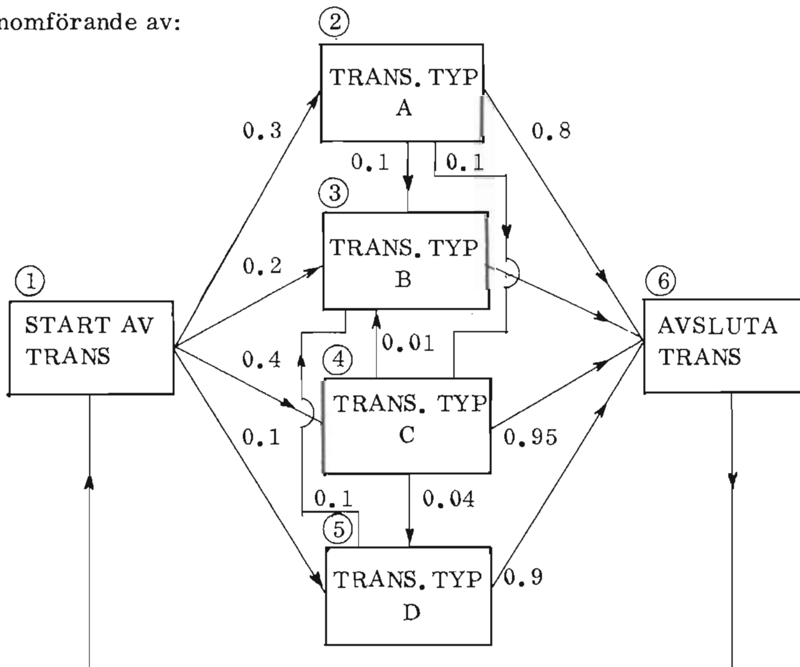
Figur 6.1:10
Exempel på bearbetningsjobbs behov av sekundärminnen av bestämd typ. Tabellen anger det procentuella antal jobb som behöver "i" direktminnesenheter (av viss typ) och "j" magnetbandsenheter.

1) Representationssättet hämtat från ett arbete utfört av B. Marions, Bonnierdata AB.

6.1.5 Reelltidsjobb och tidsdelning

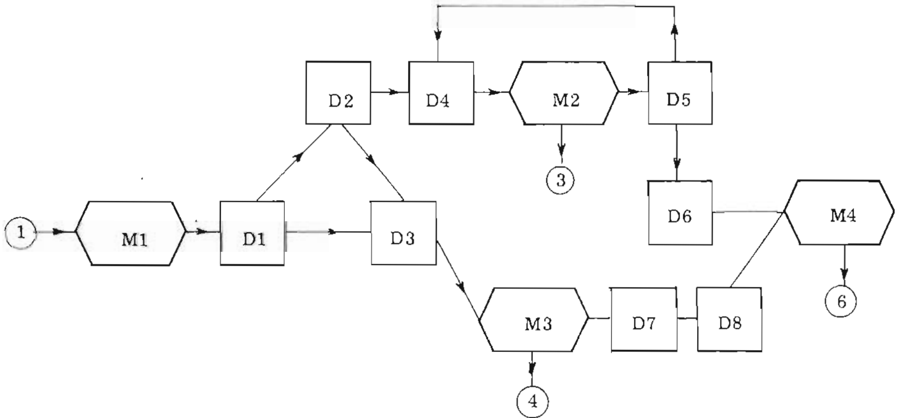
När det gäller karaktäristik av reelltidsjobb är litteraturen om möjligt ännu mer förtegen än avseende sats- och kövis bearbetning av administrativa jobb. Vissa faktorer har vi inledningsvis berört i avsnitt 6.1.1. Avseende tidsdelningssystem har studier utförts av ankomstintervalls fördelning för inmatning från terminaler vid bl a System Development Corporation (2).

Människa-datorsystemets interaktion vid reelltidssystem karaktäriseras av att olika slag av applikationsbundna transaktioner skall bearbetas. Sannolikheten för att en transaktion är av en viss typ och att bearbetning av en transaktion för en kund följs av en viss annan transaktion för samma kund kan åskådliggöras enligt ett hypotetiskt exempel i figur 6.1:11. Figuren kan också ses som en mängd av "tillstånd", 1 t o m 6, där pilarna anger sannolikheten för övergång från ett tillstånd till ett annat. Den approximation av realiteten som denna modell introducerar är att övergången till ett nytt tillstånd endast är beroende på det aktuella tillståndet. I många fall torde denna approximation dock ej vara alltför grov.



Figur 6.1:11
Exempel på ett transaktionsbearbetande system där transaktionsbearbetande tillstånd har angivits "i pilarna".

Varje tillstånd i 6.1:11 kan i sin tur brytas ner i ytterligare deltillstånd. Figuren 6.1:12 visar en tänkbar sådan nedbrytning, där M anger manuella faser (vars tidsutsträckning beror på terminaloperatören) och där D anger datoriserade faser (exekvering av olika delar i programsystemet). Ytterligare nedbrytning kan sedan göras.

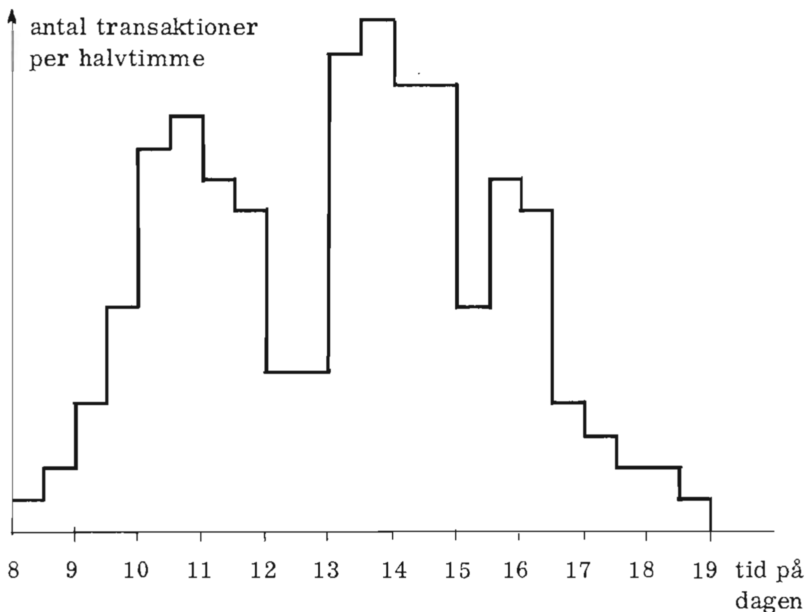


Figur 6.1:12
Exempel på ytterligare nedbrytning av tillståndet A enligt figur 6.1:11. M anger manuella "faser"[1] och D anger datoriserade processer.

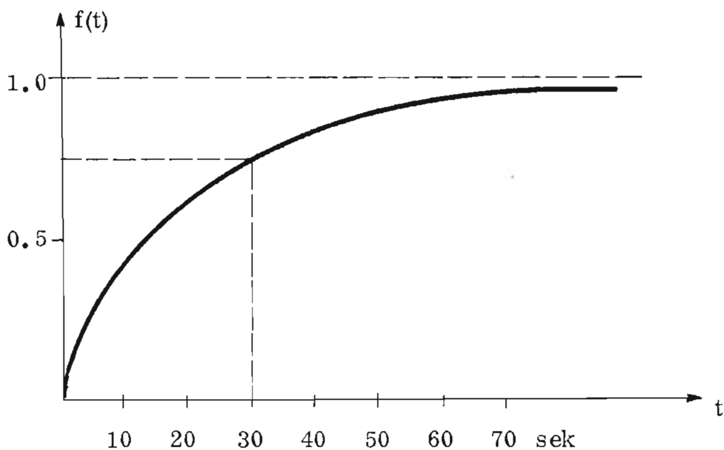
För varje datoriserad fas på tillräckligt detaljerad nivå kan dess krav på olika slag av resurser (program, primärminnesutrymme, sekundärminnesutrymme, antal accesser till olika filer osv) beräknas eller uppskattas. Om man känner ankomstfrekvensen av transaktionerna och tidsutsträckningen för de olika manuella faserna kan totalbelastningen på systemets resurser därmed beräknas.

Figur 6.1:13 visar exempel på transaktionsankomstfrekvensens variation under ett reelltidssystems dagliga "tjänstgöringsperiod". Normalt är ankomstintervalls fördelning och manuella interaktionsfasers tidsutsträckning (fig 6.1:12) svår att uppskatta innan systemet tagits i drift. Nya typer av terminaler och arbetsrutiner för operatörer gör dessutom gissningar vanskliga.

1) Eller manuella processer.



Figur 6.1:13
Exempel på variationen av ankomstfrekvensen av transaktioner till ett reelltidssystem.



Figur 6.1:14
Exempel på fördelningsfunktion för terminalanvändares tänketid. Figuren säger t ex att sannolikheten för att tänketiden är mindre än 30 sekunder är ca 0.75.

När det gäller tidsdelningssystem är situationen likartad. För ett speciellt fall (2) redovisas fördelningen av en terminalanvändares "tänketid" (tiden mellan två inmatningar från terminal). Man fann att längden av ett sådant intervall kunde approximativt anses som statistiskt oberoende av längden av tidigare intervall. Fördelningen f(x) av intervallängderna ansågs där väl approximerad med en s k "hyperexponentialfördelning", i detta fall den linjära kombinationen av två ordinära exponentialfördelningar enligt

$$f(t) = a\lambda_1 e^{-\lambda_1 t} + (1 - a)\lambda_2 e^{-\lambda_2 t}$$

med $a = 0.615$, $\lambda_1 = 0.03$ och $\lambda_2 = 0.148$ (se fig 6.1:14).

6.2 Jobbplaneringsproblem

När vi nu bekantat oss med olika typer av jobbprofiler och jobbsammansättningar har vi skaffat oss en bas för att inse att jobbplaneringsproblemen vid olika datacentraler i många fall skiljer sig från varandra. I en miljö där t ex in/utmatning väsentligen dominerar måste stor vikt läggas vid samspelet mellan operativsystem och operatörer vad gäller hantering av fysiska datamedia (kort, remsor, band, skivpackar osv). Planeringen av jobbströmmen måste då framför allt ta hänsyn till operatörernas förmåga att i tid hinna montera/demontera erforderliga datamedia. Dylik hänsyn kommer givetvis i blickpunkten i de flesta, för att inte säga alla, jobbmiljöer, men dess betydelse för planeringen är inte lika markant i en miljö som t ex huvudsakligen kännetecknas av talrika kompileringar plus tunga processorbundna tillämpningar där små krav ställs på montering av sekundärminnen eller iordningsställande av övrig in/utmatningsutrustning.

Jobbens karaktäristika, eller förväntade karaktäristika, har givetvis betydelse även för planeringen av en datacentral. Lokalmässigt kan en datacentral utformas både mer eller mindre lämplig för en viss jobbsammansättning. Datorsystemets yttre enheter bör t ex placeras så att gångavståndet för operatörerna blir minsta möjliga. Ett trumminne (ett icke utbytbart sekundärminne) kan således gärna placeras avsides, medan t ex kortläsare, radskrivare, bandstationer och skivminnen med utbytbara skivpackar, till vilka operatörerna beräknas behöva flitig åtkomst, bör få någorlunda central placering i lokalen. (Detta framgår bl a av många fotografier på datorsystem som visar konsol samt magnetbandstationer och ofta skivminnen (utbytbara skivpackar) centralt placerade.)

Vi har noterat att ur driftsledningens synvinkel är det ett önskemål att helst i förväg ha reda på typ av resursbehov för den inkommande jobbströmmen. Därmed kan datorsystemet "anpassas" för de inkommande jobben så att ett så gott systemutnyttjande som möjligt erhålles. Svårigheter förekommer emellertid att ombesörja denna inplanering.

Följande problem kan därvid noteras.

1. Svårigheter att någorlunda exakt, förutsäga jobbens resursbehov.
2. Multikörning (parallell bearbetning) av program gör bearbetningstider svåra att uppskatta.
3. Ankomsttiden för de flesta jobb är ej kända.
4. Programmen "spårar ur" eller också kan andra fel uppkomma som rycker sönder en planering.
5. Många jobb (vid administrativ jobbsammansättning) har precedensrelationer till andra jobb. Vissa jobb kräver "gemensamma resurser" (exklusiv tillgång) vilket kan innebära risk för "låsning". Dessa problem medför givetvis restriktioner och andra svårigheter vid planering.
6. Operatören är ingen "maskin" och dennes tid att utföra manuella moment varierar ofta starkt från fall till fall.

Resursbehoven

Även om jobbprofilerna och sammansättningen i stort är någotsånär känd på förhand för driftsledningen, kan givetvis inte enskilda jobbs resursbehov förutsägas tillräckligt exakt. I driftssystem, där kraven på kort omloppstid inte var lika uttalade som idag, var det, som vi tidigare noterat, vanligt att operatörerna insamlade inlämnade jobb med likartade resursbehov i <u>satser</u>, som, sedan de uppnått visst omfång, kördes i en följd. I t ex system som arbetar med fixa regionindelningar av primärminnet kunde då en regionindelning väljas som på lämpligt sätt motsvarade den just aktuella satsens behov av primärminne, eller en sådan sats köras först då lämplig regionindelning bestämts. På liknande sätt kunde systemets <u>yttre</u> enheter "anpassas" från sats till sats.

I andra system tillåter inte kraven på omloppstid dylik buntning av jobb. Jobben måste behandlas så snabbt som möjligt, exempelvis med hänsyn tagen till deras ankomsttid, prioritet eller "dead-line" när jobbet måste vara färdigt.

Driftsledningens möjligheter till anpassning av datorsystemet till den inkommande jobbströmmen begränsar sig därför oftast till åtgärder som är betingade av erfarenheter från tidigare bearbetningar av de aktuella jobben. Det är av detta skäl viktigt att följa upp datorsystemets utnyttjande genom att insamla körningsstatistik. För vissa system hänvisas härvid driftsledningen till studier av och kalkyler gjorda med hjälp av den logg som de flesta system erbjuder. Denna logginformation kan för större system bli mycket detaljerad och omfatta sådana data som

- projektbeteckning
- kontonummer
- prioritet
- datum
- tidpunkt för start av bearbetning
- uppskattad maxtid (processor)
- uppskattat antal rader (sidor) utmatning
- uppskattat antal kort (eller remstecken) utmatning
- verklig processortidsförbrukning
- verkligt antal rader (sidor) utmatning
- verkligt antal kort (remstecken) utmatning
- total primärminnesresidenstid
- total väntetid (på in/utmatning, systemservice m m)
- total systemprogramtid (ägnad åt aktuellt jobb)
- antal accesser till olika sekundärminnen
- utnyttjande av sekundärminnen av direktaccesstyp (t ex genom produkten (reserverat utrymme) x (tid))
- antal fel vid in/utmatning (maskinvarufel)
- utnyttjande av primärminne eller sekundärminnen av direktaccesstyp (t ex genom produkten (reserverat utrymme) x (tid))
- utnyttjande av övriga in/utmatningsenheter

m m

Viss begränsad del logginformation utmatas ibland automatiskt på systemets skrivmaskinskonsol eller radskrivare. Den mer omfattande körningsstatistiken kan uppsamlas i ett direktaccessutrymme som med jämna mellanrum kan "tömmas" på hålkort eller band. Dessa kan sedan periodiskt uppdatera konto- och projektfiler eller användas för framställning av statistik över resursutnyttjandet, uppföljning av felfrekvensen vid in/utmatning m m. Vissa system levererar med hjälp av speciella servicerutiner i operativsystemet automatiskt statistik i form av diagram och tabeller över behandlade jobb. Dylik information är alltså väsentlig för planeringen av datorsystemets utnyttjande. Ett exempel på den typ av statistik vi avser, finner vi i figur 6.1:3 och 6.1:4 som visar jobbfördelningen vid Stockholms Datacentral (datorsystem IBM 360/75) under oktober månad 1969. Många andra former av statistik är naturligtvis av värde att framtagas, exempelvis

- primärminnesutnyttjande
- utnyttjande av yttre enheter
- antal jobb behandlade simultant (i multiprogrammeringsmiljö) m m

En annan möjlighet av värde för planeringen av driften rör hanteringen av periodiskt återkommande jobb. Många datacentraler, kanske speciellt de med "administrativ" jobbprofil, har sig förelagda att med jämna tidsmellanrum bearbeta vissa givna program. Löner, redovisningsrutiner etc körs periodiskt, och driftsledningen kan härvid i förväg inplanera datorsystemet för de aktuella bearbetningarna (se även avsnitt 6.1.4). Dylika periodiska jobb är således populära hos drifsledningen i så måtto att tidpunkten för deras körning kan förutsägas; dock leder jobbens vanligen mycket höga tillförlitlighets- och säkerhetskrav icke sällan till driftsproblem. En lönerutin måste vara behandlad i tid för utsädande av lönebesked till de anställda. Personalen har oftast synnerligen liten förståelse för driftsstörningar o d i samband med försenade lönebesked. I reelltidssystem är dessa nödvändighetsproblem särskilt starkt markerade.

I många fall söker en datacentrals driftsledning planera för driften genom att indela dygnet i förbestämda tidsdelar då systemet utnyttjas på olika sätt. Den enklaste formen är här att konfigurera systemutnyttjandet olika vid olika skift.

Vanligt är dessutom att olika timmar på dygnet dimensionera systemet med prioritet för vissa typer av körningar. Ofta ges möjlighet till kortare omloppstider under dagtid än nattetid. Nätterna används då med fördel för tyngre jobb, med stora resurskrav men med lägre krav på omloppstid.

Vid system där inmatning av jobb sker från såväl avlägset belägna terminaler som inom datacentralen (det senare kallat "on site") kan inte jobb manuellt samlas i satser beroende på deras resurskrav, eftersom ingen manuell länk existerar mellan terminalerna och in-kön på sekundärminne. Någon sorts automatiserad buntning av jobb med speciella resurskrav måste då införas, om buntning alls skall förekomma.

Vid bl a ASEA's datorsystem General Electric GE-635 har detta realiserats, och t ex jobb med stort behov av processortid (1/2-flera timmar) placeras av systemet direkt in en väntekö som f n aktiveras först efter kl 18 på dagen, oberoende av vid vilken tidpunkt dessa jobb matats in till systemet.

Multikörning

Multikörningsteknik medför ofta planeringsproblem. Utväljningen av vilka

jobb som skall behandlas samtidigt i systemet kan antingen ske inom systemet (med hjälp av operativsystemsrutiner), enligt förbestämda algoritmer som bl a styrs av jobbens resursbehov eller också kan valet åläggas operatörerna. Det låter sig oftast inte göra att på förhand "manuellt" bestämma vilka arbetsuppgifter som skall behandlas samtidigt, av det skälet att driftsledning/operatörer inte löpande hinner hålla sig informerade om behandlingssituationen inom systemet, och agera som följd därav. Därför arbetar datorsystemen alltför snabbt.

Även om vissa operativsystem säges inkludera rutiner för löpande "optimering" av datorsystemets utnyttjande är det svårt, ofta ogörligt, för operatörerna att tillräckligt exakt kunna förutsäga tidpunkter då jobb färdigbearbetats. Man kan t ex i vissa system på anslutna "systembildskärmar" avläsa vilka jobb som är under bearbetning simultant, vilka som väntar i in-kön osv, men har relativt få möjligheter att påverka multiprogrammeringen på ett effektivitetsfrämjande sätt. Man kan från konsolen manuellt avsluta ("döda") jobb under bearbetning, men har i realiteten blott begränsade möjligheter att "knuffa in" nya jobb till behandling och små möjligheter att påverka inladdade jobbs resursbehov.

Som vi tidigare noterat leder användning av multiprogrammeringsteknik till principiella svårigheter att förutsäga ett jobbs exakta behandlingstid i systemet. Växlingen av processortid mellan program i primärminnet är på ett komplicerat sätt beroende av programmens enskilda utseende, och varierande administrationstider (overhead) noteras vid körningar av samma jobb i samma system beroende på vilka andra jobb som behandlas parallellt. Därför kan generellt varken exakta behandlingstider eller exakt debitering fastställas på förhand.

Även om variationen i bearbetningstiderna av dessa senare skäl ibland är måttlig, samverkar den med övriga planeringsproblem till att göra driftsledningens jobbplanering komplicerad.

Ankomsttider

Likaväl som resursbehov ej tillräckligt exakt kan greppas i förväg, är jobbs ankomstfrekvens oftast okänd för driftsledningen. Ankomstfrekvensen är huvudsakligen beroende av hur ofta de till jobben motsvarande problemen blir aktuella utanför företagets dataavdelning. Det är därför ett naturligt intresse för datordriftsledningen (främst vid "administrativa miljöer") att hålla en god kontakt med övrig verksamhet inom företaget.

Lösning av ett visst problem kan kräva en tillfällig omdisposition av datorresurserna av mer eller mindre omfattande slag. Härförutom kan nya pro-

blemtyper aktualisera tilläggsbeställningar till datorutrustningen tidigare än planerat. Även återlämnande av hyrda utrustningsdelar till leverantörer kan naturligtvis bli aktuella icke-planenligt, som en följd av ändrade jobbprofiler och jobbsammansättningar.

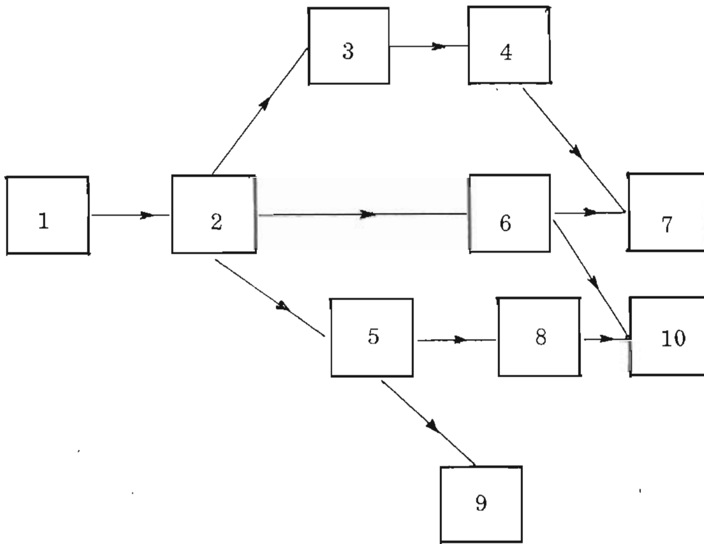
Programfel

Ytterligare en svårighet för jobbplanering uppkommer som en följd av ofullkomligheten hos i produktion varande program. Det är ett känt faktum att en stor del av programmeringsarbetet åtgår för att bygga in kontroller och andra slag av säkerhetsmekanismer i rutinerna. Programmering av den "centrala" algoritmen upptar vanligen endast en mycket måttlig del av programframställningsarbetet som helhet. Programinkörningen består till stora delar av undersökningar huruvida programmet kan anses "fool-proof" för olika tänkbara logiska kombinationer av data. Vad händer om t ex ordningen av misstag omkastas mellan vissa indatavärden, vad göra om en person i samband med folk- och bostadsräkning angivit att han/hon äger 10 personbilar, men ingen spis osv.

Erfarenhetsmässigt kan fastslås att mycket få program någonsin uppnår ett fullständigt säkert stadium, där urspårning vid exekvering p g a fel i data aldrig kan uppkomma. Bl a av detta skäl är en systematisk och komplett programdokumentation fundamental. Körningar av jobb kan givetvis även bli utsatta för irrationell påverkan från datorn. Här kan såväl icke fullt uttestade egenarter hos operativsystemet som rena maskinvarufel vara orsaker. Det faller utanför denna beskrivnings ambition att närmare gå in på dessa driftsstörningar, samt hur man söker avhjälpa dem. Vi nöjer oss med att konstatera att jobburspårningar är icke ovanliga företeelser, som medför icke ringa problem för driftsplaneringen. Uppläggningen av en "optimal" drifsplan måste ta hänsyn till omkörningar, betingade av driftsstörningar och oförutsedda programavbrott.

Precedensrelationer

Datorsystemets begränsade resurser gör att ej två eller flera jobb vilka som helst kan köras samtidigt. Vid administrativa system uppkommer ytterligare en restriktion vid planering - jobbens precedensrelationer. Ett jobb "i" säges vara en precedent till jobb "j" om "i" förutsättes exekverat och klart innan "j" kan påbörjas. Figur 6. 2:1 visar ett exempel på en tänkbar precedensstruktur för 10 jobb. Två jobb mellan vilka det råder en precedensrelation (av en viss ordning) kan ej exekvera parallellt. Av figuren framgår t ex att precedensrelation råder mellan jobb 2 och jobb 10 men ej mellan jobb 4 och jobb 8. Om inga andra restriktioner förekommer kan 4 och 8 exekvera parallellt.



Figur 6. 2:1
En tänkbar precedensstruktur för 10 jobb.

jobb nr

jobb nr	1	2	3	4	5	6	7	8	9	10
1										
2										
3					x	x		x	x	x
4					x	x		x	x	x
5						x	x			
6								x	x	
7	symmetri							x	x	x
8									x	
9										x
10										

Figur 6. 2:2
De med x märkta rutorna anger parvis multikörbara processer (dvs om man bortser från andra restriktioner för multikörning av två jobb såsom minnesbrist, sekundärminnesbrist, in/utmatningsenheters begränsning osv).

Av figuren framgår att en mängd kombinationer är ej tillåtna. Som tidigare nämnts är de olika jobbens bearbetningstider inbördes starkt vari-

erande, endast approximativt bestämbara och beror dessutom på vilka andra jobb som samtidigt bearbetas.

Problem i samband med jobbplanering är av det "icke triviala slaget" och har ej attackerats nämnvärt i den av oss genomsökta litteraturen. Även om publicerad statistik över datorutnyttjande saknas för administrativa tillämpningar torde kunna påstås att få system uppvisar processortidsutnyttjande över 50 %.

Operatörspersonalen

Vi har ovan betraktat en inkommande jobbström huvudsakligen "ur datorsystemets synvinkel". Det är nu naturligt att fråga sig hur jobbströmmens styrning beror på aktuell personal, dvs operatörer och programmerare.

Låt oss först betrakta operatörens position. Han/hon har som sin huvudsakliga uppgift att ansvara för att till datorsystemet inkommande jobb verkligen bearbetas i avsedd utsträckning. Dessa arbetsuppgifter innefattar satsvis och kövis arbetande system med lokal jobbinmatning:

1. tillhandahållande av kort/remsor etc vid inläsningsenheterna

2. framtagning och montering av för jobbens bearbetning erforderliga sekundärminnesenheter, iordningsställande[1] av radskrivare, kortstansar m m

3. tillvaratagande av resultat (på papper/kort/remsor/band/skivpackar etc) vid utmatningsenheterna

samt för satsvisa system med fjärrinmatning (avlägsna terminaler) eller för reelltids eller time-sharing system enbart punkten 2. av de tre nämnda. Det är normalt fallet att möjligheter finns för operatören att påverka jobbens bearbetning. Emedan han/hon (normalt) ej känner till de inmatade programmens interna beteende berör påverkansmöjligheterna enbart programmens externa uppträdande. Operatören har, åtminstone i mera rikligt utrustade och påkostade system, inte enbart möjlighet att inspektera in- resp ut-köernas löpande utseende, utan även från konsolen i viss utsträckning påverka dessa köer. Ett lågprioriterat jobb kan av speciella skäl behöva ges snabb genomloppstid, eller tvärtom. Operatören kan via styrorder från konsolen i vissa system ge instruktioner härom.

---

1) T ex kan förtryckta blanketter eller hålkort av viss typ erfordras för ett jobb.

Genom att i förväg planera effektiv montering av för jobben erforderliga sekundärminnesenheter kan givetvis den totala omloppstiden för jobbet påverkas.

Än väsentligare påverkansmöjligheter för operatörerna föreligger i de icke sällsynta fall när skilda operativsystemsversioner, eller helt olika operativsystem, används vid olika tider på dygnet. Även om dessa skilda systempass är väl schemalagda blir det ofta en avvägningsfråga för ansvarig operatörspersonal att avgöra när byte av system skall ske. Bearbetningen av alla de jobb som kräver ett visst system eller en viss systemversion blir här beroende på när operatörerna "bryter". Under sextiotalet har det vid många installationer verkligen varit aktuellt med körning under olika operativsystemsversioner, varför synpunkterna här ingalunda är av "teoretisk" natur. IBM 360-system som körts under operativsystemen MFT, MFT2 eller EMFT har ofta styrts med t ex varierande partitionsindelning av primärminnet vid olika tider på dygnet. Här har för minneskrävande jobb dessa problem varit speciellt aktuella. I en framtid, när väsentligt skilda dedikerade programvarusystem för samma dator sannolikt kan finnas tillgängliga kan dessa problem ytterligare komma att accentueras.

För system där in-kön genom kvarhållande finns tillgänglig "retroaktivt", kan omkörningar av jobb som av systemet behandlats styvmoderligt (partiella systemfel) initieras av operatörspersonalen. Härförutom kan vid urspårning hos produktionsjobb speciella minnesutskrifter o dyl manuellt initieras av operatör och bifogas de eventuella körningsresultaten. Dessa kan sedan bli föremål för ansvarig programmerares granskning. Jobb, som belägger datorsystemet i synbarligen icke avsedd utsträckning eller som på annat sätt förbrukar oförutsett mycket resurser, "dödas" antingen automatiskt eller av operatör (efter anmodan från operativsystemet).

Vi noterar att icke ringa möjligheter föreligger för operatörspersonalen att påverka jobbströmmens bearbetning. Även programmerarna har möjlighet därtill. En programmerare kan emellertid endast direkt påverka sitt eget jobb (även om andra jobb därmed kan indirekt påverkas). Vi har nämnt att denna direkta påverkan är avhängig de resurskrav som av programmeraren genom ett slags språk - styrspråket - anges för varje jobb eller jobbsteg (deljobb). Det blir nu aktuellt att närmare studera det principiella utseendet hos dessa styrinstruktioner i praktiken.

På grund av existensen av en mängd olika styrspråk (i stort sett ett för varje operativsystem) är vi tvungna att betrakta området relativt ytligt. Syftet är att ge en överblick över området, som sedan kan ligga till grund för läsarens egna fördjupade studier av vissa bestämda språk.

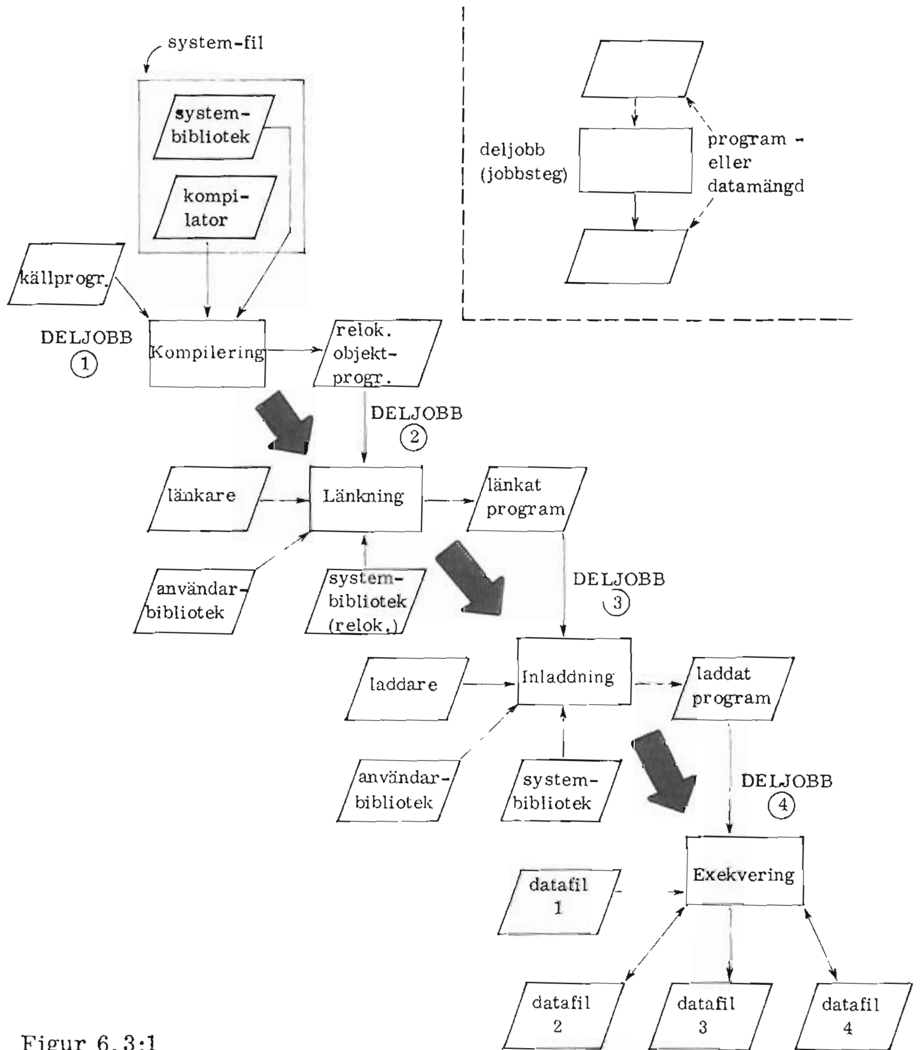
6.3 Styrspråk

De olika typer av order som vi ger datorn för att beskriva det sätt på vilket vi vill att vårt användarprogram skall behandlas ger vi i form av en följd av styrinstruktioner. Programmet är exempelvis skrivet i Algol, och vi vill att det skall kompileras och sedan exekveras. Vi måste då be operativsystemet om hjälp med att hämta fram Algol-kompilatorn (som kanske från början befinner sig på ett trum- eller skivminne) för inladdning till primärminnet innan kompileringen kan börja, och så småningom begära att länkning, laddning och initiering av det av kompilatorn genererade objektprogrammet skall göras. Dylika anvisningar (Algol-kompilering och därefter exekvering) ger vi maskinen i form av speciella instruktioner - s k "styrinstruktioner".

De för ett datorsystem tillgängliga styrinstruktionerna brukar sammanfattas under benämningen styrspråk (job control language, executive control language o dyl). Detta innebär ett "språk" på en annan nivå än de vanliga programmeringsspråken. Vi talar sålunda med styrspråkets hjälp om bl a vilket programmeringsspråk vårt program är skrivet i. Dessutom talar vi om på vilket lagringsmedium programmet befinner sig (på hålkort, på magnetband, på skivminnet, om det matas in från terminal etc). De data som skall bearbetas av programmet, måste också beskrivas och uppgifter såsom filnamn, blockstorlek, blockningsfaktor, fysisk lagringsenhet m m anges. Detta och åtskilligt annat sker för jobb och jobbsteg med inledande styrspråksinstruktioner.

Ett operativsystems jobbövervakare reagerar sålunda på en given mängd sådana instruktioner. Mellan olika operativsystem existerar emellertid tyvärr små likheter vad gäller styrspråkens syntax och semantik. Vi hade egentligen kunnat hoppas att standardiseringen vid 1970-talets början skulle ha kunnat driva fram viss enhetlighet i styrspråksutformning. Några principiella skäl av större tyngd häremot kan knappast sägas existera. Det är emellertid ett beklagligt men dock faktum att datorleverantörerna på denna punkt än så länge klart går skilda vägar. Detta får till följd att t ex ett byte av datorfabrikat ställer ofta avsevärda krav på användarna i form av arbete för inlärning av det nya operativsystemet och styrspråket.

Då antalet mera frekvent använda programmeringsspråk är begränsat kan det kanske synas som en överdrift att benämna "ett antal styrorder" för ett helt språk. Det är emellertid ett faktum att allt eftersom maskinerna under årens lopp blivit allt flexiblare och mera generellt användbara har antalet erforderliga styrinstruktioner för en maskin vuxit till en mängd som numera med rätta må rubriceras som ett eget språk. Komplikationsgraden hos dessa språk har också vuxit, så att det numera praktiskt taget enbart är specialisterna, som med rätta kan säga sig helt behärska ett styrspråks samtliga möjligheter.



Figur 6.3:1
Figuren visar att ett jobb består av exekvering av ett antal deljobb. Varje deljobb karaktäriseras av olika "behov" av bl a datamängder. Dessa kan vara:

- program att exekvera
- indatamängder
- utdatamängder
- permanent lagrade datamängder
- temporära (arbets-) datamängder

Vilka dessa är, var de residerar eller skall residera eller tilldelas utrymme är några av de uppgifter som måste anges via styrinstruktionerna.

Bearbetning av program på femtiotalets datorer kunde styras med ett fåtal styrinstruktioner. Detta kan sägas ha samband med den jämfört med nuläget begränsade flexibilitet som dessa maskiner ägde. En stor del av styrinstruktionerna gavs då med hjälp av manuella handgrepp (tryck på knappar, hantering av reglage etc), utförda av antingen en maskinoperatör eller av programmeraren själv. Dessa manuella handgrepp ersattes på sextiotalets datorer i ökande utsträckning av "programmerad" ordergivning, allt eftersom operativsystemen gavs vidgade administrativa och styrande uppgifter. När vi nu vid sjuttiotalets ingång är benägna att uttrycka viss kritik mot existerande styrspråks ofta alltför stora omfång, ofta inkonsekvent syntax och krav på mycken detaljinformation bör vi emellertid beakta att operativsystemen numera utför väsentligt mer omfattande arbete än förr. Större mängd kodifierad styrinformation krävs a priori för t ex datorsystem arbetande i "closed shop" än i "open shop". Att styrspråken successivt vuxit i omfång är alltså begripligt och i viss mån motiverat. Det är emellertid befogat att säga att de tillåtits växa något för "okontrollerat". Ett samarbete mellan datorleverantörerna när det gäller styrspråksutformning hade onekligen varit användarna till stor nytta.

Styrinstruktionerna skall ge anvisning för operativsystemet om hur aktuellt jobb skall behandlas inom datorsystemet. Det är därför naturligt att olika typer av dylika instruktionsmängder noteras lämpliga för olika driftsformer för datorsystem. Vi kan sålunda i viss utsträckning skilja mellan styrspråk för satsvis och kö-vis bearbetning och för t ex time-sharing. Flera principiella likheter existerar emellertid. Vi belyser dessa nedan. Två principer att tillhandahålla styrinstruktioner förekommer. Antingen tillhandahålls i det närmaste samtliga instruktioner för jobbet före bifogade program och data, eller också "blandas" instruktionerna med program och data, i den ordning som jobbstegen avses bli utförda. Intrycket av styrkoden som ett språk är förstärkt i fallet av den första av dessa principer. Den reella skillnaden är emellertid obetydlig mellan principerna, då styrkod och program/data ändå skiljs åt av operativsystemet. Några operativsystem tillåter att vissa styrinstruktioner "inbakas" i programmet. I detta fall har ett program möjlighet att kommunicera med en delmängd av operativsystemfunktionerna.

Kommunikation mellan användare och operativsystem sker med hjälp av instruktioner i form av styrsatser vilka normalt ingår i jobb-strömmen. Vanligt är att varje styrsats formellt utmärks av att den inleds med en eller två specialtecken, exempelvis $, /, ∇ eller *. Detta har valts för att unikt skilja styrsatser från exekverbara programinstruktioner eller data. Generellt kan sägas att en styrsats beskriver en användares krav på service (av något slag) från operativsystemet.

De för ett jobb angivna styrsatserna överlämnas till, och avkodas (tolkas internt) av en speciell operativsystemsrutin, styrkodsinterpretatorn som vanligen är en del av jobbövervakaren. Detta sker omedelbart efter inläsning av jobbet till in-kön. Vanligt är att de avkodade styrsatserna för de jobb, som befinner sig i in-kön, skiljs från själva kön fysiskt, och placeras i en speciell styrkodsfil, vanligen resident på ett sekundärminne av direktåtkomsttyp. Härigenom borgas för så litet tidsspill som möjligt uppkommer i samband med inspektion av prioritet för in-kön väntande jobb. Två huvudtyper av styrsatser kan urskiljas:

- aktiva
- passiva

En aktiv styrsats ger order om att en system- eller användarprocess skall utföras inom datorsystemet: ett program skall laddas, en datamängd skall förflyttas, exekvering av ett program skall initieras etc. En passiv styrsats är däremot deklarativ till sin natur och ger t ex information om läget av eller form för program eller datamängder.

Den inledande styrsatsen

Den för varje jobb inledande styrsatsen kan sägas vara passiv, och utgör identifikation av användare och jobb. För olika datorsystem inleds denna (efter styrkaraktären) t ex av ordet JOB eller ordet IDENT eller ordet RUN. Härefter anges, obligatoriskt, i denna sats vanligen unikt jobbnamn och kontonummer (för debitering) följt av viss frivilligt angivbar information.

Denna senare varierar givetvis beroende på vilket system som gäller och kan innefatta:

- prioritet för jobbet
- maximal processortid jobbet tillåts tilldelas
- maximalt minnesresursbehov för jobbet
- maximal resultatmängd jobbet kan leverera (antal tryckta radskrivarrader, sidor, stansade hålkort etc)
- önskan om viss undertryckning av från operativsystemet lämnade meddelanden rörande jobbets bearbetning
- villkor rörande jobbets avslutning (koppling mellan bearbetning av tillhörande jobbsteg)
- önskemål rörande eventuell senare återstart av jobbet (generering av checkpoint vid eventuellt onormalt jobbavslutande)
- önskemål om separering av jobbresultat i skilda klasser m m.

För samtliga ovan nämnda frivilliga parameterangivelser gäller att om någon eller några utelämnas i styrsatsen så antager systemet ett (vid systemgenereringen på förhand angivet) standardvärde på denna/dessa parametrar. Denna princip gäller vanligtvis för alla styrsatser.

För de fall den inledande styrsatsen tillhandahålls på hålkort (inmatning av jobbet via kortläsare) är det uppenbarligen möjligt att alla erforderliga parameterangivelser ej ryms på ett enda kort. Möjlighet finns emellertid i de flesta styrspråk att fortsätta med parameterangivelser, hörande till samma styrsats, på ett närmast följande kort. Control Datas operativsystem för 3000-serien, kallat MASTER, använder regeln att nära nog alla jobbparametrarna anges i en speciell SCHED-sats (schedulesats), sålunda separerad från JOB-satsen.

Initiering av exekvering

Den inledande styrsatsen anger sålunda vissa parametrar som gäller för hela jobbet. Varje jobb består, som tidigare nämnts, vanligen av flera delar (program), jobbsteg (deljobb eller aktiviteter). Specifikationer för det första av dessa steg följer efter den inledande JOB-satsen. Den bearbetning som skall utföras inom vart och ett av jobbstegen identifieras vanligen med ett jobbstegsnamn, följt av angivelse av vilket program jobbsteget sammanhör med. Dessa jobbstegssatser betraktas som aktiva styrsatser, då de ger information om att ett visst system- eller användarprogram skall initieras.

Utseendet av styrsatser som beskriver avsedd bearbetning inom jobbstegen varierar mellan olika leverantörer. Ett enhetligt beskrivningssätt har valts av IBM för 360-serien: varje jobbstegs bearbetning skall där definieras i en s k EXEC-sats. Denna inleds med en jobbstegsidentifikation, följd av ordet EXEC, följt av angivelser om vilket program som skall utföras samt under vilka betingelser detta skall ske. Parametrarna efter programidentifikationen kan här beröra:

- villkor rörande koppling mellan det aktuella jobbstegets utförande och andra jobbsteg
- tekniska specifikationer rörande programmet
- önskemål om speciell debitering för körkostnader för det aktuella jobbsteget
- önskemål rörande senare återstart av det aktuella jobbsteget
- speciell prioritet för jobbsteget
- processortids- samt primärminnesutrymmes-begränsning för jobbsteget

- information rörande jobbstegets möjlighet att kunna "rullas ut" eller "rulla ut" andra jobbsteg

m m

Med "program" avses här användarens program, språköversättare, servicerutiner, "procedurer" m a o exekverbar kod. Begreppet "procedur" avser här en benämning på en tidigare till systemet inmatad sekvens av styrsatser, som i sin tur beskriver ett eller flera jobbsteg. Det är sålunda fråga om en möjlighet att kunna förkorta styrinstruktionsgivandet, en värdefull egenskap, som även återfinns i vissa andra leverantörers system.

ICL 1900 GEORGE kallar sina motsvarigheter till IBM's "styrprocedur" för MACRO's, och medger även möjlighet att deklarera MACRO's med användarvalda styrparametrar.

Leverantör, datorsystem	Operativsystem	Styrorder	Parametrar anges
IBM 360	OS (MFT, MVT)	EXEC	inom EXEC-satsen
UNIVAC 1106/1108	EXEC-8	XQT	i huvudsak i inledande RUN-sats
General Electric 600-serie	GECOS	EXECUTE	i huvudsak i åtföljande LIMITS-sats
Control-Data 3000-serie	MASTER	TASK	i huvudsak i åtföljande parameter-satser
ICL 1900-serie	GEORGE	LOAD följt av ENTER	i åtföljande satser
Data SAAB D22	MK-DIRIGENT	DO	inom DO-satsen

Tabell 6.3-1
Terminologi för styrsats motsvarande "utför jobbsteg" för några olika system.

Flera leverantörer har i sina system valt att ge styrkoden en viss språkmässig karaktär (dvs motsvarande ett problemorienterat språk) vad gäller jobbstegsutförande. I användarvänlig riktning har här sannolikt ICL

1900 GEORGE gått längst och använder ett utpräglat "decentraliserat tänkande" för parameterangivelserna tillhörande jobbsteget. I vissa system skiljes mellan exekvering av ett användarens objektprogram och "exekvering" av en kompilator/assembler: de senare har givits speciella styrverb (av typen ALGOL etc).

Sekvenskontroll i styrkoden

Såvida ej annat anges utföres styrinstruktioner i den ordning de återfinns i inströmmen. Eventuellt logiskt samband (koppling) mellan utförande av olika jobbsteg i ett jobb (till exempel: jobbsteget PUTTE skall utföras "om och endast om" det föregående jobbsteget AMANDA kunnat utföras helt normalt (dvs utan felleller andra avbrott), o dyl) beskrivs, som ovan nämnts, antingen via parameterangivelser i EXEC-satsen (motsv) eller i separata styrsatser. Möjligheten till dylika angivelser varierar mellan olika operativsystem. Som exempel kan nämnas att ICL 1900 GEORGE bl a medger möjligheterna:

GO TO ⟨lägesangivelse⟩ (dvs ovillkorligt hopp)
AT ⟨händelse⟩ GO TO ⟨lägesangivelse⟩ (dvs villkorligt hopp)
AT ⟨händelse⟩ ENTER n (n = jobbstegsangivelse eller läge inom ett jobbsteg)
AT ⟨händelse⟩ RESUME (återstart)
AT ⟨händelse⟩ END

där "händelse" kan ha innebörden:

HALTED	n	(jobbsteget n avbrutet)
DELETED	n	(jobbsteget n överhoppat)
DISPLAY	m	(m = operatörsangivelse)
TIME	t	(t = uppnådd tid)
FAIL		(maskinvarufel av icke fundamental betydelse)

Flera av dessa möjligheter att villkorligt styra jobbets bearbetning återfinns även i andra system, dock är där givetvis syntaxen en helt annan.

Beskrivning och deklaration av datamängder

Vi har ovan beskrivit aktiva styrsatser rörande exekvering av jobbsteg. De datafiler som avses sammanhöra med jobbstegen definieras i en annan typ av styrsatser. Dessa satser kan betraktas som passiva. Den svaga men dock existerande likhet mellan de aktiva styrsatsernas utseende i olika operativsystem som vi ovan betraktat, existerar nära nog ej alls be-

träffande de databeskrivande styrsatserna. Konventionerna varierar kraftigt. Det låter sig därför inte göras att draga generella slutsatser om beskrivningsprinciper på denna punkt, utan vi tvingas betrakta några olika system var för sig. Utrymmet tillåter ej att fler än ett fåtal beskrivningar studeras.

IBM 360 OS (MFT, MVT)

Datamängder går här under benämningen "data sets". De beskrivs i styrsatser, kallade DD-satser (Date Definition Statements), av följande principiella utseende:

//⟨datadef-namn⟩ DD ⟨parametrar⟩

De två inledande snedstrecken anger enligt IBM-konvention att det är fråga om en styrsats. Datadefinitionsnamnet namnger styrsatsen, för eventuell referensmöjlighet från andra styrsatser inom jobbet. Parametrarna som anges efter karaktärerna DD kan bl a avse:

- datamängdens benämning
- information om fysisk perifer enhet där datamängden befinner sig
- information om datamängdens tillhörighet, ev sekretess, o dyl
- beskrivning av datamängden (organisationsprincip, postlängd, blockningsfaktor m m)
- information om etikett för datamängd på magnetband
- information om status för datamängden samt vad som skall göras med den efteråt, t ex om den skall katalogiseras
- datamängdens utrymme (på direktaccessminne)
- önskan om separat datakanal för överförande av datamängden (inom jobbsteget)

General Electric GE-600 GECOS

Detta system använder sig av ett flertal olika styrsatser för definition och användning av data. Vi omnämner nedan några av dessa:

$ DATA <parametrar>

där parametrarna bl a avser:

- filkod (2 karaktärers alfanumerisk programmerarvald filbeteckning)
- checksumma kontroll

Satsen $ DATA används för lagring av temporära filer. Andra satser för definition av datamängder är t ex:

$ DISC <parametrar>
$ DRUM <parametrar>
$ TAPE <parametrar>

där parametrarna avser:

- filkod
- hänvisning till logisk enhet
- angivelse om erforderligt utrymme (gäller för direktaccessminnen)
- angivelse om filen skall sparas eller ej
- angivelse om filen är länkad eller randomiserad
- filserienummer (serieminne)
- nummer på det band där bearbetningen skall börja (serieminne)
- filnamn, för operatörsmontering (serieminne)

Satserna ovan används normalt för icke-temporära filer.

ICL 1900 GEORGE

En mångfald olika satser för filbeskrivning existerar. Flera av dessa gör bruk av referens till lägesangivelser i styrkoden. Den passiva styrsatsen

⟨läge⟩ DATA ⟨perifer enhet⟩ (dokument-namn, ev serieminne-nummer),

refereras till via endera/båda av de aktiva satserna

IN ⟨läge⟩,
OUT ⟨läge⟩

och data läses/skrivs då IN/OUT-sats påträffas i styrkoden. IN/OUT-sats kräver angivelse av DATA-sats.

Univac 1108 EXEC-8

Bland en mängd olika möjligheter att specificera datafiler väljer vi för presentation satsen

▽ ASG ⟨indikatorer⟩ ⟨filnamn⟩/⟨nyckel 1⟩/⟨nyckel 2⟩,
⟨typ⟩/⟨reserv⟩/⟨l-modul⟩/⟨max⟩

som används för tilldelning av en direktminneslagrad (FASTRAND) datamängd till ett program eller för etablering och katalogisering av en ny datamängd.

⟨indikator⟩-fältet används för att katalogisera eller "avkatalogisera" en datamängd. Olika möjligheter och varianter finns härvid t ex

- datamängden katalogiseras endast om programmet avslutas normalt
- datamängden får status "endast läsning tillåten"
- datamängden får status av en allmänt tillgänglig fil
- datamängden får status "endast skrivning tillåten"
- programmet kräver exklusiv åtkomst till datamängden under dess bearbetning

osv

"nyckel"-fälten används för att vid katalogisering låsa filen för otillbörlig läsning resp skrivning. Dessa nycklar måste sedan anges för att få åtkomst till datamängden. ⟨typ⟩-fältet anger huruvida sekundärminnen av viss annan typ önskas "simulerade" på FASTRAND. ⟨reserv⟩-fältet anger hur många moduler av längden ⟨l-modul⟩ skall reserveras för datamängden i fråga. ⟨max⟩-fältet anger att körningen skall avbrytas om datamängden under exekvering av programmet överskrider ⟨max⟩ antal moduler.

Andra typer av styrinstruktioner

En generell princip, som kan användas inom nära nog samtliga system, utgörs av mediaoberoendet eller oberoendet av fysisk yttre enhet, vilket bl a diskuterats i ett tidigare avsnitt. Genom att i användarprogrammet referera till en logisk filbeteckning, samt genom att via fil-styrsats koppla denna logiska beteckning till vid körningen aktuell fysisk yttre enhet, medges ett programmeraroberoende av frågor rörande den fysiska datalagringen. Vid en senare körning av programmet kan, om så skulle önskas p g a tillfälliga testdata etc, andra data därmed associeras till programmet utan att filbeteckningar i detta behöver ändras.

Förutom här nämnda styrsatser existerar i styrspråken en flora av mera speciella satser. Utrymmet tillåter inte att vi går närmare in på dessas utseende, utan nämner blott att de bl a berör:

- flyttning av laddade objektprogram i primärminnet (relokering)
- ändring av önskat/tilldelat primärminnesutrymme

- kommunikation med operatör vid konsol
- begäran om minnesutskrifter
- ledigförklarande av filer under exekvering
- programlänkning (användarstyrd)
- avslutning av styrkoden m m
- angivelse om "stanskod" (IBMF, EBCDIC, ASC II etc) som program är tillhandahållna i.

Exempel på styrsekvenser

Som avrundning av dessa avsnitt om några allmänna drag i existerande styrspråks utseende väljer vi att exemplifiera hur styrkoden för en enkel bearbetning ser ut för några av de ovan nämnda existerande operativsystemen. Vi betraktar kompilering och efterföljande exekvering av ett Algolprogram, med både program och data tillhandahållna på hålkort. Detta jobb består alltså logiskt av två jobbsteg (eller aktiviteter): kompilering samt exekvering av det genererade objektprogrammet.[1] Inmatning avses ske på standard kortläsare och utmatning av resultat på standard radskrivare. Inga filer sparas efter körningen. Standardiserade systemparametrar avses användas. Varje styrsats tillhandahålls här på ett hålkort.

För General Electric GE-600 GECOS blir utseendet:

```
$       SNUMB       00001
$       IDENT       12345, TOJB
$       ALGOL
<Algol-kod>
$       EXECUTE
<Egna data>
$       ENDJOB
***  EOF
```

For Control Data 3000 MASTER:

```
$JOB, 12345, TOJB
$ALGOL (L,X)
<Algol-kod>
        FINIS
$NAMN, LGO
<Egna data>
End-of-file-kort
```

1) I vissa op. system krävs ett ytterligare steg mellan dessa två-länkning av det kompilerade objektprogrammet.

För ICL 1900 GEORGE:

```
JOB NAMN, 12345, TOJB
O  DATA  /0 (STEG 1)
1  DATA  /0 (EGNA DATA)
            IN 0
            ALGOL
            LOAD
            IN 1
            ENTER
            END JOB
⟨Algol-kod⟩
⟨Filslut-kort⟩
⟨Egna data⟩
⟨Filslut-kort⟩
```

För DataSAAB D22 MK-Dirigent:

```
JOB, TOJB, 12345
DO, ALGOLGENIUS, CORE, 5000;
TAPE, 1001, TM0, OPR, S;
DATA, ALGOLDATA;
DO, FILEMAN, IP, REG, OPR, SCAN, TM1, CSYS, TM0;
CSYS, 5, S;
TAPE, 1001, TM1, OPR, S;
DO, LINKER, IP, SEGMENT, KODNAMN, /, REWIND;
OSYS;
SYS, TM4, OPR;
SYS, 2, TM0, KODNAMN, P;
TAPE, 1001, TM4, OPR;
DO, KODNAMN;
DATA, KODNAMN;
EOC;
DATA, ALGOLDATA;
⟨Algol-kod⟩
⟨Filslut⟩
DATA, KODNAMN;
⟨Egna data⟩
⟨Filslut⟩
EOJ;
⟨Filslut⟩
```

(Möjlighet att bilda makroinstruktion av huvuddelen av ovanstående kod finnes.)

För UNIVAC 1106/1108 EXEC 8:

```
▽ RUN      12345, TOJB
▽ ALG
  ⟨Algol-kod⟩
▽ XQT
  ⟨Egna data⟩
▽ FIN
```

För IBM OS/360:

```
//NAMN     JOB      12345, TOJB
//         EXEC     ALGOFKLG
⟨Algol-kod⟩
//GO.SYSIN DD       *
⟨Egna data⟩
```

Angivelsen i EXEC-satsen, ALGOFKLG, har här betydelsen av ett "procedur"-anrop (Algol F, compile-link-go). Beteckningen står för ett större antal i sammanhanget frekvent förekommande styrsatser (i detta fall ca 25 st), som gemensamt tidigarelagrats i systembiblioteket under denna beteckning (se nedan).

```
MEMBER NAME  ALGOFKLG
//ALGOFKLG PROC SYSLIB='SYS1.ALGLIB',GOSET='&GOSET',MEMBER='(GO)',
// LDISP1=NEW,LDISP2=PASS,LDISP3=,UNIT=SYSDA,VOLUME=,PRIM=190,SEC=15,
// DIR=2,TRACE=DUMMY,PUT=DUMMY,CA=A,LA=A,GA=A,GAS=A
//ALGOL  EXEC  PGM=ALGOL,PARM='NCTEST',REGION=104K
//SYSPUNCH DD  SYSOUT=B,DCB=(RECFM=FB,LRECL=80,BLKSIZE=80)
//SYSPRINT DD  SYSOUT=&CA,DCB=(RECFM=FM,LRECL=91,BLKSIZE=91)
//SYSUT1 DD DSN=&&POOLB,SPACE=(1000,30),UNIT=SYSDA,DISP=(OLD,PASS)
//SYSUT2 DD DSN=&&POOLC,SPACE=(1000,30),UNIT=SYSDA,DISP=(OLD,PASS)
//SYSUT3 DD DSN=&&POOLD,SPACE=(1000,30),UNIT=SYSDA,DISP=(OLD,PASS)
//SYSLIN DD    DSNAME=&&LOADSET,UNIT=(SYSDA,2),SPACE=(1600,(100,10)),
//             DISP=(MOD,PASS),DCB=(RECFM=FB,LRECL=80,BLKSIZE=1600)
//LKED  EXEC PGM=IEWL,PARM=(MAP,LET,LIST),COND=(5,LT,ALGOL),REGION=104K
//SYSPRINT DD  SYSOUT=&LA,DCB=(RECFM=UA,BLKSIZE=121)
//SYSUT1 DD DSN=&&POOLA,SPACE=(1000,100),UNIT=SYSDA,DISP=(OLD,PASS)
//SYSLIB    DD  DSNAME=&SYSLIB,DISP=(SHR,PASS)
//          DD  DSNAME=SYS1.ALGLIB,DISP=SHR
//SYSLIN    DD  DSNAME=&&LOADSET,DISP=(OLD,DELETE)
//          DD  DDNAME=SYSIN
//SYSLMOD DD SPACE=(1024,(&PRIM,&SEC,&DIR)),UNIT=&UNIT,
// VOLUME=&VOLUME,DISP=(&LDISP1,&LDISP2,&LDISP3),DSNAME=&GOSET.&MEMBER
//GO     EXEC  PGM=*.LKED.SYSLMOD,COND=((5,LT,ALGOL),(5,LT,LKED)),
// REGION=104K
//ALGLDD01 DD  SYSOUT=&GA,DCB=(LRECL=132,BLKSIZE=132)
//SYSPRINT DD  SYSOUT=&GAS,DCB=(LRECL=91,BLKSIZE=91)
//SYSUT1 DD &TRACE,DSN=&&POOLA,SPACE=(10,10),UNIT=SYSDA,DISP=(OLD,PASS)
//SYSUT2 DD &PUT,DSN=&&POOLB,SPACE=(1000,10),UNIT=SYSDA,DISP=(OLD,PASS)
```

Styrspråk för tidsdelningssystem

Då operativsystem för på marknaden tillgängliga time-sharing system funnits i drift väsentligt kortare tid än för system för satsvis eller kövis bearbetning har ännu 1970 endast ett fåtal gemensamma nämnare för dessa styrspråk konkretiserats. Vår belysning av styrspråk för time-sharing får därför bli något kortfattad.

I time-sharing miljö startas "konversationen" mellan användare och system med att användaren anmäler sig till systemet (som "sökande" av datorresurser). Detta brukar kallas att anmälaren "loggar in sig". Härvid åsyftas anmälan till den debiteringsbokföring (logg), som systemet alltid är utrustat med. Vid inloggningen måste användaren ge systemet uppgift om identifikation, kontonummer, ev "lösenord" samt vilket programvarusystem han/hon avser använda. Vissa mindre driftssystem tillåter användning av endast ett programvarusystem åt gången. Härvid kan exempelvis tiden på dygnet vara vägledande för användaren rörande programvarusystem:

mellan kl 08 och 12 körs BASIC, mellan 12 och 14 "conversational" FORTRAN osv.

Systemet svarar på inloggningen med en kommentar (eventuellt om aktuella systemkaraktäristika samt) att användaren är välkommen påbörja "dialogen" (programmering eller programinitiering). I samband härmed reserverar systemet, beroende på driftsprincip, sekundärminnesutrymme för användaren samt visst buffertutrymme i primärminnet.

Vissa time-sharingsystem gör det möjligt för användaren att vid inloggningen frivilligt specificera om mer än standardmässiga systemresurser avses beläggas. Därvid förutsätts att användaren har tillräckligt detaljerad kännedom härom före programmeringens början. Då detta ofta inte är fallet (en av idéerna med time-sharing) blir standardmässiga resurser (satta vid systemgenereringen) initialt använda.

Vi noterar att inloggningen i time-sharing miljö ungefär motsvarar JOB/IDENT-styrsatsen i satsvis/kövis bearbetningsmiljö.

Efter det att systemet besvarat inloggningen, exempelvis genom att på terminalen utmata ordet READY eller INPUT, kan dialogen börja. Time-sharingteknikens normalt interpretativa karaktär medger emellertid, till skillnad från så som är fallet i satsvis eller kövis miljö, att användaren när som helst under programmeringens gång kan anropa operativsystemet. Det vanligaste av dessa anrop utgörs av instruktionen för att mata in en programsats till det reserverade sekundärminnesutrymmet (via primär-

minnesarean). Denna styrinstruktion är , givetvis, så frekvent förekommande att den i nära nog alla system ges genom tryck på en därför speciellt avsedd tangent på terminalens tangentbord, normalt vagnretur eller SEND-knapp (där dylik finns). Denna tangent används f ö även vid inmatningen av övriga styrsatser, då den ofta utgör det enda sättet att från terminal sända meddelanden till datorsystemet.

Vissa system omtalar för användaren att den sända programsatsen korrekt har tagits emot, och lagrats, genom att t ex utföra vagnretur på terminalen samt genom att automatiskt generera ett programsatsnummer på nästa rad (exempelvis 5 eller 10 enheter större än den inmatade programsatsens nummer, för att möjliggöra enkel inskjutning av senare eventuellt tillkommande satser). Andra system levererar blott vagnreturen som tecken på acceptans av inmatad sats.

Användaren har normalt ett flertal styrinstruktioner (här ofta kallade "kommandon") till sitt förfogande under programmeringens gång samt därefter. Vi ger nedan exempel på några sådana kommandon.

NAME	Tilldelning av identifierare (namn till det aktuella[1] programmet
SCRATCH	Aktuellt program förstörs, och motsvarande sekundärminnesutrymme ledigförklaras.
DELETE	Radering av en del av aktuellt program.
RENUMBER	Omnumrering av lägesnummer av aktuellt programs satser.
LIST	Ger utskrift av aktuellt program på terminalen.
PUNCH	Genererar vid användning av teletype/teleexterminal utstansning av aktuellt program på hålremsa.
TAPE eller INPUT	Meddelande om inläsning av program från terminalläsare
RUN	Startar exekvering av aktuellt program.

1) Med aktuellt program avses det program som för tillfället är inladdat i körbart tillstånd/eller som är under uppbyggnad.

SAVE	Aktuellt program sparas på sekundärminne och katalogiseras.
KILL	Raderar ut program från sekundärminne.
GET	Överför program från sekundärminne till primärminne.
SUSPEND	Ledigförklara tillfälligt aktuell terminal, utan att bryta kommunikationen med aktuell användare.
READMIT	Återupptagande av "SUSPEND"-ad konversation.
BYE eller FINISH eller LOGOUT	Avslutar kommunikationen mellan systemet och aktuell användare, dvs ledigförklarar samtliga användaren tilldelare resurser (utom permanent reserverade sekundärminnesfiler).

På dessa, och andra, användarinmatade styrkommandon svarar systemet på olika sätt. Vi nämner blott några få:

- Efter inloggning (HELLO, LOGON e d) svarar datorn med några "välkommande ord", diverse upplysningar och därefter t ex READY.

- Efter avslutad exekvering ges i flera system meddelandet DONE.

- Efter inmatning av felaktig styrsats svarar flera system med ett eller flera frågetecken (?).

- Efter inmatning av felaktig programsats svaras med lämplig felutskrift.

- Efter avslutad konversation (BYE/FINISH eller LOGOUT) svarar systemet med en uppgift om hur lång terminaltid och ibland hur lång processortid som använts, samt eventuellt hur lång tid som finns kvar på aktuellt konto.

Ett mycket enkelt exempel på konversation i time-sharing miljö ges nedan. Programmeringsspråket BASIC har använts och körningen är utförd på en Hewlett Packard 2000B minidator med 8 terminaler, som ännu i oktober 1970 befann sig i institutionens för informationsbehandling, KTH, Stockholm, lokaler.

```
(1)  HELLO-████,CADIS

*            VAELKOMMEN INF.BEHANDLINGS TIME-SHARING BASIC
*            *****************************************

*        1. TORSDAGEN 24/9-70 INSTALLERADES ETT NYTT TIME-SHARINGSYSTEM
*           VILKET INNEBAER ATT ALLA PROG. SOM FANNS PA SKIVANFOERSVANN

*        2. 20 POANGS-STUDERANDE I ADB KAN FORTFARANDE EJ LAGRA SINA PROGRAM

*        3. UNDER EN PERIOD SKALL AEVEN DISC-OPERATING SYSTEM ANVAENDAS
*           VILKET INNEBAER MINSKAT UTRYMME PA SKIVMINNET

                          LYCKA TILL

(2)  10 LET A - 13
 *   ERROR
(3)  10 LET A = 13
(4)  20 PRINT "A=",A
(5)  30 END
(6)  RUN

 *   A=                 13

 *   DONE
(7)  NAME-MIDI
(8)  SAVE
(9)  BYE
 *   004 MINUTES OF TERMINAL TIME

(10) HELOO--LO-████,CADIS

*            VAELKOMMEN INF.BEHANDLINGS TIME-SHARING BASIC
 *   STOP
(11) GET-MIDI
(12) LIST
 *   MIDI

 *   10  LET A=13
 *   20  PRINT "A=",A
 *   30  END
(13) 15 LET A=A+12345678/12345678-1
(14) RUN
 *   MIDI

 *   A=                 13

 *   DONE
(15) LIST
 *   MIDI

 *   10  LET A=13
 *   15  LET A=A+1.23457E+07/1.23457E+07-1
 *   20  PRINT "A=",A
 *   30  END
(16) BYE
 *   003 MINUTES OF TERMINAL TIME
```

Vi kommenterar nu människa-maskindialogen enligt ovan kortfattat.

De *-märkta raderna är sådana som utmatats från systemet. Övriga rader är inmatade styrinstruktioner eller programsatser. Endast raderna (2)-(5) och (13) är "vanliga" programinstruktioner. Övriga rader är styrinstruktioner (kommandon).

(1) Användaren "loggar in". Kontonumret är av sekretesskäl överstruket.

(2) Programmets uppbyggnad kan börja. Tyvärr blev första raden felaktig.

(5) Programmets sista rad.

(6) Anmodan att börja exekvering av det skrivna programmet. Efter exekvering utskrivs beskedet DONE.

(7) Det aktuella programmet ges namnet MIDI.

(8) MIDI undanlagras på direktminne.

(9) Körningen avslutas.

Någon dag senare:

(10) Användaren "loggar in" sig igen (← "suddar" ett felaktigt skrivet tecken).

(11) Gör MIDI till aktuellt program.

(12) Lista MIDI.

(13) En programändring i MIDI görs. Raden 15 inskjutes automatiskt mellan rad 10 och 20.

(14) Exekvera!

(15) Lista senaste version av aktuellt program

(16) Körningen avslutas.

Den programversion av MIDI son ligger kvar på direktminnet är fortfarande "den ursprungliga", dvs den utan rad nr 15.

Litteratur

1. Walter, E.S. & Wallace V.L., "Further Analysis of a Computing Center Environment". Communications of the ACM, Vol. 10, No. 5, May, 1967 (p. 266).

2. Coffman Jr, E.G. & Wood, R.C., "Interarrival Statistics for Time Sharing Systems." Communications of the ACM, Vol. 9, No. 7, July, 1966 (p. 500).

3. Leverantörers handböcker om resp operativsystem.

7. STYRPROGRAM OCH STYRPROGRAMFUNKTIONER

7.1. Introduktion

Ett styrprograms funktion är att utgöra en logisk länk (interface) mellan maskinvarufunktioner och övriga delar av operativsystemet samt applikationsprogrammen. Hos multiprogrammerbara system administrerar det bl a de maskin- och programvaruresurser som erfordras för att utföra flera processer simultant. Då multiprogrammering idag är allmänt förekommande, även på mindre datorer, kommer vi att i huvudsak behandla styrprogram med multiprogrammeringsmöjlighet. Den logiska strukturen hos styrprogram liksom de funktioner som de anses omfatta varierar beroende dels på tillverkare och dels på avsedd driftsfilosofi. Det är en praktisk omöjlighet att här belysa alla förekommande varianter. Det finns två skäl varför vi här ej i detalj behandlar ett styrprograms uppbyggnad och funktion.

1. Några generella, allmänt vedertagna principer kan knappast anses existera. En mindre detaljbunden framställning gör att vi i viss mån kan undvika speciella problem och använda speciell terminologi. Problemen har lösts olika från tillverkare till tillverkare och detta försvårar en diskussion av problemställningar som kan anses vara gemensamma för flera styrsystem. En detaljerad framställning skulle tvinga oss att välja ett eller flera bestämda operativsystemfabrikat.

2. En sådan framställning skulle dessutom bli synnerligen omfattande och dessutom duplicera den information som återfinns i respektive leverantörers manualer.

Vår avsikt är därför att i detta avsnitt belysa några principer för ett styrsystems funktionssätt (dock utan att göra anspråk på dessa principers allmängiltighet) för att på så sätt underlätta läsarens studier av speciella styrprogrambeskrivningar.

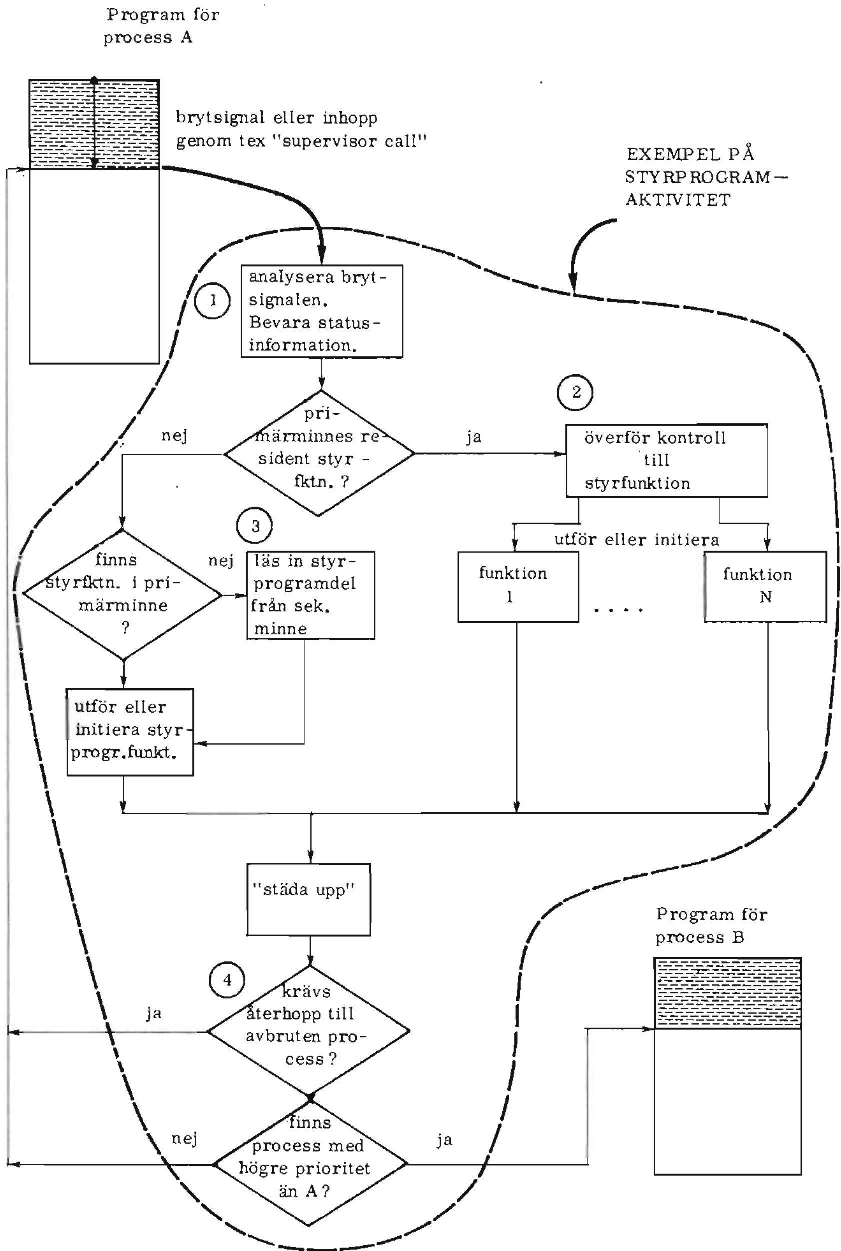
Deljobb som identifierats av jobbövervakaren på nivå 1 (se fig 4.1:1) överlämnas av denna i form av processer till exekvering under kontroll av styrprogrammet. Styrprogrammet ombesörjer att erforderliga resurser löpande tilldelas processen. Tilldelningen kan vara prioritetsstyrd och avse resurser såsom processortid, datatransportservice, primärminnesutrymme osv. Styrprogrammet håller vid varje tidpunkt

reda på resursutnyttjandet och tilldelningen samt ser till att resurser återlämnas efter det att aktuell resursförbrukande process avslutas. Hos multiprogrammerade system (med möjlighet till exekvering av flera parallella processer) kommer resurskonflikter att uppstå. Resurskraven från parallella processer måste då, om en resurs ej simultant kan delas av flera processer, inordnas i köer. Administration av dessa köer ombesörjes av styrprogrammet. För operativsystem, vilkas styrsystem endast kan administrera ett maximerat, för ett driftspass definierat antal parallella processer används ibland terminologin "multiprogramming with a fixed number of tasks" (MFT enl IBM's terminologi). På samma sätt kallar IBM system som kan administrera ett dynamiskt föränderligt antal processer parallellt för MVT - "multiprogramming with a variable number of tasts".

En aktiv fas i styrprogrammet påbörjas antingen genom att en brytsignal, t ex från an avslutad datatransport, överför kontrollen till en brytsignalsrutin (som är en del av styrprogrammet) eller genom att ett program i skyddstillstånd begär service från styrprogrammet genom ett hopp (med undanlagring av returadress) till lämplig ingång för motsvarande servicefunktion. Hos många datorsystem existerar instruktioner som kan exekveras av ett program i skyddstillstånd (dvs av ett applikationsprogram) och som därmed alstrar en brytsignal (s k "supervisor calls") som då överför kontroller till styrprogrammet. Några styrprogramaktiviteter som kan initieras av en brytsignal eller ett inhopp till styrprogrammet illustreras grovt i figur 7.1:1.

Antag att ett program exekveras och därmed utgör processen A. Processen A avbryts av en brytsignal som kan vara orsakad av datatransportavslutning (avseende processen A eller någon annan process), programfel el dyl eller också kan processen A själv ha utfört ett anrop till någon styrprogramfunktion. Vid ingången till styrprogrammet analyseras brytsignalen för att avgöra dess orsak och typ, och för att bestämma vilken styrprogramfunktion som skall initieras (1). Varje brytsignal genererar en mängd statusinformation om den "avbrutna" processen som lagras kompakt i ett bestämt primärminnesområde. Denna "statuslagring" är normalt en maskinvarufunktion och omfattar bl a nödvändiga data, såsom returadress, innehållet i vissa register, indikatorer o dyl, för att återstarta processen vid ett senare tillfälle.

Med hjälp av den avbrutna processens statusinformation kan styrprogrammet sedan återinitiera denna när dess resurskrav kan tillfredsställas. Brytsignalen orsakar också normalt en övergång från skyddstillstånd till (icke avbrytbart) beordringstillstånd.



Figur 7.1:1
Grov programplan som visar tänkbar aktivitet i ett styrprogram efter avbrott av (eller inhopp från) en process A.

Om programavbrottet orsakats av in/utmatning, programfel (t ex minnesskyddsbrott), maskinvarufel eller dylikt lämnas kontrollen till styrfunktioner vars program normalt alltid ingår i den residenta delen av styrprogrammet (2). Vissa andra servicekrav kan gälla funktioner vars program eventuellt befinner sig i primärminnet vid aktuell tidpunkt (dvs transienta program). Om motsvarande funktionsprogram ej finns i minnet måste det först inläsas från systemprogramutrymmet på sekundärminnet (3).

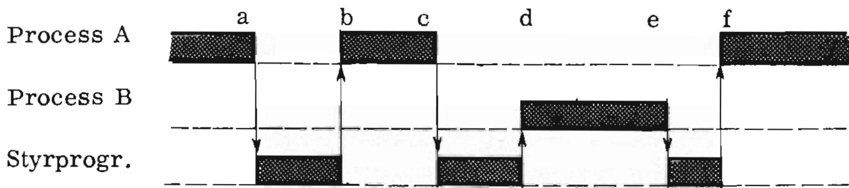
Denna inläsningsoperation erfordrar arbete av andra styrprogramresurser (= funktioner) vilket innebär att en del av styrprogrammet själv anropar andra delar i styrprogrammet på motsvarande sätt som process A. Dock har denna process i beordringstillstånd normalt högre prioritet. Inläsning av ett transient styrprogram kan vara tidsödande (storleksordning 10-100 millisekunder beroende på accesstiden hos aktuellt sekundärminne). Motsvarande processortid kan dock nyttjas av andra processer i skyddstillstånd än A om sådana existerar i systemet vid tidpunkten ifråga. När sedan laddning av program för styrfunktion skett utföres denna och därefter sker övergång till att "städa upp" och avsluta aktuell styrprogramaktivitet. Till slut gäller det att bestämma vilken process i skyddstillstånd som skall återinitieras. Hos vissa system kan den anropande processen kräva att, så fort funktionen är utförd, omedelbart återfå kontrollen. I annat fall initieras den process som väntar på processortid och har högst prioritet. Figur 7.1:2 avser att tidsmässigt illustrera detta.

Det genomgångna exemplet på samspelet styrprogram och användarprogramprocesser kan sägas vara typiskt för de flesta av dagens operativsystem som tillämpar driftsprincipen sats- eller kövis bearbetning. Implementeringen av styrsystemet blir något annorlunda för tidsdelnings- och reelltidssystem på grund av de speciella krav som ställs på dessa systems driftsegenskaper.

De centrala frågorna för samtliga system är dock den löpande resursadministrationen och resursplaneringen främst gällande

- processortid
- primärminne
- datatransporter

Datatransporterna kommer att beröras i kapitel 8 och därför ägnas detta kapital åt principer och problem avseende processortids- och primärminnesadministration. Det bör betonas att dessa två problemområden har ett beroendeförhållande. Beroendet ökar ju mer "avancerade" system vi betraktar. Till exempel kan man knappas diskutera tidsdelningssystems processortidstilldelning utan att beröra dess principer för tilldelning av primärminne och vice versa.



Figur 7.1:2
Tänkbart tidsförlopp för samverkan mellan styrprogrammet och två processer A och B. Växlingen mellan olika delar i styrprogrammet ej visad.

a. Process A begär initiering av en servicefunktion hos styrprogrammet.

b. Funktionen är initierad och kontrollen återlämnas till A då den antas ha högre prioritet än B.

c. Processen A överlämnar frivilligt kontrollen till styrprogrammet för att vänta på att den vid a initierade funktionen skall avslutas. Processen A övergår därmed i ett väntestatus (passivt tillstånd, s k "wait-state").

d. Styrprogrammet finner att process B står närmast i tur att få processortid och överlämnas kontrollen till B.

e. Servicefunktionen som initierats vid a är nu avslutad.

f. Kontrollen återlämnas till A. Processen B, som här avbrutits, återgår till bearbetningsklart tillstånd ("ready-state").

Innan vi går in på processortids- och primärminnesproblemen skall vi uppehålla oss vid några grundläggande begrepp såsom processer, adressrum, processinteraktion, processtillstånd m m. Vissa av dessa begrepp har berörts i kapitel 3. Emellertid skall vi betrakta dem här ur en mindre abstrakt och mer databehandlingsteknisk synvinkel.

De tids- och minnestilldelningsalgoritmer som brukar användas hos "konventionella" sats- och kövis arbetande system är vanligtvis föga sensationella. Litet finns skrivet om dem. Litteraturen är däremot riklig vad avser tids- och minnesplanering for multipel-access tidsdelningssystem. Den metodik och de principer som dessa system idag representerar torde bli basen för framtida datorsystem. Av den anledningen har vi ägnat dessa metoder och principer relativt stort utrymme i detta kapitel.

7.2 Några grundläggande begrepp

7.2.1 Processer och deras egenskaper

Som tidigare diskuterats i kapitel 3, utföres allt arbete i ett datorsystem i form av exekvering av processer. Vi har valt att översätta den engelska termen "task" med process. Begreppet process är ej lätt att definiera entydigt. Den vanligast förekommande definitionen är att en representerar en - ur styrprogrammets synvinkel - given mängd arbete.

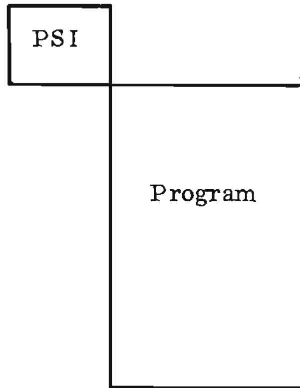
Processer kan förekomma på olika "styrningsnivåer" - såväl styrprogrammet som övriga "styrda" program (användarprogram) utför sitt arbete i form av processer.

För att en process skall kunna exekveras (äga rum) krävs resurser, vilka alltså kan sägas bli "förbrukade" under processens gång. Förbrukningen bestäms av det program som styr processen. Resurserna kan, som tidigare nämnts, vara av såväl maskinvarumässig som programvarumässig natur.

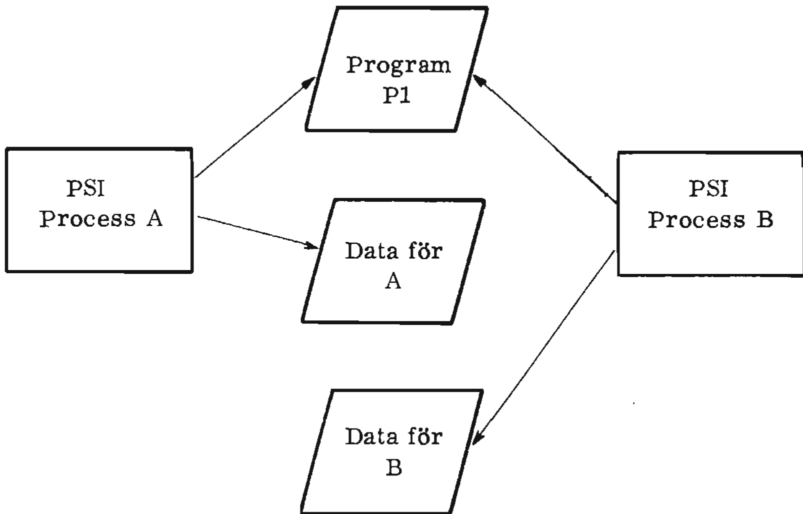
Det är viktigt att skilja mellan en "process" och ett "program". Programmet bör ses som en beskrivning som bestämmer vad processen skall utföra, dvs en nödvändig förutsättning för att processen skall kunna äga rum. Det innebär bl a att det vid en tidpunkt kan existera (med viss tidsförskjutning) två eller flera lika processer som utför samma slags arbete (på skilda datamängder) och att det i primärminnet behövs endast en kopia av ett program som beskriver processernas arbete. I detta fall ställs dock krav på att programmet måste vara multianvändbart (skrivet i reentrant-form) - en vanlig egenskap hos många av de program som ingår i ett operativsystem.

En illustrativ grafisk representation av en process tillämpas av IBM (figur 7.2:1).

Bilden avser att illustrera att för styrning av en process erfordras minst två informationsmängder associerade med densamma. Dels behövs programmet som beskriver arbetet som skall utföras, dels behövs en mängd tillstånd- och annan information (PSI - processens styrinformation) som i huvudsak används för att samordna processens arbete med andra eventuellt parallellt pågående processer och lagra referenser (länkar) till olika resurser som används av processen. Bilden 7.2:1 är möjligen något missvisande då den ger intrycket att en process alltid erfordrar eget primärminnesutrymme både för PSI och för program. Om programmet och erforderliga datamängder, liksom maskinella faciliteter, betraktas som en resurs torde två processer som styrs av samma program kunna illustreras enligt figur 7.2:2.



Figur 7.2:1
Grafisk representation av en process. PSI = processens styrinformation.



Figur 7.2:2
Processerna A och B säges tillhöra samma klass (P1) av processer då de styrs av samma program. Normalt är då att även datamängderna A och B har strukturella likheter.

Processers adressrum och namnrum

Att en process exekveras innebär att instruktioner hos den programmängd som definierar processen utföres. Utförande av instruktioner inkluderar referenser till adresser. Varje process kan sägas ha ett eget adressrum med vilket det är förknippat. Vi kan skilja mellan tre slag av adressrum

1. fysiskt adressrum, bestående av mängden av för processen adresserbara primärminnesplatser.

2. virtuellt eller logiskt adressrum, som utgöres av mängden av adresser som kan användas av programmeraren för att referera till olika informationselement.

3. namnrum (vanligen symboliskt), ett mer generellt begrepp än logiskt adressrum och som utgöres av mängden av de namn (identifierare) som en process, beskriven i ett symboliskt språk, kan referera till.

I litteraturen har normalt endast (1) och (2) diskuterats och man har i vissa fall satt ett ekvivalenstecken mellan namnrummet och det logiska adressrummet.

Det förefaller lämpligt att skilja på dessa begrepp. Ett program skrivet i ett problemorienterat språk förutsätter ett symboliskt namnrum. T ex har Algolprogrammet

```
begin real a, b; array c [1:3]; Boolean d;
      .......
end;
```

det symboliska namnrummet (på "problemnivå")

{a, b, c[1], c[2], c[3], d}

Vid kompliering transformeras namnen till ett logiskt adressrum, vanligen innehållande numeriska element. Exekvering innebär att adresserna transformeras (vid laddning eller, genom adressavbildning, vid exekvering) till fysiska adresser.

I tidiga datorsystem överensstämde det logiska adressrummet för en process med det fysiska adressrummet. Programmet var då skrivet i ren maskinkod och man arbetade med absoluta adresser. Assemblykodning och, senare, programmering i problem- eller procedurorienterade språk medgav möjlighet att skriva program där logiska adress- eller namnrum, skilda från det fysiska adressrummet, kunde användas.

För att en process skall kunna exekveras krävs att för exekvering aktuella instruktioner och data befinner sig i primärminnet (med nuvarande allmänt accepterade principer för uppbyggnad av datorer). En processor kan endast acceptera operandadresser som ingår i det fysiska adressrummet och därför krävs att programmets namn i namnrummet först transformeras till fysiska adresser.

Det fysiska adressrummet representeras vanligen som en mängd av heltal, $M = \{0, 1, 2, \ldots, m-1\}$. Det logiska adressrummet kan ha olika struktur. I det enklaste och vanligaste fallet kan vi även representera det logiska adressrummet av mängden av heltal, $L = \{0, 1, 2, \ldots, l-1\}$, där l ej behöver (generellt) vara mindre än antalet tillgängliga primärminnesplatser (m). Detta kallas för ett linjärt adressrum. Ett annat vanligt adressrum benämns det segmenterade, vilket består av en mängd av linjära adressrum, $S = \{L_1, L_2, \ldots, L_s\}$, där $L_i = \{0, 1, \ldots, l_i\}$. Referens till ett informationselement (variabel, konstant, programinstruktion m m) i adressrummet L sker genom att ange dess nummer (a) i den representerande talmängden. I det segmenterade adressrummet består en adress av talparet (b, a), där b anger segmentnumret och a anger elementet (adressen) inom segmentet.

Om segmentnamnen och/eller namnen för element inom ett segment ej är numeriska begrepp talar vi om ett symboliskt segmenterat adressrum. Ett symboliskt, segmenterat adressrum är ej linjärt och aritmetriska manipulationer med segment- och elementnamn kan vanligen ej göras.

När vi programmerar i ett högre-nivå språk (Algol, FORTRAN, COBOL, PL/I, ...) arbetar vi i ett symboliskt namnrum, som utgöres av explicit eller implicit deklarerade identifierare och lägen i programmet. Vid kompilering översättes programmets namn till namn i ett linjärt, beroende på datorsystem eventuellt segmenterat, adressrum.

När programmet skall exekveras kan förfaras på två principiellt olika sätt[1]:

1. vid inladdning av programmet (eller delar därav) översätts namnen till adresser i det fysiska adressrummet.

2. programmet (eller delar därav) inladdas med bibehållande av dess (logiska) adresser och under exekvering översätts adresserna till deras motsvarighet i det fysiska adressrummet.

---

1) Förfarandet att vid kompilering anpassa namnen i programmet till bestämda (absoluta) minnesadresser är numera ovanlig och vi tar därför ej med det här.

Den senare principen är på 60-talets datorsystem den vanligaste. Beroende på restriktioner avseende adressrummets storlek och struktur tillämpas olika översättningsprinciper. Den enklaste innebär modifiering av varje adress vid exekvering med innehållet i ett basregister. Om programmet är större än tillgängligt utrymme faller det här på programmeraren att ombesörja programadministration och inläsning samt överlägring av programdelar. Vissa datorer, främst sådana för tidsdelning, har inbyggda, mer avancerade funktioner för avbildning av namn från ett programs logiska adressrum till det fysiska adressrummet, med vilkas hjälp programmeraren kan bibringas illusionen att arbeta med ett mycket stort primärminne. Dylika minnessystem kallas "virtuella" och vi återkommer till dem längre fram.

Vi har i detta avsnitt introducerat begreppen namnrum, adressrum och belyst förhållandet att varje process P_i arbetar i ett adressrum A_i med vilket det är förknippat. A_i kan, beroende på vem som betraktar processen vara ett namnrum eller ett linjärt logiskt adressrum. Exekvering förutsätter att en avbildning $f{:}A_i \rightarrow M$, det fysiska adressrummet, kan åstadkommas. Det är underförstått att avbildningsfunktionen f generellt måste ses som variabel med tiden. Flyttning av processens data eller program i primärminnet, eller till/från sekundärminnet orsakar att f förändras.

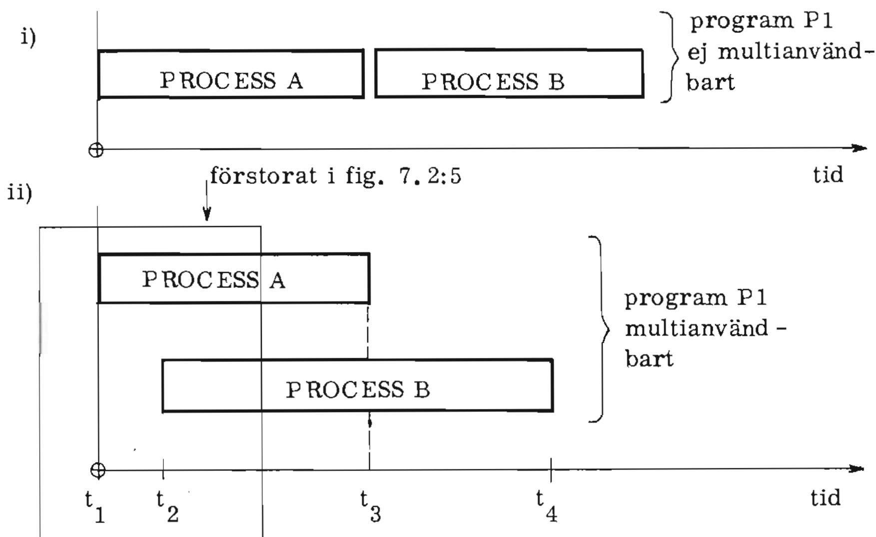
7.2.2 Samverkan mellan processer och styrprogrammet

Om processerna A och B pågår simultant och programmet P1 är multianvändbart blir det sammanlagda minnesbehovet bestämt av storleken för PSI_A, PSI_B, P1, $Data_A$ och $Data_B$. Beroende på om programmet P1 är multianvändbart eller ej kan processerna A och B utföras med viss överlappning. Detta illustreras i figur 7.2:3.

Om programmet P1 icke är multianvändbart kan processen B initieras endast när processen A avslutats, dvs A och B kommer att utföras i serie.

För att en process skall kunna utföras krävs även andra resurser än t ex program. Parallella processer kommer därvid att konkurrera om en eller flera resurser.

Styrprogrammets uppgift är bl a att lösa dessa resurskonflikter genom att styra och i möjligaste mån planera resurstilldelningen till processerna. Vissa "fasta" resurser tilldelas en process vid initiering och behålls sedan av processen under dess "livstid", Exempel på sådana resurser är primärminne för PSI-blocket (processtyrinformation), program och data-



Figur 7.2:3
Tänkbart tidsförlopp för två processer av samma klass P1 i) när programmet P1 endast får användas exklusivt av en process åt gången och ii) när programmet P1 är multianvändbart. Det antas att B initieras vid tidpunkten t_2. Observera att i fallet ii) tidsutsträckning för processerna ofta är längre än i fallet i) då resurskonflikter kan uppstå. Styrprogrammets aktiviteter är ej visade här utan framträder i den förstorade bilden 7.2:5.

mängder. Ovanstående hindrar ej att primärminneslagrad information tillfälligt matas ut till sekundärminne för att bereda plats till processer med högre prioritet. Andra resurser kan tilldelas dynamiskt under processens liv i systemet och återlämnas när de ej längre behövs eller när processer, som har högre prioritet, behöver dem. Även vissa av de resurser som ovan betecknats som "fasta" kan hos system med mer flexibel driftsfilosofi betraktas som dynamiskt tilldelbara.

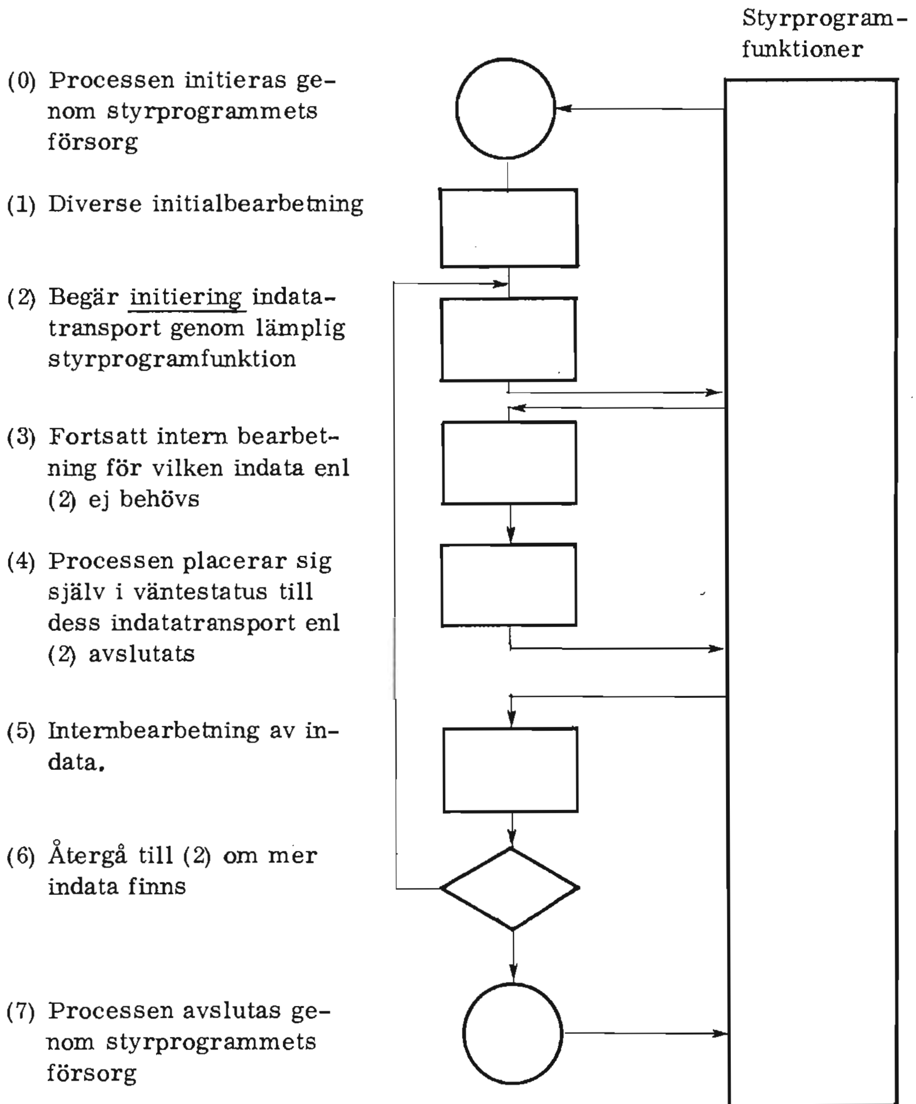
Beroende på vilka resurskonflikter som uppstår samt vilken prioritet som tilldelats processerna uppträder vid simultan exekvering alltid en viss tidsfördröjning för den individuella processen.[1)]

---

1) Framgår t ex av figur 7.2:5 och tillhörande diskussion.

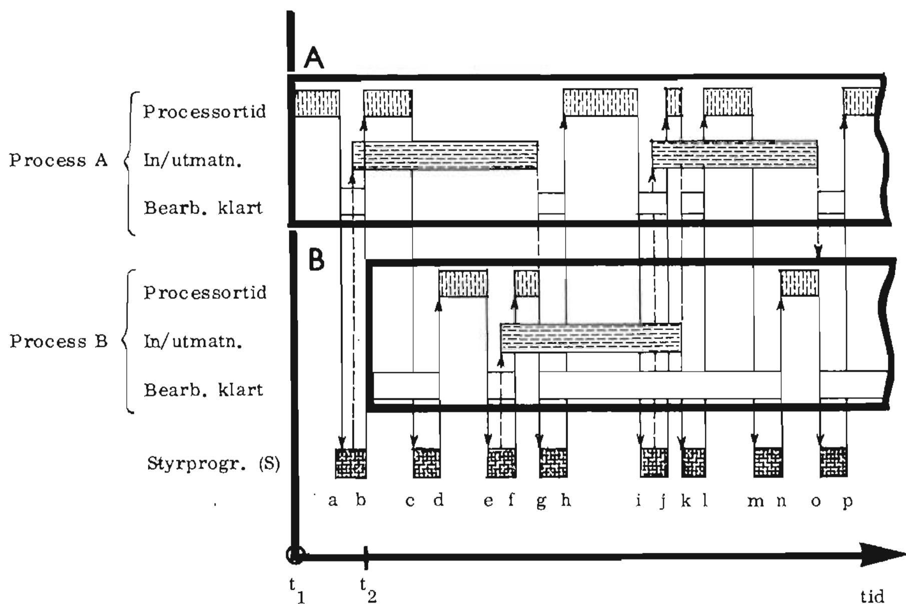
Figur 7.2:3 avser ligga till grund för den följande diskussionen om styrprogrammets styrning av två mycket enkla processer samt dessa processers olika tillstånd.[2]

Processer A och B antas tillhöra samma klass och bestå av internbearbetning och inmatning av data enligt programplan i figur 7.2:4.



Figur 7.2:4
Grov programplan för processer av klassen P1.

2) Betr processtillstånd, se även kapitel 3.



Figur 7.2:5
Samverkan mellan två processer och styrprogrammet. Styrprogrammets arbete (dvs växlingen mellan dess olika processer) är ej visad här. Det antas att datorsystemet (endast) har en processenhet. Processen A antas ha, med avseende på processortid, absolut prioritet före process B - dvs varje gång A är bearbetningsklar avbryts eventuella andra processer och kontrollen återgår, efter erforderlig styrprogramaktivitet, till A.

Vid en tidpunkt $t < t_1$ har styrprogrammet etablerat processen A. För att en process skall etableras krävs bl a följande

- processens styrinformationsmängd (PSI) skall etableras och initialiseras vilket bl a kräver minnesutrymme för lagring av densamma.

- programmet som definierar processens arbete skall, om det ej redan finns i primärminnet inläsas dit. Detta kräver primärminne och kan innebära att etablering fördröjs.

- processen skall tilldelas diverse andra resurser (som krävs för att starta exekvering) såsom primär- eller sekundärminnesutrymme för data.

När processen på så sätt har etablerats är den "redo" för exekvering. Om ingen annan process vid denna tidpunkt är under exekvering kan processen A omedelbart startas genom att styrprogrammet (rättare: den del av styrprogrammet som ombesörjer tilldelning av processortid) överför kontrollen till första instruktionen i det program som representerar processen A. Detta kan för datorsystemet innebära en övergång från beordringstillstånd till skyddstillstånd. I styrprogrammet måste ju bevakas vilken process som är under exekvering, [1] vilken exempelvis kan göras genom att det upprättas en länk till det till processen motsvarande PSI-blocket.

Om, däremot, när A etablerats, en annan process, säg X, är under exekvering måste A placeras på "väntelista" till processortid. Vi säger då att A placeras i bearbetningsklart tillstånd. Egentligen är dock situationen vid A's etablering mer komplicerad. Under etablering av A ligger kontrollen hos en styrprogramprocess, och processen X, enligt ovan, är tillfälligt avbruten (befinner si i bearbetningsklart tillstånd). Givetvis kan, förutom X, ytterligare processer befinna sig i bearbetningsklart tillstånd. När A etablerats är den då eventuellt en av flera processer som är bearbetningsklara. Styrprogrammet måste nu avgöra för vilken av processerna A, X eller någon annan, som exekveringen skall startas. Detta beslut fattas av processortidsfördelningsfunktionen hos styrprogrammet, den s k "dispatchern" eller tidsplaneraren, och det kan ske på basis av de olika bearbetningsklara processernas prioriteter och eventuella andra processtidstilldelningsprinciper såsom t ex "timeslicing". Mer om detta anges i avsnitt 7.3.

I figur 7.2:5 har vi antagit att process A vid t_1 omedelbart kan försättas i exekveringstillstånd, dvs vid t_1 existerar inga andra bearbetningsklara processer med högre prioritet än A.

I det följande skall vi behandla några illustrativa tillståndsförändringar hos processerna A och B. Bokstäverna hänför sig till figuren 7.2:5.

---

1) Vid en-processorsystem kan vid varje tidpunkt endast en process i systemet vara under exekvering, oberoende av om processen tillhör styrprogramsystemet eller ett applikationsprogram.

a) A begär via datatransportfunktionen hos S inmatning av en datamängd till utrymme associerat med A, dvs A's PSI[1]. S initierar datatransportoperationen och etablerar även en ny process B. Då emellertid antagits B tillhöra samma klass av processer som A (redan etablerad) finns programmet P1 redan i primärminnet. Då P1 antagits vara multianvändbart (reentrant) kan B "dela" det med A och styrprogrammet behöver varken

   i) läsa in en ny kopia av P1 till primärminnet (efter att ha krävt utrymme för detta) eller

   ii) betrakta P1 som en sekvensiellt användbar resurs och ställa A i kö till densamma.

Under intervallet (a, b) är A i bearbetningsklart tillstånd och under denna tid utförs de styrprogramprocesser som ombesörjer det arbete som kortfattat beskrivits ovan.

Vi inser att tidsplaneraren ("dispatchern") kan bli aktiverad åtskilliga gånger under intervallet (a, b). Vid vart och ett av tidsplanerarens beslutstillfällen under (a, b) är A bearbetningsklart - i princip - men först måste de styrprogramaktiviteter som sammanhänger med A's "servicekrav" och eventuella andra styrprogramprocesser med högra prioritet avslutas. Därför inordnas dessa nya processer före A's krav (att efter styrprogramanropet "få tillbaka processtiden") i kön till resursen "processortid".

b) Då vi antar att A's prioritet är högre än B's återgår A till exekvering och B blir en bearbetningsklar process.

c) A kan ej längre utföra någon intern bearbetning utan tillgång till vid a) begärda indata. A överlämnar därför frivilligt kontrollen till S och talar om att processtid önskas åter när vissa resurs/servicekrav tillfredsställts - i detta fall inmatning av begärda data. Styrprogrammets arbete under (c, d) består därför i huvudsak av att registrera återhoppsvillkoren för A, notera A som en process i "väntestatus" och undersöka vilken process som står i tur att få processortid.

d) I detta fall ligger B i tur och övergår nu till exekvering.

e) B framställer till S krav på indata på motsvarande sätt som A gjorde

---

1) PSI innehåller normalt inga buffertutrymmen men väl länkadress till för processen tillåtna buffertar.

vid a). Erforderlig styrprogramaktivitet sker under (e, f) intervallet.

f) Kontrollen återgår till B då A's inmatning ännu ej är avslutad.

g) En brytsignal, orsakad av avslutad datatransport, överför kontrollen från B till styrprogrammet. Det konstaterar nu att datatransporten är associerad med processen A och att A befinner sig i väntestatus. Ett "servicekrav" för A har nu tillfredsställts. A har, i detta exempel, inga ytterligare "uteliggande" krav och därför kan A åter aktiveras. Vid denna tidpunkt befinner sig således både A och B i bearbetningsklart tillstånd.

h) Resursen processortid tilldelas processen A (då A har antagits ha högre prioritet än B, vilken under intervallet (h, i) förblir bearbetningsklar). Vi noterar dock att för B pågår den vid e) initierade datatransporten.

i) A anhåller, via S, om initiering av en datatransport på motsvarande sätt som vid a).

j) Kontrollen återgår till A.

k) Processens B indatatransport är avslutad vilket orsakar en brytsignal och kontrollöverföring från A till S. Under intervallet (k, l) noterar S detta - dvs att ett servicekrav från B har tillfredsställts.

l) Kontrollen återgår till A, på grund av högre prioritet.

m) A återlämnar frivilligt kontrollen till S och övergår till väntestatus.

n) Vid m) fanns endast B i bearbetningsklart tillstånd varför kontrollen överförs till densamma. Någon gång under (n, o) kräver B indata, Dessa är dock redan inlästa (vid k)) och därför kan B fortsätta direkt. Beroende på styrprogrammets utformning kan ett inhopp till S därvid erfordras för att konstatera om servicekraven redan är uppfyllda.

o) Indatatransport för A är avslutad. Brytsignal gör att kontrollen överförs till A och att A blir bearbetningsklart.

p) Prioriteten avgör att A får resursen processortid.

7.2.3 Processtillstånd

Processtillstånd har generellt behandlats i avsnitt 3.4. Vi upprepar här delvis resonemanget men med vissa utökningar och med anknytning till föregående exempel.

Exemplet i figur 7.2:5 har illustrerat hur en process under sin "livstid" kan alternera mellan olika tillstånd. Det tillstånd som processen befinner sig i är, som framgår ur exemplet, beroende dels på det program som styr processens arbete dels (vid multiprogrammering) på tillstånd hos andra processer i systemet och på de algoritmer som styr resursfördelningen. Vi har sammanfattningsvis kunnat notera följande tillstånd[1) (figur 7.2:6)

I Processen är icke etablerad, dvs inget PSI-block (processtyrinformation) för densamma är upprättat.

E Processen är etablerad och PSI är upprättad. Det är dock ej nödvändigt, i princip, att till processen hörande program vid denna tidpunkt befinner sig i primärminnet. Programmet betraktas ju som en resurs och kan krävas för att sedan försätta processen i bearbetningsklart tillstånd.

När processen väl är etablerad kan olika "deltillstånd" urskiljas:

P Processen är passiv, den är inte initierad i syfte att utföra någon bestämd arbetsuppgift.

A Processen är aktiv (initierad) och arbetar på en given arbetsuppgift. Återigen kan här olika deltillstånd urskiljas.

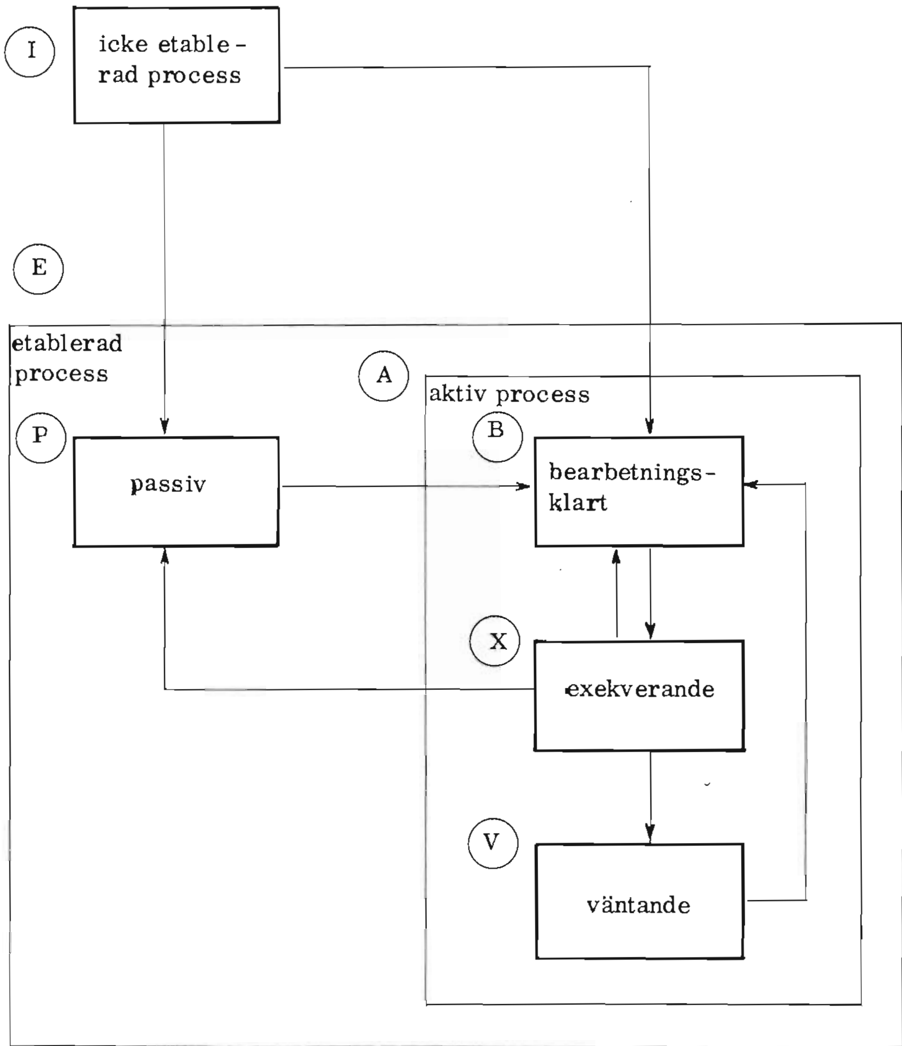
B Processen är bearbetningsklar och väntar, eventuellt med någon tillordnad prioritet, i en kö till processortid.

X Processen är exekverande på någon av datorsystemets processorer - dvs instruktioner hörande till processens program är under utförande.

V Processen är väntande på att ett eller flera av dess, via styrprogrammet fastställda, servicekrav är tillfredsställda.

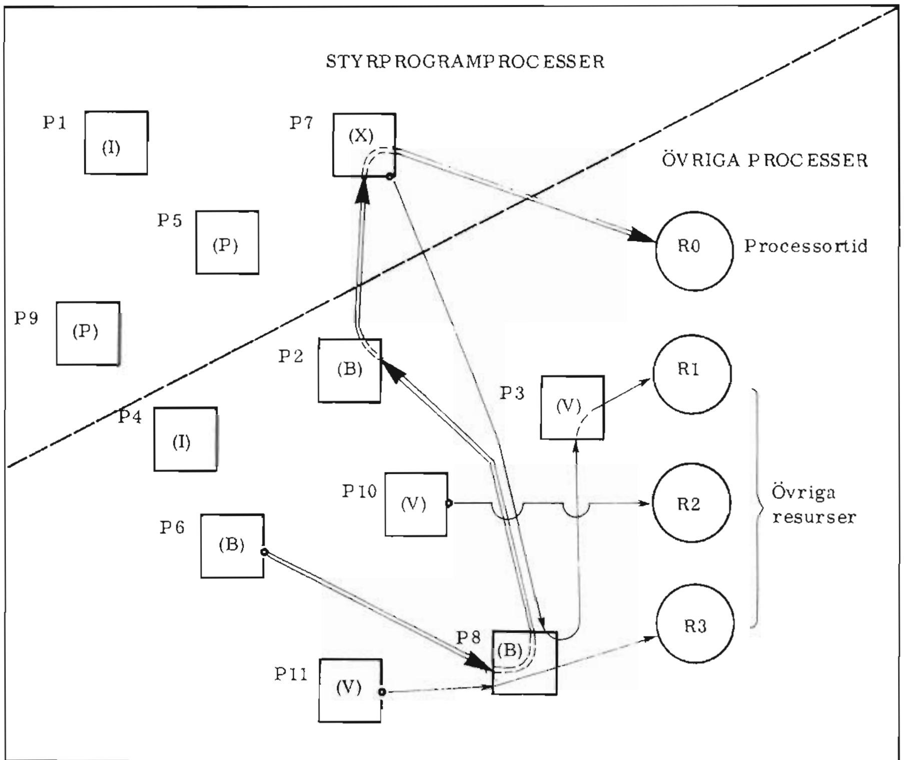
---

1) Vissa operativsystemtillverkare har valt att definiera ytterligare deltillstånd - dvs en "förfining" av nämnda tillstånd.



Figur 7.2:6
Transitionsgraf för olika processtillstånd.

Exemplet enligt figur 7.2:5 har endast kunnat illustrera en bråkdel av de aktiviteter som förekommer i datorsystem vid simultan bearbetning av processer. I multiprogrammerade dator/operativsystem existerar i varje ögonblick en mängd processer vilka kan, ur tillståndssynpunkt, klassindelas enligt grafen i figur 7.2:6. Vissa av dessa processer tillhör styrsystemet. Andra representerar arbete som utföres dels av övriga systemprogram, och dels av under körning varande användarprogram.



Figur 7.2:7
I varje ögonblick kan flera processer existera i systemet. Dessa befinner sig i olika tillstånd och är inordnade i köer till aktuella erforderliga resurser. En process kan i princip vara medlem av flera resursköer. Bokstäver inom parentes anger de tidigare (figur 7.2:6) diskuterade processtillstånden.

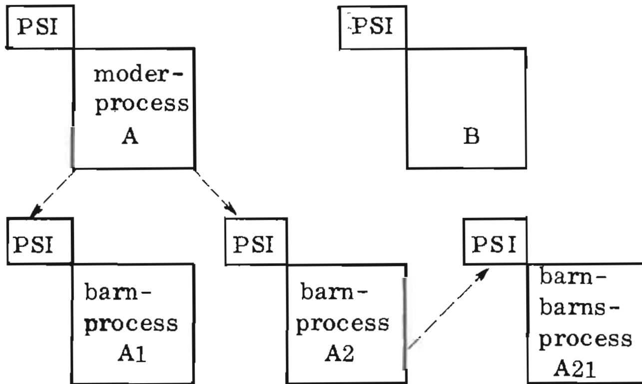
Figur 7.2:7 avser att med ett exempel illustrera detta förhållande. En process kan vara länkad till en eller flera köer. Den process som ligger främst i kön antas vara under service från resursen i fråga. Därför är t ex P7 i figuren under exekvering (X) medan P8 är dels bearbetningsklar (B) dels väntande på R1 och under service från R3.

Situationen växlar normalt snabbt och efter några mikrosekunder kan P7 ha exekverat färdigt. Då blir P7 i stället väntande på resursen R1 och en serie av styrprocesser blir initierade för att registrera och administrera den uppkomna tillståndsförändringen.

Vi har särskilt markerat kön till resursen processortid (R0). Denna kö administreras av den styrprogramfunktion som, som nämnts, på engelska vanligen kallas "dispatcher" och som vi försvenskat kallar "tidsplanerare". R0 bör anses som systemets mest kritiska resurs.

7.2.4 Processer - subprocesser - processinteraktion

Generellt kan en process under exekvering anmoda styrprogrammet att etablera och initiera en ny process. Denna kan vara helt "självständig" eller också kan den utgöra en subprocess (under-process) till den process som begärde etableringen (figur 7.2:8). Den process som initierade en subprocess, "moderprocessen", kan vara för sin egen fortsatta exekvering vara beroende på arbetet som utförs av "barnprocessen". T ex kan moderprocessen placera sig själv i väntestatus som ändras till bearbetningsklart när barnprocessen avslutat arbetet och signalerat detta.



Figur 7.2:8
A och B är två parallella processer. A har skapat ett antal barn- och barnbarnsprocesser.

En subprocess kommer vanligen att dela vissa resurser med moderprocessen. Moderprocessen har då i samband med etableringen att ange vilka av dess egna resurser som dess barnprocesser har tillgång till. Det kan exempelvis vara fråga om att barnprocessen får läsa/skriva på vissa filer som "tillhör" modern.

En process kan befinna sig i väntestatus på grund av att den väntar på en

eller flera uteliggande servicekrav, avslutning av subprocesser eller, generellt, att ett eller flera specificerade händelser skall inträffa. När händelserna så småningom inträffar en efter en undersöker styrprogrammet varje gång huruvida processen kan överföras till bearbetningsklart tillstånd. En central roll i detta sammanhang spelar processens PSI-block som bl a innehåller information om vilka händelser som processen väntar på. Process-subprocessinteraktion diskuteras även i kapitel 9.

7.2.5 Mekanismer för resursadministration

Det bör ha framgått att resursadministration bl a innebär hantering av köer till motsvarande resurser. Arbetet som en resursadministratör vanligen utför är

- anslutning av nya resurskrav till kön

- när service till en process avslutats, selektering av nästa element ur kön. Selekteringen kan vara prioritetsstyrd eller också kan andra principer tillämpas (first in - first out e d)

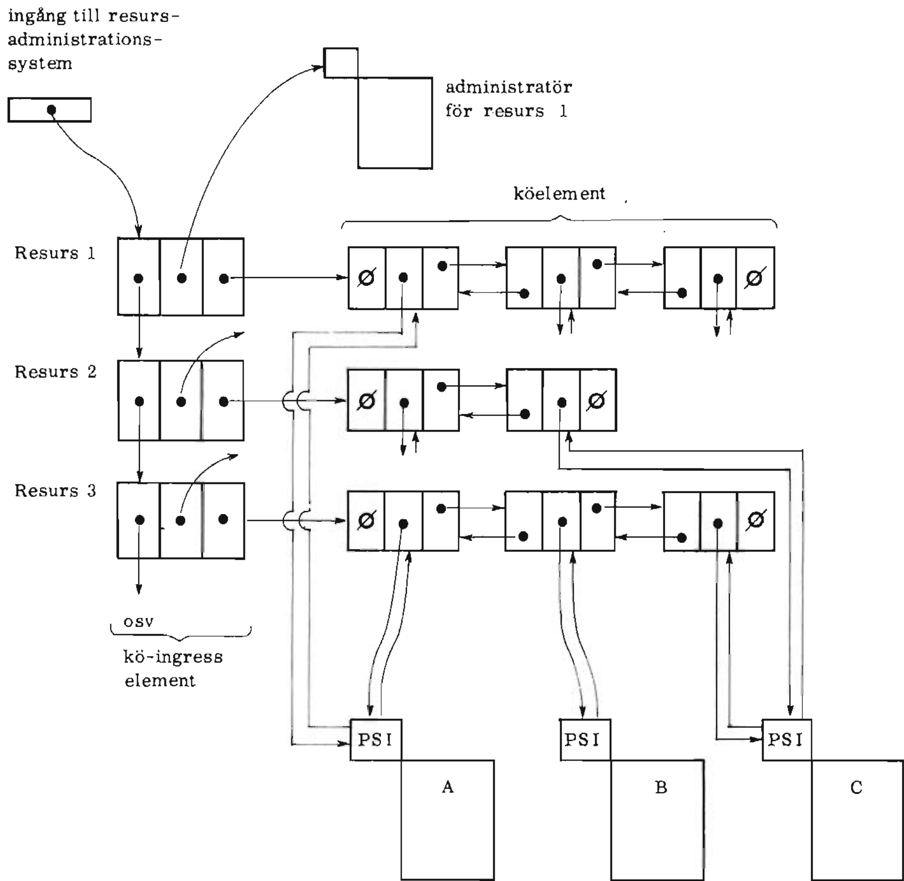
- undersökning huruvida den process som avslutades kan överföras till bearbetningsklart tillstånd, samt överföring av diverse annan information till processen i fråga (dess PSI).

Implementeringsprinciper avseende resurshantering är olika beroende på aktuell operativsystemkonstruktör. Normalt tillämpas någon form av länkade listor, t ex enligt figur 7.2:9.

PSI-processens styrinformation

PSI-blockets funktion har berörts tidigare i detta avsnitt. Vi skall här sammanfatta dess innehåll. Detta måste göras på en grov nivå ty detaljerna är starkt varierande beroende på operativsystemets implementeringsprinciper. Mängden av detaljuppgifter som ett PSI-block omfattar är vanligen stort. Exempelvis omfattar "Task Control"-blocket för IBM OS/360 MVT 190 bytes och innehåller ca 80 olika detaljuppgifter.

Varje process sades exekveras i sitt namnrum. Vidare kan sägas att en process är associerad med en mängd (för processen tillåtna) instruktioner. Processen får kommunicera med vissa bestämda in/utmatningsenheter och datafiler. Dessutom skall den ha möjlighet att kommunicera med andra processer. Processen representeras således av dels uppgifter om ovanstående sakförhållanden, möjligheter eller restriktioner, vilka samlats i PSI-blocket, dels av för processen erforderliga program och data.



Figur 7.2:9
En tänkbar principiell lösning för resursadministration. Köerna till varje resurs är länkade i två riktningar för att möjliggöra fram och baklängessökning. Observera att processerna A och C är medlemmar i två resursköer.

Ett PSI-blocks innehåll är i grova drag följande

- processens identifikation
- hänvisningar till för processen aktuella program
- information om processens prioritet(er)

- information för debitering och körningsstatistik

- information om vilka resurser som processen får bruka

- processens tillstånd

- avslutningsinformation (avslutningskod)

- utrymme för lagring av tillståndsdata vid avbrott (registerinnehåll, indikatorer och status för in/utmatningsenheter)

- hänvisningar till aktuella datafiler och buffertutrymmen

- information om medlemskap i köer till olika resurser och relationer till andra element i dessa köer

- information för att möjliggöra interaktion/samarbete med andra processer ("överordnade" såväl som subprocesser)

Kommunikation mellan processen och styrprogrammet samt mellan olika processer löses praktiskt genom ett utbyte av "meddelanden" eller "signaler" av olika slag. Meddelandena inordnas i olika mängder eller köer och behandlas av ansvarige mottagaren som sedan eventuellt signalerar tillbaka när begärd service utförts eller bestämd typ av händelse inträffat. Vanligen finns i ett operativsystem en mängd olika standardblock som är avsedda för dylik kommunikation. Blocken inordnas i olika köer (se t ex figur 7. 2:9) och utgör där länkar till anropande processer.

Vid de flesta sats- och kövis bearbetande system, är det förutsatt att en process' PSI-block ständigt befinner sig i primärminnet. Antalet parallella processer är här begränsat. Vid tidsdelnings- och reelltidssystem, där varje aktiv terminal sammanhör med minst en etablerad process, måste PSI-blocket och tillhörande buffertar vanligen lagras på sekundärminne under de tidsintervaller då terminalen är i "tänkestatus". Normalt är PSI-blocket ej explicit åtkomligt för det program, till vilket det hör.

Brytsignaler, systemtillstånd

De flesta operativsystem kan anses "brytsignal-styrda". Med detta avses att en process utför sitt arbete till dess den antingen frivilligt överlämnar kontrollen till styrprogrammet (genom att via en instruktion alstra en brytsignal) eller till dess att den avbryts av någon, av en annan händelse i systemet orsakad, brytsignal. En brytsignal överför kontrollen till ett delsystem i styrprogrammet - brytsignalsadministratören (BA). BA har

att avgöra orsaken till avbrottet och därefter överlämna kontrollen till lämplig systemrutin för åtgärd. Följande orsakskategorier av brytsignaler kan anges ((12) s. 182).

1. Maskinfel

2. Programfel (otillåten instruktion, icke tillåten operandadress o dyl)

3. Händelser vid datatransporter såsom avslutade eller avbrutna datatransporter

4. Styrprogramanrop från program i skyddstillstånd eller i beordringstillstånd som orsakar inhopp till BA

5. Externa enheter (operatörskonsoler, terminaler) önskar "kontakt" med datorsystemet

Följande systemtillstånd (ur driftssynvinkel) kan urskiljas

(S) Styrprogramtillstånd (supervisor state, master mode), vanligen innebärande beordringstillstånd ur skyddssynvinkel

(P) Problemtillstånd, vanligen under skyddstillstånd

(W) Väntetillstånd (wait-state) - ett logiskt tillstånd då intet arbete finns att utföra

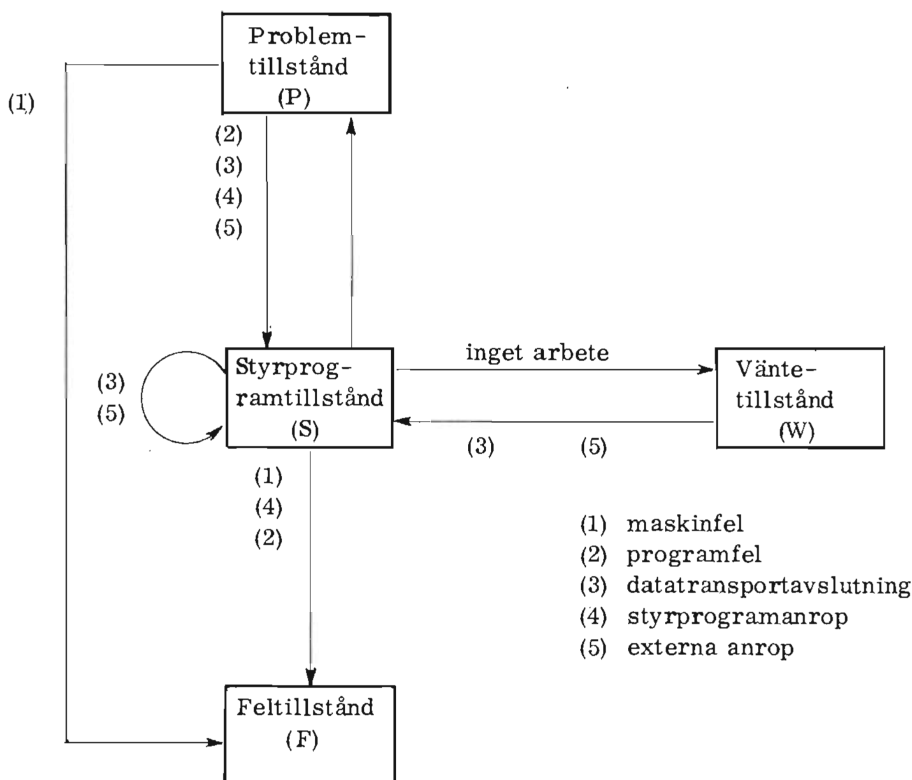
(F) Feltillstånd (orsakat av maskin- eller programfel)

Växlingen mellan dessa tillstånd är illustrerad i figur 7.2:10.

7.3 Principer för tilldelning av processortid

7.3.1 Problemställning

En process i exekveringstillstånd förbrukar resursen processortid. Antalet processer i ett datorsystem som samtidigt befinner sig i exekveringstillstånd bestäms bl a antalet processorer. Men, som vi noterat, för att en process skall kunna exekveras krävs även andra resurser, såsom program och data vilka i sin tur kräver resursen primärminne. Tilldelning av processortid kan därför ses som mer eller mindre intimt



Figur 7.2:10
Växlingar mellan systemtillstånd orsakad av brytsignaler. Siffrorna inom parentes anger brytsignalers orsakskategorier (se text).

förknippat med tilldelning av primärminne. Vid system med relativt statisk minnestilldelning (t ex IBM OS/360 MFT II) - dvs erforderligt (eller mer än erforderligt) primärminne tilldelas vid etablering av processen - är beroendet processortids- och primärminnestilldelning förhållandevis ringa. Vid system som arbetar med dynamisk minnestilldelning - t ex tidsdelningssystem med blockutbytesteknik (paging) är beroendet starkt och vi kan ofta ej behandla processortidstilldelning utan att i samband därmed även komma in på tilldelning av primärminne. Vid dylika system är ofta samma styrprogramfunktion ensam ansvarig för bägge resursernas fördelning.

Mängden av processer som är kandidater till resursen processortid ut-

göres av processer i bearbetningsklart tillstånd (B-tillstånd). Beslut om vilken eller vilka (vid multiprocessorsystem) processer som skall försättas i exekveringstillstånd (X-tillstånd) fattas av "dispatchern"-tidsplaneraren. Tidsplaneraren aktiveras vid två tillfällen:

1. när en ny process ansluts till mängden av processer i B-tillstånd

2. när en process i X-tillstånd avslutas - antingen "frivilligt" eller genom tvång (t ex dess beskärda kvantum av processortid har förbrukats)

Vid dessa tillfällen har tidsplaneraren uppgiften att besluta om vilken av processerna i B-tillstånd som skall ges processortid.

7.3.2 Mål för tidsplaneringen

Här, liksom i andra beslutssituationer, styrs beslutsalgoritmen för tilldelning av processortid mer eller mindre medvetet av

a) mål avseende systemets driftsegenskaper

b) systemets och processernas aktuella tillstånd

c) systemets och processernas tidigare beteende, under en historisk period.

Mål och operativsystem diskuteras allmänt i kapitel 10. Vi berör här enbart mål när det gäller tidsplanering och utformning av tidsplaneringsalgoritmen. Speciella mål och villkor som måste satisfieras givetvis upp för varje särskilt system. Två allmänt formulerade och generellt gällande önskemål kan dock anges

1. processorerna önskas utnyttjade i så hög grad som möjligt

2. den totala bearbetningstiden, inklusive väntetid, för de enskilda processerna skall hållas inom acceptabla gränser.

Önskemålet (2) kan ibland också ses som mängd av villkor där skilda toleransgränser kan anges för olika klasser av processer. Önskemålet (2) är oftast i konflikt med (1). Ett ökat utnyttjande av processorn kräver vanligen en hög grad av multiprogrammering. Denna i sin tur medför, bland annat p g a därmed nödvändigt administrationsarbete, en fördröjning för den individuella processen och därmed längre bearbetningstid.

Speciella krav och önskemål kan, som ovan nämnts, uppställas för varje särskilt system. Några exempel:

- kompileringsprocesser skall alltid ges förtur
- in/utmatningsbundna processer skall ges förtur
- maximal tillåten responstid för enkel konversation (styrsatser, programsatser) från terminal (tidsdelningssystem) är x sekunder
- reelltidsprocesser skall bryta alla processer av icke-reelltidskaraktär

osv.

Viktigt i sammanhanget är tidsplanerarens <u>eget behov</u> av processortid och andra resurser. Planeringen medför "overhead" (administrativ belastning) på systemet. Denna måste vara mindre än de effektivitetsvinster som tidsplaneringen medför. Vi bör också notera att komplicerade planeringsalgoritmer kan vara mer känsliga för exceptionella situationer och störningar, och eventuellt även vara svåra att få helt fria från fel.

Önskemål, mål och krav avseende tidsplaneringsalgoritmer diskuteras bl a i ref. (1) s. 140, (4), (6), (7) och (8).

Systems "godhetstal"

<u>Hellerman</u> (8) har föreslagit ett mått på tidsdelningssystems "godhetstal" (avseende systemets <u>responstid</u>) som har följande egenskaper:

1. det skall vara dimensionslöst
2. det skall kunna anta ett maximivärde för ett "idealsystem"
3. ett "bättre" system skall ge högre godhetstal. Ett bättre system antas här innebära att processer med små resurskrav ges snabbare service än större bearbetningar.

Om t_i är totaltiden för att bearbeta en process på en tom dator - dvs inga parallella processer - och T_i är processens totaltid ($t_i \leq T_i$) uppfyller följande faktor villkoren

$$\varphi = n / \sum_{i=1}^{n} (T_i / t_i)$$

där n är antalet processer i systemet som betraktas. φ beror således på n, t_i, datorn, tidsplaneraren och även andra systemprogram. Stora T-värden för små t minskar φ.[1] Enligt (8) kan en tidsplanerare fås att sträva efter högsta möjliga φ-värde. Påpekas bör att φ är ett relativt mått och kan ej användas för jämförelser mellan olika datorsystem.

Observera att systemets "throughput", som enligt (8) kan definieras som antal jobb per tidsenhet dvs $\rho = n/(\tau_n - \tau_0)$(där τ_n = tidpunkt då sista processen avslutades och τ_0 = ankomsttid för första processen) ej har egenskaperna (1) - (3) ovan utan är ett absolut mått som i bästa fall kan användas för jämförelser mellan olika system. Jämförelsen bör givetvis påverkas även av resp φ-värden.

7.3.3 Planeringsprinciper

Grundprincipen är att

> "den vid varje tidpunkt "viktigaste" processen i B- eller X-status skall tilldelas processortid"

Vad menas då med "vid varje tidpunkt" och vad avgör vilken process som är "viktigast?

Den beslutsregel som avgör en process' betydelse härvidlag baseras givetvis på de mål som uppställts (se föregående avsnitt). Vi kan också jämställa betydelsen med processens prioritet, π, som kan kvantifieras, och där en process med högsta prioritet har förtur i tidstilldelningskön.

Problemet har då återförts till att vid varje tidpunkt kunna avgöra eller beräkna en process' prioritet. Vi skall här betrakta några planeringsalgoritmer från utgångspunkten prioritet.

Prioritering

Vi kan allmänt skilja mellan tidsplaneringssystem som arbetar med fix respektive variabel prioritet.

---

1) De mindre processernas vikt kan accentueras genom att exempelvis ändra sambandet till $\varphi = n/\sum (T_i/t_i)^k$, $k \geq 1$. Ju större k desto större vikt lägges vid processer med $T_i >> t_i$ (förf:s anmärkning).

Om planeringsfunktionen ej ändrar en process' prioritet som funktion av dess tid i systemet[1] (och andra egenskaper hos processen som förändras med tiden) säges fix prioritet tillämpas. Denna princip är vanlig framför allt vid sats- och kövis arbetande system. Exempelvis arbetar IBM OS/360 MFT II (se avsnitt 5.4.1), ICL's George 1 och 2, DataSAAB's MK Dirigent, m fl enligt denna princip.

Om planeringsfunktionen ändrar en process' prioritet som funktion av bl a dess tid i systemet[1] säger vi att variabel prioritet tillämpas. Denna teknik tillämpas framför allt vid tidsdelningssystem och en av huvudorsakerna där är att man önskar uppnå vissa service-mål (t ex att en process ej får vänta för länge på exekvering). Tekniken tillämpas även hos vissa sats- och kövis arbetande system och avsikten här är ofta en strävan att öka systemets "throughput".

Många faktorer tas hänsyn till vid beräkning av en process' prioritet (relativt andra processer). Här anges några:

1. initialprioritet (dvs den som tilldelats vid etablering)
2. resurskrav av typen primärminne, sekundärminne
3. uppskattat behov av processortid
4. " " " datatransporttid
5. tidsgräns för resultat (dead-line) eller för initiering av processen
6. tid sedan processen etablerades
7. andelen redan utfört arbete i förhållande till uppskattad arbetsmängd
8. andelen arbete som återstår (uppskattning)
9. tid sedan processen sist exekverades
10. processens beteende (mätning av vissa karaktäristiska tillståndsvariabler) under en historisk period

m fl.

Vi kan definiera en process' prioritet som en funktion av en eller flera av ovanstående faktorer. Då dessa ändras med tiden ändras processens prioritet.

Vi återvänder nu till problemet att vid "varje tidpunkt" bestämma prioriteten. Beteckna processens P_i prioritet med $\pi_i(t)$ och antag att P_i exekveras vid tiden t_0.

---

1) Med detta avses "i etablerat (E) tillstånd", jfr figur 7.2:6.

Vid t_0 ändras emellertid prioriteten för en annan process Pj så att $\pi_j(t_0) > \pi_i(t_0)$. Ändringen kan exempelvis bero på att faktorn (9) ovan, tiden sedan P_j sist fick exekvera, blivit dominerande och medfört prioritetsändringen.

Om detta förhållande, $\pi_j(t_0) > \pi_i(t_0)$, på något sätt kan signaleras, kan P_j tilldelas processortid. Signaleringen är möjlig om P_j's prioritetsökning beror på någon händelse i systemet som orsakat en brytsignal. I fallet med faktorn (9) sker ingen signalering utan P_i behåller processorn till dess att en brytsignal inträffar.

För att motarbeta att en process på detta sätt konsumerar processortid och hindrar annan aktivitet i datorn, trots att processer med högre prioritet existerar, har flera system infört intervallklockor vilkas uppgift är att vid bestämda tidpunkter alstra brytsignaler och därmed inhopp till tidsplaneringssystemet, för tilldelning åt mest behövande process.

7.3.4 Planeringsalgoritmer

Först-in-först-ut principen innebär att prioriteten är en funktion av tiden för en process i exekverande eller bearbetningsklart tillstånd.

Cirkulär kö (eller köer) är vanlig vid tidsdelningssystem. Man arbetar här eventuellt med flera köer i olika prioritets-klasser och därmed olika servicegrad. Processer i en kö tilldelas ett visst kvantum processortid, som vanligen skiljer sig från kö till kö. Köer i lägre prioritetsklasser har normalt längre tidskvanta. I dessa placeras processer vars resurskrav förutom responstid är större men vars krav på responstid är mindre (vanligen s k "produktionsjobb"). Processer kan flyttas mellan de olika köerna. Om en process A t ex visar sig behöva mycket processortid kan detta registreras och nästa gång samma klass av processer initieras överflyttas A till en kö med lägre prioritet. Aktuella beslut i sådana situationer är beroende av de mål som uppställts för systemet. Cirkulära köer innebär att en process' prioritet i kön endast är en funktion av antal tidskvanta sedan den sist fick en tilldelning.

En viktig tidsplaneringsparameter är ett kvantums storlek. Den kan för en cirkulär kö vara fix eller variabel dvs en funktion av egenskaper hos den enskilda processen, samtliga processer eller datorsystemet i övrigt.

Ett enkelt men illustrativt exempel hämtar vi från (1) (s. 142). Antag att vi har 5 processer som etableras samtidigt och som var och en kräver 1 sekunds processortid. Om ett kvantum på 0,1 sek tillämpas kan den genomsnittliga "responstiden" beräknas till 4.8 sekunder, minsta res-

ponstiden till 4.6 sek och maximala responstiden till 5.0 sek. Tidsplaneraren får ingripa 50 gånger och eventuellt flytta data ut och in. Detta senare har dock ej tagits hänsyn till vid beräkningen ovan.

Om kvantumtiden ändras till 1 sek blir den genomsnittliga responstiden 3 sek (min 1 sek, max 5 sek). Tidsplaneraren aktiveras här endast vid 5 tillfällen.

Även om exemplet ej kan betecknas som realistiskt då

- flera processer sällan etableras samtidigt
- processers tidsbehov varierar starkt

bör det ge en god inledande belysning av problemet att välja kvantumtidens längd, och dess konsekvenser.

En slutsats kan här sannolikt dras. När antalet processer i systemet är litet och deras resursbehov måttliga förefaller det lämpligt att temporärt öka kvantumtiden för att öka systemets effektivitet. Detta problem diskuteras även av Mullery och Driscoll (6) vilkas metod är beskriven längre fram i detta avsnitt.

Ett 4-nivås kösystem för tilldelning av processortid beskrivs av Watson (ref (1), s. 148). Mekanismen tillämpas för ett XDS-940 tidsdelningssystem (Xerox Data Systems). Nivå-1 kön används för in/utmatningsprocesser för kommunikation från/till terminalskrivmaskiner. Denna kö har högsta prioritet då en "omedelbar" respons alltid förväntas vid inmatning av tecken från en terminal. Kön på nivå-2 används för processer som svarar för datatransporter till/från sekundärminnen. Nivå-3 är kön för tilldelning av korta tidskvanta och nivå-4 används för länga tidskvanta. Processer kan under vissa förhållanden växla mellan nivå-3 och nivå-4.

Vi skall avsluta detta avsnitt om styrprograms tidsplanering genom att från den rikhaltiga floran av olika publicerade metoder välja ut två algoritmer av vilka den första (7) är i huvudsak avsedd för sats- och kövis bearbetning och den andra (6) i huvudsak för tidsdelningssystem. Det bör betonas att algoritmerna har valts på grund av att de demonstrerar vissa intressanta ideer och icke därför att de visat sig överlägsna andra metoder.

Heuristisk metod

Algoritmen (enligt Ryder (7)) är avsedd att arbeta på hela mängden av

processer i aktivt tillstånd eller en delmängd därav. Kalla den betraktade mängden för M. Man skiljer här på processorbundna (PE) och datatransportbundna (DT) processer. Graden av överlappning och resursutnyttjande antas öka om DT-processer ges företräde vid tidstilldelning och därför är detta det mål som algoritmen eftersträvar.

Av denna orsak är det därför nödvändigt att avgöra huruvida en process från tid till tid kan karaktäriseras som PE- eller DT-bunden. Därvid tas hänsyn till processens uppförande (historik) under en viss period (T) tillbaka i tiden från beslutstillfället.

Mängden M indelas i två fullständigt ordnade delmängder (köer) M_{PE} och M_{DT}. Processer i B-tillstånd i mängden M_{DT} har förtur till processortid. Prioriteten inom M_{DT} avgörs av processens ordningsnummer. En process' medlemsskap och ordningsnummer i någon av dessa mängder kan ändras på basis av processens beteende under tidsintervallet. Villkoren för överföring av processer mellan M_{DT} och M_{PE} är följande. En process i M_{DT} överförs till M_{PE} om processen utnyttjat processortid under hela intervallet T. Processen ansluts som första element i M_{PE}-kön. En process i M_{PE} flyttas till M_{DT} och blir där sista elementet om processen under perioden T frivilligt placerat sig i väntande tillstånd. En process' läge i M_{PE} kan ändras. Om processen utnyttjar processortid under hela intervallet T flyttas den och blir sista elementet i M_{PE}. Samma sak inträffar med processen om den avbryts av en annan process med högre prioritet. Den cykliska förflyttningen i M_{PE}-mängden tillförsäkrar samtliga processer viss "jämlikhet".

Algoritmen arbetar med sex parametrar. Arbetet börjar med att tidsintervallet (kvantum) sätts till ett värde $T = T_0$. T kan sedan ökas eller minskas med inkrementet ΔT, dock så att $T_{min} \leq T \leq T_{max}$, där T_{min} och T_{max} är fasta gränsvärden. Under ett statistikintervall $T_s \gg T$ insamlas vissa data om systemets beteende som sedan bl a ligger till grund för bedömning av huruvida T behöver ändras. Av intresse är bl a förhållandet Q mellan antal gånger som processerna utnyttjade processorn hela tidsintervallet T och totala antalet tidstilldelningsbeslutssituationer.

Om Q för ett statistikintervall är tillräckligt lågt och lägre än för närmast föregående period minskas T med ΔT för att ge algoritmen en bättre chans att upptäcka PE-processer. Det omvända förhållandet gäller om Q är för högt - dvs T ökas med ΔT. En "balanserad" fördelning mellan M_{PE} och M_{DT} eftersträvas.

Lämpliga värden för nämnda parametrar och variabler varierar från system till system, beroende på kriterium för en "god planering".

I (8) redovisas experimentresultat från tillämpning av algoritmen vid körning av FORTRAN- och COBOL-jobb på IBM's system 360 modell 50, 65 och 195. En minskning av totala körtiden med omkring 10 procent kan noteras för dessa system. Mängden M omfattade vanligen 3-4 processer. Värden för de parametrar som användes för 360/65 experiment var

$T_0 = 150$ ms

$\Delta T = 5$ ms

$T_{min} = 50$ ms

$T_{max} = 500$ ms

$Q \approx 0.5$

$T_s = 1000$ ms

Författarens (8) kommentar är att sannolikt föga vinnes vid homogena belastningar, dvs om enbart PE- eller DT-processer förekommer.

En "overhead"-reducerande metod för tidsdelningssystem

Mullery och Driscoll har presenterat en metod som nedan beskrivs (6). Författarna (6) börjar med en diskussion om möjliga åtgärder för att hos tidsdelningssystem ge tillfredsställande service (responstid) till de processer som befinner sig i aktivt tillstånd. Bland annat redovisas förslagen att

1. nya användare kan (under kortare perioder) vägras inträda i systemet

eller

2. man kan avsiktligt förlänga responstiden för vissa användare för att minska antalet nya terminalinmatningar per tidsenhet.

Anledningen till det senare förslaget är en önskan att hålla antalet växlingar mellan olika processer så lågt som möjligt. Därigenom kan systemets "overhead" hållas på en acceptabelt låg nivå.

"Overhead" problemen hos tidsdelningssystem är ofta stora. Varje processväxling kräver åtskilligt administrativt "städarbete", och även datatransporter till/från sekundärminnet. Det förefaller sålunda "vettigt" att ej i onödan avbryta en process efter det att dess tidskvantum förbrukats. Vad som kan anses vara onödigt är givetvis an avvägningsfråga. Det synes dock försvarbart att låta en process fortsätta ytterligare ett tids-

kvantum om, säg, endast en eller två mindre responstidskrävande eller mindre "viktiga" processer befinner sig i kvantumkön vid den aktuella tidpunkten.

Grundtanken i algoritmen (6) är följande:

"varje aktiv process skall låtas exekveras antingen till dess att den avslutar sig själv (normal avslutning) <u>eller</u> till dess att den måste avbrytas för att garantera tillfredsställande service åt övriga väntande processer i kvantumkön."

Låt t beteckna det minsta kvantum sammanhängande processortid vi vill garantera en process. Låt ω beteckna den längsta tid vi vill låta en process vänta på att börja exekveras. Antag att systemet är ett en-processorsystem.

Om under intervallet (0, t) en process A kräver service och inga andra processer är aktiva i systemet, tilldelas A omedelbart processortid (eller rättare sagt "processorn"). Om strax därefter under samma intervall process B kräver processortid inträffar följande. B planeras in på "planeringstidsaxeln" ω/t enheter framåt. Om A ej skulle vara avslutad vid ω/t kommer A då att avbrytas för att garantera B en start inom ω tidsenheter.

Om A avslutas före ω/t kommer B därvid omedelbart att tilldelas processortid.

Antag nu att ytterligare en process C anländer under intervallet (0, t). A är under exekvering och B väntar. För att ej överskrida C's krav på start inom ω tidsenheter måste nu B flyttas upp till $(\omega - t)/t$ och C tar B's plats. B får sålunda snabbare service efter flyttningen. På detta sätt sker enligt (6) tilldelning av processortid, och varje process som avbryts betraktas som ett nytt krav och placeras längst bak på listan. Det kan inses att metoden torde spara processväxlingar om det genomsnittliga antalet processer i systemet är mindre än ω/t. Vid hög belastning på tidsdelningssystemet torde denna metods effektivitet ej bli så uttalad.

För en mer detaljerad beskrivning av metoden hänvisas till ref. (6), där även fallet med multipla processorer diskuteras.

7.4 Principer för administration av primärminne

7.4.1 Problemställning

Primärminnesadministrationsproblemet[1] kan grovt sett uppdelas i två delproblem.

1. att vid varje tidpunkt hålla kontroll över tilldelat minne till de olika pågående processerna i systemet och hur element i deras logiska adressrum motsvaras av element i datorns fysiska adressrum (dvs avbildningsfunktionen f_i för varje etablerad process P_i).

2. att besluta om ytterligare tilldelning av minne till minnesbehövande processer och att besluta hur minnesutrymmet skall disponeras av dessa.

Hos många datorsystem är minnestilldelningsstrategin mer eller mindre intimt förknippad med strategin för tilldelning av processortid. Detta samband är, som påpekats i 7.3, särskilt accentuerat vid tidsdelningssystem. Därför är även de mål man kan uppställa för minnestilldelningen i överensstämmelse med målen för tilldelning av processortid (se 7.3).

På grund av sambandet med tilldelning av processortid, och även på grund av att minnet (ännu så länge) är en av ett datorsystems mest dyrbara resurser (och därför alltid av begränsad storlek), måste minnestilldelningsproblemet anses som fundamentalt i samband med operativsystem. Endast en relativt ytlig behandling av problemet presenteras i denna bok.

Metoderna att administrera primärminne kan klassificeras med avseende på två faktorer.

Den första faktorn skiljer mellan metoderna med avseende på huruvida sammanhängande eller blockvis (fragmenterat) utrymme tilldelas en process. Faktorn avspeglar sålunda minnets _dispositionsprincip_.

Den andra faktorn speglar dynamiken i _tilldelningen-tilldelningsprincipen._ Vi kan här tala om dels _statisk_ dels _dynamisk tilldelning_. Vid statisk tilldelning erhåller en process ett för _hela_ dess exekvering tillräckligt stort minnesutrymme vid exekveringens _början_. Vid dynamisk tilldelning kan processen starta med minsta möjliga utrymme och sedan efter behov be-

---

1) För enkelhets skull kallas i fortsättningen av detta kapitalavsnitt _primärminne_ för _minne_.

gära och tilldelas ytterligare utrymme, eller återlämna ej längre erforderligt utrymme. Krav på ytterligare minne resp återlämnande av icke längre erforderligt minne planeras och administreras i vissa fall av processprogrammet, dvs kraven är programstyrda. Minnestilldelningen kan också vara helt automatiserad i den meningen att programmeraren ej behöver bekymra sig för hur avbildningen mellan det logiska adressrummet och det fysiska adressrummet går till, och där det logiska adressrummet även kan vara större än tillgängligt primärminne. Den senare principen kallas för "virtuellt minne" och är av stort intresse ty principen kan förväntas bli vanlig på de flesta framtida datorer. Virtuell minnesteknik är normalt knuten till blockvis fragmenterad utrymmestilldelning och blockutbytesteknik (paging, swapping). Planeringsproblem i samband med denna teknik kommer att belysas i detta avsnitt.

<table>
<tr><td colspan="2" rowspan="2">tilldelningsprincip
(faktor 2)</td><td colspan="2">dispositionsprincip
(faktor 1)</td></tr>
<tr><td>samman-
hängande
utrymme</td><td>blockvis
fragment-
terat ut-
rymme</td></tr>
<tr><td colspan="2">statisk tilldelning</td><td>A</td><td>B</td></tr>
<tr><td rowspan="2">dynamisk
tilldel-
ning</td><td>program-
styrd</td><td>C</td><td>D</td></tr>
<tr><td>automa-
tisk</td><td>E</td><td>F</td></tr>
</table>

Figur 7.4:1
Olika metoder för administration av minne. T ex innebär metod A att en process erhåller (vid etablering) ett sammanhängande minnesutrymme som sedan behålles under processens hela existenstid i systemet. Minnesutnyttjandeproblem i samband med olika administrationsmetoder diskuteras längre fram i detta avsnitt. (Bokstäverna A, B, ..., F ovan är endast avsedda som beteckningar för resp principer.)

En generell betraktelse över minnesadministrationsproblemet

Följande betraktelse är delvis baserad på en artikel av Denning (2). Vi bortser här från skillnader mellan det logiska adressrummet och det symboliska namnrummet (behandlat i avsnitt 7.2) och betraktar enbart förhållandet mellan

- det fysiska adressrummet $M = \{0, 1, 2, \ldots, m-1\}$ och

- det "generella" namnrummet $N = \{n_1, n_2, \ldots, n_i, \ldots\}$

där n_i är namn på informationselement i en process' namnrum. Ett linjärt logiskt namnrum kan ses som ett specialfall av N där $n_1 = 0$, $n_2 = 1$ osv $n_i = i$. För ett segmenterat (styckvis) linjärt, logiskt adressrum motsvaras n_i av talparet (a, b) där a är segmentnumret och b adressen inom segmentet. Generellet kan N betraktas som en mängd identifierare som en process nyttjar för identifiering av sina informationselement av olika slag (lägen i programmet och element i datastrukturer m m).

En process exekveras, ur programmerarens synvinkel, i det generella namnrummet. Vid referens måste element i N vara associerade med element i M. Om detta göres under exekvering krävs, i samband med denna, en mer eller mindre komplicerad namn- eller adresstransformation.

Adresstransformationen[1] $f_i : N_i \to M \cup \{\Omega\}$ for processen P_i med namnrummet N_i definieras som

$$f_i(n_{ij}) = \begin{cases} a_i & \text{om elementet } n_{ij} \text{ finns i } M \text{ vid adressen } a_i \\ \Omega & \text{om } n_{ij} \text{ ej finns i } M \end{cases}$$

där n_{ij} ($j = 1, 2, \ldots$) är processens P_i informationselement. Ω indikerar att en styrprogramfunktion aktiveras för att motsvarande element skall beredas plats och inläsas till M.

Antag att det existerar två parallella processer P_1 och P_2 associerade med namnrum N_1 resp N_2. Vid varje tidpunkt gäller

$$f_1 : N_1 \to M \cup \{\Omega\}$$

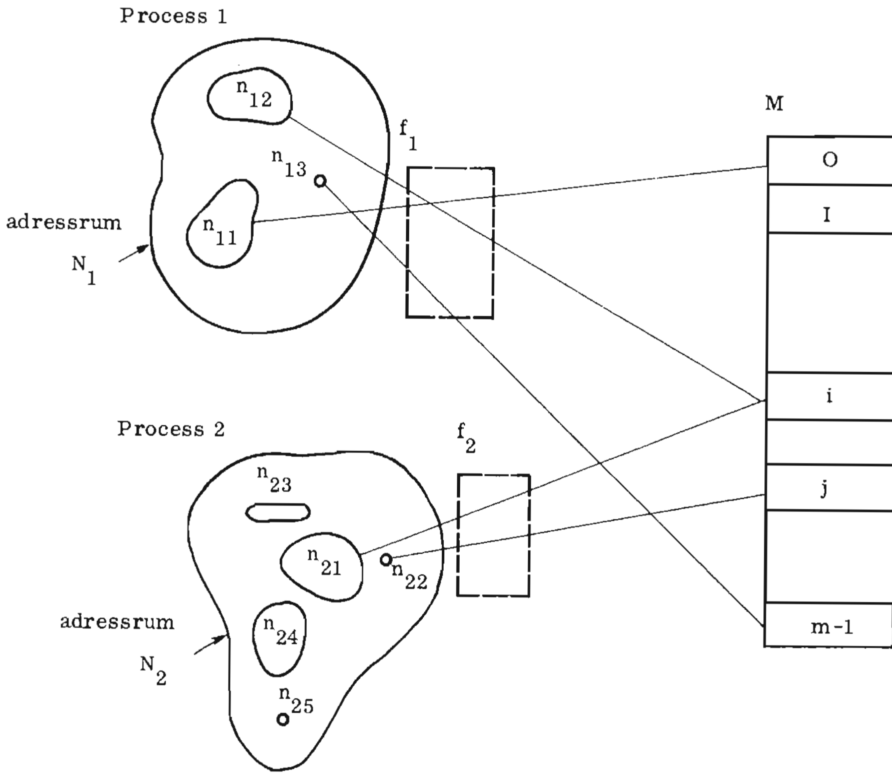
$$f_2 : N_2 \to M \cup \{\Omega\}$$

så att

$$f_1(n_{1j}) = \begin{cases} a_1 & \text{om } n_{1j} \text{ finns i } M \text{ vid adressen } a_1 \\ \Omega & \text{om } n_{1j} \text{ ej finns i } M \end{cases}$$

---

1) $\to$ utläses "avbildas på"



Figur 7.4:2

Adresstransformation mellan två processers namnrum, N_1 och N_2, och det fysiska adressrummet M.

$$f_2(n_{2j}) = \begin{cases} a_2 & \text{om } n_{2j} \text{ finns i M vid adressen } a_2 \\ \Omega & \text{om } n_{2j} \text{ ej finns i M} \end{cases}$$

Här kan konstateras att om $f_1(N_1) \cap f_2(N_2) \neq \emptyset$ så delar P_1 och P_2 ett eller flera i M befintliga element. Om element i N_i alltid avbildas på samma adresser i M är avbildningsfunktionen f_i konstant. Vid datorer som tillämpar blockutbytesteknik varierar f_i med tiden. För tidpunkterna $t_1 \neq t_2$ är generellt $f_{i,t1}(n_{ij}) \neq f_{1,t2}(n_{ij})$, dvs vid tidpunkterna t_1 resp t_2 är n_{ij} avbildade på olika fysiska adresser.

En avbildningsmekanism av detta slag kan bli komplicerad och dyrbar beroende på namnrummets struktur. För linjära eller styckvis linjära (segmenterade) adressrum finns idag maskinvarumässiga avbildningsmekanismer. Det gäller här att översätta ett numeriskt begrepp (logisk adress) till en fysisk adress, eller konstatera att mot adressen svarande element först måste inläsas från sekundärminne. För att klara symboliska namnrum med mer komplicerad struktur måste idag programmerade mekanismer tillgripas. Dylika system är ännu så länge ovanliga.

Trenden mot databaser och separering av program och databeskrivningar (se kapitel 8) kommer sannolikt att aktualisera filosofin med virtuella processer som har hela eller delar av mängden identifierare i databasen som sitt namnrum.

Låt oss nu betrakta fallet när $f_i(n_{ij}) = \Omega$, dvs när elementet n_{ij} i N_i ej finns lagrat i det fysiska adressrummet M. Här kan två situationer uppkomma:

1. M innehåller tillräckligt stort ledigt utrymme och, på bekostnad av transporttid, kan n_{ij} inläsas från sekundärminne och enligt någon placeringsstrategi inlagras i M.

2. M är "fullt" eller innehåller ej tillräckligt stort ledigt utrymme. För att berede plats för n_{ij} måste något element i M återlagras till sekundärminne (sparas) eller också får n_{ij} vänta till dess erforderligt utrymme återlämnats av andra pågående processer. Här aktualiseras minnesadministrationens algoritmer för hämtning av erforderliga element från sekundärminne och ersättning av befintliga element i M med nya element (s k "fetch" resp "replacement" strategier).

När det gäller algoritmer eller strategier för inläsning av nya element till M kan man skilja på i förväg planerad (prognosstyrd) och behovsstyrd (kravstyrd) inläsning. Behovsstyrd inläsning initieras av att $f_i(n_{ij}) = \Omega$ inträffar. Den i förväg planerade inläsningen arbetar, som namnet säger, med principen att i god tid förutsäga behovet av att vissa element i N_i skall behövas lagrade i M. Den som bäst kan bedöma detta är givetvis programmeraren. Vid automatisk administration är prognosticering av detta slag normalt vansklig då referens till olika element i N_i i hög grad beror på datastyrda, villkorliga förgreningar i programmet.

Processens beteendemönster

En process' beteende är beroende av de program och data som den är

associerad med. En process' P_i beteendemönster kan beskrivas på ett maskinoberoende sätt genom dess referenssträng (2)

$$\omega_i = r_1 r_2 r_3 \ldots r_k \ldots, \quad r_k \in N_i, \quad k = 1, 2, \ldots$$

dvs där r_k är processens referenser till element i processens namnrum N_i. En delmängd $\eta_i \subset N_i$, sådan att samtliga element i η_i kan rymmas i det fysiska adressrummet M, kallas för ett möjligt tilldelningstillstånd. Referenssträng-begreppet kommer att tillämpas längre fram i detta avsnitt.

7.4.2 Grundläggande principer för disposition och adressering av primärminne

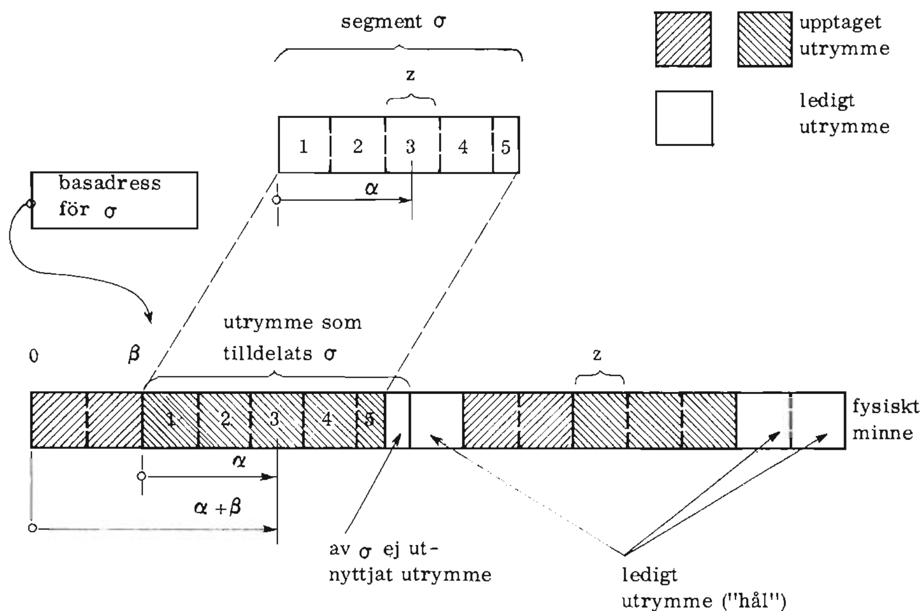
Ett programsegment σ, i ett linjärt adressrum bestående av ett antal programelement och dataareor uppfattas av programmeraren vanligen som ett sammanhängande utrymme av l minnesceller med logiska adresserna $0 \leq \alpha \leq (l-1)$. När primärminne skall tilldelas ett dylikt segment kan det ur minnesutnyttjandesynvinkel göras på två principiellt olika sätt:

1. σ tilldelas ett sammanhängande minnesutrymme som är minst av storleken l, vanligen ett antal, säg λ, sammanhängande moduler om av fast längd (z) så att

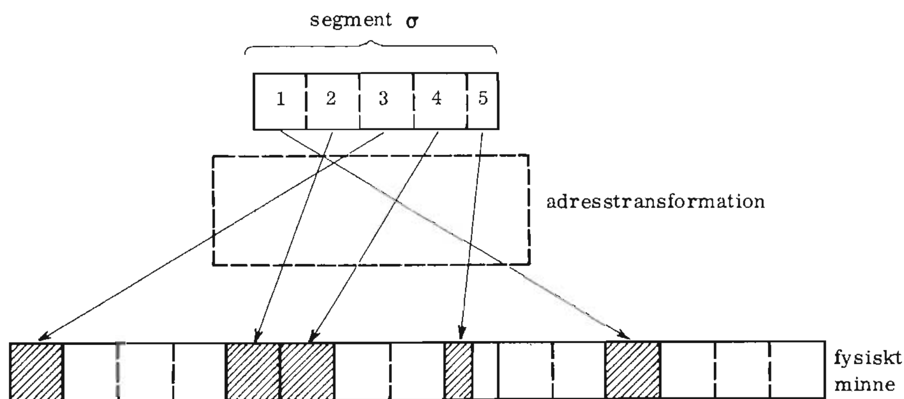
$$(\lambda - 1) \cdot z < l \leq \lambda \cdot z$$

2. σ tilldelas ett antal (λ) minnesmoduler enligt ovan dock utan kravet att de skall vara sammanhängande eller ligga i någon bestämd ordning i minnet.

I fallet (1) är adresseringen av informationselementen i segmentet σ enkel (figur 7.4:3). Om ett element i σ har det relativa ordningsnumret (adressen) α får adressen för dess fysiska minnesplats genom att till α addera adressen för det tilldelade utrymmets första cell - basadressen (β). I fallet (2) måste en mer komplicerad omvandling av den logiska adressen äga rum (figur 7.4:4). Det logiska adressrummet för segmentet uppfattas som sammanhängande vilket kallas för "artificiellt granskap" (eng. "artificial contiguity").

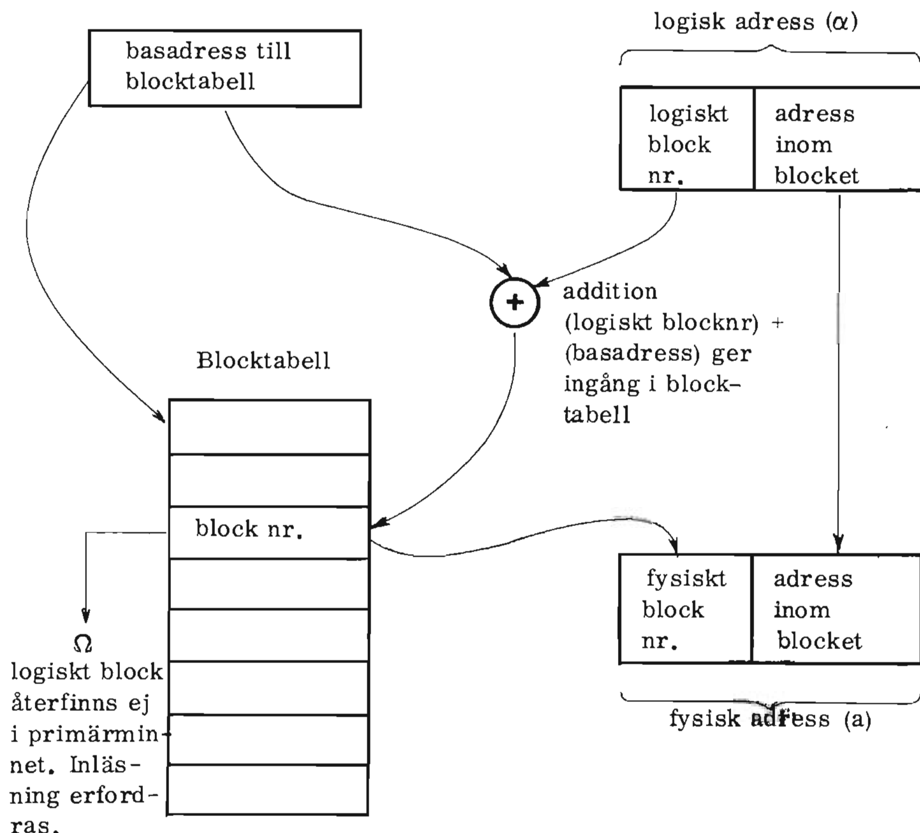


Figur 7.4:3
Tilldelning av sammanhängande minnesutrymme.



Figur 7.4:4
Tilldelning av fragmenterat minnesutrymme.

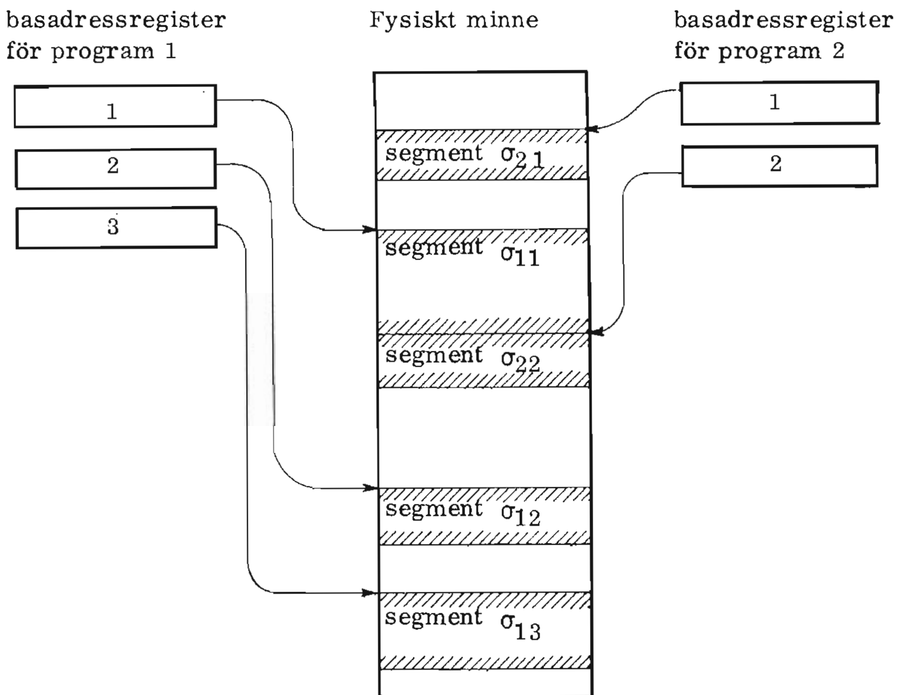
Den vanligaste metoden att i fallet (2) omvandla en logisk adress α till motsvarande fysiska adress a är med hjälp av en s k "blocktabell" (figur 7.4:5). Det linjära adressrummet indelas i ett antal angränsande block (pages) av längden z. Den logiska adressen får då utformningen $\alpha = \langle \alpha_1, \alpha_2 \rangle$ där α_1 anger blocknumret och α_2 adressen inom blocket, $0 \leq \alpha_2 \leq (z - 1)$. Adresseringen är sådan att om t ex den logiska adressen $\langle 3, (z-1) \rangle$ ökas med 1 fås den logiska adressen $\langle 4, 0 \rangle$.



Figur 7.4:5
Omvandling av en logisk adress α i ett linjärt adressrum till fysisk adress a. Om α ej befinner sig i primärminnet sker uthopp till minnes-administrationsalgoritm. Varje linjärt adressrum har en egen blocktabell. Denna princip kallas vanligen för en-stegs "paging".

Vi har hittils diskuterat principen där minnet antogs vara indelat i ett antal block av fast längd (z). Denna princip är den vanligaste, men det förekommer system där, för att minska spillet,[1] två olika blockstorlekar (z_1, z_2) förekommer i syfte att öka minnesutnyttjandet. Administrationen blir här mer besvärlig.

Ett program består ofta av flera segment vilka har logiskt avgränsade arbetsuppgifter. Till exempel kan ett segment innehålla det "styrande" programmet och andra segment innehålla dataareor, buffertareor och olika delprogram (subrutiner). På grund av den funktionella avgränsningen kan de uppfattas som en mängd av linjära adressrum och behöver ej lagras samtliga i ett sammanhängande minnesutrymme.



Figur 7.4:6
Ett program kan bestå av flera segment vilka kan tilldelas skilda, icke sammanhängande utrymmen i det fysiska minnet. Referenser mellan programsegmenten ombesörjes av att man även anger vilket basadressregister som skall modifiera adressen inom resp segment. Flyttning av ett segment i minnet kräver att resp basadressregisters innehåll uppdateras.

1) = outnyttjat minnesutrymme i ett block.

Figur 7.4:6 visar tilldelning av primärminne till två segmenterade program där kravet på lagring i sammanhängande utrymme gäller för varje segment.

Om vi vill tillämpa en blockvis fragmenterad lagring (fall (2)) och även behålla ett segmenterat logiskt adressrum kräver adresstransformationen av en logisk till en fysisk adress något mer arbete. En blocktabell krävs då för varje segment och för hela adressrummet krävs en segmenttabell. En logisk adress utgöres här av komponenterna $\alpha = \langle \alpha_1, \alpha_2, \alpha_3 \rangle$ där

α_1 = segmentnummer

α_2 = blocknummer

α_3 = adress inom block

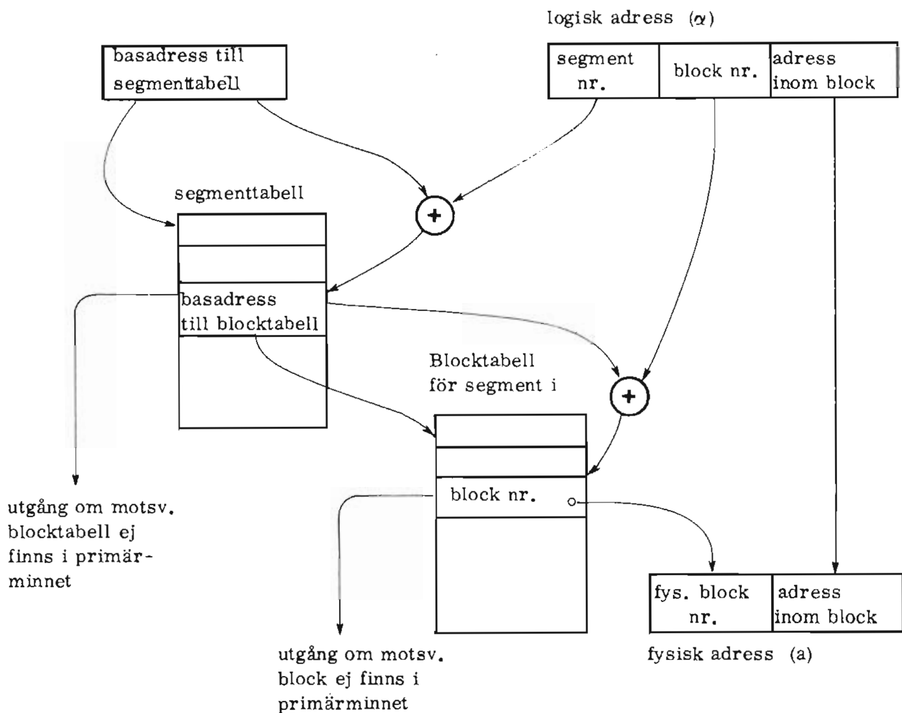
Adressomvandlingen här innebär att den fysiska adressen $a = \langle \beta, \alpha_3 \rangle$ skall beräknas där β är en funktion av α_1 och α_2 och anger numret på ett block i det fysiska adressrummet (figur 7.4:7).

Bokföring av minnesdispositionen

Vi har hittills diskuterat problemet att avbilda adresser i en process' logiska adressrum till fysiska adresser och konstaterat att detta ofta görs med hjälp av en eller två slag av tabeller (segment- och blocktabeller). För styrprogrammets minnesadministrationsfunktion är det emellertid lika viktigt att veta vilken eller vilka processer som har blivit tilldelade ett visst fysiskt minnesutrymme. Likaså behöver minnesadministrationen veta vilka delar av minnet som är lediga och möjliga att tilldela till andra processer.

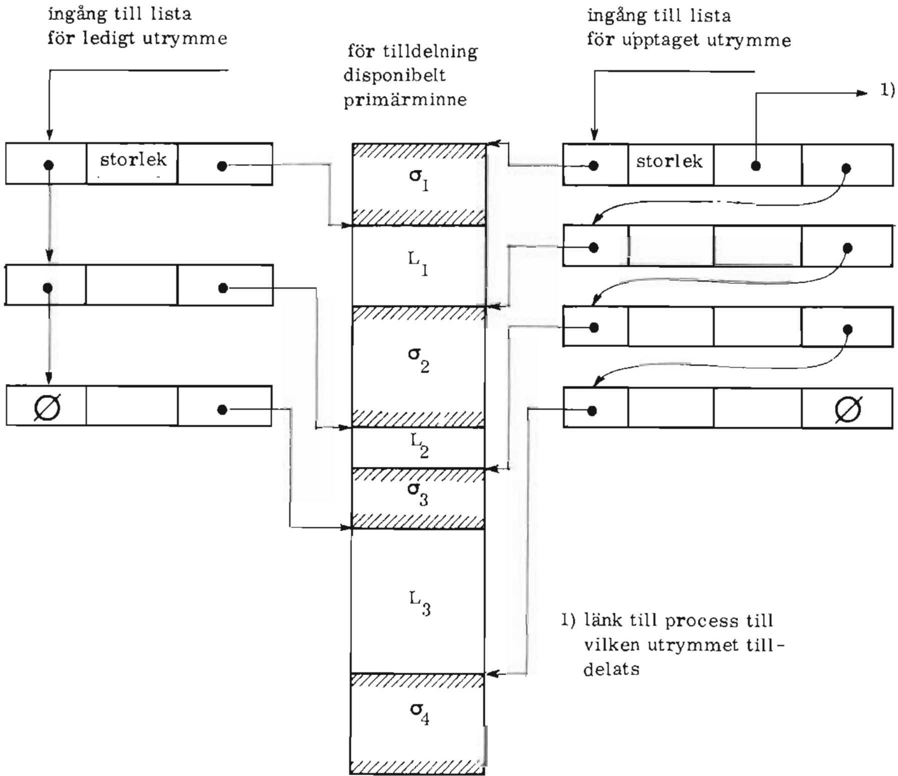
Denna bokföring kan skötas med hjälp av en eller flera tabeller. Olika metoder förekommer beroende på tillämpad minnesdispositionsprincip. Vi beskriver nedan principiella lösningar för de två vanliga dispositionsprinciperna (figur 7.4:1).

Om minne tilldelas i form av segment av variabel längd (t ex enligt figur 7.4:6) erfordras dels en lista för att hålla reda på alla tilldelningar, och dels en lista för att hålla reda på alla lediga utrymmen. Alltefter som processer startas eller avslutas förändras minnesbilden. Ett flexibelt sätt att hantera tabeller vars omfattning växer resp krymper är en representation av dessa i form av länkade listor. Ett möjligt förfarande är visat i figur 7.4:8. När ett segment återlämnas till minnesadministrationen kan ibland ett större sammanhängande utrymme skapas.



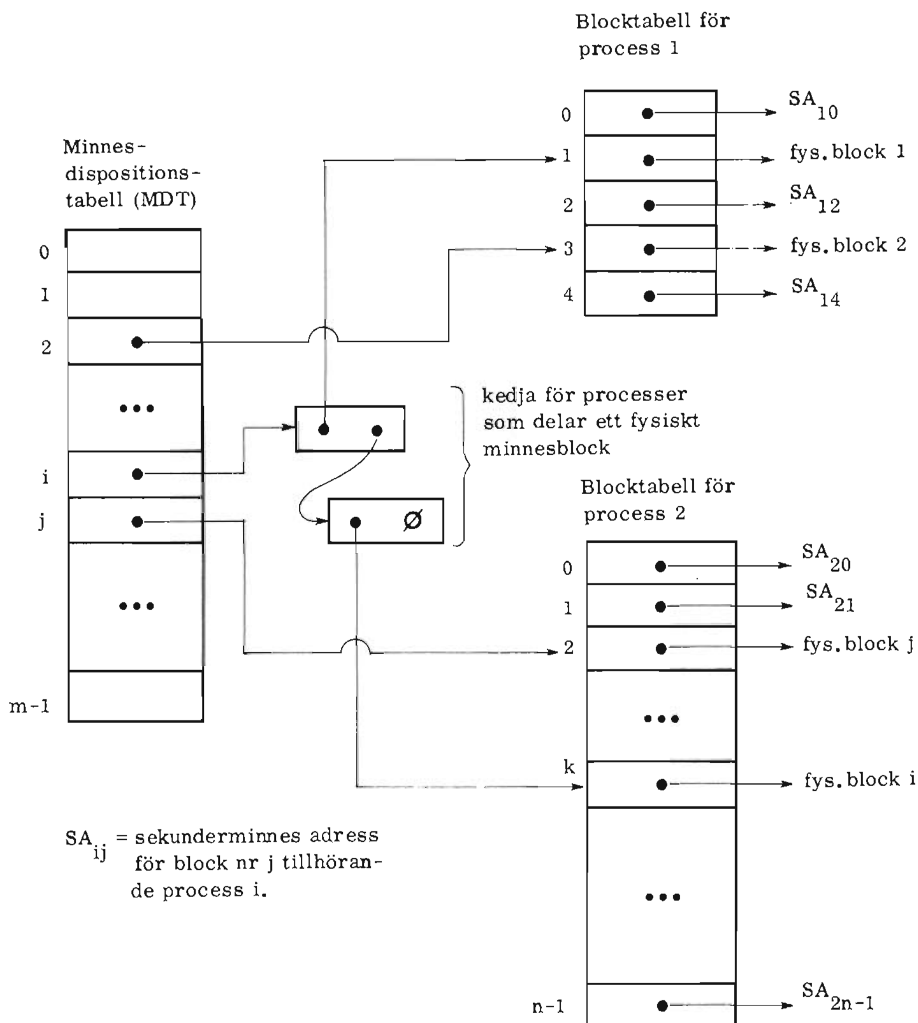
Figur 7.4:7
Adresstransformation vid segmenterade logiska adressrum är en tvåstegsprocess och kräver två nivåer av tabeller.

T ex om σ_3 i figur 7.4:8 blir ledigt sammanslås L_2, σ_3 och L_3 till ett enda utrymme. I annat fall skapas ett nytt element som inlänkas i listan för ledigt utrymme.



Figur 7.4:8
Bokföring av minnesdisposition när principen med "tilldelning av sammanhängande utrymme av olika längd" tillämpas.

En metod för att hålla reda på minnets disposition när blockvis fragmenterad lagring och blockutbytesteknik tillämpas är visad i figur 7.4:9. Åtskilliga andra metoder existerar i samband med denna teknik beroende på att skilda datorer har olika maskinvarufunktioner med vars hjälp adresstransformation m m sker. Vissa sådana metoder redovisas i (1). Den metod som visas i figur 7.4:9 bör anses som relativt administrationskrävande när flera processer gemensamt använder (delar) vissa fysiska minnesblock.



Figur 7.4:9

Ett minne antas bestå av m block av fast längd (z). En minnesdispositionstabell (MDT) bestående av m element upprättas. Om en process har tilldelats ett fysiskt block "i", pekar element nr "i" i MDT på motsvarande process' blocktabell. Om ett fysiskt block delas av flera processer kan referens till resp blocktabell ske via en länkad lista. Ett element i en blocktabell pekar antingen på ett fysiskt block i primärminnet eller på motsvarande blocks lagringsplats på sekundärminnet.

Slutligen bör påpekas att tabellerna och listorna i fig 7.4:8 och 7.4:9 saknar, för överskådlighets skull, en mängd för praktiskt bruk nödvändiga detaljer (flaggor, indikatorer o dyl), då de ej är av direkt intresse i detta sammanhang.

7.4.3 Placeringsstrategier och minnesutnyttjande

Antingen minne tilldelas i form av segment av variabel storlek eller i form av ett antal "spridda" block av fix längd uppträder spill - dvs minnesutrymme som ej kan utnyttjas. Man brukar skilja på (ref. (15), (2))

- intern fragmentering, när minnestilldelningen rundas av till närmast större antal block

och

- extern fragmentering, vid segmentvis tilldelning av minne när ett antal "luckor" uppstår vilka på grund av sin ringa storlek ej kan utnyttjas till att lagra andra programsegment.

Den negativa effekten - minskat minnesutnyttjande - av intern fragmentering kan minskas genom att minska blockstorleken (z). Minskad blockstorlek innebär å andra sidan krav på ökat utrymme för blocktabeller och även en ökning av transportarbetet för överföring av block till eller från sekundärminne.

Den externa fragmenteringens nackdelar kan minskar antingen genom att tillämpa någon sofistikerad algoritm för val av "luckor" vid inplanering av nya segment, eller genom att kosta på extra administrativt arbete och utföra s k minneskomprimering. Minneskomprimering innebär att vid varje avslutad process, när minne återlämnas, återstående segment relokeras (förskjuts i minnet) så att största möjliga sammanhängande lediga utrymme skapas.

Placeringsstrategier vid segmenterad tilldelning diskuteras av Denning (2) under antagandet att "systemet befinner sig i jämvikt" dvs att under en lång tidsperiod antalet krav på tilldelning av nya segment är lika stort som antalet ledigförklarade segment.[1] Antalet segment som vid en viss tidspunkt finns i minnet betecknas med n och antalet luckor betecknas med h. Om luckornas storlek är x_1, x_2, ..., x_h och ett krav på utrymme för ett segment av storleken s uppstår, kan bl a någon av följande placeringsstrategier (2) tillämpas:

---

1) Förutsätter att minnesutrymmet ej är den kritiska (begränsande) faciliteten.

1. Om $x_1 \leq x_2 \leq x_3 \leq \ldots \leq x_h$, finn det minsta "i" sådant att $x_i \geq s$. Strategin strävar efter bästa minnesutnyttjande. Listan över luckor är sorterad i stigande storlek på luckorna.

2. Luckorna är sorterade i stigande ordning efter minnesadress. Man väljer luckan med lägsta minnesadress som är tillräckligt stor för att rymma segmentet. Metoden är mindre söktidskrävande än 1. Efter någon tid tenderar dock små luckor att samlas i början på listan vilket ökar söktiden. Detta kan avhjälpas genom att ordna luckorna i en cirkulär lista där ingångselementet varje gång framflyttas en länk.

Knuth (17) redogör för simuleringsexperiment där resultaten förordar strategin 2. I ref. (17) diskuteras även en annan algoritm som innebär att man delar upp luckorna i k listor med storlekarna 2^i, $i = 1, 2, \ldots, k$. En lucka inom lista j kan överföras till två luckor i listan med luck-storleken 2^{j-1}.

Under antagandet att jämvikt råder anger Knuth (17) två regler avseende minnesutnyttjande vid segmenterad tilldelning.

1. Antalet luckor i minnet är approximativt $h = n/2$

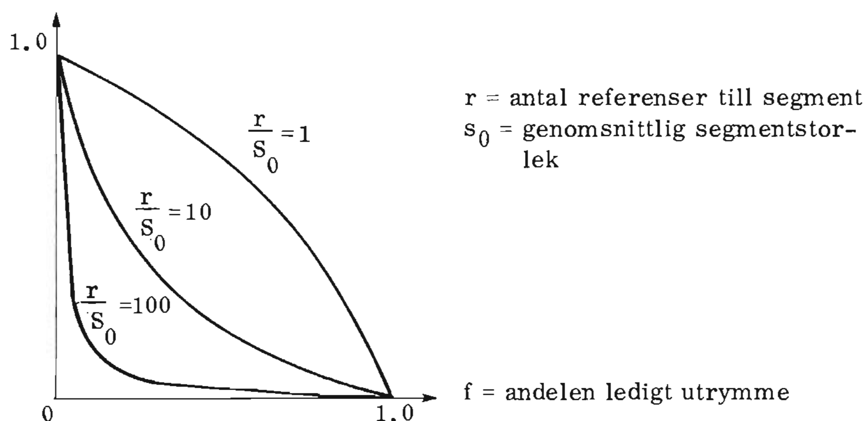
2. Andelen av minnet som upptas av luckor är $f \geq k/(k + 2)$, där k är förhållandet mellan genomsnittlig storlek för luckor och genomsnittlig segmentstorlek.

För en effektiv drift anges att minnesstorleken bör vara minst 10 gånger större än genomsnittlig segmentstorlek (17).

Är det bättre att låta minnet externfragmenteras eller skall man komprimera? Det är en fråga som ej kan besvaras här då kostnaden för dåligt minnesutnyttjande skall vägas mot behov av och kostnad för processortid (innebärande bl a fördröjningar) för komprimering. En approximativ modell för beräkning av andelen processortid som erfordras för komprimering är redovisad i (2). Antag att jämvikt råder, låt minnet bestå av m celler och låt f vara andelen av minnet som är ledigt. Antag vidare att varje segment får r referenser under sin "livstid" och att genomsnittlig segmentstorlek är s_0. Jämviktsvillkoret innebär att ett nytt segment inlagras i minnet vid var r:te referens. Den av segment belagda delen av minnet "expanderar" med hastigheten s_0/r minnesceller per tidsenhet. Tiden för att "spräcka" minnet är då $t_0 = fm/(s_0/r)$.

Komprimeringen kräver två minnesreferenser per flyttad cell (plus viss

administrativ spilltid). Antal ord som skall flyttas vid komprimering är $(1 - f)m$ och antalet referenser till minnet blir $t_c = 2(1 - f)m$. Andelen av tiden som åtgår för komprimering är då $\Psi = 1 - t_0/(t_0 + t_c)$. Sambandet är grafiskt åskådliggjort i figur 7.4:10 (källa: ref. (2)).



Figur 7.4:10
Komprimering måste anses som ogynnsam vid högt minnesutnyttjande (f litet) och för "kortlivade" segment (r/s_0 litet).

Den interna fragmenteringen kunde, som tidigare påpekats, minskas genom att minska blockstorleken. Å andra sidan ökar detta blocktabellens storlek och minskar transporteffektiviteten. Om man tar hänsyn till minnesutnyttjandet och kostnader för spill i block resp kostnad för tabeller kan en "optimal" blockstorlek härledas (2). Låt c_2 vara kostnaden för en icke utnyttjad cell i ett minnesblock och c_1 kostnaden för ett rad-element i blocktabellen. Beteckna genomsnittlig segmentstorlek med s_0 och blockstorleken med z. Antag att segmentstorleken s är en slumpvariabel med det förväntade värdet (medelvärdet) $E(s) = s_0$. Då förväntas ett segment uppta s_0/z block. Blocktabellen för segmentet "kostar" därför $c_1 s_0/z$. Om $\underline{z \ll s_0}$ kan man approximativt anta att $z/2$ celler av segmentets sista block icke är utnyttjade. Detta "kostar" $c_2 z/2$. Den förväntade totalkostnaden blir $E(C/z) = c_1 s_0/z + c_2 z/2$.

$dE(C/z)/dz = 0$ ger $-c_1 s_0/z_0^2 + c_2/2 = 0$ och optimal blockstorlek

$z_0 = \sqrt{(2 s_0 c_1/c_2)}$.

Om man betraktar att en normal segmentstorlek s_0 är ca 1000 minnesceller och antar att $c_1 = c_2$ fås z_0 till ca 50 celler. Anledningen till att

normal blockstorlek vid existerande virtuella minnessystem håller sig kring 500-1000 celler är transportekonomin. Endast stora yttre kärnminnen kan försvara så små block som ca 50 celler. Dylika kärnminnen anses emellertid ännu så länge alltför dyrbara för "vanliga" installationer. Trummor och snabba skivminnen (accesstider ca 10 ms) används vanligen för lagring av programblock. Dessa är, på grund av den initiala accesstiden, ineffektiva för små blockstorlekar. Men, som ovan diskuterats, innebär stora block ett dåligt minnesutnyttjande vilket verkar kapacitetshämmande. Många tidsdelningssystems relativt (i förhållande till maskinstorlek) dålig prestanda är orsakad av detta förhållande.

7.4.4 Minnesadministration vid blockutbyte (paging)

I detta avsnitt skall vi betrakta de tidigare nämnda strategierna för inhämtning från sekundärminne av nya block och strategierna för beslut om vilka block i minnet som skall ersättas med nya block (dvs "fetch and replacement strategies").

En mängd olika metoder för ovanstående ändamål har föreslagits och analyserats (bl a ref. (2), (4), (8), (11), (13), (14), (16), (18)). Utrymmet i denna bok tillåter endast en ytlig betraktelse av problemkomplexet.

Problemställning, processers beteende m m

Utbytestekniken (swapping) introducerades hos de tidiga tidsdelningssystemen (ca 1960) och gick ut på att program i sin helhet var föremål för utbyte. Ett program inlästes, gavs ett kvantum exekveringstid och utmatades därefter tillbaka till sekundärminnet. Metoden används fortfarande med god effektivitet av flera minidatorers tidsdelningssystem. Genom att dela upp minnet i två delar kunde exekvering ske i den ena delen samtidigt som "programbyte" pågick för den andra delen.

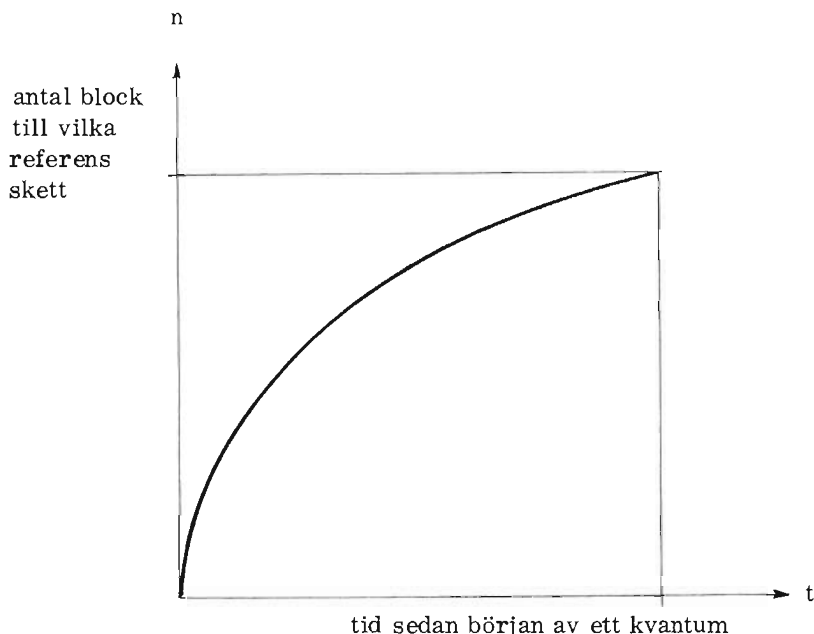
Blockvis fragmenterad lagring av program innebar inledningen till metoden att endast byta ut ett eller flera block allteftersom sådana behövdes (kravstyrt blockutbyte) eller beräknades bli erforderliga (prognosstyrt blockutbyte). Genom att i primärminnet hålla endast sådana block som direkt erfordrades för att en eller flera processer skall kunna exekveras räknade man med att öka minnesutnyttjandet och processortidsutnyttjandet, och samtidigt minska transportvolymen till/från sekundärminnet.

Prestanda hos system som tillämpar denna princip har i få fall motsvarat förväntningarna. Anledningarna har sannolikt varit mindre goda

planeringsalgoritmer i kombination med relativt långa åtkomsttider till block på aktuella sekundärminnen. Processers beteendemönster i denna miljö har heller inte tillräckligt klarlagts. Metodiken har dock potentiella möjligheter att, med hjälp av goda planeringsalgoritmer, bli mer allmänt accepterad.

Populärt uttryckt kan problemet vid blockutbyte beskrivas som att "minimera risken för att inget arbete blir utfört på grund av att det är svårt att åstadkomma "produktiva" block-sammansättningar i primärminnet och systemet riskerar att bli i huvudsak sysselsatt med att skyffla block fram och tillbaka mellan primärminnet och sekundärminnet". Mycket låga värden avseende processortidsutnyttjande för användarprogram har i vissa fall noterats ($<$ 10-20 %).

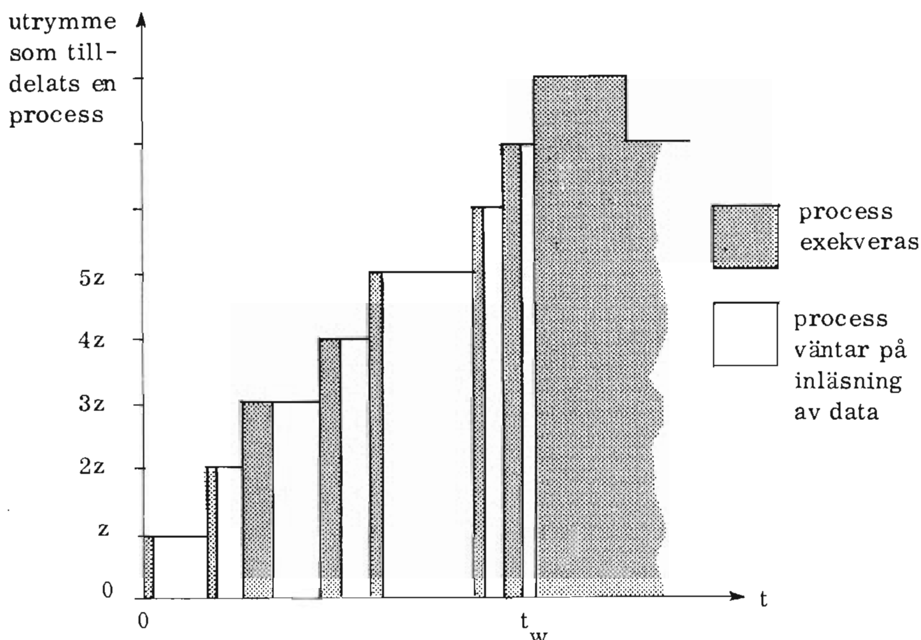
Till att börja med är det intressant att betrakta en process' referenser till informationselement i olika block under tidskvantum. Tidigare i detta kapitelavsnitt diskuterade vi en process' beteendemönster i samband med att vi angav den referenssträng $\omega = r_1 r_2 r_3 \ldots r_k \ldots, \; r_k \in N_i, \; k \geq 1$. Varje referens r_k innebär referens till ett block. Ett typiskt samband mellan förfluten tid sedan ett tidskvantums början och antal av en process refererade block är visat i figur 7.4:11.



Figur 7.4:11

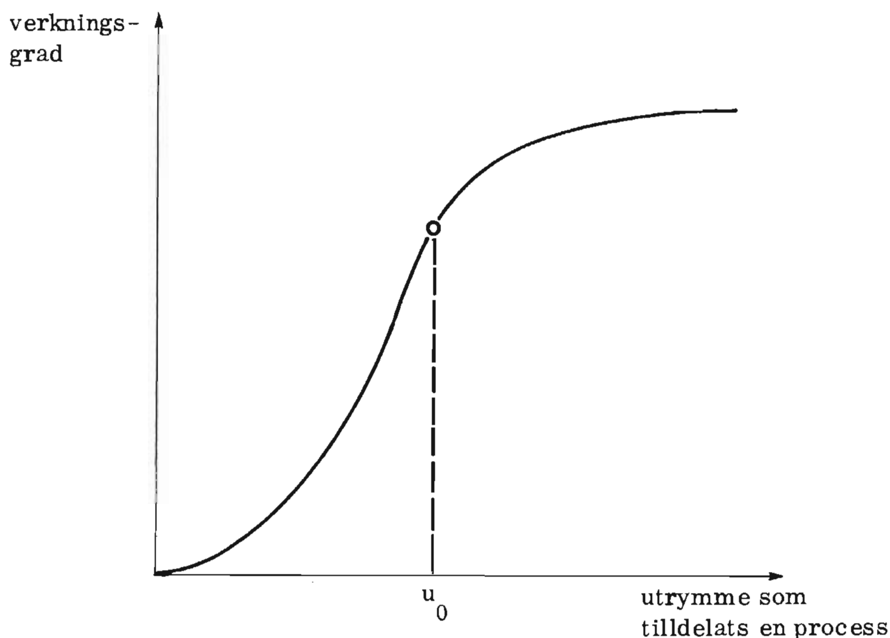
Experiment med kravstyrt blockutbyte (demand paging) (bl a ref. (1), (2), (13), (18), (19)) har visat att när en process börjar exekveras är referensfrekvensen till "nya block" stort men att den avtar med tiden.

Figuren 7.4:11 visar att processen har referat till merparten av under tidsintervallet använda block under tidsintervallets första hälft. Referensfrekvensen kan i början anta sådan storlek att föga produktivt uträttas (figur 7.4:12). En gammal regel i programmeringsbranschen säger att större minnestilldelning till ett program brukar medföra snabbare exekvering. Detta gäller i stort även för processer som arbetar under kravstyrt blockutbyte, även om vissa undantag har konstaterats (ref. (11)). Verkningsgraden, definierad som antalet instruktioner som utförs innan ett nytt block krävs inläst stiger med antalet block som tilldelats processen (figur 7.4:13).



Figur 7.4:12
Vid kravstyrt blockutbyte är processortidsutnyttjandet, under de inledande inläsningarna av erforderliga block, lågt (ref. (18)). Samtidigt belastar processen primärminnet. Vid t_w förefaller processen ha fått ett tillräckligt antal block inlästa för att kunna exekveras under en något längre tid.

Av figur 7.4:13 framgår dock att ett kritiskt minimiutrymme u_0 kan urskiljas. Om mindre utrymme än u_0 tilldelas blir verkningsgraden låg. Om minne skall tilldelas ett flertal processer finns det kanske skäl att vägra tilldelning till samtliga processer på en gång för att ej riskera en



Figur 7.4:13
Verkningsgraden definieras här som genomsnittligt antal utförda instruktioner mellan kraven på inläsning av nytt block från sekundärminne. Av figuren framgår att det ej lönar sig att tilldela en process alltför mycket minne. Däremot blir systemets verkningsgrad dålig om processer tvingas exekveras med en lägre tilldelning än u_0.

alltför dålig verkningsgrad för hela systemet. När en process initieras för exekvering är det väsentligt för verkningsgraden att processens för tillfället "allra nödvändigaste" mängd av block finns i primärminnet. Denna mängd av block, vars innehåll givetvis varierar med tiden, kan kallas för "arbetsunderlaget" ("working set", Denning ref. (2), (20), (22)). Vi återkommer till arbetsunderlaget i samband med diskussion av några styralgoritmer för blockutbyte.

Styralgoritmer för blockutbyte

Åtgärder som kan vidtas för att höja systemets effektivitet är främst (22):

1. Att förbättra programmens struktur (ur blockutbytessynvinkel) dvs att genom (användarens eller) kompilatorns försorg minska frekvensen för krav på en mängd nya block.

2. Att förbättra algoritmerna för styrning av system som arbetar med blockutbytesteknik.

3. Att använda snabbare medium för lagring av utbytta block - dvs snabbare sekundärminnen.

Olika skäl talar för att (1) är en besvärlig lösning. Programmen är oftast modulärt uppbyggda - man känner ej alltid till beteendet hos andra moduler. Dessutom är processens beteende beroende av dess data, vilka blir kända för systemet endast när exekveringen väl har startat. Alternativet (3) är i dagens läge dyrbart, men kan förväntas bli ett mindre problem i framtiden.

Resten av detta avsnitt ägnas åt några styralgoritmer för blockutbyte.

Val av lämplig styralgoritm är beroende på en mängd faktorer, såsom (18)

- kostnaden för icke produktivt utnyttjande av systemresurser
- systemets arbetsbelastning och belastningens struktur (jobbprofiler)
- kostnaden och resurskravet för styralgoritmen själv

m m

Utvärdering av olika algoritmer är ett besvärligt problem, som vi avstår från att gå djupare in på i denna bok.

Beslutet om att byta ut ett eller flera block i primärminnet kan fattas vid två tidpunkter

1. när ett nytt block krävs av en pågående process

2. när ett block från sekundärminnet har blivit inläst till ett buffertutrumme i primärminnet.

En "god styralgoritm" strävar bl a mot att utnyttja datorns processor till så mycket produktivt arbete som möjligt.[1] Ett delmål här är att ständigt ha en exekverbar process "till hands". Ett allmänt accepterat mål hos många system, som arbetar med blockutbytesteknik, är att vid beslutstillfället ((1) eller (2) ovan) söka byta ut det block i primärminnet som har den lägsta sannolikheten att inom en nära framtid bli efterfrågat av någon pågående process (block med lägsta referenssannolikhet).

---

1) Med beaktande av diverse andra önskemål såsom bättre service till högprioriterade processer osv.

Även om denna princip ej alltid leder till "optimal styrning" anses den som en god "heuristik", som kommer optimala lösningar nära.

Följande algoritmer kommer att betraktas

- FIFU-strategin (först-in-först-ut)
- LTSA-strategin (längst tid sedan använt (block))
- arbetsunderlagsstrategin

FIFU-strategin innebär att det eller de block avlägsnas som längst befunnit sig i primärminnet. Metoden är enkel att implementera t ex genom att varje block i minnet associeras med ett "åldersregister" i en ålderstabell. Tabellen uppdateras vid varje beslut om blockutbyte. En kanske ännu enklare metod att administrera FIFU-strategin är genom att tänka in block i en "ålderslista" eller en "åldersstack".

LTSA-strategin innebär att det eller de block avlägsnas vars sista referens befinner sig längst tillbaka i tiden. Metoden är svårare att implementera men förefaller (intuitivt) överlägsen FIFU när det gäller att finna blocket med lägsta referenssannolikhet. Olika metoder, att implementera LTSA har föreslagits (se ref. (1), s. 170) vilka samtliga går ut på att associera ett åldersregister med varje block av fysiskt minne. Olika mekanismer, maskin- som programvaruorienterade, kan tänkas för uppdatering av dessa åldersregister. Uppdateringen kräver givetvis datorkapacitet.

Problemet med båda ovannämnda strategier är att de inte gör någon direkt åtskillnad mellan till vilka processer de olika blocken hör, och därför ej tar hänsyn till övriga data om processerna för att avgöra vilka block som lämpligen bör bytas ut. Sådana strategier kan ge icke önskvärda effekter, såsom avlägsnande av vissa av den blockkrävande processens egna block (som krävs för vidarebearbetning) eller andra processers block så att deras "arbetsunderlag" blir alltför litet och systemets verkningsgrad katastrofalt låg.

Arbetsunderlagsstrategin ("working-set strategy", Denning (22)) lämnar praktiska problem kvar att lösa innan den kan allmänt implementeras. Idén baseras på observationer i figurerna 7.4:11 t o m 7.4:13 att referensfrekvensen till nya block avtar när ett visst "underlag" av block för en process befinner sig i primärminnet.

Arbetsunderlaget för en process betecknas med $W(t,\tau)$ och utgöres av de block som processen refererat till under tidsintervallet $(t-\tau, t)$. τ bör väljas med hänsyn till faktorer såsom

- primärminnesstorlek
- åtkomsttid till sekundärminnet
- transporthastighet m m

$W(t, \tau)$ bestäms genom åldersregisterteknik motsvarande dem som diskuterats i samband med LTSA-strategin. Endast block som tillhör den aktuella processen betraktas.

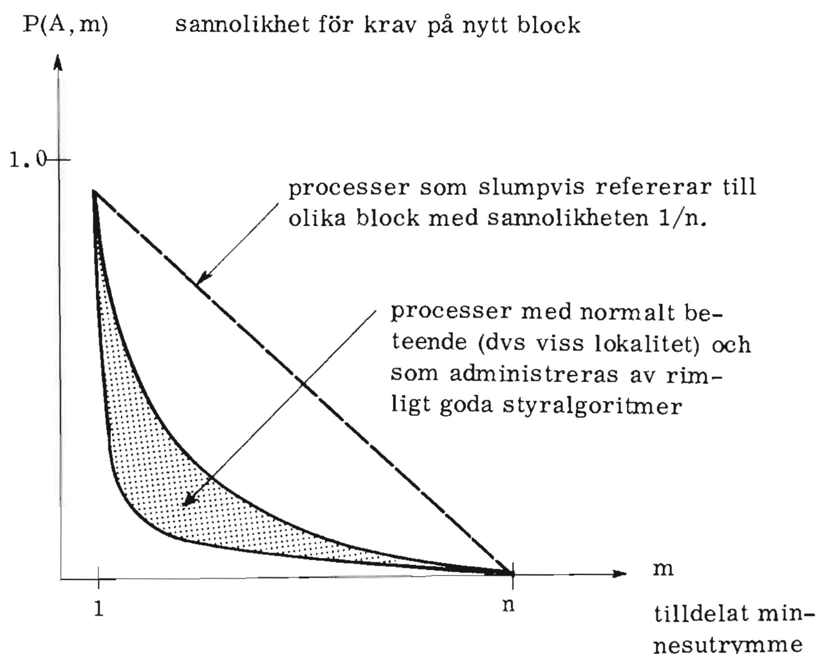
Målet med denna strategi är att hålla i primärminnet endast sådana processer vars $W(t, \tau)$ befinner sig i primärminnet och att vid blockutbyte göra minsta möjliga åverkan på andra processer.

1. Ingen process får laddas till primärminnet för exekvering om inte plats finns för lika många block som ingick i dess senaste arbetsunderlag.

2. Vid blockutbyte, avlägsna först de block från den aktuella processen vilka faller utanför processens just då aktuella $W(t, \tau)$.

För att ge plats till en ny process bör hela arbetsunderlaget för en eller flera processer utmatas till sekundärminnet. Strategin syftar till att minska icke önskvärd interaktion avseende minneskrav mellan processer i primärminnet, och förefaller, teoretiskt sett, intressant.

Vi avslutar detta avsnitt med ett påpekande (Denning, (2), s. 179) att "även om valet av styralgoritm är viktigt så är valet av lämplig minnesstorlek kritiskt". Detta illustreras av figur 7.4:14. P(A,m) betecknar sannolikheten att krav på inläsning av nya block (när en viss referenssträng $\omega = r_1 r_2 r_3 \ldots$ bearbetas) inträffar när minnet administreras av styralgoritmen A och när processen är tilldelad ett minnesutrymme av storleken m block. Processens logiska adressrum antas omspänna n block. Det framgår av figuren att P(A, m) kan vara mer känslig för variationer i m än för val av olika algoritmer. Figuren gör ej anspråk på att vara generell då algoritmer förekommer som ej i alla lägen minskar P(A,m) genom att m ökas.



Figur 7.4:14
Sannolikheten att en process begär ett nytt block när det blivit tilldelat ett minnesutrymme omfattande m block, $1 \leq m \leq n$, där n är antalet block i processens logiska adressrum. (källa: ref (2), sid 179)

Litteratur

1. Watson, R.W., Timesharing System Design Concepts. McGraw-Hill Computer Science Series, New York, 1970.

2. Denning, P.J., Virtual Memory, in Computing Surveys (Survey and Tutorial Journal of the ACM), Vol. 2, No. 3, Sept. 1970.

3. Randell, B. and Kuehner, C.J., Dynamic Storage Allocation Systems. Communications of the ACM, Vol. 11, No. 5, May 1968.

4. Oppenheimer, G. and Weizer, N., Resource Management for a Medium Scale Time-Sharing Operating System. Communications of the ACM, Vol. 11, No. 5, May 1968.

5. Daley, R.C. and Dennis, J.B., Virtual Memory, Processes and Sharing in MULTICS. Communications of the ACM, Vol. 11, No. 5, May 1968.

6. Mullery, A.P. and Driscoll, G.C., A Processor Allocation Method for Time-Sharing. Communications of the ACM, Vol. 13, No. 1, Jan. 1970.

7. Ryder, K.D., A Heuristic Approach to Task Dispatching. IBM Syst. J., Vol. 9, No. 3, 1970.

8. Belady, L.A., A Study of Replacement Algorithms for a Virtual-Storage Computer. IBM Syst. J., Vol. 5, No. 2, 1966.

9. Hellerman, H., Some Principles of Time-Sharing Scheduler Strategies. IBM Syst. J., Vol. 8, No. 2, 1969.

10. Hansen, P.B., The Nucleus of a Multiprogramming System. Communications of the ACM, Vol. 13, No. 4, April 1970.

11. Belady, L.A., Nelson, R.A. and Schedler, G.S., An Anomaly in Space-Time Characteristics of Certain Programs Running in a Paging Machine. Communications of the ACM, Vol. 12, No. 6, June 1969.

12. Katzan Jr, H., Advanced Programming. Programming and Operating Systems. Van Nostrand-Reinhold, New York, 1970.

13. Coffman, E.G. and Varian, L.C., Further Experimental Data on the Behavior of Programs in a Paging Environment. Communications of the ACM, Vol. 11, No. 7, July 1968.

14. Smith, J.L., Multiprogramming under a Page on Demand Strategy. Communications of the ACM, Vol. 10, No. 10, Oct. 1967.

15. Randell, B., A Note on Storage Fragmentation and Program Segmentation. Communications of the ACM, Vol. 12, No. 7, July 1969.

16. Wallace, V.L. and Mason, D.L., Degree of Multiprogramming in Page-on-Demand Systems. Communications of the ACM, Vol. 12, No. 6, June 1969.

17. Knuth, D.E., The Art of Computer Programming. Vol. 1, Addison-Wesley, Reading, Mass., 1968.

18. Kuehner, C.J. and Randell, B., Demand Paging in Perspective. AFIPS 1968 Fall Joint Conference Proceedings, Vol. 33, Part 2, Thompson, Washington, D.C., 1968.

19. Fine, G.H., Jackson, C.W., McIsaac, P.V., Dynamic Program Behavior under Paging. ACM 21st National Conference Proceedings, Thompson, Washington, D.C., 1966.

20. Denning, P., Thrashing: Its Causes and Prevention. AFIPS 1968 Fall Joint Comp. Conference Proceedings, Thompson, Washington, D.C. 1968.

21. Brawn, B.S. and Gustavson, F.G., Program Behavior in a Paging Environment. AFIPS 1968 Fall Joint Comp. Conference Proceedings, Thompson, Washington, D.C., 1968.

22. Denning, P.J., The Working Set Model for Program Behavior. Communications of the ACM, Vol. 11, No. 5, May 1968.

23. Leverantörers beskrivningar och handböcker om resp operativsystem - ingen nämnd och mången glömd.

8. DATATRANSPORT- OCH FILÖVERVAKNINGSSYSTEM

8.1 Inledning och begrepp

Från att ha bestått i huvudsak av rutiner för hantering av in/utmatningsoperationer till sekundärminnen och andra perifera enheter har operativsystems datatransportfunktion utvecklats till relativt omfattande filövervakningssystem med en mängd olika funktioner. En bidragande faktor här har varit tillkomsten av direktminnen med rimligt "kapacitet-åtkomsttid-pris" förhållande. Härigenom har en allt större andel av en installations datafiler och programfiler kunnat lagras permanent i datorsystemet och sökning i dessa har varit (teoretiskt) möjlig med relativt kort åtkomsttid. Datatransport- och filövervakningssystems struktur och olika vanligen förekommande funktioner skall vi belysa i detta avsnitt. Först är det dock lämpligt att diskutera några begrepp i samband med filer och datatransporter.

Begreppet fil (på svenska även register) kan ännu så länge normalt sägas kännetecknas av följande egenskaper:

1. En fil innehåller ett antal poster

2. En post består av värden för ett antal (data-)termer. Dessa värden kan antingen vara kvantitativa eller bestå av en hänvisning (länk) till en annan term eller post.

3. Två poster säges vara av samma klass (slag) om de har samma postbeskrivning. Detta hindrar inte att poster av samma klass kan vara av variabel längd (vid maskinrepresentation) och/eller innehålla ett olika antal datatermvärden. Postbeskrivningen hos idag existerande system är normalt explicit angiven eller implicit antagen i de program som avses bearbeta postklassen ifråga.

4. Vanligen brukar en fil utgöras av poster av samma klass, dvs alla poster i filen har samma postbeskrivning.

5. Datatermers värden representeras i datorers lagringsmedia av en sekvens av binära siffror. Beroende på datortyp kan de sedan sägas vara grupperade i 6-bitstecken, "bytes" (oktader) eller ord. En post representeras följaktligen (i ett lagringsmedium) av en sekvens av

binära siffror. Normalt anges termvärdenas och postens längd genom det antal 6- eller 8-bitarstecken som dess maskinrepresentation kräver.

Postbeskrivning:

Personpost:

Personnummer	term
Namn:	termgrupp
- Efternamn	term
- Förnamn	"
Adress:	termgrupp
- Gatuadress	term
- Postnummer	"
- Postadress	"
Längd	"
Vikt	"
Hårfärg	"

Aktuella poster av klassen "personpost"

471025-1235	390513-1811	370802-3953
ANDERSSON	PETTERSON	LUNDSTRÖM
FRANCOIS	ALEXANDER	SEVED
LILLA BOMMEN 2	BLÅSVÄDERSGRÄND 7	
18953	15385	
GÖTEBORG	FILSTAD	
175	190	
65	93	
MELLANBRUN	BLOND	osv

Figur 8.1:1
Exempel på postbeskrivning och några tillhörande "aktuella" poster.

6. För att man skall kunna företa en rimligt effektiv utsökning av en viss bestämd post i en fil måste filen vara organiserad och dess organisationsstruktur och åtkomstprincip känd. Det är normalt att en eller flera termer i en post utgör postens identifieringsbegrepp (ID-begrepp, t ex personnummer i figuren 8.1:1). De vanligaste organisationsformerna är

   - sekvensiell organisation, eventuellt sorterad efter ID-begreppet som medger en sekvensiell behandling av poster i stigande eller fallande ID-begreppsföljd.

   - direktåtkomstorganisation vilken, om man känner adressen eller ID-begreppet, medger en mer eller mindre direkt åtkomst till motsvarande post. Denna organisationsform förutsätter att filen kan lagras på ett direktminne (trumma, skiva o dyl).

   Andra, mer komplicerade, organisationsformer börjar få spridning tack vare det begynnande intresset för databaser[1] och önskemål om mer komplicerade utsökningar. Vi skall belysa ytterligare organisasonsformer i avsnitt 8.3.

7. De flesta operativsystems filhanteringssystem tillåter endast enklare datahantering. En utsökning av en fil resulterar i antingen en unik träff eller ingen träff alls. Det senare får, i normala fall, ses som en felaktig operation. De utsökningsmetoder, som vanligen förekommer hos operativsystems filhanteringsdel, kan därför klassificeras som[2]

   a. utsökning sekvensiellt efter nästa post (eller postmängd)

   b. utsökning efter adress (dvs man vet motsvarande posts adress)

   c. utsökning efter namn (dvs efter ID-begrepp)

   Utsökningarna a och b ger unika svar. För att utsökningen c skall ge unikt svar krävs unika namn - i vårt fall identifieringsbegrepp. Om vi i exemplet i figur 8.1:1 använder personnummer som sökbegrepp

---

1) Det ligger utanför denna boks syfte att mer i detalj behandla databas-databank-problematik. Några allmänt vedertagna definitioner om vad en databas eller vad en databank är existerar ej. Vi gör dock troligen inget större fel om vi definierar en databas som ett system av filer.

2) Se (betr. definitioner av ytterligare utsökningsprinciper) Langefors, B., "Introduktion till Informationbehandling". sid. 41.

fås vid utsökning efter namn unikt svar. Används däremot "Namn" som sökbegrepp består svaret, i det generella fallet, av en mängd av poster som tillfredsställer sökvillkoret då flera personer kan ha samma namn. Vi kan också tänka oss en mer komplicerad utsökning efter egenskap och formell beskrivning, t ex "sök efter alla poster som tillfredsställer sökvillkoret: [efternamn = Andersson] och [postadress = Filstad] och [vikt ≥ 100 kg]". Som påpekats ovan, tillhandahåller operativsystem normalt ej verktyg för utsökningar enligt ovan som kan resultera i multipla svar. Det existerar dock hos vissa leverantörer "applikationspaket för databashantering" som kan disponeras av användaren och inkorporeras som en länk mellan applikations- och operativsystemet. Normat är dock användaren ännu så länge själv ansvarig för hantering av mer komplicerade utsökningar och gör då bruk av kombinationer av de utsökningsmekanismer (av typen a, b och c) som op. systemet tillhandahåller.

8. En fil ges normalt ett namn och ett datum som anger tidpunkten för dess upprättande eller senaste ändring.

9. Hos vissa filer, vars innehåll regelbundet ändras, kan man tala om "generationer" av desamma. Ett antal "senaste generationer" av en fil med samma namn sägs då utgöra en generationsgrupp. T ex om personfilen enligt figur 8.1:1 regelbundet uppdateras en gång i veckan och vi alltid sparar de 4 senaste versionerna utgör dessa en generationsgrupp.

10. För vissa applikationer är i systemet förekommande filer med skilda namn stort. För att underlätta administration av dessa, och tillåta lokala namn, oberoende av lokala namn hos andra filer, tillhandahåller vissa filhanteringssystem en mekanism att namnge och katalogisera filerna. Systemet ombesörjer då också reservering av plats i sekundärminnessystemet och lagring av filen. Detta belyser vi ytterligare i avsnitt 8.6.

11. Som framgår av pkt 5 kräver datorlagring av en fil ett lagringsutrymme som normalt uttrycks i antalet erforderliga 6- eller 8-bitarstecken. En standardenhet av sekundärminne kallas för en lagringsenhet. Exempel på lagringsenheter är magnetband, skivminnen o dyl. Lagringsenheter av direktminnestyp kan normalt lagra flera filer vilket kräver att de håller sig med en innehållsförteckning över i enheten ingående filer.

12. Viss information, lagrad i form av filer, kan vara av konfidentiell natur eller sekretessbelagd. Hos vissa filhanteringssystem kan en datafil skyddas mot icke auktoriserad åtkomst genom mekanismer av skilda slag. Detta diskuteras närmare i avsnitt 8.7.

Hittills har vi huvudsakligen behandlat sekundärminneslagrade filer. Generellt kan emellertid varje datatransportoperation anses som en filoperation. Datatransportövervakningssystemet kan allmänt sägas ombesörja transporter såväl till/från sekundärminnen som till/från lokala och avlägsna perifera enheter. I det senare fallet avses rutiner för telekommunikation. En telekommunikationslinje kan generellt anses som en fil där vi arbetar sekvensiellt med utsökning resp sändning av "nästa post". När det gäller datatransporter kan ett användarprogram grovt sett skilja mellan tre servicenivåer[1] hos datatransportövervakningssystemet:

- postnivå
- blocknivå
- styrprogramnivå

Om ett applikationsprogram arbetar på postnivå betraktar den in/utgående datamängder i form av poster. Blockning (dvs sammanfogning) av en eller flera poster till ett block och begäran av en in- eller utdatatransportoperation avseende blocket ombesörjes av en posthanteringsrutin. Användaren kommunicerar med posthanteringsrutinen genom instruktioner av typen "giv mig nästa post" och "tag emot denna post för uttransport" (länk 1 i fig 8.1:2). Rutiner på posthanteringsnivå ombesörjer också normalt buffring (inklusive köadministration) av datablock och, beroende på tillgängligt utrymme, eftersträvar normalt mesta möjliga överlappning av datatransporter och internbearbetning. Ett applikationsprogram kan också i de flesta system arbeta på "block-nivå" och därvid själv ombesörja buffring och ansvara för överlappning av internbearbetning med datatransporter. Applikationsprogrammet kommunicerar då med blockhanteringsrutinen genom order av typen:

- läs nästa block från band med logiskt nr x till minnesutrymme med adress y

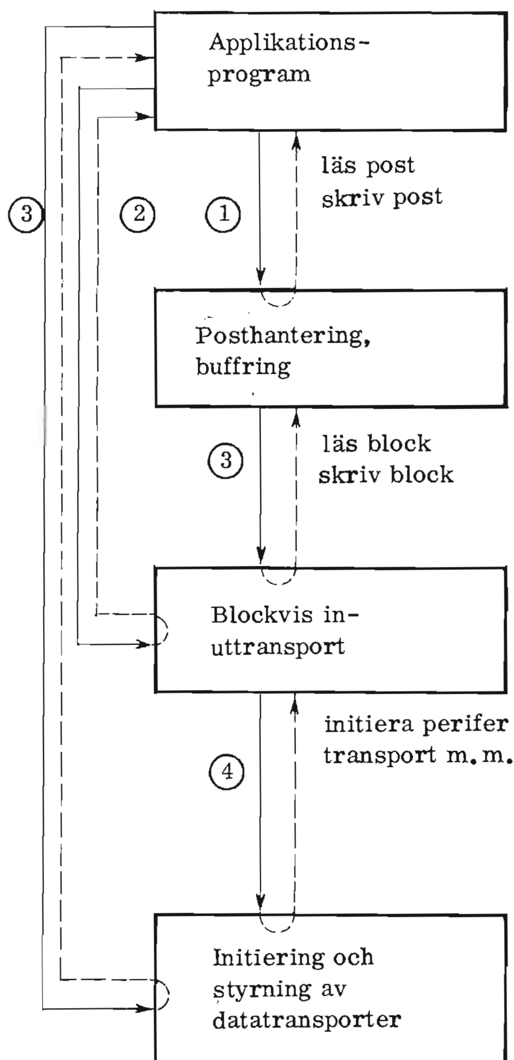
- skriv x dataenheter med början från primärminnesadress y till direktminne, logisk enhet z, direktminnesadress w

- läs meddelandeblock från terminal x till buffertutrymme med adress y.

Normalt brukar även posthanteringsrutiner kommunicera med blockhanteringsrutiner som då ombesörjer den blockvisa in/uttransporten (länk 3 i figur 8.1:2). På denna nivå ankommer det på applikationsprogrammet att kontrollera när initierad blocköverföring är avslutad.

---

1) En ytterligare, mer ambitiös, servicenivå kunde betecknas arbeta på termnivå. Här närmar vi oss s k databashanteringssystem och en separering av program och datadefinitioner.



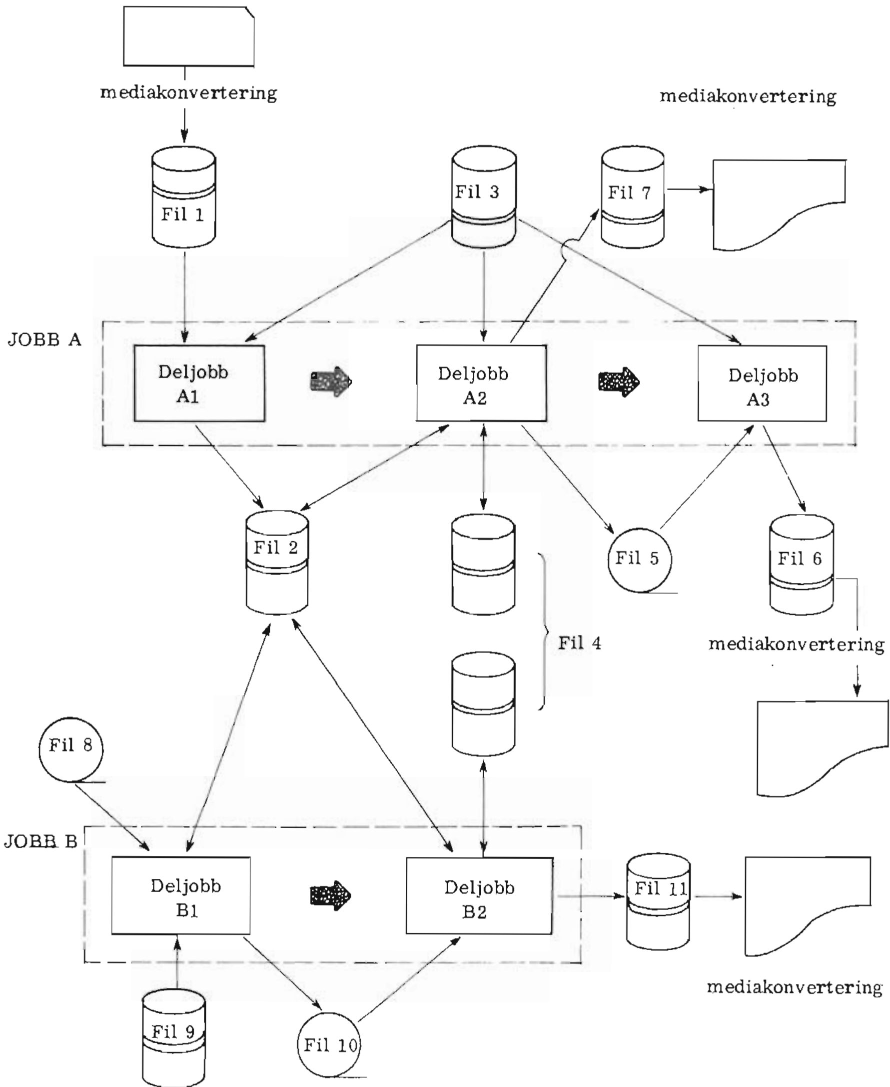
Figur 8.1:2
In-utmatningsoperationer på olika nivåer.

Blockhanteringsrutiner är ansvariga för att den aktuella datatransportoperationen rent "maskinvarumässigt" initieras och utföres. Det innebär, att det normalt faller på dessas lott att med hjälp av de styrprogrammets olika funktioner, som är relevanta i detta sammanhang, initiera datatransportoperationer och även ansvara för en del av de aktiviteter som följer på brytsignaler när datatransporterna avslutats. I undantagsfall förekommer det att även program på applikationsnivå arbetar med in/utmatning på denna "låga" nivå. Anledningen till detta är vanligen att blockhanteringsrutiner saknas i operativsystemet för en ovanlig eller "främmande" (dvs av annat fabrikat) perifer enhet. I sådana fall brukar dock användaren, ev med assistans från leverantör, utforme en för den perifera enheten avsedd blockhanteringsrutin och "infoga" den i operativsystemet. Datatransportoperationer betraktas även i avsnitt 8.5.

Figur 8.1:3 avser att visa datatransport- och filövervakningssystemets roll i samband med bearbetning av en följd av jobb eller deljobb. Varje deljobb opererar på en eller flera filer antingen genom läsning, skrivning eller bådadera. Filer kan "överlåtas" från ett deljobb till ett annat och utgör på så sätt en kommunikationslänk mellan dem. Vissa filer krävs under behandling av hela jobbet. Varje deljobb antas före sin exekvering (eller under exekvering genom kommunikation med styrprogrammet) mer eller mindre i detalj beskriva de filer som den avser att arbeta på. Beskrivningen kan omfatta komponenter, såsom namn, serienummer på lagringsenhet, typ av lagringsenhet, utrymmeskrav (om tillämpligt), auktoriseringskod, blockfaktor m m. Kravet på detaljerade uppgifter varierar, förutom mellan olika operativsystem, även beroende av typ av enhet och beror dessutom på huruvida filen redan är katalogiserad (och därmed beskriven) i systemet eller ej, samt diverse andra applikationsberoende faktorer. Vid system med multiprogrammering uppstår vid denna resurstilldelning (av filen) en potentiell risk för "låsning" (deadlock) - ett problem som vi har behandlat i kapitel 3.

8.2 Datastrukturer

Syftet med databehandling kan allmänt sägas vara att insamla, lagra, bearbeta och distribuera fakta och utsagor om ett fysiskt eller abstrakt system. Vi kan kalla detta för "objektsystemet". Data i en fil kan sägas representera en mängd fakta (sakförhållanden) om en del av detta objektsystem. Faktamängden kan vara mer eller mindre fullständig beroende på vilka typer av sakförhållanden eller variabler vi har valt att registrera.

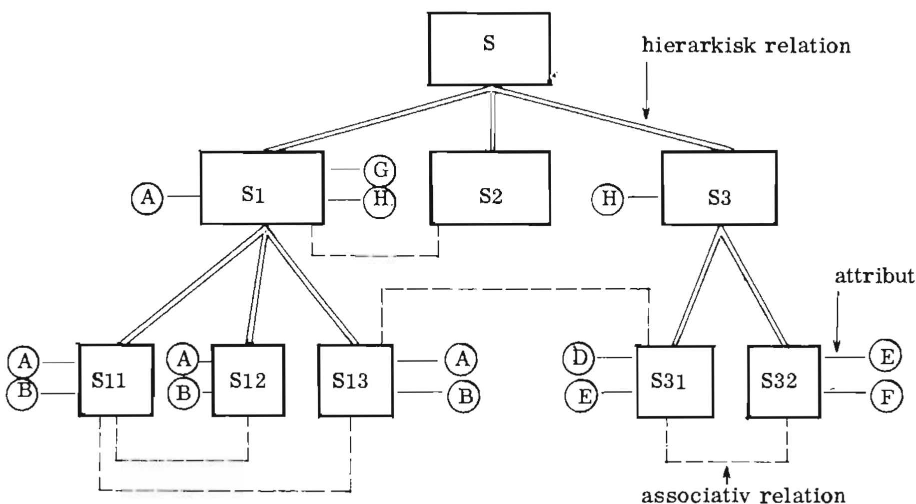


Figur 8.1:3
Exempel på två jobbs, A och B, utnyttjande av filer vid bearbetning. Man inser att, vid bearbetning, en potentiell risk för interferens eller låsning föreligger eftersom både A och B använder filerna 2 och 4.

Objektsystemet har alltid en viss struktur (explicit angiven eller implicit antagen). Om vi definierar ett system[1] som

> "en mängd av delsystem, där delsystemen karaktäriseras av vissa egenskaper (attribut) och där det råder vissa relationer mellan delsystemen"

kan sägas att strukturen beskrivs dels av hierarkiska relationer (system-delsystem relationer) dels av associativa relationer (attribut och relationer till andra delsystem). Både hierarkiska och associativa samband som attribut torde kunna betraktas som relationer.



Figur 8.2:1
Strukturen hos ett system S visad genom delsystem, dessas attribut (egenskaper) och relationer till andra delsystem.

Om ett delsystem S11 (se fig 8.2:1) sägs besitta en egenskap av typen A och denna egenskaps aktuella värde för S11 är xyz kan vi säga att S11 är relaterad till en "punkt" xyz i en "värdemängd" med relationen med namnet A.

---

1) Näraliggande definitionen i Langefors: "Theoretical Analysis of Inf. Systems", ref. (1).

Exempel: S11 är en artikel som lagerförs av ett företags (S) lageravdelning (S1). "A" betecknar attributet (- relationen) "antal i lager" och det vid en viss tidpunkt aktuella antalet i lager är då attributets värde.

Vi har genom denna diskussion närmat oss begreppet "elementar-meddelande"[1] som kan sägas karaktäriseras av det minsta antal termer som erfordras för att uttrycka ett sakförhållande, i vårt fall

1. objektet eller delsystemet (här S11)

2. attributet (-relationen) (här A)

3. attributets värde (här xyz)

4. tidpunkten för observation av ovanstående sakförhållande eller utsaga

De termer, relationer och deras kopplingar som vi, på ett explicit eller implicit sätt, bygger en fils innehåll på kan sägas beskriva den logiska (eller abstrakta) datastrukturen. Den logiska datastrukturen är helt beroende av hur vi har betraktat och strukturerat det aktuella objektsystemet, dvs vilka delsystem vi har definierat och vilka relationer och attribut som vi har tagit hänsyn till.

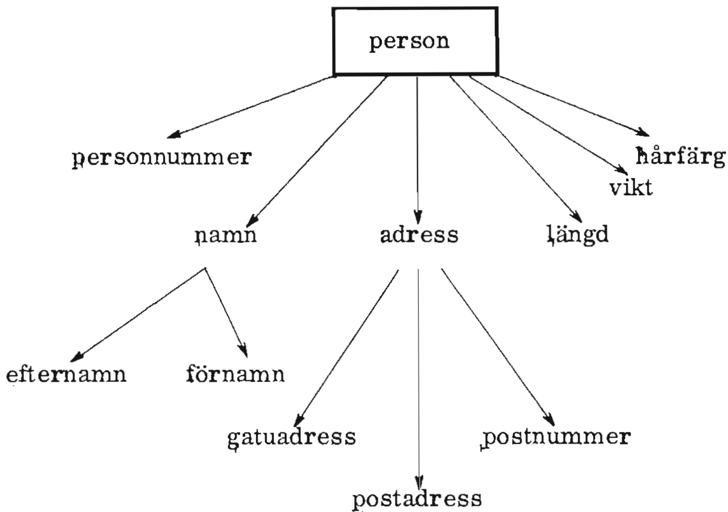
Låter vi två personer oberoende definiera en delsystemstruktur på grov nivå av ett så allmänt vedertaget och bekant system som en personbil finner vi endast i undantagsfall någon nämnvärd överensstämmelse i resultaten.[2] Vi kan således konstatera att även tämligen enkla system kommer att få relativt olika delsystemstrukturer definierade beroende på vem som utför struktureringsarbetet. Innebörden av detta är att även den logiska datastrukturen i flera avseenden kommer att bli olika från fall till fall, beroende på person, tid m m. På något längre sikt implicerar detta sannolikt betydande svårigheter för ett databassystem att kommunicera med ett annat.

Betrakta "personposten" i figur 8.1:1. Vi kan säga att den avser att bära viss information om "ett delsystem" i detta fall en person. Personen karaktäriseras här av ett antal attribut såsom namn, adress m m. Andra data om en person har ej ansetts erforderliga i detta system.

---

1) Langefors, B., "Introduktion till informationsbehandling".

2) Detta har empiriskt verifierats av förf.



Figur 8. 2:2
Datastrukturen för delsystemet person enligt figur 8. 1:1.

En person kan betraktas som ett delsystem i olika "överordnade" system. Till exempel kan en person ses som en komponent i samhället, som anställd i ett företag eller som medlem i en förening. Alla dessa överordnade system (som givetvis själva kan ingå som komponenter i system på ännu högre nivå) ställer olika krav på vilka attribut som skall associeras med delsystemet.

Den samlade mängden data som via olika attribut kan knytas till delsystemet "person" är utomordentligt stort och genom en kombination, vägning och värdering av attributvärden är det möjligt att enkelt få fram en mer eller mindre grov och subjektiv "profil" av personen i fråga.[1]

Enkelt exempel: även den sparsamma personinformationen i figur 8. 2:2 kan samlat ge oss ytterligare information (subjektivt upplevd). Antag t ex att för en viss person X följande attributvärden registrerats:

---

1) För att bidra till debatten om individers integritet i databasers tidsålder har vi valt att illustrera datastrukturer m m med exempel som anknyter till samhällets sannolikt viktigaste komponent - den självständigt tänkande individen.

längd = 160 cm

vikt = 125 kg

hårfärg = skallig

Beroende på vår bakgrund, vår miljö, våra "värderingar" m m ger oss de dessa tre meddelanden olika information om X, i pragmatiskt avseende. En reaktion (tolkning) kan t ex vara slutsatsen att personen i fråga ej är speciellt tillfreds med sin tillvaro. En mer medicinskt-fysiologiskt inriktad individ kanske i stället reagerar med slutsatsen

1. personen X har troligen rörelsesvårigheter
2. personen X har troligen andningsbesvär
3. personen X borde bli föremål för medicinsk undersökning

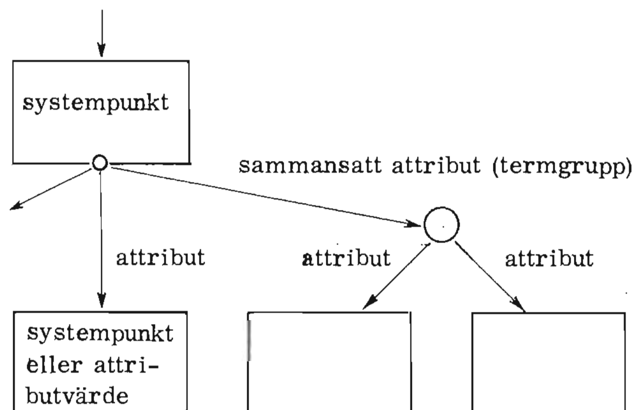
Det intressanta är här ej att olika slutsatser kan dras men att olika och för det mesta alltid subjektiva, värderingsbundna slutsatsdragningsmekanismer relativt enkelt kan byggas och låtas operera på datafiler.

Det torde vara lätt att inse att datafiler smidigt och enkelt kan nyttjas i positivt syfte, men att det är lika enkelt att använda dem på ett för individen och därmed samhället skadligt och kanske ödesdigert sätt.

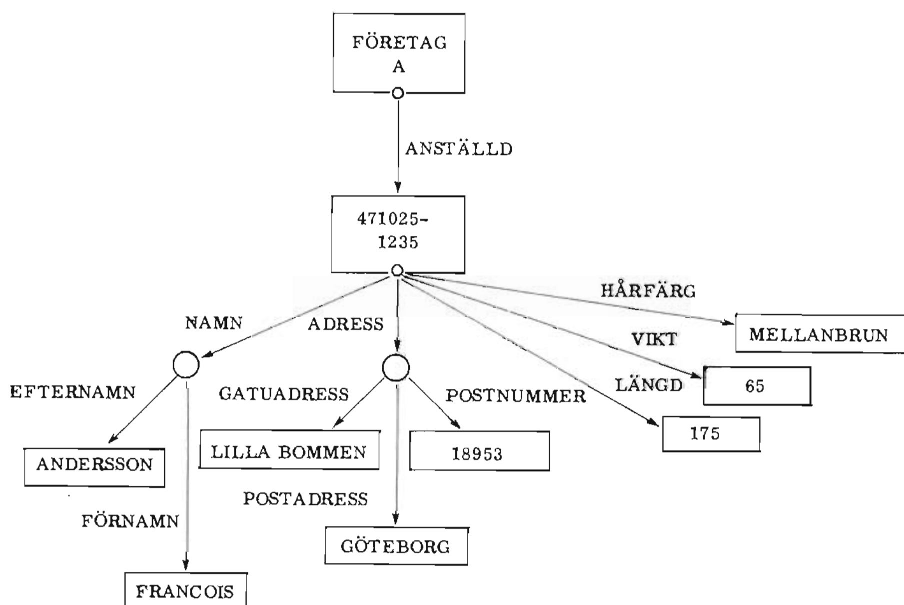
Figuren 8.2:2 antyder ingenting om hur data om personen skall representeras i ett lagringsmedium. Endast sambanden mellan olika termer och termgrupper är visade och vi har använt vedertagna[1] begrepp såsom attribut. Visserligen är dessa termer samlade i en "personpost" enligt figur 8.1:1 men denna visar endast exempel på ett av flera möjliga sätt att samla data om en person i lagringsbara enheter. Ett systemekonomiskt betingat alternativ kunde tänkas vara att samla personens namn och adress för sig i en posttyp och övriga data (längd, vikt, hårfärg) för sig i en annan posttyp. För att dessa två poster vid behov skall kunna entydigt bearbetas tillsammans, krävs en unik identifiering av posterna. Vad vi väljer som unik identifiering är normalt beroende på det överordnade systemet och diverse praktiska faktorer. Om det gäller en personalfil, resp kundfil för ett företag kan ett personalnummer resp ett kundnummer väljas att unikt, inom aktuellt system, identifiera personen. Men även personnumret kan i de flesta praktiska situationer väljas som identifikationsbegrepp och därmed kan en s k integreringsmöjlighet med andra personfiler i samhället skapas.

---

1) Här måste reservation göras beträffande vad som kan anses "vedertaget".



Figur 8. 2:3a
Möjlig grafisk representation av datastrukturer



Figur 8. 2:3b

I figur 8. 2:3a har vi valt en annan form att grafiskt representera en logisk datastruktur. Grundkomponenten i denna struktur är den "associativa trippeln" som kan uttryckas som "⟨attribute⟩ of ⟨object⟩ is ⟨value⟩" där ⟨object⟩ avser ett delsystem (eller systempunkt) och även ⟨value⟩ kan beteckna en systempunkt. Vi antar att varje systempunkt på något sätt[1] är unikt identifierbart - i vårt fall (när det gäller systempunkten "person) förslagsvis genom personnummer. Motsvarande gäller för attributnamnen.

Figuren 8. 2:3b är ett exempel på en datastruktur där ovannämnda representationsform tillämpats och aktuella attributvärden infogats. Om vi skulle vilja beskriva datastrukturen, dvs utan hänsyn till några aktuella delsystem och attributvärden, kunde det göras genom att vi angav mängden av attribut och deras succedensrelationer, t ex att möjliga succedenter fill "anställd" är {"namn", "adress", "längd", "vikt", "hårfärg"}.

De relationer som visats i figur 8. 2:3b kan sägas vara av "lokal natur" och centrera sig kring objektet "anställd person". En något mer omfattande datastruktur visas i figur 8. 2:4 där PN1, PN2, ..., PN5 betecknar fem individers personnummer. Vi har här introducerat ett antal olika relationer mellan de olika personerna samt mellan personerna och "diverse andra system" såsom företag och ideella organisationer. För att ej göra figuren alltför svårläst och, för vårt syfte, svåröverblickbar har en mängd ytterligare tänkbara attribut och relationer utelämnats eller endast vagt antytts. Observera igen att dataorganisationen i ett eller flera datorsystems sekundärminnen, avseende denna struktur, alls ej betraktats. Endast de logiska sambanden är studerade. Återigen är vi hänvisade till att använda begrepp och beteckningar som är i tillräcklig grad vedertagna (inom betraktad systemram) såsom

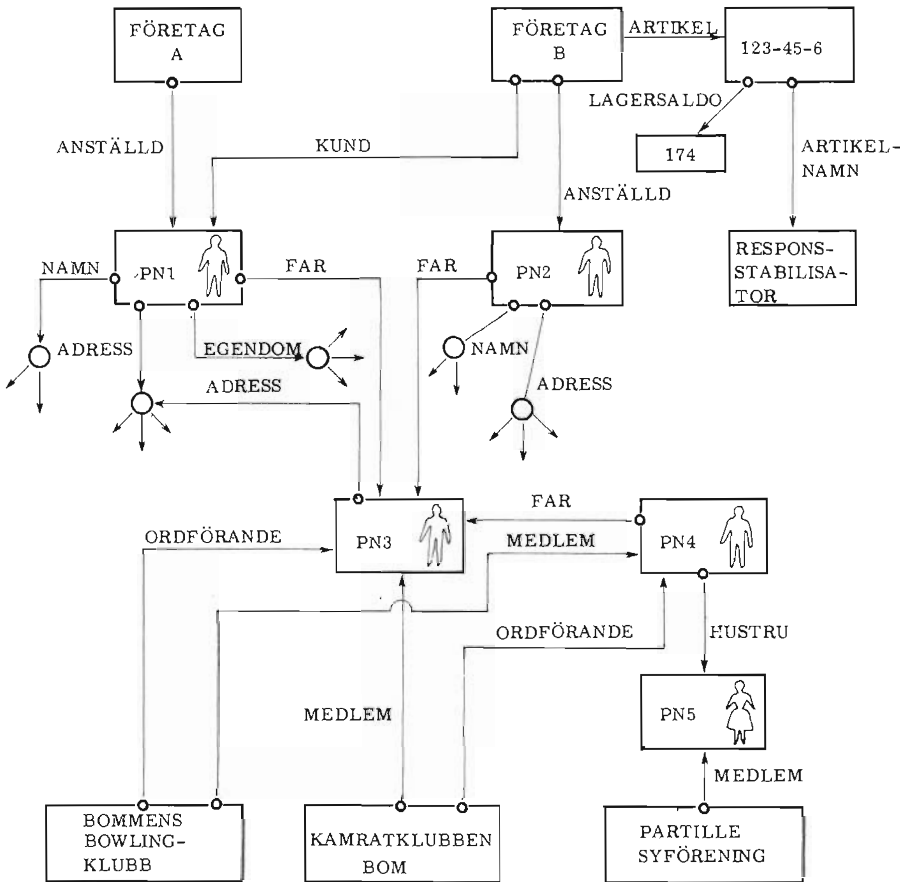
- KUND
- LAGERSALDO
- ORDFÖRANDE
- MEDLEM
- FAR

osv

Om samtliga uppgifter enligt 8. 2:4 återfinnes i ett enda integrerat databassystem öppnas önekligen möjlighet till intressanta utsökningar och, som vi tidigare illustrerat, mer eller mindre subjektiva värderingar

---

1) Detta berör det s k "namngivningsproblemet" som dock ej behandlas i denna bok men som är av fundamental betydelse i samband med databaser.



Figur 8. 2:4
Några tänkbara relationer mellan personer, företag, organisationer.

och "slutsatser" därav. Eftersom syftet med denna bok ej är att behandla denna problematik i detalj (vilken dock bör förtjäna betydligt mer uppmärksamhet i andra sammanhang) anger vi här blott några exempel (läsaren uppmanas att följa figuren och konstatera sannolikheten i nedanstående subjektiva utsagor):

1. PN1, PN2 och PN4 är bröder, eller åtminstone halvbröder

2. PN1 bor troligen i föräldrahemmet

3. Far (PN3) och son (PN4) har ett stort intresse för föreningsverksamhet och torde ägna åtskillig kvällstid åt denna

4. Det är troligt att PN4's hustru har funnit det ensamma kvällslivet hemma tråkigt och därför gått med i en syförening

5. Fru PN5 tycker ej om bowling

6. PN1 har (i sin folk- och bostadsräkningsuppgift) ej markerat huruvida han äger någon responsstabilisator. Finns det skäl att undersöka om lagbrott begåtts?

Naturligtvis är den "databas" som skisserats alltför informationsfattig för att med någon nämnvärd grad av sannolikhet kunna dra slutsatser och göra utsagor. En icke alltför stor ökning av attribut och relationer skulle dock öka denna utsago-kapacitet betydligt. En fara ligger i den eventuella frestelsen att låta subjektiva värderings- och utsago-algoritmer operera på en databas av detta slag.

Exempel på några slag av utsökningar som generellt kan göras på en datastruktur av typen enligt fig 8. 2:4 kan omnämnas:

1. Samtliga anställda i företaget A?

2. Vilka bor på en given adress?

3. Vilka är anställda i företag A och samtidigt kunder i företag B?

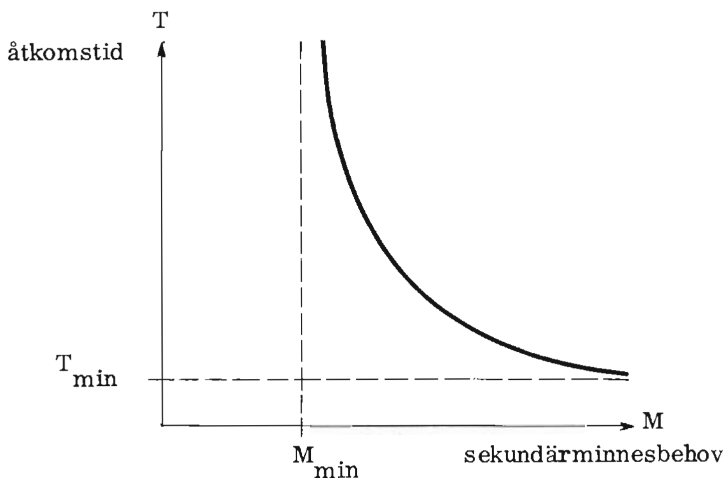
4. Vilka anställda i företag A har fäder som spelar bowling?

osv.

Operativsystems filhanteringssystem kan normalt ej utan vidare varken lagra eller operera (enligt ovan) på en associativ datastruktur. Man är då vanligen hänvisad till att i operativsystemet inkorporera speciella databashanteringspaket eller skriva egna program för detta ändamål.

8.3 Dataorganisation

När vi talar om ett operativsystems funktioner för organisation av datamängder (filer) och åtkomst av data i dessa, förutsätts vanligtvis att filerna är namngivna och innehåller ett antal poster som alla har samma postbeskrivning. Posterna sammanhör med ett sökbegrepp, dvs en av deras termer används för att identifiera posten och anknyta dess övriga uppgifter (termer) till någon punkt i objektsystemet. Bestämning av postens innehåll och disposition ur lagringssynpunkt är hänvisat till användaren och kan ses som ett utrustningsanpassningsmoment i samband med systemering. Normalt har användaren en logisk datastruktur, t ex enligt fig 8.2:3b eller 8.2:4 att utgå ifrån och postkonstruktionen får ses som en till stor del intuitiv process där konstruktörens erfarenhet, insikter i databehandlingsteknik m m samt strävan att balansera mellan rimliga minnesbehov och krav på korta åtkomsttider styr arbetet (figur 8.3:1). Det är dock ej denna boks syfte att i detalj diskutera dylika konstruktionsproblem. Något kan dock belysas och vi väljer persondatastrukturen enligt figur 8.2:3b som underlag för några exempel.



Figur 8.3:1

Figuren avser att illustrera att korta åtkomsttider kan åstadkommas på bekostnad av sekundärminnesutrymme. Omvänt kan en fil "packas" så att ett nära 100%-igt minnesutnyttjande åstadkommes, men då på bekostnad av åtkomst- och även processortid.

Med utgångspunkt från 8. 2:3b skall vi belysa olika möjligheter att utforma poster. Tilläggas bör att endast en liten del av det totala antalet olika möjligheter har betraktats.

Följande elementarposter kan skapas (8. 3:2).

Posttyp 1	företag	personnr.	
exempel:	A	471025-1235	
Posttyp 2	företag	personnr.	efternamn
exempel:	A	471025-1235	ANDERSSON
Posttyp 3	företag	personnr.	förnamn
exempel:	A	471025-1235	FRANCOIS
osv	⋮		
Posttyp 9	företag	personnr.	hårfärg
exempel	A	471025-1235	MELLANBRUN

Figur 8. 3:2
Elementarposter baserade på strukturen i figur 8. 2:3b. Observera att posttypen 1 uttrycker sakförhållandet att person 471025-1235 är anställd i företaget A. Om samtliga filer tillhör företaget A är uppgiften om detta (i posterna) redundant.

Det är rent teoretiskt ingenting som hindrar att vi lägger upp vårt system enligt ovan med en stor mängd elementarfiler innehållande elementarposter. För att sedan kunna utföra bearbetning måste ett lexikon på något sätt upprättas så att vi vet vilka uppgifter som finns i vilka filer, eller också får detta implicit beaktas inom varje program.

I praktiken kan en datorrepresentation enligt 8. 3:2, främst av datatransport- och sekundärminnesekonomiska skäl, ej accepteras. Del av kon-

struktionsarbetet utgöres därför av en (mer eller mindre intuitionsstyrd) konsolidering (hopslagning-sammansmältning) av elementarposter (1). Kombinatoriskt sett kan här givetvis en stor mängd konsolideringar tänkas. Praktiskt sett kan åtskilliga konsolideringar genast uteslutas. Det som styr konsolidering är informationssystemets (och applikationsprogramsystemets) processtruktur (1). Accessfrekvensen för de olika e-posterna[1] och accessernas samhörighet är här några avgörande faktorer. Om vi till exempel räknar med 100 accesser per tidsenhet till "efternamn"-uppgifter och en access per tidsenhet till data om "hårfärg" inses att om dessa två e-poster sammanslås "hårfärgsdata" transporteras till primärminnet 99 eller 100 gånger i onödan[2]. Några tänkbara konsolideringar är visade i figur 8.3:3.

Posttyp 1'	Före-tag	pers. nr.	efter-namn	för-namn	längd	vikt	hår-färg
Posttyp 2'	Före-tag	pers. nr.	efter-namn	för-namn	gatu-adress	post-adress	post-nummer

Figur 8.3:3a
Posttyp 1' är en konsolidering av typerna 2, 3, 7, 8 och 9.
Posttyp 2' är en konsolidering av typerna 2, 3, 4, 5 och 6.
Observera att namn-data dubbellagrats - kanske av åtkomsttidsskäl.

Posttyp 1''	Före-tag	pers-nr.	namn-data	adress-data	längd	vikt	hår-färg

Figur 8.3:3b
En total konsolidering av datastrukturen enligt fig 8.2:3b.

1) I fortsättningen förkortas "elementar" med "e".

2) Langefors har i ref. (1) en utförlig diskussion om datatransportminimering i samband med systemkonstruktion. Vi får 100 "onödiga" transporter om "hårfärg" aldrig behövs i samma process som "efternamn".

Lagring av poster i ett datamedium

Antag att en post har den principiella uppbyggnaden

ID, T1, T2, T3, T4, ..., Tn

där ID = identifikationsbegrepp och T1, ..., Tn betecknar postens termer som var och en upptar ett datafält av visst längd. Ett vanligt sätt att lagra posten är att ha samtliga termer samlade i konsekutiva minnesplatser och i den ordning som angivits i beskrivningen ovan.

ID	T1	T2	....	Tn

Varje term (inkl ID-begreppet) kräver härvid ett visst minnesutrymme som kan vara av fast eller variabel längd. Lagringssättet är nödvändigt vid sekvensiella lagringsmedia (hålremsa, hålkortsfiler, magnetband). Om posten kan lagras på lagringsmedia av direktåtkomsttyp är variationsmöjligheterna flertaliga. Posten kan lagras i konsekutiva minnesplatser som i det sekvensiella fallet. Andra alternativ innebär att postens termer indelas i två eller flera segment (eller grupper) vilka sedan kan lagras "spridda" över ett eller flera direktminnesenheter. Exempel på dylik spridd lagring är visat i figur 8.3:4a och b. Två alternativ är visade där termerna har indelats i segment S1 = {T1, T2, T3}, S2 = {T4, T5, T6, T7} och S3 = {T8, ..., Tn}. Om vi betecknar åtkomstfrekvensen med f_S kan följande antagande göras

$$f_{S1} \gg f_{S1 \cup S2} \gg f_{S1 \cup S2 \cup S3} \quad ^{1)}$$

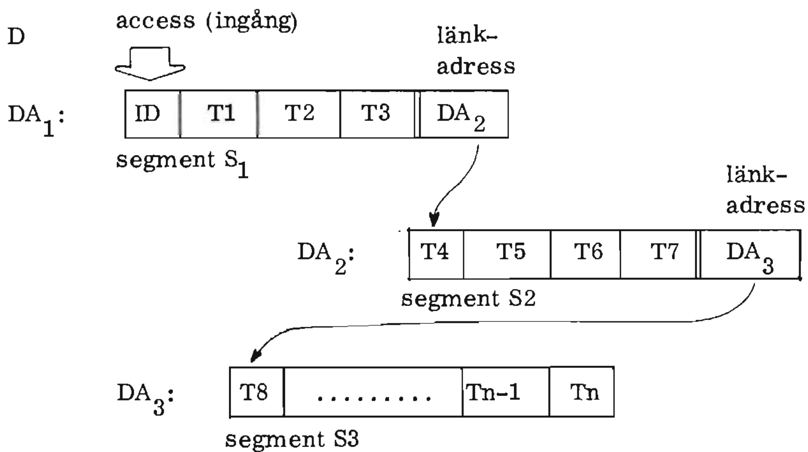
Ovanstående säger att åtkomstfrekvensen till posten i sin helhet (unionen av S1, S2 och S3) är avsevärt lägre än frekvensen till de första 3 termerna (S1).

En minnesbesparande åtgärd skulle då vara att placera S3 på ett långsammare (avs åtkomsttid), men också billigare, lagringsmedium. Om vi antar att DA_i är adressen till en plats på ett direktminne "i" med genomsnittliga åtkomsttiden t_i blir åtkomsttiden till en term T_i (approximativt[2])

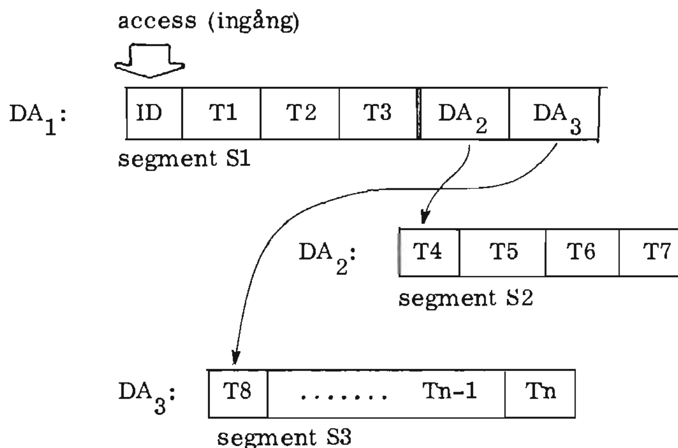
---

1) >> innebär "mycket större än".

2) Approximativt - ty åtkomsttiden är en funktion av bl a åtkomstmetod och sekundärminnets utnyttjandegrad.



Princip a



Princip b

Figur 8.3:4
Två alternativa länkade lagringar av en post i ett direktminnet. DA_1, DA_2 och DA_3 betecknar direktminnesadresser och kan generellt sett avse skilda minnesenheter. Ingång till posten antas via DA_1 genom en åtkomstmetod (dessa diskuteras längre fram) som baseras på ID-begreppet.

	princip a	princip b
$T_i \in S1$	t_1	t_1
$T_i \in S2$	$t_1 + t_2$	$t_1 + t_2$
$T_i \in S3$	$t_1 + t_2 + t_3$	$t_1 + t_3$

Varje länk antas här kräva en extra access.

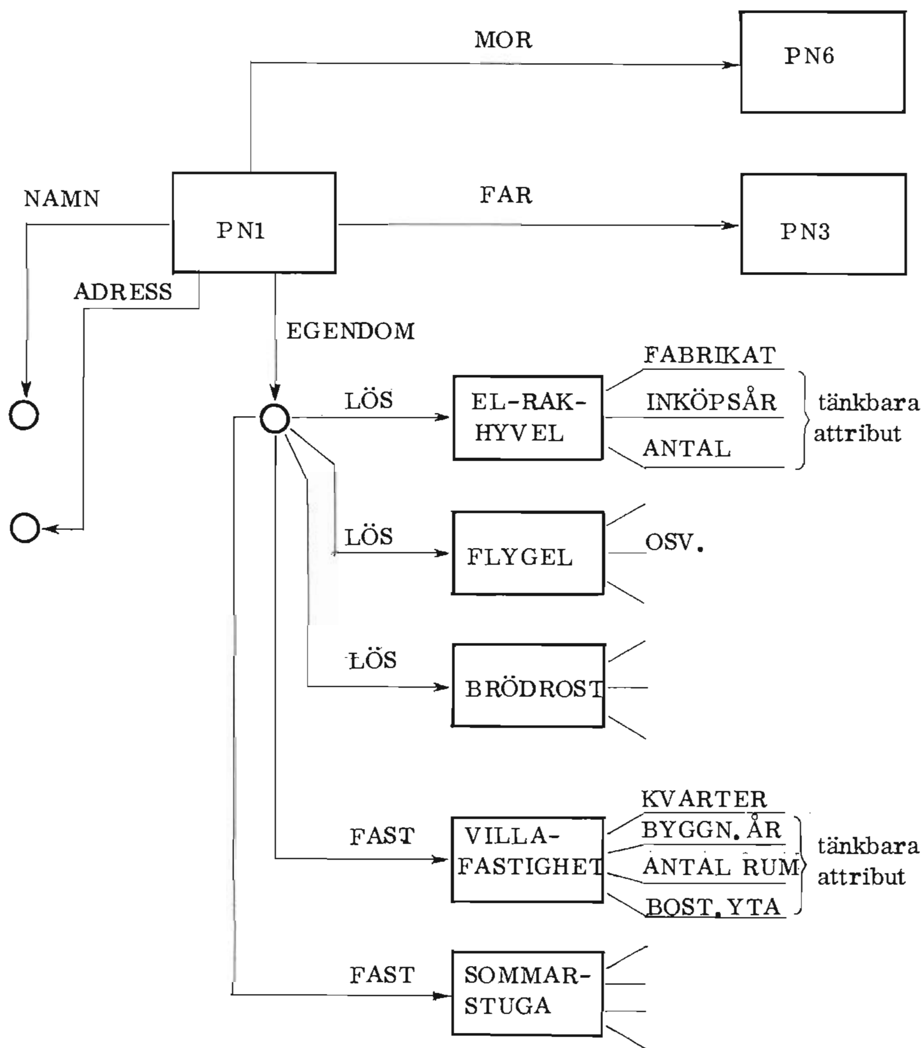
Skillnaden mellan principen a och b är länkadressernas placering. Om poststrukturen är fast, dvs samtliga termer T1, ..., Tn alltid är närvarande (eller plats för dem alltid reserveras), kan principen b vara att föredra av åtkomsttidsskäl. Om posten däremot indelats i ett stort antal segment och många av dessa ej innehåller aktuella data och därför ej reserveras plats blir länkadressdelen enligt b av variabel längd och därmed blir hela S1 variabelt. För att underlätta programmeringen kan då ibland fast post-segmentstruktur eftersträvas och princip a föredras.

Representation av poster med starkt variabel struktur och/eller relationer till andra poster

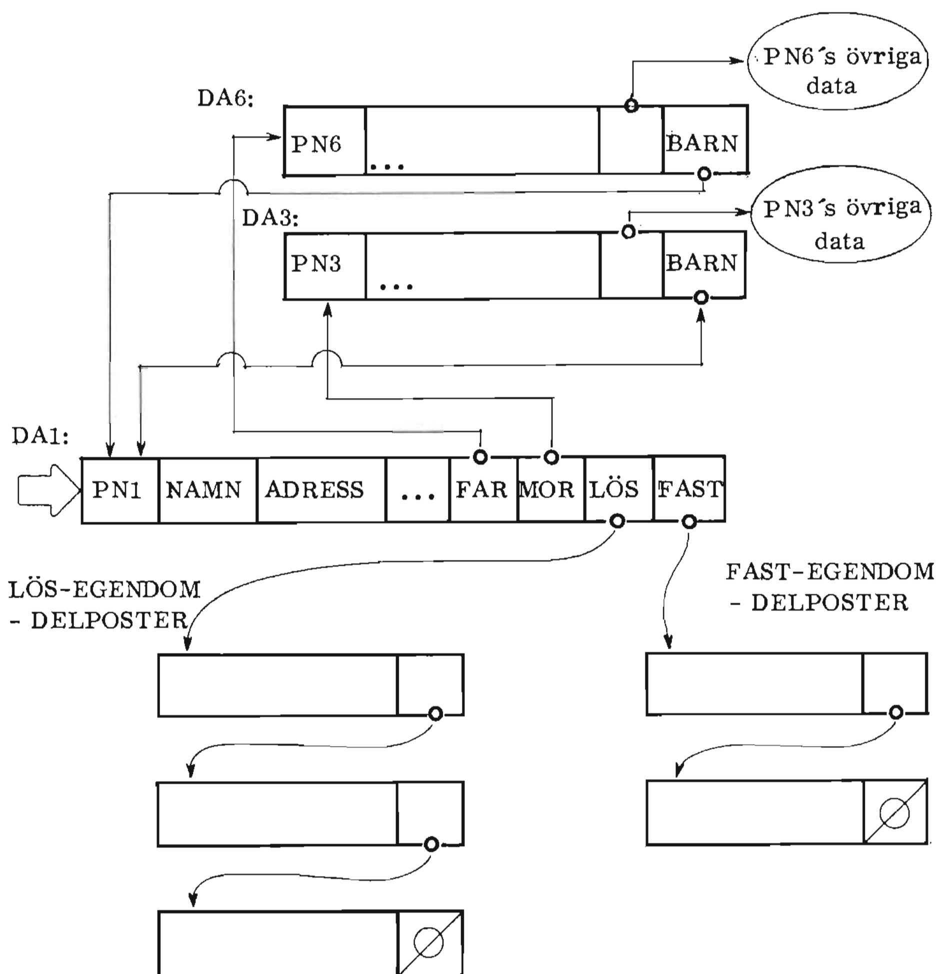
Integrationssträvanden inom ADB och önskemålet att i en databank kunna lagra stora mängder data om objektsystemet resulterar i filer där posterna dels har starkt variabel struktur, och dels har ett variabelt antal skilda relationer olika poster emellan. Om data om ett system av typen enligt figur 8.2:4 skall lagras i en databank och skall tillmötesgå olika krav på utsökning, inses att en tämligen komplicerad lagringsstruktur erhålles. Figur 8.3:5 visar ytterligare några uppgifter som kan tänkas knutna till ett personnummer. Eftersom egendomsdata från person till person kan variera starkt är en fast lagringsstruktur här sannolikt oekonomisk i de flesta fall. Givetvis finns inga standardmetoder för hur man skall förfara i ett sådant sammanhang.

Den vanligaste metoden torde vara att införa en mängd delposter, av ett erforderligt antal olika slag, och sedan länka dessa till huvudposten. Länkningen kan exempelvis utformas som en kedja (fig 8.3:6) eller genom att i huvudposten tillåta ett variabelt antal länkadresser (figur 8.3:4b).

När det gäller relationer till andra poster (dvs ej "egna delposter") kan dessa ibland även vara åtkomliga via sitt identifikationsbegrepp - i vårt fall personnumret. Man står då inför valet huruvida länkningen skall ske via ID-begreppet eller genom att lagra den relaterade postens "exakta" adress i sekundärminnet. Sker länkning via direkt adress åstadkommes normalt en åtkomsttidsvinst jämfört med länkning via ID-begreppet efter-



Figur 8.3:5
En utökning av datastrukturen i figur 8.2:4.

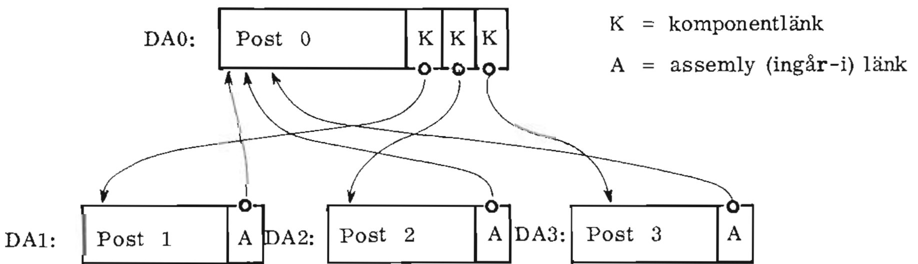


Figur 8.3:6
Tänkbar maskinrepresentation av datastrukturen i fig 8.3:5.

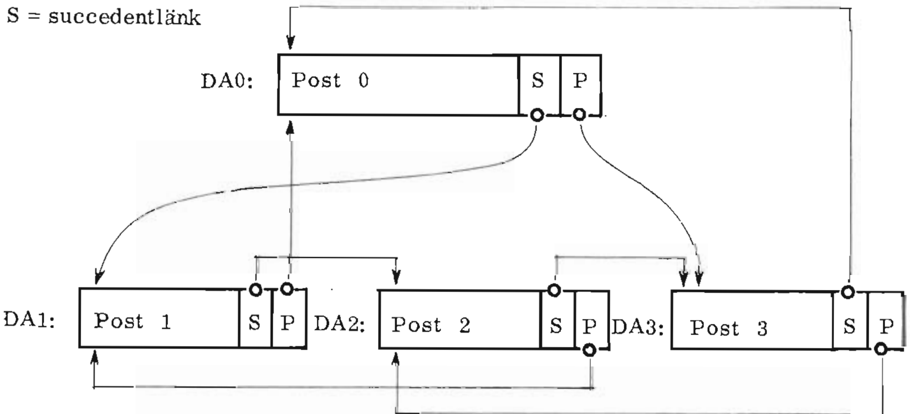
som en sökprocess i det senare fallet måste genomlöpas. Nackdelen med den direkta länkningen är främst att vid flyttning av den relaterade posten eller reorganisation av filen ett omfattande arbete måste utföras för att ändra alla berörda länkadresser. Andra faktorer som kan påverka valet här är utrymmesbehovet för ID-begrepp vs. länkadresser. Till

exempel i figur 8.3:6 kan FAR resp MOR utgöras av antingen ID-begreppen PN3 resp PN6 eller av de "direkta" adresserna till motsvarande poster, DA3 resp DA6.

Två andra metoder att länka poster till varandra är visade i figur 8.3:7 och 8.3:8. Den första figurens länkningar indikerar att man här sannolikt har räknat med behov av åtkomst "parvis" enligt (0,1), (0,2) och (0,3). Om man till primärminnet har läst in post 1 och behöver post 3 kan denna kommas åt via post 0, dvs sannolikt på bekostnad av extra access till sekundärminnet. Länkningen innebär sålunda en trädstruktur.



Figur 8.3:7



Figur 8.3:8

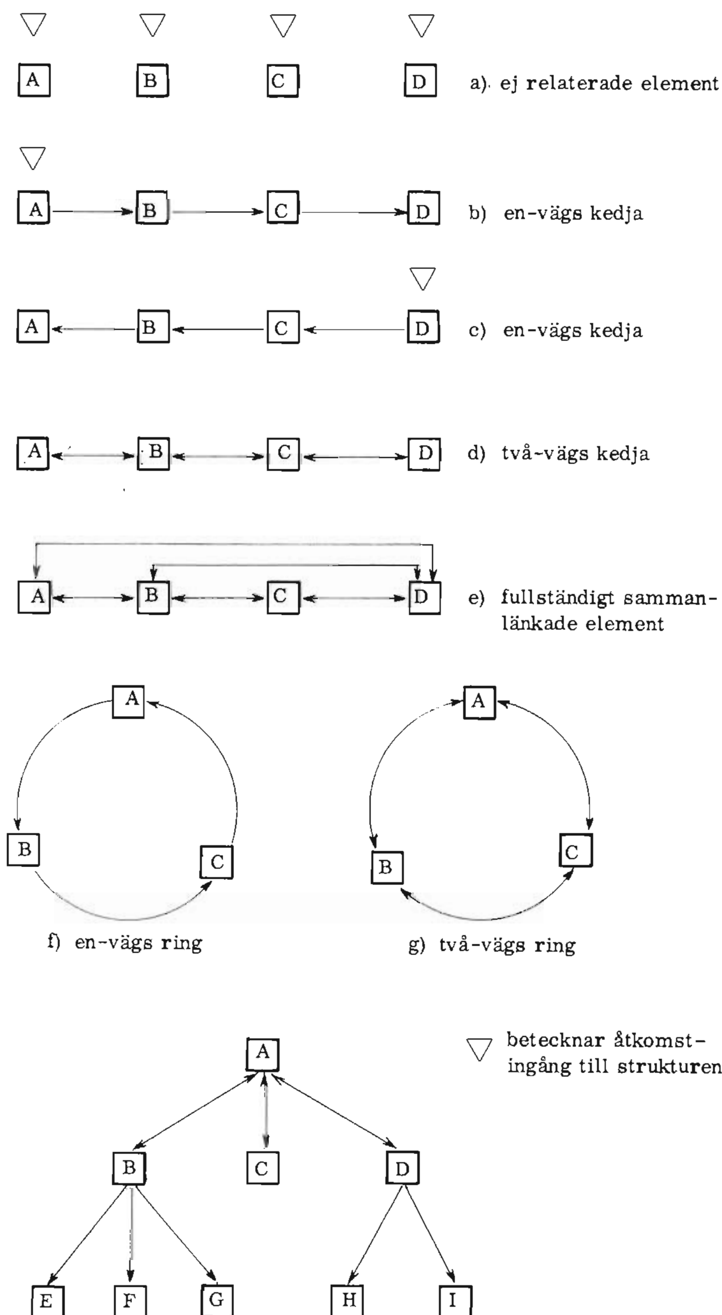
I figuren 8.3:8 kan vald länkningsprincip indikera användning i samband med någon form av "kö" eller "stack". Man kan anta att post 0 representerar ett "baselement" och posterna 1, 2, 3 till 0 anslutna element[1]. Länkningen, här i två riktningar, gör det möjligt att avsöka listan fram- eller baklänges. Dock kan från ett element (ej det första eller det sista) en direkt åtkomst till baselementet ej göras.

Om posterna 0 t o m 3 har egna unika identifieringsbegrepp och sålunda är åtkomliga via en sökprocess kan även här länkadresserna (S, P, A och K) utgöras av motsvarande ID-begrepp.

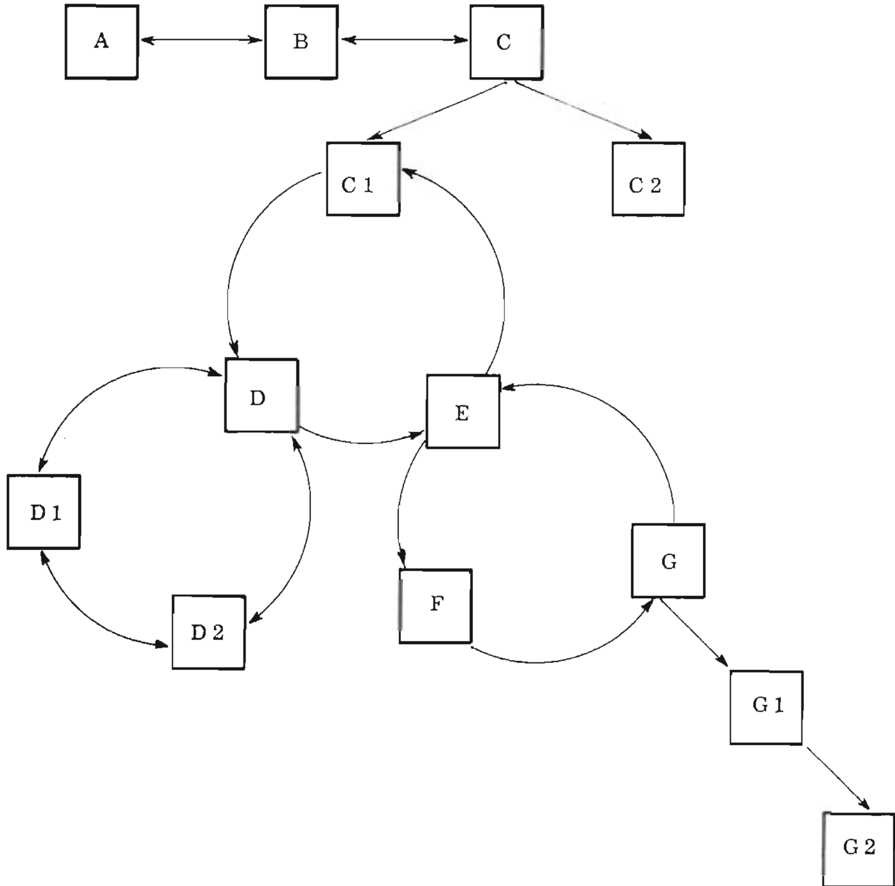
I figur 8.3:9 visas några grundläggande metoder att länka dataelement mellan vilka samband existerar. I fallet a) - ej relaterade element, måste varje element normalt ha någon form av unik identifiering så att det kan utsökas. Vid en envägs lista måste åtminstone det "ledande elementet" kunna identifieras. Vid flervägs sammanlänkade strukturer (d, e, f, g) kan länkarna följas i olika riktningar och lämpligt (eller lämpliga) "ingångselement" beror därför på applikationen. En kombination av olika grundläggande länkade strukturer är visad i figur 8.3:10.

---

1) Givetvis kan dessa i sin tur utgöra baselement för sig "underordnade" element osv.



Figur 8.3:9
Några grundläggande metoder att länka "besläktade dataelement.



Figur 8.3:10
En "kombinerad" länkningsstruktur

8.4 Åtkomst- och utsökningsprinciper

I detta avsnitt bortser vi från det triviala fallet "utsökning efter nästa post" vid sekvensorganiserade (fysiskt, eller genom länkning) filer och betraktar

(1) utsökning efter adress

(2) utsökning efter namn

(3) utsökning efter egenskap och efter formell beskrivning.

Operativsystems filhanteringssystem kan vanligen ombesörja utsökningar enligt (1) och (2). Principen (3) är något mer "avancerad" och återfinnes ännu så länge endast hos vissa fristående databashanteringssystem. Arbete pågår dock hos flera leverantörer att inkorporera även denna utsökningsmöjlighet i resp operativsystem.

Utsökning efter adress

Med adress avses här antingen motsvarande posts fysiska adress i sekundärminnet eller dess logiska adress. Den fysiska adressen uppbyggs normalt, beroende på typ av sekundärminne, av komponenter såsom

- enhetsnummer
- cylinder (accessgrupp)
- spår
- sektor (eller postens relativa plats på spåret).

Normalt arbetas här på block-nivå, dvs posten antas uppta hela blocket.

Med logisk adress anses vanligen motsvarande posts eller blocks ordningsnummer i filen. Operativsystemets filhanteringsrutin har då att utföra en enkel omräkning från logisk till fysisk adress och användaren behöver här ej känna till filens exakta, fysiska lagringsplats och lagringsstruktur. Utsökning efter adress innebär mest arbete för användaren men ger samtidigt den bästa möjligheten att bygga egna avancerade filstrukturer (jfr föregående avsnitt 8.3).

Utsökning efter namn

I denna kategori inräknar vi

- utsökning genom access-tabeller (index-tabeller)
- utsökning genom adressberäkning.

Accesstabellprincipen innebär att med postens sökbegrepp givet, sökning i en hierarki av tabeller på olika nivåer göres. Successivt närmar man sig därigenom det fysiska läget av den sökta posten. Exempelvis kan den första tabellen ge besked om i vilken del av direktminnet posten finns och hänvisa till en nästa tabell, som ger närmare anvisning om vilken cylinder (eller motsv) som är att söka samt adress till nästa tabell, som kan ge besked om på vilket spår den sökta posten befinner sig. Det inses att denna åtkomstmetod normalt kräver ett antal accesser ≥ 2 till direktminnet beroende på antal tabellnivåer och deras placering. Om metoden skall bli tillräckligt effektiv förutsätts att filen ej ständigt undergår förändringar genom att poster raderas eller lägges till. Förändringar skapar luckor och flera länkar till överskottsutrymmen, dvs dels ett sämre minnesutnyttjande och dels längre åtkomsttider. En omorganisation av filen inkl tabellerna är därför med vissa tidsintervaller nödvändig.

Det är vanligt att posterna i en fil enligt denna åtkomstmetod är sekvensiellt organiserade efter sökbegreppet (dock med länkar till överskottsutrymmen). Filen kan därför även bearbetas sekvensiellt. Lagrings- och åtkomstsättet kallas då vanligen för "index-sequential data set (file) organization".

Sekvensorganisation är dock ej något krav för principen i fråga och filen kan generellt sett ha godtycklig organisation av posterna. Som sökbegrepp kan godtycklig term i en post väljas. Om sökbegreppet ej är unikt associerat med posten i fråga bör tabellsökningen resultera i en mängd av länkar till sådana poster som tillfredsställer sökbegreppet. Antag att vi som sökbegrepp för en fil med poster enligt figur 8.3:3a väljer "efternamn". En sökning här bör resultera i en mängd av poster som tillfredsställer exempelvis "efternamn" = ANDERSSON. Dylika index-filer kallas vanligen för nyckelfiler. För varje sökbegrepp kan en sådan nyckelfil upprättas och därvid en egenskapsstyrd sökning genomföras.

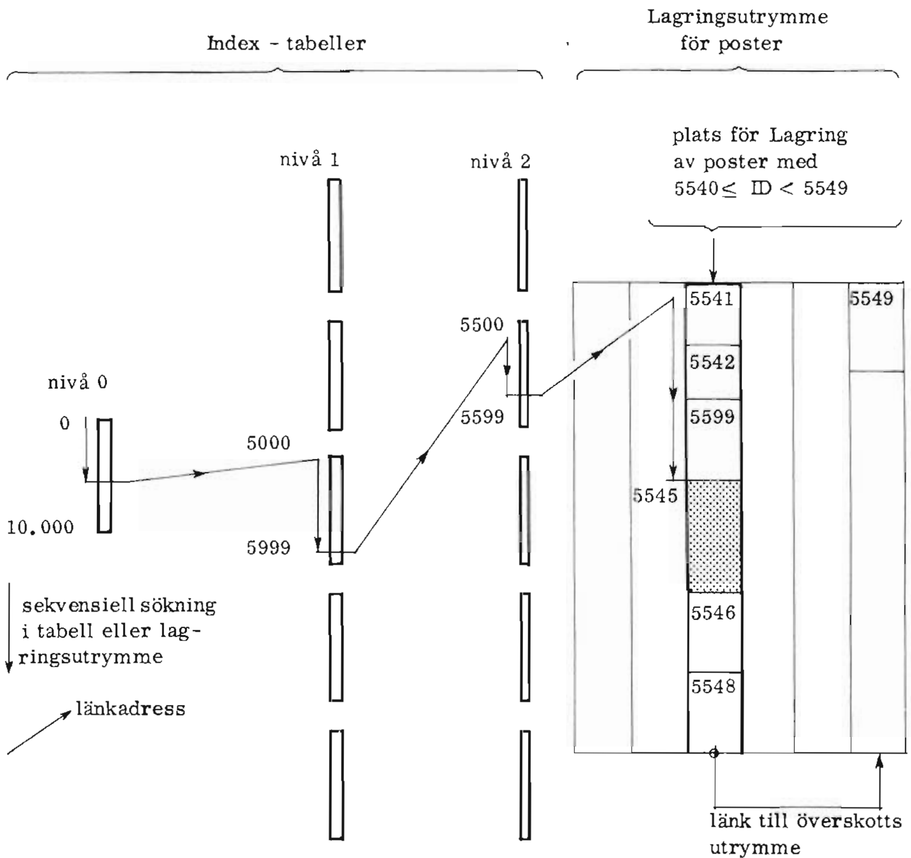
Utsökning genom adressberäkning kallas även för pseudoadressering eller randomiserad adressering. Metoden bygger på principen att beräkna postens adress i sekundärminnet som funktion av dess ID-begrepp.[1]

$$DA_i = F(ID_j)$$

Adresseringen är ej omvänt entydig. Det finns alltid en viss sannolikhet att flera olika ID-begrepp leder till samma direktminnesadress. Därför

---

1) Med ID-begrepp avses här generellt en term i posten eller godtycklig kombination av termer eller term-delar. ID-begreppet behöver ej vara unikt för posten i fråga för att metoden skall kunna tillämpas.



Figur 8.4:1
Indexerad utsökning efter namn. Sökt post har det numeriska ID-begreppet 5545.

måste vanligen en sökning företas efter det att man kommit fram till den beräknade minnesplatsen. Detta kan innebära extra accesser till andra minnesplatser.[1] Metoden finns mer i detalj beskriven i (2) och (3). En "god" adressberäkningsalgoritm anses vara en sådan som fördelar filens

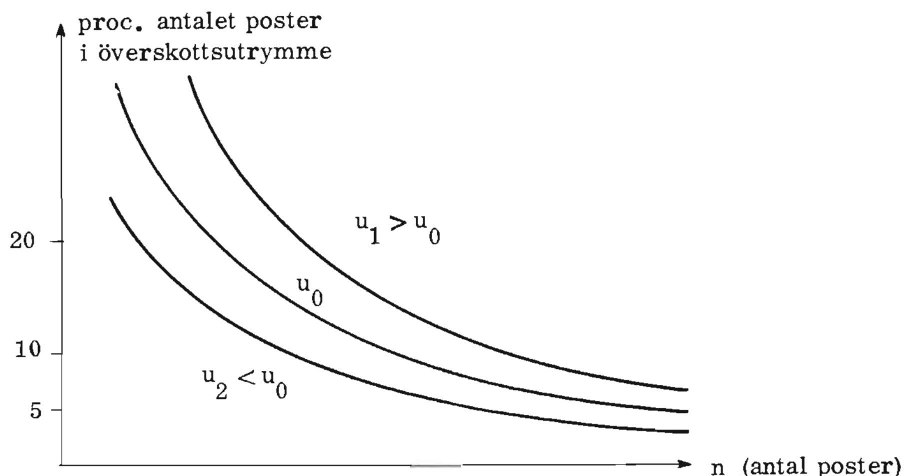
1) Dvs om posterna ej får plats inom beräknad minnesplats, lagras "överskottsposter" i ett annat utrymme (se (3)).

ID-begrepp så jämnt som möjligt över tillgängliga minnesplatser. I de flesta fall innebär adressberäkningen en "slumptransformation" **av** ID-begreppet till en av de direktminnesadresser som reserverats för filen i fråga. Om filen tilldelats N adresserbare platser i sekundärminnet $A_1, A_2, \ldots, A_N$ är en "god" algoritms egenskap den att ett ID-begrepp med en lika stor sannolikhet ($p = 1/N$) avbildas till någon av adresserna $A_1, \ldots, A_N$.

En fördel med utsökning genom adressberäkning kan sägas vara dess enkelhet vid programmering och relativt snabba åtkomst genom att en access till sekundärminnet i många fall är tillräcklig för att lokalisera den sökta posten. Antal erforderliga accesser är normalt beroende på det förväntade antalet poster i överskottsutrymmet. Detta beror i sin tur på

- antal poster i filen (M)
- antal minnesplatser (N) för adressberäkning
- minnesplatsens storlek, uttryckt i antal poster (n)
- minnesplatsens utnyttjandegrad u (N/M, vanligen $< n$)
- antal överskottsplatser m m.

Det förväntade procentuella antalet av filens poster som kommer att lagras i överskottsutrymmen blir större ju större utnyttjandegrad eftersträvas (figur 8.4:2).

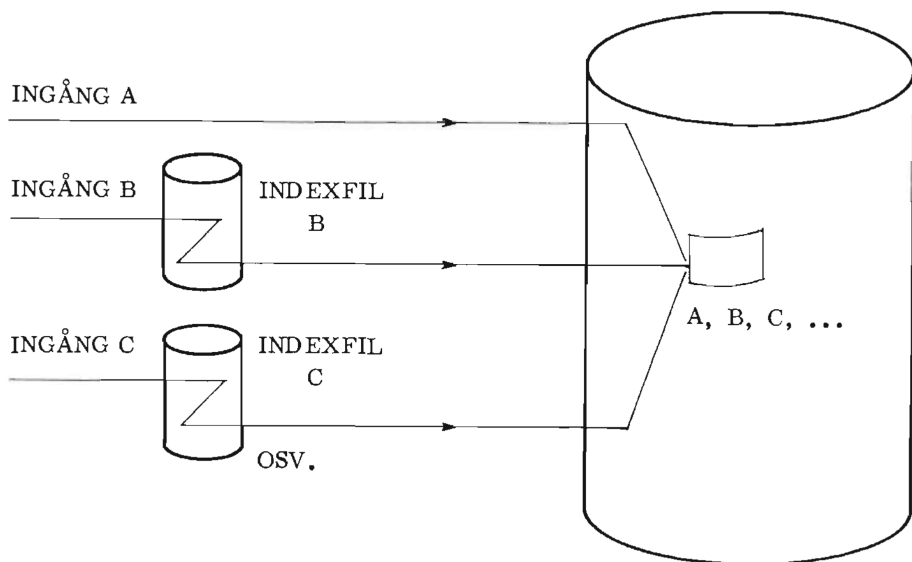


Figur 8.4:2
Det procentuella antalet poster i en fil som lagras i överskottsutrymme är beroende av platsstorleken (n) och platsens utnyttjandegrad (u). Figuren visar enbart tendensen. Exakta samband återfinnes bl a hos (4).

Som framgår av fig 8.4:2 kan metoden bli relativt accesskrävande vid små minnesplatser och höga värden på u. En annan egenskap som ibland kan anses som ofördelaktig är att posterna i filen ligger fysiskt blandade och ej utan vidare kan bearbetas sekvensiellt. Ibland kan ID-begreppets uppbyggnad vara sådan att en god slumptransformation är svår att finna och utnyttjandet av filens minnesplatser kan då bli mycket ojämnt.

Operativsystems filhanteringsrutiner omfattar ej ofta utsökning genom adressberäkning och användaren är här normalt hänvisad till egna adressberäkningsrutiner och därefter "utsökning efter adress".

Vi observerar att metoden kan bli relativt effektiv om en minnesplats kan rymma ett stort antal poster. Detta är fallet om postlängden kan göras mycket kort vilket är vanligt vid indexfiler där posten normalt innehåller sökbegrepp plus länkadress till sökt post. En fils poster kan organiseras för direkt åtkomst via adress eller adressberäkning för ett sökbegrepp A och för indirekt åtkomst via indexfiler avseende sökbegreppen B, C, ... osv (figur 8.4:3). Indexfilerna kan ofta med fördel organiseras för åtkomst genom adressberäkning.



Figur 8.4:3
En post i filen antas direkt åtkomlig via sökbegreppet A. För andra sökbegrepp B, C, ... upprättas indexfiler. Om sökbegreppen ej är unika ger en utsökning multipla svar.

Utsökning efter egenskap och efter formell beskrivning

Vid informationssökning (information retrieval) kan en post "i" sägas karaktäriseras av en mängd söknycklar E_i t ex

$$E_i = \{A, P, Q, W\}$$

I enklare fall är dessa nycklar ej associerade med numeriska eller symboliska värden, utan nycklarna kan sägas associera posten med en delmängd (E_i) av universalmängden (E) av söknycklar. Sålunda är $E_i \subseteq E$ och två poster i och j kan sägas ha termer gemensamt och $E_i \cap E_j \neq \emptyset$, där $\emptyset$ betecknar den tomma mängden. Ovanstående problemställning är aktuell vid litteratursökning där E_i anger den mängd av nyckelord som en publikation associerats med. En utsökningsinstruktion kan då t ex få beskrivningen

$$A \wedge P \wedge (B \vee Q) \wedge \neg W$$

som kan tolkas som

> "sök samtliga poster där nyckeln A och P och antingen B eller Q förekommer. Uteslut härvid sådana poster där även W förekommer."

Det inses att utsökningsoperationer av detta slag är mycket resurskrävande om antalet poster och söknycklar är stort. Principen att organisera en fil för utsökning av detta slag innebär att skapa mängder av referenser till poster som innehåller en viss söknyckel. Kalla mängden av referenser till poster med nyckeln A för R_A.

Den aktuella återvinningen av posterna föregås då av en operation på dessa mängder. I exemplet ovan bildas mängden R.

$$R = R_A \cap R_P \cap (R_B \cup R_Q) \cap W'$$

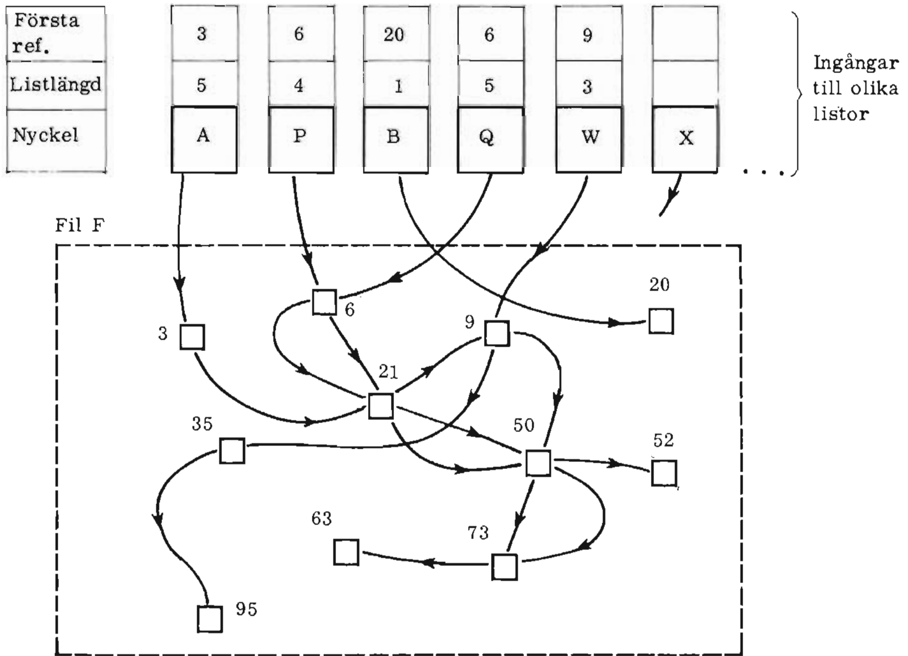
där W' betecknar komplementmängden till W. Därefter utmatas samtliga poster vars referenser återfinnes i R.

Praktiskt kan dessa operationer på mängderna implementeras på olika sätt. Referens (5) (Lefkowitz) diskuterar flera praktiska lösningsmetoder av vilka vi här kortfattat betraktar två ur filorganisationssynvikel. Den första metoden förutsätter att filen är organiserad enligt "multi-list" principen. Den andra bygger på principen "invertering av en fil".

I multi-listfallet inlänkas varje post i ett antal listor som bestäms av an-

talet av dess nyckelord (figur 8.4:4). Utsökning sker genom avsökning av de listor som representerar motsvarande nyckelord. Alla listor behöver dock normalt ej avsökas. I exemplet (fig 8.4:4) behöver endast den kortaste listan (P) avsökas och varje post i denna undersökas m a på övriga nycklar A, B, Q och W.

Om vi upprättar en fil F^{-A} som består av referenser till alla poster i en fil F, vilka innehåller en nyckel (eller ett attribut) A, kallar vi F^{-A} för en "invers av F med avseende på A". Några sådana inverterade filer är visade i figur 8.4:5. Dessa motsvaras av "referensmängderna" enligt tidigare diskussion.

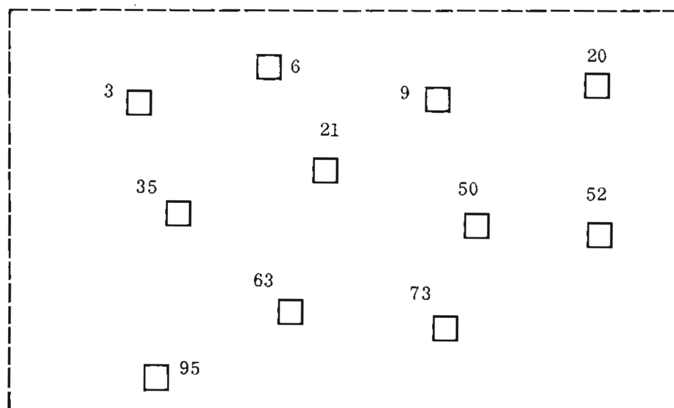


Figur 8.4:4
Multi-list organisation av fil. Varje post är inlänkad i en eller flera nyckel-ordskedjor. Utsökningen $A \wedge P \wedge (B \vee Q) \wedge \neg W$ ger posterna med ref. nr (eller adress) 21 och 50.

	F^{-A}	F^{-P}	F^{-B}	F^{-Q}	F^{-W}	F^{-X}
Nyckel	A	P	B	Q	W	X
Referenser till poster som har denna nyckel	3 9 21 50 73	6 21 50 52	20	6 21 50 63 73	9 35 95	

. . . . Inverterade filer

Fil F



Figur 8.4:5
Inverterad filorganisation. Varje nyckel är associerad med en lista av referenser (s k inverterad fil) till poster som innehåller denna nyckel.

De inverterade filerna (referenserna) kan lämpligen vara organiserade sekvensiellt i stigande referensordning.

Om vi arbetar med kvantifierbara söknycklar, dvs varje söknyckel kan anta en ändlig mängd olika värden, kan dessa värden ingå i de inversa filerna för att möjliggöra utsökning där även nycklars värden ingår i det villkorliga utsökningsuttrycket.

Antag att filen F avser totalbefolkningen och att attributen betecknar

A högre utbildning (kodifierad 00, 01, ... osv)
Q antal kylskåp
P månadslön (kr)

Om en post saknar någon av ovanstående uppgifter antas motsvarande person sakna motsvarande "egenskap". Ett tänkbart (sociologiskt intressant) utsökningsvillkor kan här vara:

$$(A = 13) \wedge (2 \leq Q \leq 4) \wedge (P < 1000)$$

dvs personer vilka har utbildning av kategori 13, har mellan 2 och 4 kylskåp och en månadslön mindre än 1000 kronor.

Som tidigare nämnts är det få (6) leverantörer som har utsöknings/filorganisationsmetoder av detta slag inkorporerade i sina OS-filhanteringssystem. En trend att inkorporera mer avancerade filhanteringssystem i operativsystem är dock klart märkbar.

Den i allmän filorganisation och filhantering intresserade läsaren hänvisas till ref. (5), (7) och (8).

8.5 Datatransporthantering

Datatransportoperationer på <u>post</u>-, <u>block</u>- och <u>styrprogramnivå</u> har kortfattat berörts i avsnitt 8.1. För att kunna lagra in/utdata under den tid datatransporten pågår eller väntar på bearbetning eller utmatning krävs buffringsutrymme. Härvid kan olika principer tillämpas. Varje fil i ett program kan tilldelas en eller flera egna buffertar eller också kan vissa filer "dela på" en "pool" av buffertar. Om filerna är relativt inaktiva kan det vara oekonomiskt att permanent tilldela en eller flera buffertar till varje fil. Ett illustrativt exempel på detta är telekommunikationsfiler - dvs förbindelser till fjärrterminaler. Då terminalerna kan förväntas vara inaktiva stor del av tiden (dvs ej vara igång med sändning/mottagning av meddelanden) är en vanlig lösning här att låta telekommunikationsfilerna dela en s k "buffert-pool" bestående av ett lämpligt antal buffertar. Det lämpliga antalet bestäms genom en analys av de kösituationer som kan uppstå. Buffert-pool-principen kan ofta med fördel användas vid transporter till/från lokala in/utmatningsenheter eller sekundärminnesfiler.

Filtabeller

Varje fil som ett användarprogram refererar till måste på något sätt beskrivas och dessutom krävs någon form av förbindelselänk mellan användarprogrammet och systemets filhanteringsrutiner. För detta ändamål etableras för varje fil i ett användarprogram en s k filtabell ("file control table", "data control block", etc). Den detaljerade utformningen av en filtabell varierar givetvis från system till system och beror dessutom på vilken typ av fil och slag av perifer enhet som avses. Exempelvis innehåller filtabellen:

o uppgifter om filen
  - symboliskt namn för filen
  - lösenord (sekretessåtgärd)
  - fysisk adress (avseende sekundärminne) till början av fil
  - " " " " " slutet " "
  - organisationsprincip
  - blocklängd
  - storlek av filen (t ex uttryckt i antal spår)
  m m

o uppgifter om buffertar och köer
  läget av och antalet buffertar
  läget för aktuell buffert (under bearbetning)
  läget av början resp slutet av kö av poster eller block för in/utmatning
  m m

o länkar till användar- eller systemrutiner
  länkar till vilka uthopp skall göras om vissa förutsedda händelser inträffar vid filhantering (t ex felläsning, filslut osv)

o länkar till diverse data (om transporttillståndet) som behövs för och administreras av filhanteringsrutinerna.

De uppgifter som ingår i en filtabell kan härröra från olika källor. Visst information fylles i när användaren via programmet etablerar tabellen. Annan information fylles i när motsvarande fils datadefinitionssats påträffas i styrkoden. Ytterligare information hämtas från resp fils etikett (file label, data set label) och infogas i filtabellen.

Filhantering

Den principiella sekvensen av instruktioner i programmet för hantering av data i en fil är i stort sett följande

1. Definiera filtabell. Utrymme för en filtabell i användarprogrammet reserveras och vissa data fylls i (instruktionen normalt utformad som en makroinstruktion). När jobbsteget börjat exekveras tas här även parametrar från datadefinitionsinstruktioner som kompletterar filtabellen.

2. Öppna filen. Vid läsning: Filens första block - etiketten - läses och kontroll av densamma sker. Viss information från etiketten överföres till filtabellen. Vid skrivning av fil etableras normalt en filetikett genom "öppna fil"-satsen. Beroende på filorganisation och åtkomstmetod kan en "öppna-fil"-sats tänkas kompletterad med vissa parametrar. Filen är nu redo för användning.

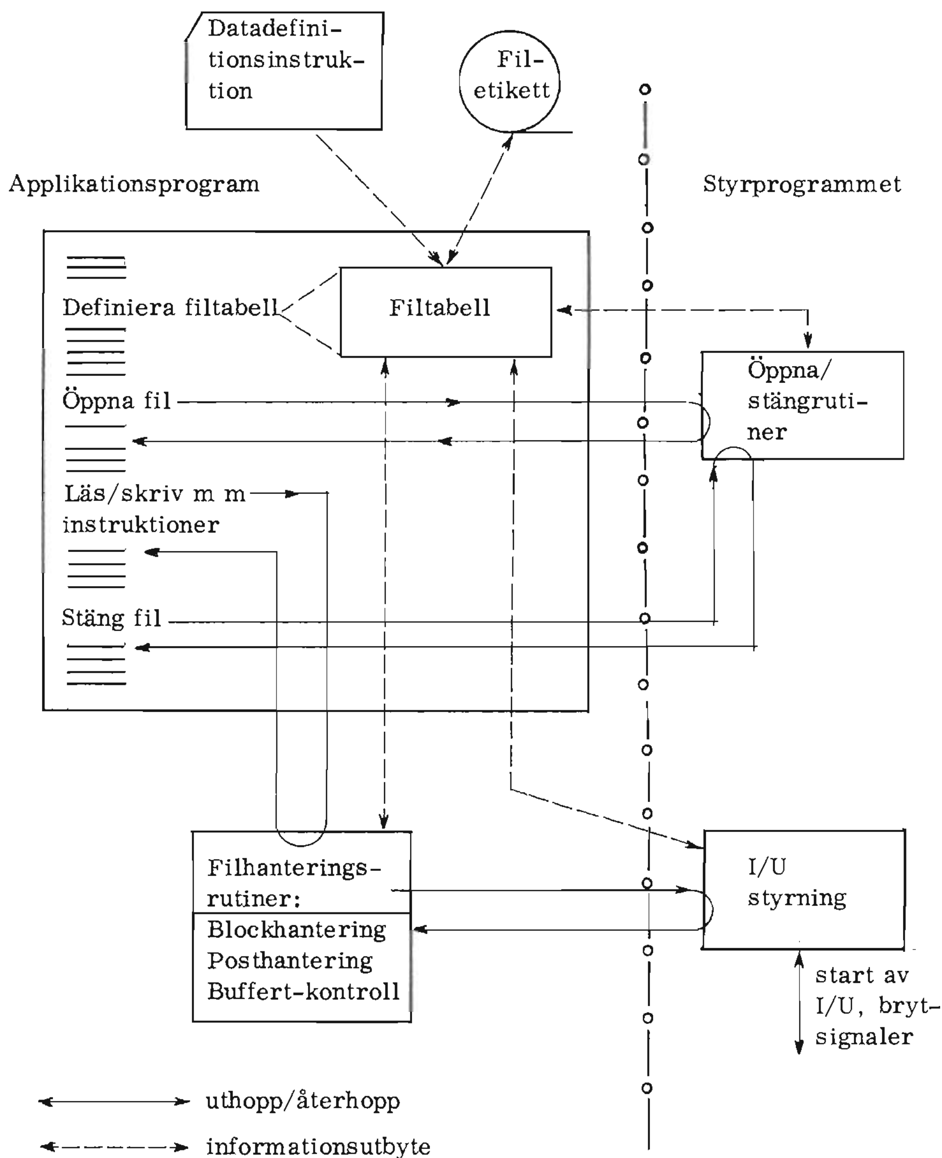
3. Upprepade läs-, skriv- eller annan typ av instruktioner[1] från programmet till motsvarande filhanteringsrutin utförs med referens till aktuell filtabell som sålunda innehåller all erforderlig information för att administrera datatransporten.

4. Stäng filen. Filen avbokas för resp användarprogram, vilket gör den åtkomlig för ett eventuellt annat program. Vid skrivning innebär "stäng-filen" instruktion att en filslutsetikett skrivs.

Mängden av olika läs-, skriv- och andra filhanteringsinstruktioner som kan användas av ett applikationsprogram är stort och varierar med typ av fil och utrustning. En fil som tilldelats till ett jobbsteg för läsning och skrivning, kan normalt tilldelas till andra parallella jobb endast för läsning (förutsätter direktminnesfiler). Inom ett jobbsteg kan dock flera processer existera samtidigt och kräva åtkomst till samma fil. För att vid uppdatering av en post eliminera risken för fel kan processen begära "exklusiv kontroll" av posten i fråga. När uppdateringen är utförd häves denna exklusiva kontroll. Dylik hantering skapar givetvis en låsningsrisk (se kapitel 3).

---

1) Beroende på aktuell maskinell enhet och dess funktioner.

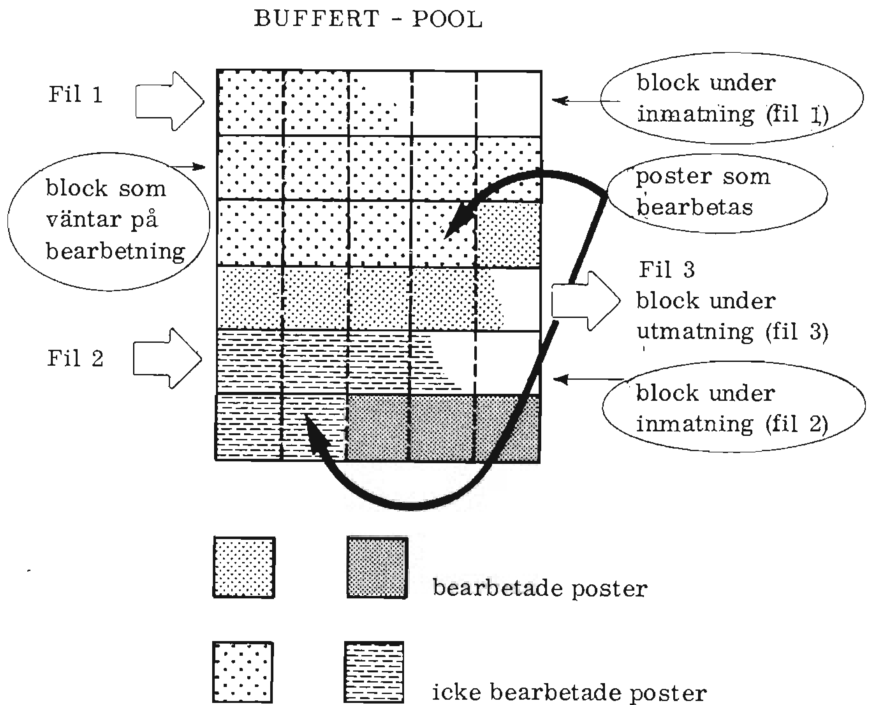


Figur 8.5:1
Grov skiss av principiellt kontroll- och informationsflöde vid filhantering.

Buffringsmetoder

En buffert, avsedd för en eller flera filer, består av ett avgränsat primärminnesutrymme stort nog att kunna lagra det största block som kan förekomma i resp filer. När en post i ett inläst block skall bearbetas kan den antingen flyttas till en arbetsarea eller kan bearbetningen utföras direkt i bufferten. Vid skrivning kan en bearbetad post antingen flyttas till en ut-buffert eller också kan posten ligga kvar och länkas till en ut-kedja av poster. De flesta datorsystemen idag har in/utmatningsinstruktioner varmed möjlighet ges att assemblera poster från spridda platser i primärminnet till ett block vid utmatning och, omvänt, sprida ett indatablock på flera platser i primärminnet.

En buffringssituation, vanlig vid filuppdatering, är visad i figur 8.5:2.

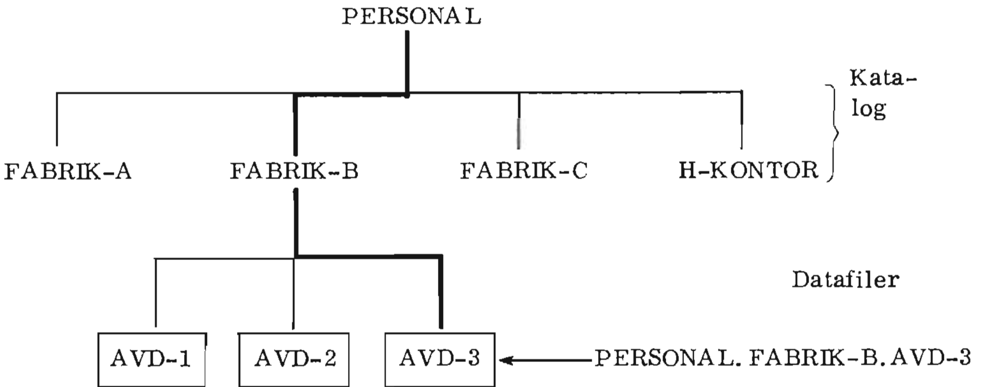


Figur 8.5:2
Poster i fil 1 bearbetas (sekvensiellt) med data från fil 2 och skrivs ut på fil 3. En buffertpool bestående av 6 block à 5 poster används och utnyttjas även som arbetsutrymme.

8.6 Katalogisering och lagring av filer

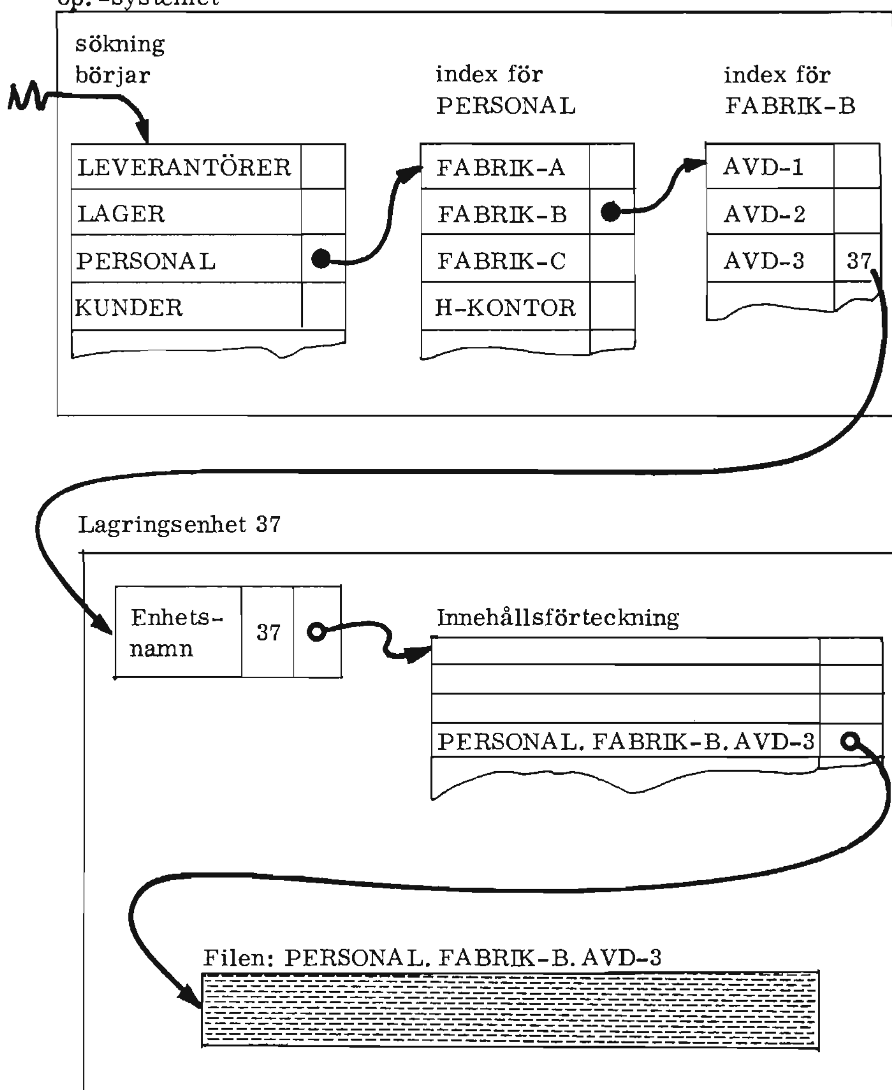
I detta avsnitt betraktar vi filer som lagras på datorsystemets sekundärminnen. De flesta operativsystem har funktioner som gör det möjligt att i systemet katalogisera en fil (datamängd) under ett givet namn. När en fil blivit katalogiserad kan användaren referera till filen enbart genom dess namn och behöver då ej ange den fysiska lagringsenhet som filen blivit placerad på, eller dess fysiska adress.

En fil kan ges ett (enkelt) symboliskt namn (ett maximerat antal tecken som normalt alltid börjar med en bokstav) som kan föregås av ett maximerat antal kvalificeringsnamn. I figuren 8.6:1 är AVD-3 namn på en fil som, för att undvika eventuella tvetydigheter, måste kvalificeras med namnen PERSONAL och FABRIK-B åtskilda av punkt.



Figur 8.6:1
Namngivning av datafiler.

Katalogens uppbyggnad och sökningen efter en viss fil illustreras i figur 8.6:2. För att ett operativsystem skall kunna hålla reda på alla filer är alla lagringsenheter försedda med namn och innehållsförteckningar. En central katalog residerar vanligen på system-sekundärminnet. I princip kan katalogens olika indextabeller vara fördelade på flera lagringsenheter. För manipulering och vård av ett systems olika filer finns vanligen ett antal olika serviceprogram att tillgå (avsnitt 9.6).



Figur 8. 6:2
Sökning efter en katalogiserad fil med namnet PERSONAL. FABRIK-B. AVD-3.

(Källa: IBM File No. S360-36, Form C28-6535-0)

Några av systemets datafiler är föremål för periodisk uppdatering. Andra filer kan vara medlemmar av en större fil som genereras vid olika tidpunkter. Exempel på sådana filer är

- o fil över företagets lagerförda artiklar, som dagligen uppdateras med kundordertransaktioner

- o transaktioner avseende olika veckodagar till en bankrutin som bearbetas veckovis

Filer av detta slag kan ha samma datastruktur och samma namn men deras innehåll gäller för ett bestämt tidsintervall. Varje sådan datamängd kallas för en generation av filen i fråga. Vissa systems katalogiseringsfunktioner tillåter en specificering av filens generation t ex genom att filens namn följs av ett (relativt) generationsnummer. PERSONAL. FABRIK-B. AVD-3 (0) avser då den senast katalogiserade versionen (farversionen). Index (+1) anger då att en ny generation (son-versionen) skall bildas och katalogiseras. På motsvarande sätt anger (-1) att det är den näst sista generationen (farfar) som avses. Möjlighet finns också att tilldela filen ett absolut generationsnummer och på så sätt kunna referera till äldre generationer utan att känna till vilken generation som just då är den aktuella[1].

8.7 Skydd av filer - säkerhetsproblem

De flesta på marknaden idag (1970) saluförda operativsystem tillhandahåller någon form av mekanismer som avser att kunna skydda systemets katalogiserade filer mot förstörelse eller mot icke auktoriserad åtkomst. Dessa mekanismer anses dock av många relativt primitiva och ej tillräckligt tillförlitliga. Dels är operativsystemen själva så svåröverblickbara att de därigenom blir "sårbara", dels är skyddsmekanismerna i många fall forcerbara för personer som besitter tillräckliga tekniska kunskaper om operativsystemet/filhanteringssystemet.

Utvecklingen mot fjärrterminalorienterade databassystem vilkas informationsinnehåll omspänner en allt större del av samhällets olika komponenter aktualiserar data-säkerhetsproblemet.

---

1) Ovanstående beskrivning är baserad på information i IBM publ. File No. S360-36, Form C28-6535-0, "IBM Operating System/360, Concepts and Facilities".

De skyddsåtgärder avseende filsystem med persondata som idag är vidtagna kan i de flesta fall anses som bristfälliga. Även lagstiftningen avseende registrering och datoriserad lagring av persondata bör anses otidsenlig. Utvecklingen i datorbranschen har varit så snabb att en yrkesetik (motsvarande läkare, advokater m fl) ej hunnit etableras för dem som arbetar i fältet.

Diskussionen i USA avseende dataskydds- och sekretessproblem intar en framträdande plats i datapressen. Åtskilliga lagförslag avseende dessa problem har framlagts och vissa är redan antagna. Diskussionen i Sverige har länge varit av tämligen blygsam omfattning och få medborgare har öppet reagerat mot och insett riskerna med etablerandet av personnummeranknutna databaser över "totalbefolkningen". 1970-års folk och bostadsräkning utlöste emellertid en livlig debatt, som bör kunna medföra åtminstone det positiva att åtgärderna för att bevaka den personliga integriteten intensifieras bl a genom arbete på en adekvat lagstiftning. I detta sammanhang är det intressant att notera att unik identifiering av varje medborgare med ett personnummer sedan länge utan vidare accepterats i Sverige. I flera andra länder skulle ett dylikt system väckt betydligt större uppmärksamhet.

Eftersom skyddsproblemet idag bör upplevas som högst väsentligt skall vi i detta avsnitt inte begränsa oss till de mekanismer (vanligen av "lösenordstyp") som finns inbyggda i de flesta filhanteringssystemen, utan betrakta problemet något mer allmänt. Integritetsproblemet, som aktualiseras i och med att vi har för mycket information om ett företag eller en person (och deras relationer till andra objekt och delsystem) ligger, trots sin betydelse, vid sidan av denna boks syfte och behandlas därför ej här (viss diskussion om detta finns dock i avsnitt 8.2).

Säkerhetsproblemet kan definieras som (9)

> "skydd av data mot oavsiktlig eller avsiktlig, icke auktoriserad
> - modifikation
> - förstörelse eller
> - exponering (eng. disclosure)."

Säkerhetsproblem uppträder givetvis även vid konventionell sats- eller kövis bearbetning, men är särskilt accentuerade vid fjärrterminal- och kommunikationsorienterade system (reelltidssystem, tidsdelningssystem). Problemet accentueras ytterligare om systemet avses lagra och bearbeta ur sekretessynpunkt känslig information.

8.7.1. Olika slag av "hot" mot säkerheten

I ref. (9) anges följande klassificering av händelser eller aktiviteter som kan utgöra ett hot mot datasäkerheten:

Oavsiktliga

1. Användarfel som innebär att vederbörande "råkar" komma förbi etablerade "spärrar"

2. Systemfel, av maskin- eller programvaruslag, som sätter en eller flera av skyddsmekanismerna ur funktion

3. Fel i datatransmission

4. Ett oavsiktligt avslöjande av skyddsmekanismer till datoroperatörer eller servicepersonal

Avsiktliga, passiva

5. Elektromagnetisk "inspelning" från diverse datorutrustning[1)]

6. Avlyssning-avspelning av "data-samtal"

7. En icke auktoriserad person, som betraktar utmatning på en terminalskrivare

Avsiktliga, aktiva

8. Systematisk avsökning och avkodning av filer på jakt efter känsliga data

9. Personifiering av en auktoriserad användare

10. En "data-tjuv" kopplar in sig på en auktoriserads terminallinje och utnyttjar den under tidsintervall då den auktoriserade är "inaktiv"

---

1) Det berättas (9) om den elektromagnetiska strålning som ett datorsystem under drift avger. Det lär t ex vara möjligt att genom sinnrik mottagningsutrustning uppfånga signaler från en radskrivare i arbete i ett angränsande rum och därvid framställa samma utdatalista.

11. Inkoppling enligt (10), "avledning" av meddelanden, substituering av meddelanden o dyl.

12. Åtkomst till data via systempersonal som känner till säkerhetsmekanismerna och vet hur de kan förbigås

13. Läsning av "slask"-data från sekundärminnesbaserade arbetsfiler (t ex efter en sorteringsoperation), radskrivarkarbon o dyl.

I ett framtida samhälle där olika slag av information kan förväntas tillmätas allt större vikt kommer "data-stöld" och "data-våld mot den enskilde" sannolikt bli relativt vanliga begrepp. Informationen kommer i allt större utsträckning att tillmätas större värde (representera pengar) och därmed kommer olika åtgärder att vidtas för att få åtkomst till databaser o dyl. De kostnader som man kommer att bli beredd på att lägga ner på utformning av säkerhetsmekanismer kommer att stå i proportion till det värde som man (på basis av skilda bedömningskriteria) tillmäter informationens säkerhet.

Det bör särskilt påpekas att <u>stöld</u> av data ibland kan vara ett mindre allvarligt problem än <u>avsiktlig förändring</u> av innehållet i en databas. Olika organisationer och företag litar i allt större omfattning på datorn som ett verktyg för operativ- och ibland även direktiv - styrning av verksamheten. En påverkan av databasen kan här få katastrofala följder för styrningen.

8.7.2. Åtgärder för att öka data-säkerheten

Det åtgärder som f n kan tänkas bli vidtagna för att öka säkerheten är av många skilda slag. Vi redogör här kortfattat för några kategorier:

Administration av åtkomst till data

I stigande följd, med avseende på ambitionsgrad, kan följande metoder tänkas

- o Användaren identifierar sig själv vid terminalen på ett för systemet acceptabelt sätt. Metoden ger en ringa grad av säkerhet.

- o Användaren måste ange ett "hemligt lösenord" som kontrolleras av systemet. Lösenorden kan stjälas, avslöjas eller också kan de vara lätta att "komma på" intuitivt.

- o Lösenordsmetoden kan förbättras på olika sätt t ex

- lösenorden kan ändras med oregelbundna tidsintervall
- användaren har en lista av lösenord och använder varje gång ett nytt ord
- datorn översänder till användaren ett flersiffrigt tal, ber användaren att utföra en icke trivial manipulation av siffrorna samt återsända resultat. I detta fall är "algoritmen"[1] att se som ett lösenord. Om lösenorden inmatas utan att motsvarande symboler framträder på terminalen minskas risken för obehörig observation.

o Ytterligare säkerhetsåtgärder som kan vidtas är att begränsa olika användare till "tillåtna" terminaler. Mer minneskrävande metoder går ut på att i systemet lagra tabeller som utvisar vilka användare som får ha åtkomst till en viss fil. Olika "åtkomstgrader" kan tänkas såsom diskuterats inledningsvis i avsnitt 3.3. Skydd enligt ovan kan tänkas dels på filnivå dels på post- och termnivå. I det senare fallet rör det sig om att göra vissa termer i en post oåtkomliga för obehörig access. Det inses att dylika skyddsmekanismer är mycket minneskrävande. En annan åtgärd som kan verka stöldförebyggande är att för varje skyddad fil uppsamla statistik (logg) över accesser till densamma. Man kan också tänka sig ett alarmsystem som signalerar till ansvarige operatör eller "filägare" så fort ett visst antal misslyckade lösenordsförsök gjorts vid någon terminal.

Det inses att ingen av de ovan redovisade metoderna kan sägas skapa ett 100 %-igt skydd av data.

För ytterligare och mer detaljerat studium av metoder för åtkomstskydd hänvisas till re. (9) och (11) som även innehåller åtskilliga referenser till publikationer över detta problemområde.

Filkonstruktion

Säkerhetskravet kan påverka filkonstruktionen på flera sätt. Som vi tidigare påpekat kan åtkomstskydd tänkas implementerat på olika nivåer såsom fil, post, delpost och term. Ju längre ner vi går desto mer får vi betala i form av administrationstid och minnesutrymme.

Om filer lagras på utbytbara/monteringsbara lagringsenheter och uppkopplas endast under den tid som de oundgängligen behövs kan detta anses som ett bidrag till säkerheten. Naturligtvis kan monterbara media bli föremål för stöld.

Ett påpekande görs i (9) att starkt länkade filer (diskuterat i avsnitt 8.3

1) dvs den manipulation som utföres av användaren.

och 8.4) är speciellt känsliga för maskin- och programvarufel. En "länk kan brista" vilket innebär att man bokstavligt talat "tappar bort" vissa poster i den länkade listan.

En annan metod att öka skyddet av lagrad information är att lagra den i kodifierad-förvrängd form. Detta kräver en dechiffrering vid läsning. Denna metod berörs, huvudsakligen från minnesekonomiska utgångspunkter, i (13).

Maskin- och programvaran

Ett system som bearbetar ur sekretessynvinkel känsliga data bör implementeras på en datorutrustning som är försedd med så många säkerhetsmekanismer som möjligt. Vi har tidigare talat om minnesskydd, paritetskontroll, loggning av in/utmatningsfel osv. Ett system är speciellt oskyddat när det arbetar i det s k "beordringstillståndet" där minnesskyddsmekanismen är inhibirad. Det är ej ovanligt att ca 10 % av processortiden åtgår i detta tillstånd. En strävan bör vara att minska denna procentuella andel. Operativsystemen är idag så komplexa att de aldrig kan anses bli 100 %-igt uttestade. En risk existerar således att systemfel kan uppstå då systemet arbetar i beordringstillstånd.

Datatransmission

Tidigare har påpekats möjligheten att

- avlyssna en telefonförbindelse
- att otillåtet koppla in sig och sända/motta data på en telefonförbindelse
- att motta signaler från elektromagnetisk strålning (radiation)

Till relativt höga kostnader kan varierande former av skydd av teleförbindelser tänkas. Vid privata, fasta telefonlinjer kan skyddet givetvis göras säkrare än när det gäller det allmänna telenätet.

En annan metod vore att sända data i chiffrerad form. För att försvåra en "avtappning" och avkodning har det vidare föreslagits att sända data utan pauser (mellan block) eller att fylla mellanrummen med redundant information, datamönster av liknande slag som meddelanden själva.

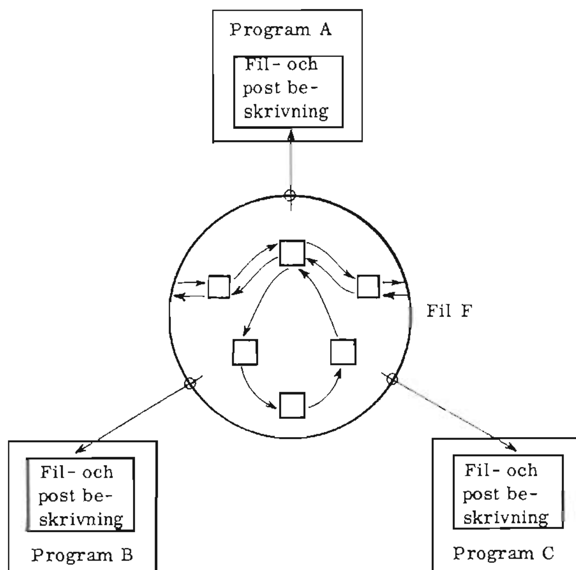
Vi har i detta avsnitt konstaterat att olika möjligheter existerar att komma åt "skyddade" data. Vi har också redogjort för några metoder att öka säkerheten. Det bör dock ha framgått av kapitlet att en allvarlig satsning krävs på olika områden för att främst skydda individen i det framtida datoriserade informationssamhället.

8.8 Program - dataoberoende

Betrakta beskrivningen av "personposten" i figur 8.1:1 bestående av termerna

Personnummer
Namn
- Efternamn
- Förnamn
Adress
- Gatuadress
- Postadress
- Postnummer
Längd
Vikt
Hårfärg

och antag att en fil F består av en mängd aktuella poster av denna postklass.



Figur 8.8:1
Illustration av program-databeroendet. För varje fil som används i ett program innehåller detta en explicit eller implicit beskrivning av filen. Varje strukturell förändring av filen innebär att berörda program måste ändras.

För att kunna läsa och bearbeta vissa data i denna fil behöver programmeraren veta dels postens maskinella representation, och dels filens organisation och utsökningsprincip. Om vi antar att det aktuella programmet endast behöver uppgifter om, säg, vissa personers efternamn och hårfärg, måste i detta programs minnesutrymme reserveras ej enbart för samtliga datafält i posten utan, i de flesta fall, även för minst ett helt block av poster. Vid läsning av data från filen måste programmet känna till filens organisation, vilken åtkomstmetod och vilket eller vilka sökbegrepp som måste tillämpas. Om vi, av någon anledning, önskar ändra posterna i filen (till exempel genom att öka antalet attribut för personer) eller ändra filens organisation (t ex från sekvensiell till pseudoadresserbar via adressberäkning baserad på personnumret) måste alla program som använder filen ändras (figur 8.8:1).

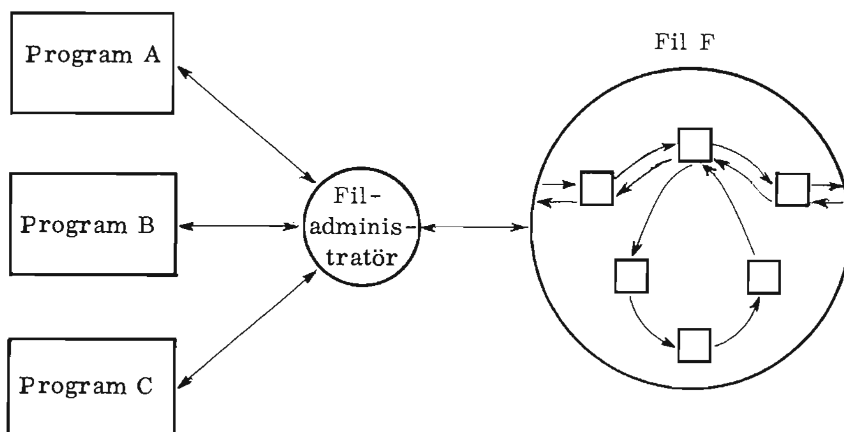
Vi har ovan illustrerat hur, vid "konventionell databehandling" program är bundna till filers maskinella organisation och representation. Behovet att känna till detaljer om filen varierar beroende på vilket programspråk som används. Till exempel är kravet på detaljer relativt stort vid programmering på assembly-nivå. Vid programmering i Algol arbetar man med datastrukturer som "fält" (array) och man är vanligen ej intresserad av dessas aktuella maskinella representation. Om man däremot lagrar fält och andra variabler i en sekundärminnesfil måste den aktuella lagringsstrukturen vara känd för andra program som önskar använda data i filen.

Trenden mot generella filhanteringssystem innebär bl a en strävan att separera program och motsvarande filbeskrivningar. I stället för att kommunicera med ett filsystem på postnivå är tanken här att endast behöva ange de termer som man är intresserad av. Detta förutsätter att programmeraren känner till motsvarande filsystems logiska datastruktur och termnamn. Däremot skall programmeraren ej behöva känna till aktuellt filsystems maskinella organisation. Kommunikation mellan program och filsystemet kommer då att ombesörjas av ett administrativt system, som arbetar, ur programmens synvinkel, på en logisk data-nivå och kan nyttjas av samtliga program. Ändras filens maskinella organisation påverkas ej den logiska datastrukturen och därmed ej heller programmen (fig 8.8:2).

Ett enkelt hypotetiskt exempel på ett fil-post-strukturoberoende programstycke (baserat på en datastruktur enligt figur 8.2:3b) är

```
for I: = all ANSTÄLLD of FÖRETAG-A do

    if HÅRFÄRG of I is MELLANBRUN
       and FÖRNAMN of I is KALLE then
    print (EFTERNAMN, POSTADRESS of I);
```



Figur 8.8:2
Program kan göras oberoende av en fils (eller "filsystems") maskinella organisation genom att arbeta på logisk data-nivå där datatransporter till/från filen ombesörjes av ett gemensamt administrativt system.

Litteratur

1. Langefors, B., "Theoretical analysis of information systems", Studentlitteratur, Lund. 1967.
2. Bubenko Jr, J. & Dopping O., "Databehandlingens XYZ", Studentlitteratur, Lund, 1970.
3. Bubenko Jr, J., "Databehandlingsteknik, del II", Studentlitteratur, Lund, 1968.
4. Martin, J., "Design of real-time computer systems", Prentice Hall, Englewood Cliffs, N.J., 1967.
5. Lefkowitz, D., "File structures for on-line systems", Spartan Books, N.Y., 1969.
6. Pettersson, S-O., "Datasaabs databankssystem OS22DB", föredrag vid NordDATA-70, Köpenhamn, 1970.
7. FILE-68, "International seminar on file organization", Working Papers 1968 (Conference sponsored by IFIP Administrative Data Processing Group (I.A.G.)).

8. Wedekind, H., "Datenorganisation", Walter de Gruyter & Co, Berlin, 1970.

9. EDP Analyzer, "Data security in the corporate data base (CDB)", Canning Publications Inc., Vol. 8, No. 5, May, 1970.

10. McGee, W. C., "Data description for data independence", ACM Special Interest Committee on File Description and Translation, Vol. 1, No. 2, Dec. 1969.

11. Hoffman, L. J., "Computers and privacy: A survey", Computing Surveys, Vol. 1, No. 2, June 1969.

12. CODASYL Systems Committee Technical Report, "A survey of generalized data base management systems", May, 1969. (Säljes av The Association for Computing Machinery, 1133, Avenue of the Americas, New York, N.Y. 10036, $ 7.00.)

13. deMaine-Springer, The COPAK Compressor. Återfinns i FILE-68, sid. 29-45, ref. (7).

14. Datorleverantörers handböcker avs filhanteringssystem m m.

9. PRINCIPER OCH HJÄLPMEDEL VID PROGRAMKONSTRUKTION

9.1 Introduktion

Vid sidan av val av lämpligt programmeringsspråk för lösning av användarproblem med hjälp av datorsystem, existerar ytterligare ett antal programmerarnära situationer att beakta för att uppnå god kommunikation mellan användare och datorsystem. Programkonstruktion innehåller ett flertal arbetsfaser där "medverkan" från operativsystemet kan bidraga till att uppnå ett gott resultat. Vi belyser nedan några dylika kontaktytor mellam (främst) programmerare och operativsystem:

Programstrukturering. Operativsystemet tillhandahåller hjälpmedel för att konstruera i olika situationer lämpliga programstrukturer samt funktioner för hantering av dessa.

Programegenskaper vid användning. Här avses "interna" egenskaper av typen icke-multi-användbara resp multi-användbara (reentrant) program.

Diagnostik, test och felsökning. Alla operativsystem tillhandahåller någon form av hjälpmedel för programinkörning (test- och felsökningsfunktioner). Olika former av diagnostik kan begäras. Ambitionsgraden hos olika system i dessa avseenden varierar dock.

Köravbrott och reetablering. Ibland är det önskvärt att ett programs exekvering avbryts avsiktligt för att låta andra aktiviteter äga rum, Ibland inträffar icke önskade eller icke förutsedda avbrott av exekveringen. Vi belyser här bl a begreppet checkpoint/restart. Reetablering leder till speciella problem i reelltidsmiljö, vilket nedan kommer att diskuteras.

Serviceprogram. Här anger vi några funktioner som huvudsakligen kan utföras med hjälp av olika typer av relativt fristående servicerutiner inom ett operativsystem.

Under den första rubriken ovan kommer vi bl a att studera det systemprogram som ofta går under benämningen länkaren (eng. linkage editor, collector), samt hur detta program samarbetar med från användaren tillhandahållna program.

9.2 Programstruktur och länkning

Det existerar skillnader mellan olika leverantörers sätt att lösa problemen kring programlänkning. Erforderliga användarkonventioner skiljer sig ofta markant från varandra. Sannolikheten att ett större segmenterat program (i ett generellt programmeringsspråk) framgångsrikt skall kunna direkt (utan "ingrepp") kompileras, länkas, laddas och exekveras på några skilda datorsystem är idag liten.

Ett objektprogram som avses laddas till primärminnet för exekvering kan fysiskt härröra från olika källor:

- Det kan tillhandahållas direkt från användaren i objektform (maskinkod).
- Det kan hämtas fram i objektform ur ett sekundärminnesresident programbibliotek.
- Det kan ha genererats vid utförande av ett föregående jobbsteg av typen kompilering eller assemblering.

Ett objektprogram som fullständigt färdigställts för laddning, ett laddningsbart program (load module), kan förekomma i olika utseenden.

Följande uppdelning kan göras:

1. Seriell struktur. Exekveringen av det laddade programmet avses förlöpa helt seriellt. Vi kan skilja mellan

   - Enkel seriell struktur. Programmet kan laddas och exekveras i ett sammanhängande stycke.

   - Statisk seriell struktur. Alla segment av programmet laddas ej samtidigt, utan vissa må under exekveringens gång av laddaren inläsas till av programmet disponerat minnesutrymme och därmed "överlagra" andra segment. Strukturens utseende fixeras av programmeraren före exekveringens början.

   - Dynamisk seriell struktur. Samma statisk (överlagrad) seriell struktur, med skillnaden att inläsningen av segment ej kan förutses före exekveringens början, då den är beroende av programmets förlopp.

2. Parallell struktur. Två eller flera underprogram inom "huvudprogrammet" tillåts exekveras samtidigt, parallellt. Ej sällan behandlas därvid varje underprogram som separat process (task eller subtask) av operativsystemet.

Sammanknytningen av olika sammanhörande program, programsegment eller -delar före laddning till primärminnet sköts av den del av operativsystemet som går under benämningen länkaren (eng. linkage editor, collector m m).

Förutom sammanlänkning av tillhandahållna programsegment i objektform, har länkaren ofta andra funktioner. Exempel på dylika är:

- uppsökning och inkorporering av till ett användarprogram erforderliga tidigare lagrade delprogram på sekundärminne.

- programsegmentmodifikation, konstruktion av en tabelliknande adresshierarki hörande till aktuellt användarprogram, via vilken laddaren under exekveringen kan administrera inläsning av erforderliga externa programsegment till tillgängligt utrymme i primärminnet.

- att ge anvisning om eventuella från användaren felaktigt angivna länkningsinstruktioner.

m m

Länkarens utformning och arbetsuppgifter varierar för olika tillgängliga operativsystem. Allmänt kan dock sägas att de uppgifter, som åvilar länkaren, utförs antingen omedelbart efter (och i samma jobbsteg som) kompilering/assemblering (motsv.) av till datorsystemet inmatat källprogram, eller, i ett följande jobbsteg, i samband med laddningen av det färdigställda objektprogrammet. Hos vissa operativsystem, t ex IBM OS/360, sker länkningen i ett eget separat jobbsteg. Härigenom vinner användaren det att ej onödigtvis behöva belastas med (och debiteras för) utförande av länkning när denna ej önskas (exempelvis i samband med av användaren i tid upptäckta logiska programfel o d) eller ej belastas med andra operationer när endast länkning önskas. Användaren belastas i gengäld med vissa ökade administrationstider vid växling mellan de erforderliga operativsystemfunktionerna.

Användarens kommunikation med länkaren utförs normalt via speciella styrsatser, tillhandahållna i samband med det jobbsteg skall utföra länkning. Ofta är standardiserade länkstyrsatser inkorporerade i "proceduren" ("makrostyrinstruktionen") för vanligt förekommande länkningssekvenser. Detta gäller t ex sammanlänkning av programsegment med frekvent anropade systemprogramsegment (standardiserade subrutiner o d). Icke-standardiserade länkinstruktioner, t ex rörande speciell överlagring av användarprogramsegment, får dock av användaren tillhandahållas separat.

Utseendet av en länknings-styrsekvens har följande principiella struktur:

1. Styrsats (ev flera) som anropar länkaren, definierar och överför till den vissa parametrar såsom namn på filer och namn på element för till länkning ingående resp från länkning resulterande program.

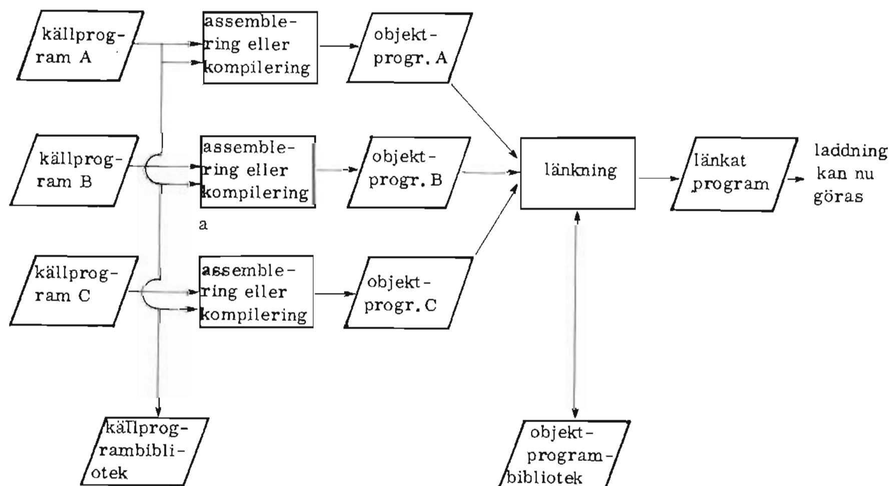
2. En sekvens av lokala styrsatser för länkaren. Dessa styrsatser betraktas som lokala och kan tydas endast av länkaren. Dessa lokala styrsatser interpreteras av länkaren och kan ha följande funktioner

   - besked om att vissa programelement skall ingå i länkningen
   - anvisning om att vissa programbibliotek skall avsökas för att från dessa infoga sådana programelement som efterfrågats i källprogrammet.
   - beskriver ett programs (segment-)överlagringsstruktur,

   m m

Vanligt förekommande är att länkaren producerar en relokerbar laddmodul, som alltså kan tilldelas godtycklig absolut plats i primärminnet. Denna tilldelning, eller fixering av plats, sker vid laddning av modulen.

Vi bör notera att länkaren (i de flesta system) arbetar i icke-priviligierat programtillstånd (skyddstillstånd, slave mode), sålunda helt jämnställd med användarprogram som löpande bearbetas i systemet. Figur 9. 2:1 avser att illustrera länkarens plats och roll i samband med programkonstruktion (se även figur 6. 3:1).



Figur 9. 2:1
Länkarens funktion vid programkonstruktion. I detta fall är det resulterande programmet sammansatt av tre källprogramelement A, B, C samt ytterligare element från de två olika programbiblioteken.

Låt oss med några exempel betrakta hur de ovan angivna programstrukturerna administreras av länkaren.

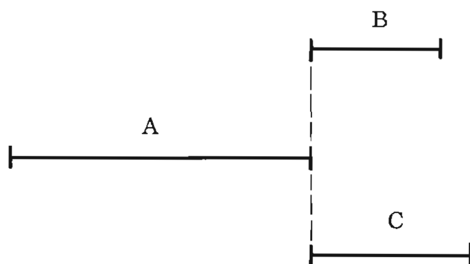
Enkel seriell struktur. En huvuduppgift för länkaren är här (emedan ingen överlagring förekommer) att sammanknyta (upprätta adresstabeller för) användarprogrammet och eventuellt begärda subrutiner, lagrade på sekundärminne, eller på annat sätt tillhandahållna.

Användarprogrammet må erfordra ett godtyckligt antal subrutiner, men den enkla seriella strukturen förutsätter att såväl huvudprogram som alla dess delprogram kan laddas till primärminnet tillsammans. Relativt omfattande primärminnesåtgång noteras alltså härvid. Laddaren är aktiv endast före starten av programmets exekvering.

Statisk seriell struktur. I de situationer då tillgängligt primärminne ej räcker för alla de delprogram som erfordras inom en process, kan statisk seriell struktur komma i fråga. Härvid förutsätts att programmeraren på förhand, dvs före överlämnande av jobbet till datorsystemet, kan förutsäga förloppet vid programexekveringen och vilka programdelar (segment) lämpligen kan dela samma utrymme (dvs överlagra varandra). Låt oss betrakta ett enkelt exempel.

Programmet må ha utseendet

där programdelarna B och C avses kunna dela primärminnesutrymme. Länkaren får instruktion att omarbeta programstrukturen till



och sammankopplar därvid A:s slut med B:s början, dvs medger att B placeras omedelbart efter A i minnet vid laddningen. Dessa sammankopplingar sker med hjälp av länktabeller där begynnelse- och slutadres-

ser för aktuella programdelar lagras. Dessa tabeller upprättas av länkaren, men disponeras, dvs används, senare av laddaren. Länkaren är sålunda endast aktiv före starten av ett jobbs exekvering.

I vårt enkla exempel ovan ges B och C samma begynnelseadress.

Om lagring av programdelar i absolutform används av operativsystemet, ges B och C samma fysiska begynnelseadress. Om lagring av segment i relativform används, sköts den fysiska adresseringen (med samma begynnelseadress för B och C) senare under exekvering, av laddaren. Utförandet av programmet enligt ovan kan ske rent sekvensiellt eller också kan en mer komplicerad interaktion mellan segmenten förekomma.

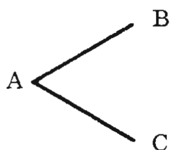
Vid den sekvensiella exekveringen kommer då B att börja utföras direkt efter A, medan vid tidpunkten för B:s slut en avbrotssignal genereras (anrop till laddaren), som startar inläsning av C till det minnesutrymme där B (som nu är färdigexekverad) befinner sig. När C är färdiginläst startas exekveringen av C, på order från styrprogrammet, och programmet löper (förhoppningsvis) normalt till slut.

Ett något mer komplicerat förlopp kan tänkas. Exempelvis kan en hoppinstruktion i segmentet A till ett läge i C förekomma. Beroende på om C eller B finns inlästa vid den aktuella tidpunkten måste eventuellt laddaren aktiveras och C inläsas. På samma sätt kan en interaktion förekomma mellan segmenten B och C.

De styrinstruktioner som användaren ger till operativsystemet för att beskriva ovanstående överlagring kan vi t ex schematiskt ange som:

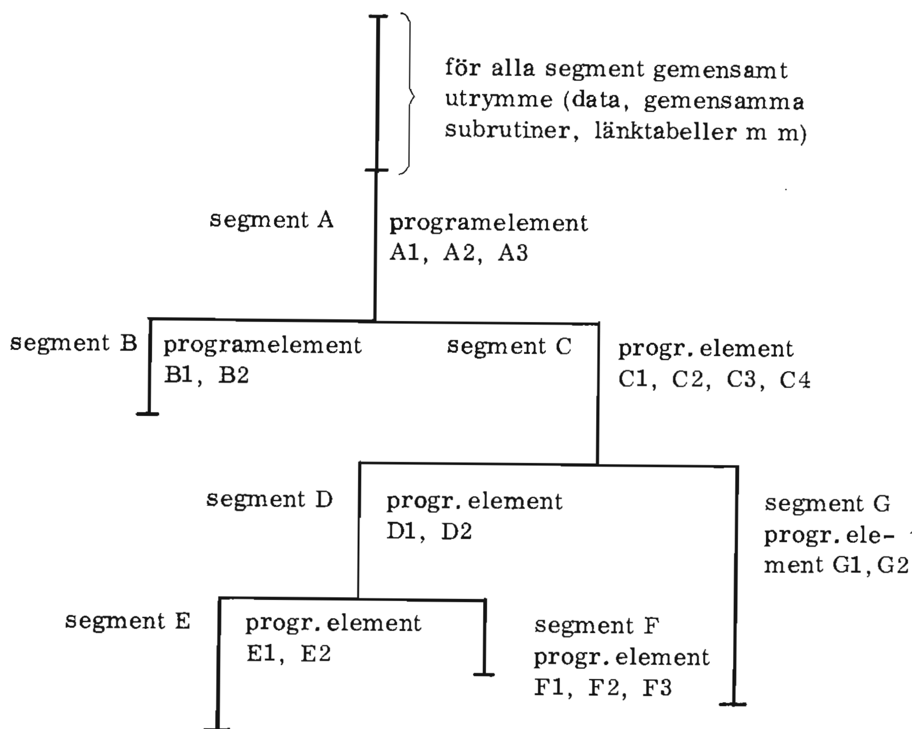
(A . (B . C))

vilket är ett konventionellt sätt att beskriva en trädstruktur i listspråk.



Härvid klargörs för länkaren att B och C avses kunna dela minnesutrymme, samt att detta utrymme skall utgöras av arean närmast efter A. Olika operativsystem använder sig av varierande utseende på länkande styrinstruktioner, och utseendet i exemplet ovan är enbart en konstruktion.

En något mer komplicerad programsegmentering och överlagring är visad i figur 9. 2:2.



Figur 9. 2:2

Länkningsinstruktionerna för denna struktur kan ha följande principiella utseende

SEG	A	def. A som rot-segment
INCL	A1, A2, A3	inkluderar progr. element i A
SEG	B, (A)	def. B som omedelbar succedent till A
INCL	B1, B2	
SEG	C, B	segmentet C får samma startadress som B
INCL	C1, C2, C3, C4	
SEG	D, (B, C)	segment D får startadress som bestäms av den längre av de två segmenten B och C
INCL	D1, D2	
SEG	E, (C)	E är omedelbar succedent till C
INCL	E1, E2	
SEG	F, E	F får samma startadress som E
INCL	F1, F2, F3	
SEG	G, D	
INCL	G1, G2	

Skälet för användning av överlagringsteknik är besparing av primärminnesutrymme. In- och utläsning av programsegment mellan sekundärminne och primärminne kräver emellertid processortid för administration, samt visst minnesutrymme för länktabeller o d. Emedan ofta den marginella tillgången på processortid och sekundärminnesutrymme är större än den marginella tillgången på primärminne (i sin tur betingat av de olika inköps/hyreskostnaderna för dessa resurser) är segmentering/överlagring mycket vanlig i praktiken.

Det förtjänar nämnas att i enstaka programmeringsspråk, t ex Algol 60 och PL/I en "automatisk" segmenteringsmöjlighet existerar. Ett Algolexempel:

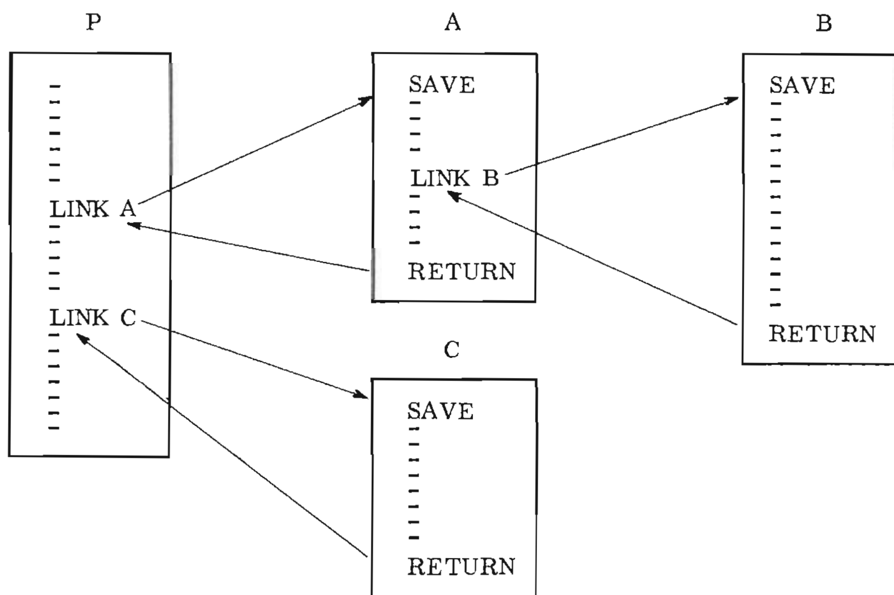
```
begin
  .
  .
  .
  begin
  .
  .
     begin    ┐
       .      │
       .      ├  A
       .      │
     end;     ┘
     begin    ┐
       .      │
       .      ├  B
       .      │
     end      ┘
  .
  .
  .
  end;
  .
  .
  .
end
```

Figur 9.2:3
Blockstrukturen för ett Algolprogram medger en minnesbesparande programkonstruktion.

Variabelfält i de båda "parallellt" förekommande programblocken A och B kommer vid exekvering att kunna disponera helt eller delvis samma primärminnesutrymme. I Algol 60-rapporten finns ej explicit anvisat att kompilatorkonstruktörer har att ta hänsyn till denna möjlighet till minnesbesparing, men i de flesta fall har detta beaktats vid framställning av kompilatorn.

Dynamisk seriell struktur. I många fall kan av programmeraren ej på förhand anges i vilken ordning programdelar kommer att anropa varandra under exekveringen. Det är här fråga om att exekveringsförloppet är beroende av inlästa data och parametrar, dvs att påverkan "utifrån" styr utförande av villkorliga hoppinstruktioner mellan programderlarna. Det låter sig därför, för programmeraren, ej göra att på förhand fastslå en viss överlagringsstruktur för de olika delprogrammen.

För flera operativsystem existerar i dessa situationer möjlighet att inom programdelar anropa andra programdelar, exempelvis subrutiner. Dessa anrop sker genom att i själva programkoden infoga länk-anrop. Ett exempel:



Figur 9. 2:4
Dynamisk seriell länkning (källa: IBM OS/360).

I fig 9. 2:4 noterar vi att "kedje-länkning" förekommer, dvs att länkning i flera steg avses utföras (huvudprogrammet P anropar delprogrammet A, som i sin tur anropar delprogrammet B, osv). Instruktionerna LINK A och LINK C i huvudprogrammet må här utföras betingat av exekveringsförloppet inom P, dvs villkorliga hopp i P kan medföra att uthopp till exempelvis delprogrammet C kommer att ske beroende på exekveringen i P.

En programmerare som har möjlighet välja mellan statisk och dynamisk seriell struktur (vilket beroende på problemutseende inte alltid är fallet) kan besinna att den förra, som en följd av sin lägre flexibilitet, vanligen utnyttjar systemresurserna bättre än den senare. Den dynamiska strukturen innebär att länktabeller måste byggas upp successivt under exekveringsförloppet. Situationer kan förekomma då en och samma länktabell behöver byggas upp vid ett flertal tillfällen, beroende på programförlopp, varvid extra processortid åtgår för administration, jämfört med situationen om denna länktabell konstrukerats en gång för alla före exekveringens början. - Det förekommer även system där den på förhand planerade statiska strukturen möjliggör för länkaren att "optimera" primärminnesutnyttjandet.

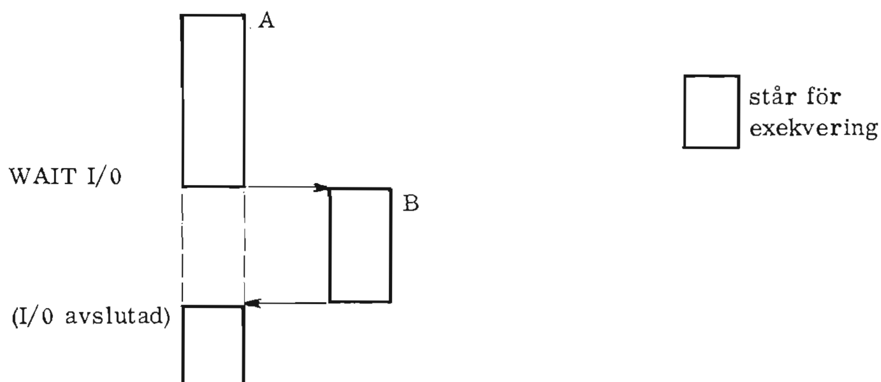
Vi noterar här alltså åter att flexibilitet kostar extra systemresurser.

Parallell struktur. I vissa system finns möjlighet att begära parallellt utförande (exekvering) av programsegment inom ett deljobb. Man skulle något oegentligt kunna benämna detta "multiprogrammering inom ett deljobb" till skillnad från vanlig "multiprogrammering mellan jobb och deljobb", vilken vi tidigare diskuterat.

Den dynamiskt parallella länkningen innebär för länkaren arbete som i mycket påminner om den dynamiskt seriella strukturen. Vid anrop av ett parallellt programsegment föds emellertid en ny underprocess, och på grund av att denna må ha speciella systemresurser tilldelade, kan extra administrationstid i samband härmed bli erforderlig.

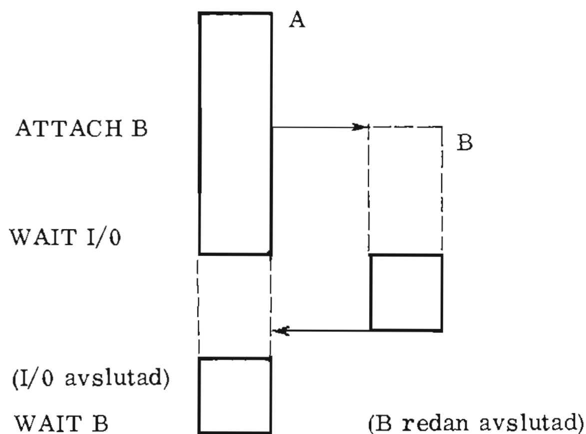
Låt oss något betrakta hur styrprogrammet samverkar med länkstrukturen i detta fall.

I enprocessorsystem kan huvudsakligen in/utmatning överlappas inom en process, medan förekomst av flera processorer i systemet möjliggör parallell bearbetning inom en process. Antag att vi har ett enprocessorsystem samt att två programsegment (eller delprogram) A och B skall exekveras inom ett deljobb. Under det att A utför in/utmatning kan då B exekveras av processorn:



Figur 9. 2:5

Anhållan om färdigställande av segmentet B för exekvering så snart processortid finns tillgänglig kan i vissa system effektueras utan direkt samband med in/utmatning. Vi kan tänka oss detta angivet med den inom A förekommande instruktionen ATTACH. Detta ord kan vi grovt översätta med "ANSLUT". Ett exempel visas i figur 9. 2:6 där det antas att A har högre prioritet än B.

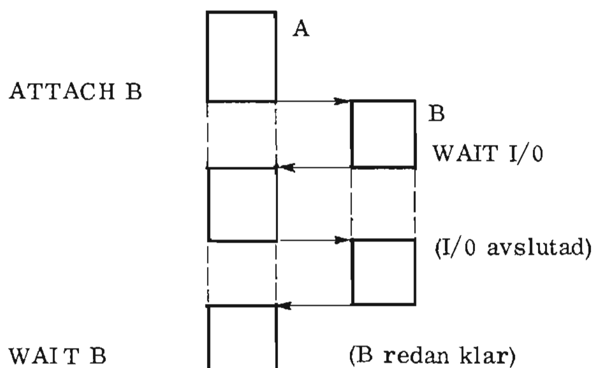


Figur 9. 2:6

Efter det att B "anslutits" kommer här dess exekvering att startas så snart A ej längre behöver processortid. Längre fram kommer A att fråga

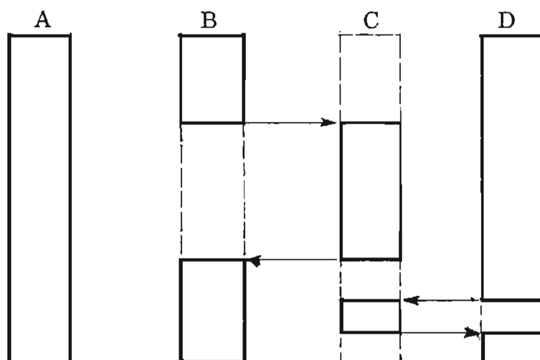
om B's arbete är utfört (WAIT B) och om så ej skulle vara fallet invänta B's avslutning.

Även B kan givetvis "lämna ifrån sig processorn" p g a erforderlig in/utmatning. Figur 9. 2:7 visar ett sådant fall där A startar upp B och ger den en högre "dispatching priority" än sig själv. A antas ej ha någon tidsfördröjande in/utmatning.



Figur 9. 2:7

I system med <u>flera</u> processorer kan en mer "sann" (fullständigare) parallellitet erhållas, enär programsegmenten då i stor utsträckning kan exekveras helt parallellt utan större hänsyn till in/utmatning.



Figur 9. 2:8

I figur 9. 2:8 har vi skisserat exekvering av fyra program samtidigt i ett

treprocessorsystem. Vi bör dock notera att ännu 1970 relativt få kommersiellt tillgängliga operativsystem medger möjlighet till styrning av flerprocessorsystem. Även p g a hittillsvarande priser på processorer har dylika system hittills visat sig ovanliga.

9.3 Programegenskaper vid systemanvändning

Relaterat till det sätt på vilket laddningsbara program, laddmoduler, exekveras kan följande uppdelning tänkas:

Självmodifierande program. Ett dylikt program ändrar sina instruktioner under exekveringen. Samma program kommer ej på ett normalt sätt att kunna exekveras igen vid ett senare tillfälle. För att återställa programmet i exekverbart skick måste antingen ett "städningsarbete" utföras eller en ny kopia av programmet inläsas från programbiblioteket.

Seriellt återanvändbara program. Som namnet säger kan program av denna typ användas i serie av olika "modersprocesser" och får ses som "exklusivt användbara" resurser. Normalt har programmen en egen inbyggd initialiseringsprocedur som genomlöpes innan användningen börjar. Om flera processer vill simultant använda sig av ett seriellt återanvändbart program måste flera kopior av detsamma inläsas till primärminnet.

Icke självmodifierande (reentrant) program. Ett program av denna typ är utformat så att det på intet sätt modifierar sig själv under exekvering. Det kan ses som ett "read-only" program. Eftersom ett reentrant program aldrig modifieras under sin exekvering kan det laddas vid ett tillfälle och sedan frekvent användas av varje process som befinner sig inom systemet och har "tillstånd" att anropa det aktuella programmet.

Typiska exempel på reentrant program finner vi inom operativsystemen själva. Ett belysande exempel utgörs av en interpretator, använd i dialog. Ett flertal användarprogram kan här samtidigt befinna sig under bearbetning av interpretatorn. Endast en kopia av interpretatorn behöver därvid befinna sig i primärminnet. Uppenbarligen är det av stor betydelse att kunna hålla systemets eget minneskrav nere på detta sätt. Som ett annat exempel på en systemfunktion av reentrant typ kan vi välja ett operativsystems jobbinläsare, som ej sällan är utformad för simultan bearbetning av flera inkommande jobbströmmar.

Minnesekonomi eller - hushållande talar således för användande av reentrant - teknik i så stor utsträckning som möjligt. Drivs denna linje för

hårt kan emellertid systemets administrationstidsåtgång komma att starkt öka. Minnesskyddsproblemet kan bli mera svårhanterligt och löpande kontroller får utföras om auktoriserat minne efterfrågas av de arbetande programmen.

Om ett reentrant program som "betjänar" en process A avbryts av något skäl undanlagras registerinnehåll och programstatus i en till den avbrutna processen A associerad minnesarea. Oberoende av hur reentrantprogrammet sedan används av andra processer kan därefter verksamheten med processen A fortsättas så snart erforderlig avbrottshantering m m avslutas.

9.4 Diagnostik, test och felsökning

Även om kostnader för inkörning/felsökning av användarprogram ofta är substantiella, kan kostnader som följd av ännu oupptäckta programfel visa sig än större. De flesta operativsystem tillhandahåller därför olika faciliteter för att söka minimera tid och kostnader för så fullständig programinkörning som möjligt. Bland olika hjälpmedel härför kan vi nämna:

1. Diagnostik. Samtliga översättare (kompilatorer och assemblers) ger mer eller mindre belysande kommentarer till de programfel som upptäcks av systemet under översättning och programinkörning. Vi kan här skilja mellan två typer av diagnostik:

- diagnostik på källprogramnivå
- diagnostik på objektprogramnivå

Den första typen av diagnostiska meddelanden, på källprogramnivå, genereras av kompilatorn/assemblern under källprogramöversättningen. Förekomst av svårare programsyntaxfel leder, i satsvis bearbetningsmiljö, till att ingen objektkod genereras, utan översättningens resultat består i stort sett endast av felmeddelanden. I situationer då (via styrsatser) exempelvis kompilering och omedelbart påföljande exekvering är begärd, ser vi exempel på hur utförandet av en process, här kompilering, kan leda till att nästföljande begärda process, här exekvering, negligeras av systemet. Systemet utför alltså här en av programmeraren ej explicit begärd "rimlighetskontrollerande" funktion. Mindre allvarliga programfel, trots vilka senare exekvering av det generarade objektprogrammet kan fullföljas, leder ofta endast till påpekanden efter översättningen, och det lämnas till programmeraren att bedöma huruvida objektprogrammet senare verkligen skall "rättas till" eller ej.

I time-sharing miljö lämnar översättaren (interpretatorn),naturligt nog, källprogramdiagnostiska meddelanden där så är möjligt direkt efter mottagandet av varje programsats. Detta ses vanligen som en stor fördel vid programmeringen.

I satsvisa system där länkning normalt utförs i ett separat jobbsteg (exempelvis IBM OS/360) lämnas vanligen diagnostiska länk-meddelanden vid utförandet av detta jobbsteg. Det kan här röra sig om att felaktig överlagringsstruktur är begärd, att man har begärt programelement vilka ej återfunnits i angivna bibliotek o dyl.

Den andra typen av diagnostik, på objektprogramnivå, ges av systemet (via delvis tidigare indikation från översättare) efter uppträdande av oförutsedda programavbrott. Givetvis kan ej alla uppdykande typer av ovälkomna programavbrott av systemet klarläggas till sitt ursprung. Icke desto mindre förekommer ofta likartade fel, om vilka systemet kan ge förhållandevis klara besked. Det kan röra sig om t ex

- felaktiga format och konventioner i datamängder
- fel hos datamängder (t ex läsning av filslutsmarkering)
- overflow/underflow vid beräkningar
- felaktiga indexgränser i variabelfält
- för litet tilldelat primärminne
- oändliga slingor i program (loopar) m m

Emedan utförliga beskrivningar om formellt utseende hos aktuell diagnostik återfinns i programmerarmanualer väljer vi att här ej vidare belysa dessa. Anmärkas kan dock att dessa diagnostiska meddelanden har från leverantör till leverantör olika grad av användarvänlighet. I vissa fall fås tämligen verbal beskrivning av felet. I andra fall lämnas endast en felkod vars betydelse får sökas i speciella kataloger.

2. Syntaxkontrollerande program. Önskemålet om korta översättningstider under programinkörning har, som vi tidigare noterat, lett till att ej sällan olika kompilatorer för samma språk tillhandahålls av samma leverantör. Ytterligare ett steg på denna väg har tagits i och med förekomst av "översättare" som enbart kontrollerar den grammatikaliska korrektheten i användarprogrammen. Sålunda genereras av dessa senare hjälpprogram ingen objektkod alls, utan deras resultatutmatning består enbart av en samling kommentarer rörande eventuella syntaktiska programfel. Erforderlig maskintidsåtgång för dylik syntaxkontroll blir därför ofta mycket kort. Tre ambitionsnivåer hos översättare (om vi räknar in den sistnämnda under denna rubrik) kan alltså f n sägas existera:

a) ren syntaxkontroll

b) "quick-and-dirty" översättning

c) optimerande översättning

Under inkörning av ett större användarprogram som sedan räknas bli frekvent använt använder man sig normalt av dessa tre översättare i ordningsföljden a) till c).

3. Speciella programsatser för testning. Vid inkörning av program uppträder under exekveringens förlopp bl a önskemål om utskrift av vissa variablers "mellanresultat". Härigenom kan exempelvis olika programdelars samspel belysas. Ett sätt att klara detta är att i programmet lägga in instruktioner som orsakar testutskrifter av mellanresultat. Instruktionerna kan sedan borttas då programmet färdiginkörts. Det har dock praktiskt visat sig mindre klokt att blanda "ordinarie" programinstruktioner med testutskriftinstruktionerna. Numera medger systemen därför i regel möjlighet att tillhandahålla testutskriftinstruktionerna som en separat och från programmet fristående del. I denna testinstruktionsdel anges bl a vilka programvariabler som önskas utskrivna, i vilka format, samt under vilka villkor (under exekveringens förlopp). Därmed uppnås att:

- testdelen enkelt kan borttas när programmet färdiguttestats
- testdelen enkelt kan ändras utan erforderlig omkompilering/omassemblering av det tillhörande programmet.
- risken att introducera nya programfel genom manipulering med testutskrifterna hålls låg.

I vissa system kan testinstruktionerna infogas i programmet under pro - ramskrivandet. De separeras dock automatiskt vid kompilering/assemblering och placeras i en fristående modul.

Man har normalt möjlighet att via speciella testrutiner få fortlöpande information om:

- variablers värden (ev. endast när dessa förändras)
- registerinnehåll
- tillståndsdata för filer m m
- programkartor (minnesdisposition)
- utskrifter betingade av speciella aritmetiska eller logiska testvillkor
- speciella testräknares (hjälpvariablers) värden
- s k trace-utskrifter, dvs listning av utförda instruktioner och därmed sammanhängande innehåll i ackumulatorregister, indexregister m m.

De mer omfattande hjälpmedlen av denna typ liksom flera andra systemfaciliteter kommer man vanligen endast åt från assemblyprogramnivå. De problemorienterade språken är därvid ännu något styvmoderlig behandlade.

4. Minnesutskrifter. Som ett komplement till ovan nämnda utskriftsmöjligheter finns systemrutiner tillgängliga för utskrift av hela eller delar av primärminnet. En gängse benämning på en dylik minnesutskrift är en dump. I tidigare dator/programvarusystem var färdighet i att läsa minnesutskrifter, ofta oktala eller hexadecimala, en nödvändighet för system- och programmerarpersonal. Nyare system fäster i stället tonvikt på sådana hjälpmedel som nämnts i föregående avsnitt. Den datornära systempersonalen är emellertid fortfarande ofta i sista hand hänvisade till (vanligen oktala) minnesutskrifter.

9.5 Köravbrott och reetablering

En annan typ av köravbrott än som ovan berörts utgörs av de ur programmerarens synpunkt avsiktliga avbrotten. Vi tänker här inte på de, indirekt avsiktliga, avbrott som sammanhänger med begärd in/utmatning e d, utan på i källprogrammet direkt begärda avbrott av programexekveringen. Ett dylikt avbrott benämns vanligen brytpunkt eller kontrollpunkt med senare åtföljande återstart. Den engelska motsvarigheten är checkpoint/restart. Skäl för användande av en dylik facilitet är bl a

- man befarar att programmet kan "gå snett" (avbrytas p g a programfel eller maskinfel) vid ett långt framskridet skede av exekveringen, och man vill spara delresultat som tidigare framräknats i programmet för att ej behöva göra om hela körningen. Situationen är typisk för administrativa rutiner där körtiden för ett program normalt är av storleksordningen en timme.

- alltför lång total maskintidsåtgång uppskattas erforderlig för programexekveringen i sin helhet för att passa ett visst driftpass e d, och programmets exekvering önskas därför uppdelad i delar, där en efterföljande programdelsexekvering kan startas där den föregående avslutats.

- programexekveringen önskas avbruten, med senare återstartsmöjlighet, av andra skäl t ex ankomst av ett högprioriterat program.

En brytpunkt av denna typ etableras vid exekvering av en speciella bryt-

punktsinstruktion i användarprogrammet. Denna typ av instruktion finns tillgänglig i de flesta större operativsystems repertoir. Brytpunkter kan inläggas antingen vid ett jobbstegs början eller på valfri plats inom ett program. I det första fallet behöver därvid resultat från tidigare jobbsteg inom jobbet inte gå förlorade.

Exekvering av brytpunktsinstruktionen innebär anrop av (uthopp till) en systemrutin som i ett avpassat sekundärminnesutrymme undanlagrar vissa hittills uppnådda delresultat samt annan tillståndsinformation för det berörda användarprogrammet. Det är givetvis av vital betydelse att denna information säkert sparas tills återstart blir aktuell. I de fall då flera brytpunkter inplaceras i samma jobbsteg kan program-information från en tidigare brytpunkt förklaras inaktuell i och med etablering av en i programmet senare uppträdande brytpunkt.

Återstart, som sker från programinstruktionen omedelbart efter brytpunktsinstruktionen, kan ske dels automatiskt, och dels manuellt. Den automatiska återstarten kan t ex vara aktuell under förutsättning att någon form av enklare systemfel inträffat under exekveringen av ett jobbsteg och datorsystemet ej kan anses vara i funktionsodugligt skick. Anvisningar om återstart kan anges i samband med ett av jobbstegets styrsatser, dvs före jobbsteget. Om jobbsteget fullföljs, dvs exekveras korrekt, kommer en dylik begäran om "automatisk återstart i händelse av systemfel" att lämnas utan åtgärd. Ibland måste en automatisk återstart auktoriseras av operatören.

Manuell återstart kan komma i fråga då ett jobbsteg avbrutits p g a dator- eller programfel och hela systemet därvid avstannat. Systemet återbildar då situationen från tillfället för brytpunktsinstruktionens utförande med hjälp av den sekundärminnessparade brytpunktsinformationen angående programstatus.

Om brytpunktsinformationen har anvisats ett direktminnesutrymme överlagrar normalt varje ny brytpunkt information om den föregående brytpunkten. Används magnetband kan information on flera brytpunkter tillbaka i tiden sparas.

Det inses att om ett program använder t ex kortläsare, remsläsare eller läsare för streckmärkta dokument som indataorgan är en "tillbakapositionering" av dessa infiler utan speciella åtgärder ej möjlig och en brytpunkt kan i dylika fall ej, utan speciella manuella åtgärder, etableras.

Den serie av händelser som vanligen aktiveras i samband med upprättande av en brytpunkt är bl a

1. alla register (ackumulator, index m m) sparas

2. alla pågående datatransporter avseende det aktuella programmet tillåts avslutas

3. jobbets ännu icke utförda styrinstruktioner sparas

4. styrprogrammets tillståndsdata avseende aktuellt program sparas

5. aktuell programs primärminnesinnehåll lagras undan

6. läget för alla magnetbandsfiler noteras (t ex genom att blocken är numrerade)

7. ev. temporära filer som tilldelats jobbet, lagras undan.

Vi vill nu kortfattat i detta kapitelavsnitt belysa några av de problem som uppträder i samband med uppkomst av fel i och reetablering i reelltidsmiljö. Det bör kunna konstateras att fel i användar- eller systemprogram betydligt lättare kan diagnosticeras, och ofta även avhjälpas, i satsvis miljö än i reelltidsmiljö. Den satsvisa bearbetningen utmärks av att användarprogrammen i stor utsträckning kan ses som autonoma (självständiga), och felverkningar fortplantar sig därför ej i samma utsträckning från ett program till ett annat. I reelltidsmiljö är interaktionen mellan program under b arbetning mycket intim och uppvisar ett komplicerat mönster. Inkommande transaktioner kan aktivera flera parallella processer och felaktiga indata om de ej upptäcks, kan skapa komplicerade följdfel även i perifert berörda program. Den kanske mest komplicerade faktorn i RT-system är tiden. Vissa typer av fel upptäcks endast när systemet är mycket hårt belastat och systemets resursadministrationsfunktioner är tidsmässigt hårt ansträngda.

Det är därför av stor betydelse att kontroller mot uppkomst av fel systematiskt inbyggs i reelltidsprogram. Dylika kontroller kan beröra:

- förebyggande av operatörsfel
- förebyggande av maskinvarumässigt uppkommande fel
- förebyggande av dataöverföringsfel
- kontroll mot partiell fil- och programförstörelse p g a fel i data
- i möjligaste mån förebyggande av att fel i ett systemprogram inducerar fel i andra systemprogram
- skapande av systemfunktioner som möjliggör felsökning under pågående bearbetning av "normala" transaktioner.
- tillhandahållande av programinkörningshjälpmedel som möjliggör så fullständigt uttestande av reelltidsprogram som möjligt

m m.

Bland hjälpmedel att upptäcka fel kan vi, förutom vad som ovan nämnts i detta avsnitt, i anknytning till reelltidssystem nämna:

- Loggning av identifikation av samtliga transaktioner under ett givet tidsintervall
- Loggning av avbrott, ett systematiskt bevarande av uppkomna feltyper och frekvenser
- Operatörstillagda eller datorsystemtillagda kontrollbeteckningar på transaktioner
- Systematiskt genomlöpande av filöversynsrutiner för kontroll emot "oegentligheter" i filsystemet

m m

De fel, som trots ovan nämnda förebyggande aktioner, uppkommer, kan åtgärdas på olika sätt. Det är uppenbart att ett klokt användande av ovan nämnda hjälpmedel för felupptäckande också ofta ger direkt indikation om hur felen bör korrigeras. Emellertid kan sådan skada ha bringats, att mera fullständig systemöversyn kan bedömas erforderlig. I reelltidsmiljö är fullständig systemöversyn ofta utomordentligt svår att genomföra p g a vikten av att kunna uppehålla transaktionsbearbetningskapaciteten.

I vissa situationer är filrekonstruktion aktuell. Även lång tid efter det att ett system tagits i full drift upptäcks sporadiskt programfel. En tillfällig överskrivning av register, ett felaktigt användande av datamedia osv uppträder då och då. För filrekonstruktion är det betydelsefullt att löpande och med jämna tidsmellanrum dumpa filer eller "filförändringar" på bakgrundsminnen t ex av magnetbandstyp. Mera omfattande reelltidssystem kan använda sig av ett hierarkiskt system av back-up-datamedia, varvid det kan bedömas erforderligt att ha kopior av en och samma fil på en hierarki av sekundärminnen (t ex trumma, skiva, cellminne, magnetband).

En aktiv och löpande revision, dvs översyn, av användarprogramsystemet är normalt en nödvändighet. Det kan ofta vara klokt att ge möjlighet för revisorsexpertis att medverka redan vid tiden för systemutformning, för att försäkra sig om adekvata kontrollmöjligheter.

Slutligen kan vi påpeka att viss löpande kontroll över operatörernas agerande är av betydelse i såväl satsvis som i reelltidsmiljö. Detta kan ske både genom "programmerad" kontrollverksamhet och genom fast styrning av operatörsarbetet. Det har sålunda i praktiken förekommit att mindre betydelsefulla systemfel med jämna mellanrum avsiktligt genererats, för att garantera att operatörspersonalen håller sig ajour med åtgärder för reetablering.

9.6 Service-program

I detta kapitel-avsnitt vill vi kortfattat redogöra för några av de funktioner som kan utföras med hjälp av ett datorsystem vanligen tillhörande serviceprogram (eng. utilities). Det kan förefalla att beteckningen serviceprogram egentligen kunda gälla för flertalet av de system-program som normalt inräknas i begreppet operativsystem. Man skulle då i begreppet kunna räkna in styrprogram, jobbinläsningsprogram, jobbplaneringsprogram, länkare, laddare, övervakare, översättare m m emedan de alla utför servicefunktioner åt användarprogram. Emellertid brukar så ej göras. Med serviceprogram avses här hjälpprogram av "general purpose"-karaktär som avser att effektivisera driften av ett datorsystem och som kan nyttjas från användarnivå. Att dessa hjälpprogram inräknas i begreppet operativsystem härrör från synsättet att i ett operativsystem inräkna alla av leverantören underhållna "general-purpose"-program i ett datorsystem som ej är att betrakta som användarprogram inriktade på viss bestämd applikation.

Serviceprogrammen kan anses befinna sig relativt "långt ut från kärnan" av ett operativsystem. De styr ej flödet av jobb genom datorsystemet, utan finns tillgängliga för assistans åt såväl andra systemprogram som åt under konstruktion eller bearbetning varande användarprogram.

Mängden och typen av olika serviceprogram som finns tillgängliga inom ett datorsystem varierar mellan olika leverantörer av datorsystem. Bland olika tänkbara sätt att kategorisera serviceprogrammen väljer vi en grovindelning baserad på sättet hur ett dylikt program anropas.

1. Serviceprogram som anropas på källprogramnivå, dvs som anropas inom en process. Dessa kallas för subrutiner eller procedurer.

2. Serviceprogram som anropas på styrspråksnivå, dvs som i och med anropet skapar ett eget jobbsteg vid utförandet. Vi kan kalla dessa program fristående serviceprogram.

Procedurer-subrutiner. Dessa program, normalt av typen procedurer (subrutiner), anropas inom källprogrammet direkt med användning av respektive rutiners namn. Härvid överförs genom parametrar, data till proceduren för bearbetning. Med hänsyn till användningsområde kan dessa procedurer indelas i

a) Matematiska hjälprutiner. Dessa rutiner befriar programmeraren från att själv tillhandahålla algoritmer för beräkning av elementära såväl som mer komplexa matematiska funktioner. Som exempel kan nämnas:

- absolutvärde
- logaritmfunktion
- exponentialfunktion
- trigonometriska funktioner
- kvadratrot
- rötter till polynom av olika gradtal och slag
- statistiska fördelningsfunktioner
- logiska funktioner

m m

b) Datamanipulerande hjälprutiner. Med hjälp av dessa rutiner har programmeraren möjlighet att enkelt "själv" utföra vissa manipulationer på "detaljerad" nivå inom programmet. Exempel:

- kodomvandling
- packning resp uppackning av tal i flytande form
- konversion av tal mellan olika talsystem
- kontroll mot spill vid aritmetiska beräkningar
- sortering av talföljder
- flyttning av datafält inom primärminnet

m m

c) Hjälprutiner för in/utmatning o d. Dessa rutiner utför bl a följande arbete:

- brytpunkt/återstart-funktion
- kontroll angående om filslut uppnåtts vid läsning på yttre minnen
- kontroll mot paritetsfel vid läsning på yttre minnen
- kontroll av status för data i in/ut-buffertar
- placering av programmet i viloläge tills all pågående in/utmatning avslutats
- utmatning av partiell minnesutskrift (under programmets exekvering)

m m

Fristående serviceprogram. Dessa program kan anses befinna sig i operativsystemets periferi. De används främst till manipulering och organisation av data, både sådana som hör till systemet självt (OS-data) och de som hör till användarprogrammen. Serviceprogrammen används på liknande sätt som standardiserade applikationsprogram, dvs de anropas normalt från styrspråksnivå på motsvarande sätt som uppstartning av ett system- eller användarprogram. I vissa system har dessa program givits speciella programnamn, för att skilja dem från gängse användarprogram. Programnamnen kan t ex samtliga ha gemensam inledande identifikatordel, i förekommande fall de två eller tre första karaktärerna i identifierarna.

Vi kan indela dessa program i två klasser:

a) Systemservice. Det är här fråga om hjälpprogram för att hålla operativsystemets datamängder aktuella. Som exempel kan bl a följande funktioner utföras:

- manipulering av systemdata, bl a
  namngivning,
  uppdatering (infogning, ändring, radering),
  katalogisering,
  sammankoppling,
  förflyttning,
- kopiering av systemdatamängder
- listning av systemdata
- standardetikettering av magnetband
- redigering och listning av uppsamlad körnings- och felstatistik

m m

b) Användarservice. De program vi här främst avser används till manipulering av användardatamängder, bl a för redigering och/eller omorganisation av användarfiler.

Vi noterar här program för:

- omorganisation av datamängder, t ex ändring från sekvensiell till partitionerad lagringsform
- kopiering
- konsolidering (sammanslagning)
- utmatning (listning)
- uppdatering (utökning, ändring, radering) av datamängder

m m

Som vi tidigare nämnt är skillnaderna mellan de av olika leverantörer tillhandahållna serviceprogrammen stora. Dessa rutiner kan dock ej anses vara av större intresse i detta sammanhang och därför uppehåller vi oss ej längre med dem.

9.7. Systemgenerering och systeminitiering

Avslutningsvis i detta kapitel vill vi orientera om två "servicefunktioner", som vi inte tidigare belyst men som är (i mer eller mindre avancerad form) nödvändiga för alla operativsystem. Vi avser då de program vars

exekvering utför generering av systemet (systemkonfigurering) och en initialisering (uppstartning) av systemet. I motsats till minidatorer, som ofta kan vara uppkopplade för användarbruk 24 timmar om dygnet flera veckor i följd, måste de stora giganterna ha ca 1 à 2 timmars preventiv service per dygn för att "överleva". Det innebär att systemen måste initialiseras minst en gång per dygn. Systemgenereringsprogrammets syfte är att "skräddarsy" ett operativsystem för en viss bestämd installation. Ingångsdata till detta program är

- information om aktuellt datorsystems konfiguration
- operativsystemets programmoduler (komponenter)
- riktlinjer (systemgenereringsinstruktioner) för generering av det installationsanpassade operativsystemet

Genereringsprogrammet accepterar också leverantörs- eller användarspecificerade modifikationer och/eller tillägg till operativsystemet. De olika (förbättrade) versionerna av leverantörernas operativsystem distribueras till användare vanligen som uppdaterings-modifierings-data till tidigare OS-versioner. Varje leverantör har härvid egna etablerade konventioner.

Systemgenereringsinstruktionerna kan gälla, vid större system, en mängd olika detaljer[1]. Vi nämner nedan i illustrationssyfte blott några av dem.

- specifikation av sekundärminnesenhet(er) där op-systemet avses redidera
- bestämning av tidskvanta (vid "time-slicing") för program av olika kategorier (sats/kövis, tidsdelning, reelltid)
- specifikation av standardvärden (om ej annat anges av användaren i JOB, RUN eller motsvarande styrkort) för max processortid, max antal rader/sidor utmatning osv
- beskrivning av till systemet anslutna fjärrterminaler och mellanliggande kommunikationsfaciliteter
- reservering (inbokning) av direktminnesutrymme för speciella ändamål (t ex in-utköer)
- flyttning av diverse mindre frekvent nyttjade OS-rutiner till långsammare direktminnen
- specifikation av max antal simultana processer under exekvering
- specifikation av sekundärminnesenhet för lagring av systemets logg och/eller debiteringsdata
- specifikation av tillgängligt primärminne

---

1) Vissa av nedan angivna specifikationer kan även ges vid initiering av systemet.

- specifikation av tillgängliga perifera enheter och deras kanalanslutning
- specifikation av buffertutrymmen för datakommunikation till fjärrterminaler
- infogning av nya systemprogram

mm

Resultatet av systemgenerering är vanligen en sekvensfil som innehåller operativsystemet inklusive dess programbibliotek. Sekvensfiler, normalt lagrad på magnetband laddas initialt genom en s k "boot-strap" metod: genom diverse operatörshandgrepp initieras en primitiv inläsningssekvens (ibland inbyggd i maskinvaran) vilken som första åtgärd läser in ett mer fullständigt inläsningsprogram.

Om datorsystemet är direktminnesorienterat sker nu en inläsning av den genererade OS-filen och överföring av dess komponenter till olika direktminnesutrymmen. Den del av operativsystemet som skall vara resident i primärminnet inläses till detsamma och systemet är nu i princip klart att kommunicera med operatören och bearbeta användarjobb. Normalt har operatören möjlighet att under drift ändra viss information i operativsystemets tabeller såsom tillstånd och tillgänglighet av olika hård- varu- och programvaruresurser.

Initiering av operativsystemet, när det redan är lagrat på ett direktminne är vanligen en enkel process som ej kräver längre tid än ca 1 minut.

Litteratur

1. Leverantörsmanualer

2. Lanzano, B.C., "Loader Standardization for Overlay Programs". CACM, Oct. 1969.

3. Martin, J., "Design of real-time computer systems". Prentice-Hall, 1967.

4. Davidson, L-O. & Josefsson, P-O., "Återstart- och räddningsfunktioner i realtidsmiljö". 3-betygsuppsats, Institutionen för Informationsbehandling, KTH, våren 1970.

10. OPERATIVSYSTEM - ÖNSKEMÅL OCH MÅL

10.1 Inledning

Det är förvånande hur litet målsättningsproblem i samband med operativsystem har diskuterats. Kanske är en av anledningarna till att operativsystem i många fall vuxit ut till svåröverblickbara, svårbemästrade och resurskrävande kolosser (inte minst vad gäller deras datorresursbehov), frånvaron av en väl genomtänkt målstruktur vid konstruktionsarbetets början. Man har ofta av olika, måhända mest försäljningstekniska, skäl skruvat upp den initiala komplexitets- och ambitionsgraden och önskat att systemet skall omfatta snart sagt alla tänkbara funktioner, för en mängd olika driftsformer. Resultatet har praktiskt taget i samtliga fall lett till en kraftig leveransförsening. Dessutom har det levererade systemet ej omfattat många av de initialt utlovade funktionerna och prestanda har normalt legat under förväntan.

Dagens i drift och planering varande datorsystem (normalt av en-processortyp) kan knappast betecknas som komplexa med tanke på vad vi kan vänta oss i framtiden med nät av olikstora datorer av skilda fabrikat, databaser och terminaler. Att tekniken för operativsystemkonstruktion i dag ligger långt efter maskinvaruutvecklingen och de möjligheter som maskinvaran generellt torde erbjuda verkar alarmerande för framtiden.

De centrala delarna av ett operativsystem arbetar med övervakning och styrning av system- och användarprograms exekvering. Varje form av styrning baseras medvetet eller omedvetet på en överordnad målstruktur. Denna struktur behöver således ej vara explicit angiven och definierad. Resultatet blir då att man har svårt, för att inte säga omöjligt, att avgöra om styrningen är effektiv eller om operativsystemet i övrigt är välkonstruerat och ändamålsenligt. Ännu svårare blir det att jämföra och utvärdera flera operativsystem eller konstruktioner om en klar och operationell målformulering saknas. Detta är dessvärre nästan undantagslöst fallet i dag (1970) och gäller ej enbart operativsystem utan de flesta "system" i allmänhet.

En mindre väl genomtänkt målstruktur medför risk för "suboptimering" såväl vid konstruktion som vid utvärdering i samband med anskaffning av datorsystem. Alltför stor betydelse kan då fästas vid funktioner som konstruktören/utvärderaren fäster stor subjektiv vikt vid. Detta sker i de flesta fall på bekostnad av "värdet" hos övriga funktioner.

Åtskilliga exempel härpå noteras även vid konstruktion eller utvärdering av kompilatorer där ofta oförtjänt stor vikt fästs vid speciella detaljer. Detta kan t ex beträffande Algol gälla språkets maskinrepresentations praktiska anslutning till Algolrapporten, own-begreppet eller rekursiva procedurer. I de flesta sammanhang[1] viktigare egenskaper, såsom diagnostik, in/utmatningseffektivitet, möjlighet till stränghantering etc har ofta givits en underordnad betydelse. (Därmed har inte sagts att situationer ej kan förekomma där denna "viktning" kan vara riktig.) På samma sätt kan vid utvärdering av operativsystem en oproportionerligt stor "vikt" fästas vid t ex det teoretiska antalet möjliga samtidigt aktiva program i primärminnet, existensen av en "roll-out/roll-in"-funktion, e d.

10.2 Allmänt om målproblem

Innan vi går in på en diskussion om önskemål och mål för operativsystem är det dock nödvändigt att beröra begreppsbildningen kring målsättning i allmänhet. Langefors har i sin bok "System för företagsstyrning" (Studentlitteratur, Lund, 1968) en ingående diskussion om målproblem. De begrepp avseende önskemål och mål som följer har vi i huvudsak hämtat från denna bok.

Langefors inför bl a följande begrepp:

Ultimärmål - de yttersta syftena för systemet eller verksamheten. Styrning, dvs fattande av beslut, anses nyttig om den ger ett positivt bidrag till ultimärmålens uppfyllelse.

Delmål- medel (mål) för tillgodoseende av andra (högre) mål. "Goda" delmål lämnar positiva bidrag till ultimärmålens uppfyllelse.

Primärmål - mål, som måste tilldelas högsta vikt vid beslut när flera mål kommer i konflikt. Primärmålen anger normalt vad som först måste uppfyllas för att ultimärmålen i önskad utsträckning skall kunna realiseras.

Vidare säges ett mål vara operativt om vi därmed avser att det tages som medveten bakgrund för styrning och planering av den operativa verksamheten (i vårt fall konstruktion av systemet). Operationellt säges ett mål vara om dess uppfyllnad eller, i vissa fall, uppfyllnadsgrad, kan avgöras (dvs på något sätt mätas eller konstateras).

---

1) Med detta avses att det dominerande antalet Algolprogram ej utnyttjar dessa funktioner.

På grund av svårigheten att formulera och identifiera ultimärmål anser Langefors det fördelaktigt att först göra en förteckning över önskemålen och därefter försöka definiera de ultimära önskemålen. Distinktionen mellan önskemål och mål innebär att önskemål uttrycker våra önskningar men målen anger vad vi beslutar oss för att uppnå eller eftersträva. Önskemål som man är beredd att tillgodose anses som ultimära. Ofta är önskemålen motstridiga. Det blir därför lämpligt att uppställa precedensstruktur och preferensstruktur för dessa. Tillgängliga resurser och önskvärd aspirationsnivå bestämmer sedan hur totalmålsstrukturen och systemet av operativa mål kan uppfyllas.

Vi bör göra klart för oss att för att kunna diskutera förflyttning i riktning mot ett betraktat operationellt mål är det fundamentalt att målet, nuläget och förflyttningsmetoden beskrivs med samma mätskala.

En generell svårighet i måldiskussionssammanhang rör separering av begreppen mål och aktivitet som förväntas bidra till måluppfyllelse. Beskrivningar tenderar också ej sällan att i avsett allmänna målresonemang infoga mer eller mindre väl dolda värderingar. Vi har i detta kapitel allvarligt sökt undvika dylika sammanblandningar och värderingar, men vill inte göra anspråk på att fullständigt ha lyckats härmed.

10.3 Önskemål och operativsystem

De önskemål som vi kan diskutera i samband med operativsystem måste ses som delar i ett större system av önskemål[1] och som berör det system som omger konstruktionen eller användningen av operatisystemet. Givetvis gör detta situationen utomordentligt komplex, och någon seriös analys av problemet har oss veterligt ej heller publicerats. Vi gör inte anspråk på att här servera en fullständig och konsistent målstruktursbelysning avseende operativsystem. Vår förhoppning är dock att det som följer kan vara av värde vid bl a principiella diskussioner gällande önskemål, krav och utvärdering av operatisystem och inspirera till mer ambitiösa utredningar av problemkomplexet.

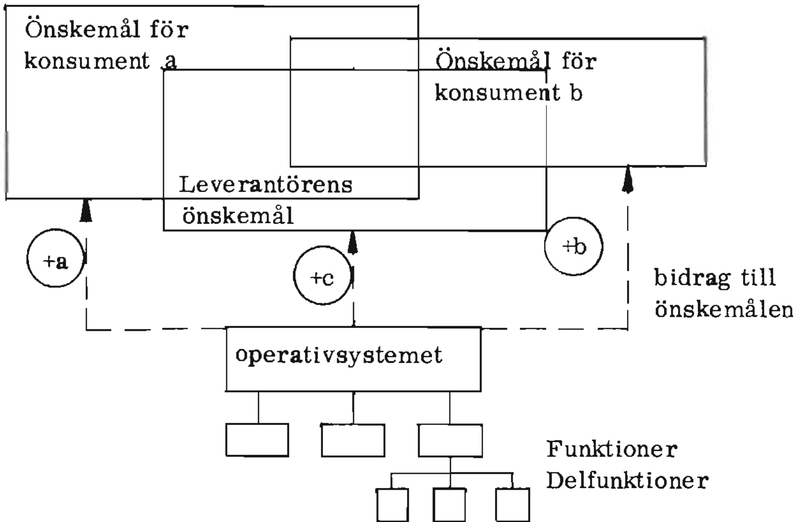
De funktioner som ett operativsystem skall innefatta samt deras egenskaper utgör operativa mål för konstruktionsarbetet. Genom att studera dessa kan vi ibland gissa oss till några av de högre mål och/eller önskemål, som, medvetet eller omedvetet, legat till grund för systemkonstruktionen.

---

1) Varje önskemål, eller önskemålstruktur, kan sannolikt ses som underordnad en "högre" önskemålsstruktur. Den totala önskemålsmängden kan därför ses som en "uppåt" icke begränsad mängd.

Ett operativsystem är med få undantag att betrakta som en produkt med ett stort försäljningsvärde. Detta medför att konstruktörens önskemål beträffande operativsystemet vanligen icke på alla punkter sammanfaller med användarens - konsumentens. Vi bör dock skilja på konstruktörer som arbetar under pressen från en försäljningsavdelning och konstruktörer i en icke-profitstyrd forskningsmiljö (t ex project MAC, MULTICS).

Ett operativsystem kan ej ha ett värde i sig självt utan får ett värde först när det betraktas i samband med avsedd datorutrustning och användarmiljö. Leverantören av datorutrustning kan anföra ett antal mål beträffande operativsystemets utformning som dock normalt bör betraktas som delmål vilka hjälper till att tillgodose leverantörsföretagets önskemål på högre nivå, såsom ökad försäljning, ökad good-will för företaget m m. Konsumenten ser (eller borde se) operativsystemt som ett av många hjälpmedel för att på ett ekonomiskt sätt förverkliga och operera en konkret del av ett informationssystem. Önskemål som berör operativsystemet varierar således från konsument till konsument beroende dels på olika förutsättningar och dels på att konsumenterna till stor del har olika högre önskemål som är överordnade önskemål avs operativsystemt. Det positiva bidrag som ett operativsystem lämnar till konsumenter resp leverantörer är därför olika (figur 3-1).



Figur 10.3:1

Ett operativsystem kan sägas till olika grad tillgodose olika avnämares önskemål. Önskemålsmängderna är normalt icke helt "överlappande". Användaren har också normalt svårt att "översätta" sina önskemål till motsvarande funktion för ett operativsystem.

En strävan från leverantörerna att vid konstruktionen av operativsystemet tillmötesgå så många förmodade användarönskemål som möjligt utan att systemet blir för resurskrävande för en viss speciell användarmiljö har utmynnat i försök att konstruera operativsystemen i starkt modulär form. Detta har medfört att de flesta operativsystem till viss grad är anpassbara till speciella kundorienterade förhållanden och önskemål. Denna fördel för vissa användare kan dock bli till nackedel eller extra belastning för andra.

Det bör påpekas att konsumenterna, åtminstone vid anskaffning och utvärdering av system, i allmänhet har haft betydande svårigheter när det gäller att specificera sina egna önskemål och krav[1]. I de flesta fall har detta berott på att man ej har haft en klar uppfattning om applikationsprogrammens önskvärda eller tilltänkta arbetssätt. Vi tar dock ej närmare upp det problemet då det närmast sammanhänger med området systemering. Vi bortser också från det mindre vanliga men dock förekommande fallet att ett mer eller mindre avancerat operativsystem i sig självt verkar attraktivt på konsumenten. Ovanstående belyser dock det svåra problemet att värdera operativsystem.

Innan vi diskuterar de konstruktionsmål ("design objectives") som redovisas av några skilda operativsystemtillverkare skall vi betrakta några tänkbara konsumentönskemål som har relationer till målen och arbetssättet hos ett operativsystem.

Vi kan tänka oss att ett informationssystem planeras vid ett företag och att ett av önskemålen är "minsta möjliga kostnad för systemet" under dess projekterade livslängd. Önskemålet är givetvis ett delönskemål sett ur hela informationssystemets och företagets synvinkel.

Vi kan anta att kostnaderna schablonmässigt kan fördelas enligt (nedanstående lista behöver här ej anses komplett):

Systemkostnad under kalkylerad livslängd

1. Systemutformnings- och omläggningskostnader
   - 1.1 Systemering
   - 1.2 Programmering
   - 1.3 Omläggning
   - 1.4 Förseingskostnader (kostnad alt "vinst" som uppstår om systemets driftsstart förskjuts).
   - 1.5 Kapitalvarukostnader

etc

1) Önskemålen har från början normalt karaktären av "krav" vilka dock mildras när man upptäcker att dessa sannolikt ej kan uppfyllas.

2. <u>Driftskostnader (löpande)</u>

2.1 Maskinvarukostnader
2.2 Servicekostnader
2.3 Personalkostnader
2.3.1 Operatörer
2.3.2 Programmerare (för progr underhåll o nyutveckling)
2.3.3 Systemerare (för kontinuerlig utveckling)
2.3.4 Övrig personal som är beroende av systemets tjänster
2.4 Övriga driftskostnader (ej direkt relaterade till operativsystem).

Önskemålet att hålla totalkostnaden nere innebär naturligtvis inte att detta behöver erhållas genom att pressa <u>samtliga</u> utgiftsposter ovan. Vi bortser från övriga faktorer som kan påverka dessa kostnader, för att här hålla oss till operativsystemen. Det är inget tvekan om att ett adekvat operativsystem på ett avgörande sätt måste påverka flera av de ovan angivna posterna. Vi anför här bara ett fåtal exempel:

<u>Systemering</u> underlättas om operativsystemet bl a tillhandahåller lämpliga delsystem för datastrukturering, filorganisation, filhantering och rutiner för hantering av katastrofsituationer

<u>Programmering</u> underlättas och effektiviseras av
- snabb omloppstid vid testkörning
- god diagnostik och lämpliga felsökn hjälprutiner
- arbetsvänligt styrspråk
- adekvata styrprogramfunktioner för speciella tillämpningar (t ex on-line drift)

<u>Omläggning</u> förbilligas och tiden därför förkortas om bl a ändamålsenliga filomläggningsrutiner finns.

<u>Driftskostnaderna</u> kan hållas låga genom att operativsystemet på maskinvarusidan ger bidrag till att höja utrustningens utnyttjandegrad för "produktiva" ändamål i stället för att självt lägga beslag på huvuddelen av resurserna. Exempel finns på att byten till nya dator/operativsystem vid i stort sett oförändrad belastning och jobbprofil krävt en kraftig ökning av maskintidskostnaden trots det nya datorsystemets överlägsna rent maskinvarumässiga prestandasiffror. Andra undersökningar har återigen indikerat det uppseendeväckande resultatet att ett visst arbete kört på samma dator med två olika operativsystem (av samma tillverkare) uppvisat skillnader i tidsåtgång på upp till en faktor 2. (dvs 100 %).

Operativsystemets bidrag, positiva som negativa, till olika kostnadsposter i det totala systemet är ännu ett tämligen outforskat område: Få empiriska[1) och än mindre teoretiska undersökningar har gjorts.

Mycket talar för att man gör alltför grova approximationer genom att behandla operativsystems olika egenskaper mer eller mindre som "intangibles" (som sedan bortses ifrån) vid datorutvärderingar.

Utför tankexperimentet att maskinvarukostnaden under systemets livslängd är, säg, 10 Mkr - en icke ovanlig siffra. Ett adekvat och effektivt operativsystem kanske kan sänka denna kostnad med - förslagsvis 20 %, dvs 2 Mkr. Vi har ändå inte talat om personalkostnader som torde vara av minst samma storleksordning. Personalens trivsel (ett önskemål) kan givetvis påverkas av operativsystemegenskaper med produktivitetsändring som en tänkbar följd.

Situationen kompliceras av att ett systemerar- och programmerarvänligt operativsystem kan vara mycket resurskrävande ur maskinsynpunkt. Vi har då problemet att evaluera eventuellt sänkta personalkostnader, ev ökat "trivselvärde", och ev tidsvinst avseende driftens igångsättning mot ev högre maskinkostnader. Eftersom ett informationssystems kostnader och intäkter infaller med olika fördelning under dess förväntade livstid bör även företagets kalkylräntefot på ett avgörande sätt påverka utvärderingen.

10.4 Några konstruktionsmål

I detta avsnitt skall vi återge och kommentera några idag existerande operativsystems s k "design objectives" eller "functional objektives" så som de framställts i litteraturen. Vi vill inte utesluta möjligheten att ytterligare, ej redovisade, mål/önskemål existerar. Vi börjar med OS/360 - IBM's operativsystem för datorserien 360. Därefter tar vi mer kortfattat upp några andra operativsystem. Anledningen till att vi uppehåller oss oproportionerligt länge vid OS/360 är att den diskussion som förts betr dess önskemål och mål i stor utsträckning kan appliceras även på övriga system.

10.4.1 OS/360

OS/360 är troligen det mest ambitiösa programvaruprojekt som någonsin startats. Antalet manår som nedlagts och ytterligare kommer att nedläggas

1) De undersökningar som gjorts avser satsvis kontra multipel-access-bearbetning (dialogprogrammering) i en starkt avgränsad problemlösningsmiljö och har där huvudsakligen betraktat sådana faktorer som total datortidsåtgång och totalt antal dagar för problemlösning.

i detta projekt är mycket stort.[1] Dokumentationen om detta system är så omfattande att det måste anses som en prestation att behärska och kunna göra bruk av mer än en bråkdel av systemets möjligheter.

Mealy (IBM Systems Journal 1, 1966) anger följande "grundläggande" mål (objectives) för operativsystemet OS/360:

(A) att "non-stop" kunna bearbeta en mängd av arbeten
(B) att kunna anpassas till olika omgivningar av applikationer och driftsformer

därefter anför Mealy ett antal viktiga "sekundära mål", nämligen

(a) ökad "throughput"
(b) kortare respenstid
(c) ökad produktivitet hos programmerarna
(d) anpassningsförmåga (för program till varierande maskinresurser)
(e) utbyggbarhet

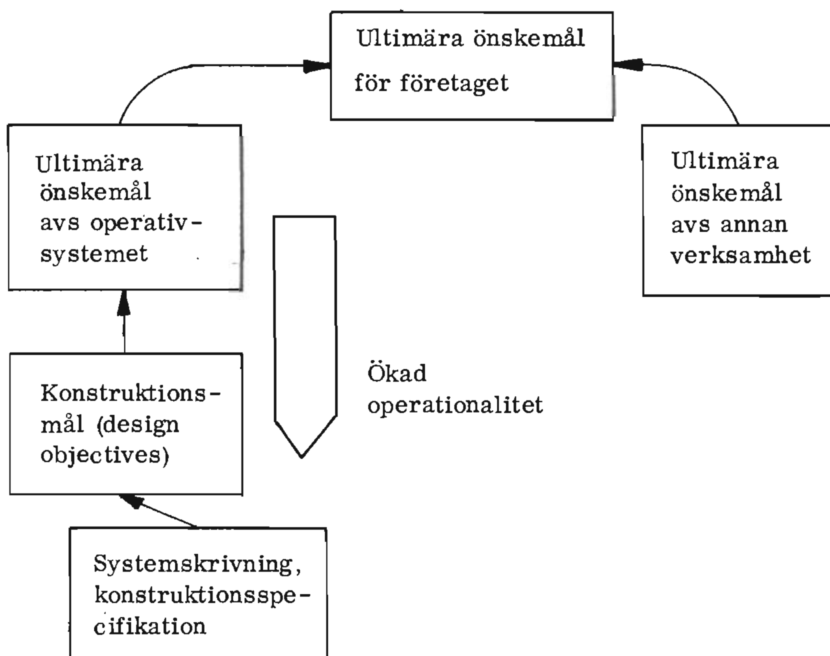
Dessa senare mål har uppenbarligen karaktären av ultimära önskemål och torde kunna tillämpas på samtliga operativsystem. Med undantag av (A) har inget av dem någon nämnvärd grad av operationalitet.

Således säger ökad "throughput" (mål a) i och för sig inte så mycket då jämförelseobjekt ej anges (men förmodligen IBM's äldre maskinserier avses). Hög throughput hade varit en naturligare formulering, men även denna är relaterad till en värdeskala, som tillsammans med själva begreppet throughput är svårdefinierad. Vi observerar här också det vanliga förhållandet att önskemålen kan till viss del vara motstridiga. Ökad "throughput", om det mätes i antalet arbeten (av viss typ) per tidsenhet, åstadkommes bl a genom att öka datorsystemets utnyttjandegrad. En sådan ökning kan medföra allvarliga köbildningar och eventuellt dålig verkningsgrad, vilket kan förlänga "responstiden"(mål b)[2]. På samma sätt kan önskemålet stor anpassningsförmåga, om det därvid ställer höga krav på programmerarna att behärska stora delar av systemets konventioner och styrinstruktioner, öka den relativa komplexitetsgraden vid programkonstruktion med lägre produktivitet som eventuell följd. Hög anpassningsförmåga och utbyggbarhet är troligen svåra att tillgodose utan att ge avkall på systemets effektivitet och "throughput" för vissa bestämda applikationer och driftsformer.

---

1) Det uppskattas att hittills (1970) drygt 1000 manår nedlagts i OS/360.
2) Den tid som förflyter från det ett arbet (eller en transaktion tillförts systemet och till dess att det behandlats.

Ett mål som måste betecknas som primärt och som sannolikt på ett negativt sätt bidrar till ovan redovisade önskemål, men lämnar positiva bidrag till andra här ej redovisade önskemål, är tidpunkten för operativsystemets (eller väsentliga delar därav) färdigställande. Detta måls relativa vikt torde i de flesta fall ökas över övriga (primär-) mål ju närmare operativsystemets annonserade färdigdatum man närmar sig.

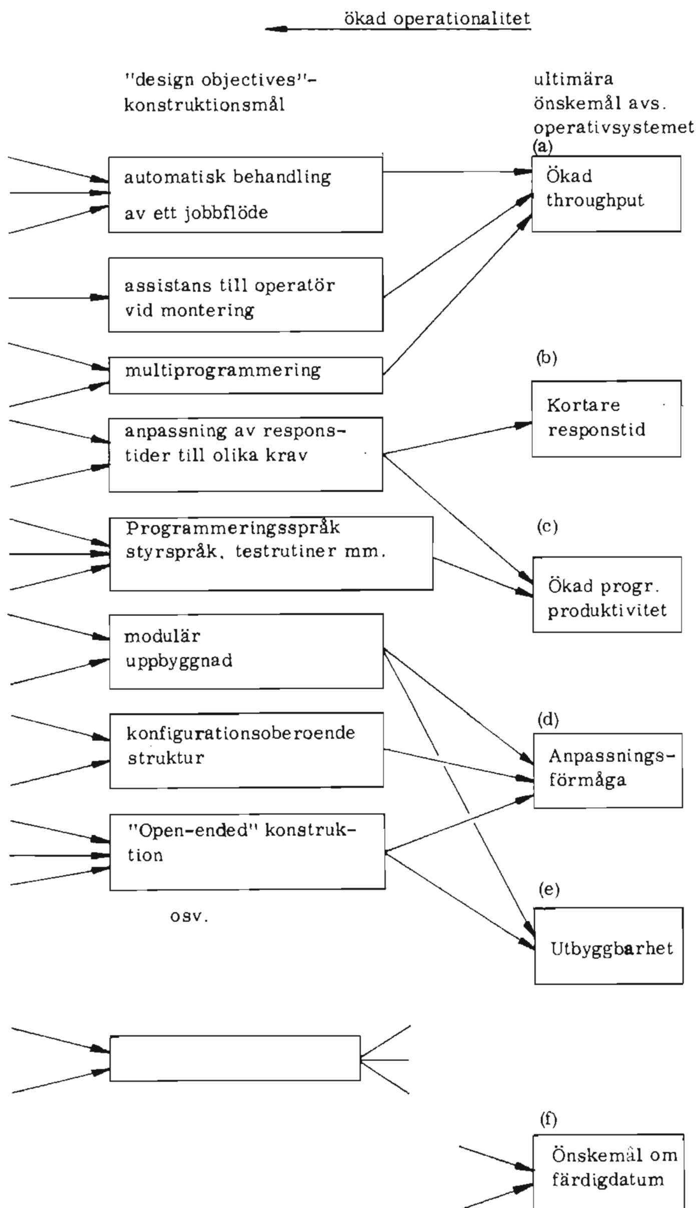


Figur 10.4.1:1

Exempel på bidragsstrukturen för önskemål och mål på olika operationalitetsnivåer.

Önskemålen ovan ges implicit ökad grad av operationalitet genom definition av delönskemål och delmål. De "slutliga" delmålen kan då ses som helt operationella och utgöres av operativsystemets konstruktionsspecifikationer - dvs en dokumentation över dess avsedda delsystem, deras relationer och egenskaper.

Mealy (2) diskuterar hur önskemålen (a) - (e) kan tillgodoses i samband med att han definerar ett antal delmål för vart och ett av dem. Vi redogör för dem här.



Figur 10.4.1:2

Exempel på konstruktionsmål och deras förväntade positiva bidrag till olika "ultinära" önskemål. "Negativa bidrag" kan givetvis tänkas men har ej exemplifierats i figuren.

Önskemålet (a) ökad "throughput" befrämjas av

a1 bearbetning automatiskt av en ström av arbeten, och assistans till operatören att utföra exempelvis monteringsarbete för ett arbete överlappat med exekvering av andra arbeten

a2 Flera arbeten skall kunna bearbetas simultant (multiprogrammering) "thus helping to ensure that resources are kept busy".

a3 "Also, recognizing that the productivity of a shop is not solely a function of machine utilization, heavy emphasis is placed on the variety and appropriateness in source languages, on debugging facilites and on input convenience".

Kommentar: målen a1 och a2 har en viss grad av operationalitet även om ytterligare specifikationer krävs för att avgöra om bidraget till önskemålet "ökad throughput" är märkbart. Formuleringen för a1 är försiktig och säger t ex ingenting om graden av assistans eller hur den är tänkt fungera. Formuleringar av detta (vaga) slag i kontraktssammanhang bör undvikas. Målet a2 är också vagt formulerat och kan tolkas på olika sätt. Parallell bearbetning av ett användareprogram och ett mediaomvandlingsprogram tillfredsställer delmålet lika väl som, säg, parallell bearbetning av 10 användareprogram. För att klara denna målsättning på olika stora datorer har olika versioner av OS/360 introducerats. Målet a3 syftar troligen mer på att öka programmerarnas produktivitet, som ju givetvis är i viss utsträckning beroende på datorsystemets throughput - datacentralens produktivitet. Den samlade effekten av maskinvara, programvara och datorcentralens driftsfilosofi påverkar i hög grad "problemthroughputen"[1]. Man kan notera att Mealy i detta sammanhang ej tar upp throughput-främjande önskemål såsom

- kompilatorhastighet och
- objektprogrameffektivitet

vilka dock IBM har försökt att tillgodose bl a genom att introducera flera slag av kompilatorer (för FORTRAN) med resp utan "optimering" av objektkoden. Vad som ovan (a3) avses med "input convenience" är ej riktigt klart men det torde syfta på lättheten till "åtkomst" ("inputmöjlighet") till systemet över fjärrterminaler av olika slag eller tillhandahållande av "timesharing" möjlighet där sådan är önskvärd.

---

1) Med detta begrepp avser vi mängden av nyttigt arbete som produceras på datorsystemet. Det torde ligga i varje datoranvändares intresse att minska andelen "onyttigt" arbete dvs programinkörning, tester eller omkörningar.

(b) kortare responstid anses tillgodoses genom (Mealy (2)):

b1 "OS/360 is designed to lend itself to the whole spectrum of response times by means of control-program options and priority conventions."

Kommentar: Med detta avses att OS/360 är konstruerat för att kunna motsvara responstidskravet från vanlig satsvis bearbetning (responstid av storleksordningen timmar) till reelltidstillämpningar (responstid av storleksordningen decisekunder) - ett inspirerande mål. Detta skall åstadkommas genom en mängd olika styrprogramvarianter och olika prioritetskonventioner. Enligt ovanstående formulering är denna delmålstruktur ej operationell utan måste brytas ner i operationella delmål. - Den invändning som kan anföras mot ovanstående formulering är att en potentiell avsändare lätt kan få det intrycket att OS/360 utan vidare kan sammansättas att motsvara alla tänkbara responstidsönskemål. Med vissa restriktioner är dock en mängd strukturer möjliga och med tilläggsprogrammering torde OS/360 kunna sägas motsvara målet enl. ovan. Därmed har dock ingenting sagts om till vilken kostnad (= resurskrav) detta åstadkommes och vad den totala effektiviteten kan bli som följd.

(c) ökad produktivitet hos programmerarna anses befrämjas genom (Mealy (2)):

c1 en hög grad av mångsidighet hos operativsystemet med hjälp av en relativt stor mängd av programmeringsspråk

c2 makroinstruktionsmöjligheter i assemblyspråket

c3 ett "koncist" (kortfattat!) språk för styrning av arbetsflödet (dvs styrspråk).

Kommentar: uppfyllelse av målet c1 torde i viss mån motverka uppfyllelse av andra mål. En alltför stor mängd olika kompilatorer skapar för leverantören ett underhållsproblem och kanske även en viss icke önskad intern konkurrenssituation. Det dröjde t ex ända till 1968 innan IBM levererade Algol ("svenskbyggd", på Nordiska Laboratorier på Lidingö) som ett av IBM marknadsfört system. Beträffande c2, har makroinstruktionsmöjligheter funnits i över 15 år hos de flesta assemblysystem, dvs är närmast självklart. Målet c3 kan anses uppfyllt såvida att ett styrspråk givetvis existerar. Huruvida detta bör karaktäriseras som koncist är en fråga som är svårbedömd, dvs c3 kan knappast anses vara ett operationellt mål. Till ovanstående delmål för att öka programmerarnas produktivitet bör läggas delmålet a3. Andra programproduktionsfrämjande (önske-) mål som ej explicit redovisats ovan, men som i viss utsträckning implementerats i systemet, är

- möjligheten att skriva delar av ett program i olika källspråk, eller separatkompilera delprogram som sedan länkas ihop av en speciell styrprogramsmodel
- möjligheten att upprätta, organisera och manipulera en "databas" av programelement på ett direktminne vilket dels underlättar den manuella hanteringen av kortbuntar och dels ger ett verktyg för integrerad styrning av stora programmeringsprojekt.
- möjlighet till dialogmässig multipel-access till systemet vid programkonstruktion och utprovning
- servicerutiner för uppläggning, organisation och manipulation av datafiler osv

(d) anpassningsförmåga anses befrämjas genom (Mealy):

d1 systemets modularitet och flexibilitet

Med detta avses att OS/360 består av ett bibliotek av programmoduler. Dessa moduler utgör byggstenarna för varje aktuellt enskilt operativsystem, som normalt endast gör bruk av vissa av dessa. Systemkonstruktionen utnyttjar följande tre grundläggande principer för att åstadkomma önskvärd grad av modularitet:

d1.1 parametrisering av modulerna. Med detta avses att programmodulerna är skrivna så att man genom att variera parametervärden kan få dem anpassade till avsedd konfiguration och omgivning.

d1.2 funktionell redundans. Med det avses att två eller flera moduler existerar för att utföra samma operationer men att dessa är konstruerade med olika "vikter" m a p mål för deras arbetssätt. Vi har tidigare nämnt exemplet att det existerar flera FORTRAN-kompilatorer var och en med olika målsättning (hög komplieringshastighet resp snabb objektprogramexekvering).

d1.3 funktionell "optionalitet". Med det avses att man med utgångspunkt från en "kärna" av funktioner kan utöka "sitt" operativsystem genom att inkorporera olika tilläggsfunktioner som kan förbättra och effektivisera driften. Det är givet att utökningen av funktionerna ej erhålles gratis. Dessa kräver marginella investeringar i olika slag av utrustning såsom primärminne, sekundärminne och kanske även snabbare processenhet(er). Det måste från fall till fall bedömas huruvida dessa maskinella utökningar betalar den, förhoppningsvis, högre kapaciteten.

Bland viktigare "optioner" (tillägg) som OS/360's styrprogram kan utrustas med är (enl Mealy)

- synkronisering av processer vilket tillåter simultanitet mellan parallella processer
- hantering av in/utköer på direktminne, vilket bl a tillåter överlappning av bearbetning med in/utmatning av andra programs in/utdata[1]. Förutom att systemets utnyttjande grad höjs erhålles kortare omloppstid för arbeten än vid bearbetning av indata från en i förväg iordningställd sekvensfil.
- olika metoder för primärminnestilldelning där de "mer avancerade" metoderna själva kräver ett större minnesutrymme för tilldelningsprogrammet och processtid för att utföra erforderlig administration
- minnesskydd, för att i ett multiprogrammerat system förebygga riskerna att oberoende program inbördes förstör varandra
- prioritetsstyrd selektering av arbeten som tillförts systemet.

De ovan anförda mekansimerna - funktionerna har delvis berörts i kap 2.2 samt mer i detalj diskuterats i kapitel 3.

Vi tar nu upp målet

(e) utbyggbarhet, som starkt hänger samman med systemets modularitet och flexibilitet. Vi citerar Mealy (2)

e1 "The system is not only open-ended for the class of functions discussed (in this paper), but is based on a conceptual framework that is designed to lend itself to additional functions whenever warranted by cumulative experience". "Moreover, from the viewpoint of system management, a System/360 installation may look upon its own application programs as an integral part of the operating system".

Kommentar: detta är ett högt ställt, svårkonkretiserbart och icke direkt operationellt mål. Målet är självklart och nödvändigt ty utan det vore konstruktions- och underhållsarbetet på ett så omfattande system tämligen hopplöst.

System/360 omfattar ett brett spektrum av datorer ur prestanda- (och möjlig konfigurations-) synpunkt till vilka effektiva anpassning av operativsystemet krävs.

När ett system (operativ- eller informations-) konstrueras föreligger oftast inga praktiska driftserfarenheter. Att efter en tids drift nya önskemål och synpunkter framträder, vilka kräver ändringar eller utökningar av operativsystemet, är oundvikligt.

Vad man kunde kritisera är att ingenstans skymtar önskemålet att en ("la-

---

1) Dvs "optionen" att övergå från satsvis till kövis bearbetning.

gom" kvalificerad) användare själv, utan större besvär, skall kunna laborera med operativsystemet för att genom ändrade planeringsalgoritmer mm kontinuerligt kunna förbättra systemets driftsegenskaper för aktuell arbetsprofil.

Ett annat önskemål som icke direkt framträder ovan är givetvis att operativsystemets funktioner skall kunna utökas/ändras utan att existerade användarprogram därför måste omarbetas. Systemets användning sett från "normalprogrammerarens" synpunkt bör heller ej radikalt förändras vid övergång från en "version" till en annan.

En viktig punkt som tidigare ej explicit berörts i samband med OS/360 och som är av stor vikt för alla operativsystemsanvändare är önskemålet avseende driftssäkerhet. Önskemålet är naturligtvis speciellt uttalat för datorsystem som utan driftsavbrott skall betjäna och övervaka eller styra annat fysiskt eller administrativt system dygnet runt, t ex vid rymdprojekt. Driftsavbrott eller störningar kan förorsakas av dels maskinvarufel dels programfel och dels personfel. Något fullständigt skydd mot dylika fel kan knappast påräknas inom överblickbar framtid. De operativsystemfunktioner som kunde bidra till detta önskemål befinner sig ännu i ett primitivt utvecklingsskede. Särskilt vid komplexa system av reelltidstyp faller ännu lotten att bygga in driftssäkerhet i systemet främst på användaren. För reelltidssystem av typen "platsbokning för flygbolag", bankapplikationer o dyl torde detta arbete överstiga 25 % av totala arbetet att implementera systemet.

Längre uppehåller vi oss ej vid OS/360's önskemål och konstruktionsmål. Som det torde ha framgått är få av dessa mål operationella (kvantifierbara). Förhållandet är likartat även för andra operativsystem.

OS/360's målstruktur enligt ovan synes tilltalande för många potentiella användare. Man får det intrycket att allt är möjligt och att systemet tack vare sin modularitet kan generera ett effektivt och resurssnålt "delsystem" för varje speciell driftsprofil. Huruvida IBM lyckats med detta är en svår fråga att generellt besvara. Systemet har onekligen fått motta viss kritik på grund av sina stora resurskrav och i vissa fall för sin ineffektivitet. Att kritiken mot OS/360 har "märkts" i större utsträckning än kritiken mot andra tillverkares operativsystem torde till största delen kunna förklaras av IBM's dominerande marknadsandel. Man bör dock komma ihåg att denna kritik ofta uteslutande har beaktat önskemålet om effektivitet i "rå"throughput-mening. Att det finns en mängd andra önskemål som ej tillgodoses av throughput enbart har vi redan konstaterat. Dessutom har IBM under de sista åren gradvis förbättrat OS/360's prestanda så att det, åtminstone för administrativa rutiner, förefaller ha en god verkningsgrad. Aktuella prestandajämförelser med s k "beachmark-tests" av administrativ karaktär (körning av fingerade eller verkliga program hämtade ur en användares jobb-

mängd) har vi sat att OS/360 står sig väl i konkurrensen. Slutsatsen måste givetvis begränsas till körning av de program som ingått i testet. Vi vill här också påpeka svårigheten av genom "benchmark-tests", omfattande ett fåtal rutiner, avspegla en realistisk driftsmiljö. Ett operativsystems möjligheter att planera driften av en mängd rutiner kan på ett rättvisande sätt endast bedömas genom mycket omfattande försök vilka normalt är praktiskt otänkbara vid t ex utvärdering av maskin/programvara. De teoretiska framstegen på detta område är för närvarande blygsamma. Simulering är ett populärt verktyg men relativt resurskrävande m a p datortid och tid för beskrivning av systemmodellen.

För sina mindre maskiner i 360-serien (mod 360/30 och 360/40/ har IBM lanserat de mindre sofistikerade men mindre resurskrävande DOS- och TOS-systemen (Disc resp Tape Operating System). Dessutom har vid en del användarinstallationer egna operativsystem konstruerats (i vissa fall med stöd av lokala IBM-krafter), vilka även funnit användning hos andra IBM-kunder.

10.4.2 ICL GEORGE

ICL tillhandahåller för sin maskinserie 1900 operativsystemen GEORGE 1, 2 och 3 där GEORGE 1 har den lägsta ambitionsgraden och GEORGE 3 den högsta. Någon detaljerad önskemåls- och måldiskussion har vi ej hittat i tillgänglig litteratur. Att döma av resultatet är många av de önskemål som diskuterats i damband med OS/360 tillämpliga även här.

Beträffande GEORGE 1 och 2 konstateras dock (se ref ICL (3)) att

"... an operating system has two important aims:

(1) the oprimum use of machine time

(2) an efficient and accurate interpretation of the user requirements"

Dessa är att betrakta som icke operationella önskemål. Önskemålet (1) leder till definition av delmål som berör datorsystemets effektiva utnyttjande bl a genom multiprogrammering och lämpliga planeringshjälpmedel. Önskemålet (2) är mera diffust formulerat men torde bl a beröra frågan om lämpliga och användarvändliga styrspråk och programmeringsspråk.

GEORGE 3 - systemet är avsett för de större systemen i 1900-serien, och är mer resurskrävande. Dess målsättning är hög. Det fastslås (Oesteicher et al 1967) att

"... the system was designed to meet the needs of users of all the 1900

series which have a miniumum of 32 K (24 binära siffrors ord) of core store and one half million words of drum or disc (memory)".

För att få närmare grepp om vad som avses med ovanstående är man tvungen att studera en GEORGE-3 (förkortat G-3) - manual (ICL (4)) och de funktioner som ingår i systemet.

- G-3 avlastar operatören från uppgiftern att instruera datorns exekutiv-system[1] att initiera och styra arbeten genom systemet
- G-3 tillåter lagring av arbeten i systemet före deras exekvering
- G-3 planerar datorns arbete
- G-3 tillåter temporär eller permanent lagring av information (program och data) i systemet, tillgänglig för ankommande jobb
- G-3 ombesörjer automatiskt buffring av transporterade data till/från långsamma perifera enheter.

Ovanstående egenskaper syftar till att öka systemets prestanda (throughput). För att öka systemets tillgänglighet för användare som ej är betjänta av en (job-shop) satsvis bearbetning enligt ovan och som kräver en frekvent människa-maskinkontakt ingår i G-3 funktioner som gör multipel-accessdrift möjlig simultant med satsvis bearbetning i "bakgrunden".

Det är uppenbart att önskemålet att tillmötesgå alla användares "behov" är svårt att uppfylla för GEORGE-3. Speciellt när det gäller reelltidssystem med anslutning av terminaler av olika utformning över telenätet torde idag knappast något operativsystem existera där användaren är helt avlastad från viss egen programmering av olika centrala styrprogramfunktioner eller infogning av ett "extra styrprogram" i systemet som då bildar en länk mellan reelltidsprogrammet och styrsystemet.

I en senare publikation (10) diskuterar Burkinshaw två, ofta motstridiga, önskemål för operativsystemen GEORGE 1 och GEORGE 2. Dessa är i sak analoga med de ovan redovisade önskemålen (1) och (2) för GEORGE 3. I ref (10) framhålls bl a att en ökning av effektiviteten (en ökning av datorsystemets utnyttjandegrad) normaltkräver att man på ett eller annat sätt inskränker en programmerares frihet genom olika "standards" vid användning av operativsystemet. Naturligtvis kan "standards" många gånger vara nyttiga för att höja den allmänna produktiviteten. Man måste dock komma

---

1) En gemensam "kärna" för samtliga GEORGE-system som kan betraktas som en utvidgning av maskinvaran och som bl a styr multiprogrammering.

ihåg att etablerade standards måste ta hänsyn till de ofta högst varierande krav på sytemets resurser som olika jobb ställer.

10.4.3 UNIVAC 1108/1106 - EXECUTIVE SYSTEM (EXEC-8)

Följande mål redovisas för EXEC8 (Univac (1))

(1) att optimera användningen av maskinenheterna och på samma gång optimera interaktionen för alla användare med hjälp av multiprogrammerings- och multibearbetningsteknik.

Kommentar: Här är ett annat exempel på hur ordet "optimering" missbrukas i många kretsar. Optimering kräver givetvis att bl a en kvantitativ målfunktion redovisas, vilket torde vara svårt i detta sammanhang. Målet (1) är ett önskemål som syftar till att åstadkomma en hög effektivitet hos systemet, samtidigt som man vill tillåta olika former för drift och åtkomst till systemet.

(2) att göra samtliga faciliteter inom datorsystemet 1108/1106 tillgängliga för icke-lokala användare.

Kommentar: Detta mål är operationellt (om "facilitet" och "tillgänglighet" definieras i förväg går det att konstatera om det uppfyllts eller ej).

(3) att tillhandahålla ett styrspråk vars struktur tillåter enkla program att på ett enkelt sätt uttrycka sina krav på systemet.

Kommentar: Målet syftar på önskemålet att underlätta programpreparationen och öka programmerarproduktiviteten.

(4) att tillhandahålla flexibilitet för komplexa program att beskriva en komplex omgivning.

Kommentar: Detta syfta på vissa faciliteter i styrspråket. Målet (4) tillsammans med (3) syftar således på ett uttryckskraftigt och flexibelt språk för styrning av flöden av arbeten genom systemet.

(5) att tillhandahålla ett brett och enkelt användbart spektrum av hjälpmedel för konstruktion, katalogisering, manipulation, och utprovning av program.

Kommentar: Uppfyllelse härav bidrar till uppfyllelse av önskemålet att öka programmeringsproduktiviteten.

(6) att möjliggöra bearbetning av program såväl satsvis, kövis, dialogmässigt (demand mode i Univac-terminologi) som i reell tid.

Kommentar: Målet syftar på önskemålet att kunna använda operativsystemet för ett brett spektrum av driftsformer och applikationer.

(7) att tillhandahålla enkla och flexibla hjälpmedel för generering och vård av systemet vid varje individuell installation.

Kommentar: Ett mål som (om det uppnås) ger användarna möjligheter att på egen hand anpassa operativsystemet till aktuell miljö. Få operativsystem har detta mål så högt placerat i målhierarkien. Ett lättskött operativsystem torde också medföra att en användares beroende av operativsystemleverantören minskar (och omvänt!)[1] Detta beroendeförhållande är idag tyvärr ett icke försumbart problem.

(8) Systemet skall vara osårbart av användarprogramfel och i mesta möjliga utsträckning även av maskinfel.

Kommentar: Detta är tyvärr oftast svårt att till fullo tillmötesgå.

(9) "to provide simplest possible operational characteristics consistent with full utilization of the capabilities of the system".

Kommentar: Alltså önskemålet om enklast möjliga operation av systemet tillsammans med bästa möjliga utnyttjande av systemets möjligheter. Målet syftar till hög produktivitet på datacentralnivå.

Erfarenheter från EXEC-8's användning är enligt uppgift ännu relativt begränsade. Vid många installationer har användning av ECEX-8 fått stå tillbaka för det mindre ambitiösa och mindre resurskrävande men för typiska beräkningsarbeten i satsvis miljö mer effektiva ECEC II-systemet. I sammanhanget kan det vara intressant att notera att EXEC I för Univac 1107, ett för sin tid (1962) mycket ambitiöst system för multiprogrammering, vid många installationer snart ersattes av det till sin struktur enklare EXEC II. Svårigheter för operatörer att med det komplicerade EXEC I få ett effektivt utnyttjande av medelstora 1107-system var en bidragande orsak härtill.

Man torde kunna påstå att EXEC-8 systemet, introducerat omkring 1966, fortfarande befinner sig under viss utveckling. Prestandamätningar utförda 1968 på jobb av "teknisk-vetenskaplig" karaktär har indikerat att systemet

---

1) Det "omvända förhållandet" är givetvis av intresse för en tillverkare som vill minska sina kostnader vad avser systemunderhåll och systemassistans.

vid den tidpunkten belastades av en relativt stor administrativ "overhead" vilket framför allt märktes vid multiprogramkörning. Systemets effektivitet m a p administrativa rutiner är svår att generellt bedöma då systemets konfiguration och den aktuella jobbprofilen är avgörande. En icke obetydlig roll i detta sammanhang spelar även t ex COBOL-kompilatorn - dvs effektiviteten hos den genererade objektkoden.

10.4.4 MULTICS-systemet

MULTICS (Multiplexed Information and Computing Service)[1] är ett omfattande programmeringssystem som utvecklas som ett forskningsobjekt vid (MIT) Massachusetts Institue of Technology, USA. Corbato´ och Vyssotsky framhåller att

> "One of the overall design goals is to create a computing system which is capable of meeting almost all of the present and near-future requirements of a large computer ultility"
>
> (Corbato´ et al. i Rosen (ed.), "Programming systems and languages" p 714).

För att konkretisera ovanstående anges några mer eller mindre operationella mål

(1) Systemet måste kunna vara i drift 24 tim om dygnet och uppvisa en hög tillförlitlighet

(2) Systemet skall kunna tillfredställa en mängd varierande servicekrav, från multipel människa-maskininteraktion till satsvis bearbetning av inlämnade arbeten, från framställning/programmering av systemet med hjälp av problemorienterade språk, från tillförsel av arbeten via (central) perifer utrustning till tillförsel via fjärrterminaler av olika slag.

Ett intressant önskemål är att systemet skall kunna anpassas till (f n okända) framtida krav: dess struktur måste vara generell och kapabel att kunna utvecklas med tiden,

> "It is not expected that the initial system, although useful, will reach the objective; rather the system will evolve with time in a general framework which permits continual growth to meet unknown future reqirements" (Corbato´ et al i Rosen (editor), "Programming systems and Languages", p 718)

---

1) Författarna vill gärna gardera sig avs MULTICS systemets aktuella status och möjligheter till fortsatt existens, då vi saknar information om detta.

Betraktar man önskemålen och målen ovan uppstår bl a frågan varför man ej i stället för att bygga ett eget system anslöt sig till något existerande operativsystem. Målen för dessa system förefaller vid en första betraktelse lika ambitiösa som MULTICS' mål. Vad är det då som gör att de ej ansetts tillräckliga för MULTICS?

Konventionella stora system av ovan nämnd typ har kanske konstruerats med viss tanke på önskemålet om interaktiv multipel-access men det slutliga resultatet har hittills ej motsvarat förväntningarna. Anledningarna till detta är många. En av dem torde vara kravet att systemet skall var ur försäljningssynpunkt anpassningsbart för ett brett spektrum av stora som små datorer med varierande konfigurationer.

Uppfyllelse av dylika krav har givetvis ej varit lika accentuerat vad beträffar MULTICS. Tyngdpunkten för MULTICS ligger på interaktiv multipel-access till systemet för personer i en (akademisk) forskningsmiljö. För dessa är datorn ej ett självändamål utan ett lättillgängligt tankestimulerande verktyg i det dagliga (förhoppningsvis) intellektuella arbetet. Erfarenheter av dylika system indikerar en explosiv tillväxt av nya, förut icke påtänkta, användningsområden för datorer. Ett av huvudönskemålen för MULTICS är således att tillhandahålla lämpliga verktyg för vad som kan kallas "intellktförstärkning" (machine-aided cognition).

Traditionella system torde knappast ha placerat någon större vikt vid uppfyllelse av ett dylikt önskemål. Det är dock uppenbart att inom en icke alltför avlägsen framtid även i icke-akademisk miljö datorer kommer att användas på ett betydligt mer progressivt sätt än idag. Detta kommer givetvis att förändra önskemålen beträffande de av leverantörerna eller "programvaruhusen" utbjudna operativsystem. Redan idag har många användar uttryckt önskemål beträffande operativsystem för reelltidssystem - önskemål som leverantörerna, med få undantag, har haft svårt att tillmötesgå.

De frågor på operationell nivå som har tillmätts största betydelse i MULTICS projektet berör

- systemets responstider
- bekväm manipulering av program- och datafiler
- lätthet att styra processer <u>under exekvering</u>
- skydd av privata filer och processer.

Några operationella konstruktionsmål för MULTICS som bidrar till ovanstående har beskrivits av <u>Daley</u> och <u>Dennis</u> ((7) 1968)

(1) att tillhandahålla för användaren ett stort och maskinoberoende virtuellt minne (adressrum) så att ansvaret och besväret med administration av det fysiska minnessystemet helt överlåtes åt datorsystemet. Med detta avses att användaren kan röra sig med ett (virtuellt) adresserbart "logiskt" minne som är större än det fysiska primärminnet utan att behöva bekymra sig för "swapping", buffring och datatransporter. På så sätt blir användarprogrammet på problemnivå praktiskt taget oberoende av egenskaperna och strukturen hos de olika minnesenheterna i systemet.

(2) att tillåta ett generellt programmeringsförfarande som tidigare ej varit praktiskt vanligt. Det omfattar t ex möjligheten för en procedur att anropa en annan (främmande) procedur endast genom att ange dess namn utan att veta dess resursbehov eller behov av andra procedurer.

(3) att tillhandahålla för användarna ett ("dyrkresistent") skyddssystem så att t ex en datamängd eller en procedur kan "delas" endast mellan sådana användare som erhållit formell "auktorisation".

Problem och mekanismer av denna principiella natur har vi behandlat i **tidigare** kapitel.

10.4.5 Några andra system

Avslutningsvis vill vi återge ytterligare några målanknutna synpunkter avseende operativsystems utformning.

Konstruktörernas av den danska datorn RC 4000 (tillverkas av Dansk Regnecentral) uppfattning om viktiga önskemål betr operativsystem redovisas av Per Brinch Hansen (1969) :

> "Det torde vara uppenbart att systemerarens / programmerarens första krav på ett operativsystem måste vara att det tillåter honom (henne) att ändra dess driftssätt. Men det är just vad dagens operativsystem ej tillåter".
>
> "När behovet uppstår finner systemeraren att det är hopplöst att modifiera systemets arbetssätt då dess grundläggande struktur är avsedd för en viss bestämd driftsform. Alternativet - att ersätta det med ett annat operativsystem - är för de flesta datorer ett svårt problem om inte en omöjlig sak att genomföra då kompilatorer m m normalt är bundna av det gamla systemets konventioner. Denna olyckliga situation visar att huvudproblemet vid konstruktion av operativ-

system ej är att definiera funktioner som tillfredsställer speciella driftsönskemål utan att tillverka en "Kärna" som sedan på ett enkelt sätt kan utvecklas av användaren och anpassas till den dynamiskt föränderliga omgivningen"[1].

Johnstone (1967) diskuterar utformningen av ett operativsystem för ett datorsystem vars uppgift är att bistå vid styrning av NASA's Gemini och Apollo-projekt. Detta är ett reelltidssystem där följande önskemål om karakteristika hos operativsystemet nämns:

(1) Modularitet - operativsystemets konstruktion skall tillåta tillägg av nya funktioner som orsakats av utrustningsändringar och/eller nya applikationer utan att detta skall påverka existerande funktioner och program.

(2) Enkelhet - endast ett minimum av träning i operativsystemets funktioner bör krävas innan programmerarna skall kunna tillverka program som skall arbeta i en komplicerad reelltidsmiljö.

(3) Mångsidighet (smidighet) - systemet skall kunna användas från den enklaste simulerade reelltidsomgivningen vid programutprovning till de mest krävande real-time "missions".

(4) Generalitet (allmängiltighet) - operativsystemet skall ej vara orienterat mot en viss applikation utan kunna användas för varierande driftsprofiler.

(5) Tillförlitlighet - systemet skall arbeta utan driftsavbrott - ett absolut krav vid bemannade rymdfärder.

Det är, som vi vid det här laget klart noterat, knappast avsevärda skillnader mellan de olika betraktade systemens önskemål och mål på den nivå vi här har diskuterat dem. Då dessa mål för det mesta är icke operationella och okvantifierbara kan man genom att studera dem många gånger knappast ens avgöra om ett satsvis arbetande system eller ett multipel-accessystem avses.

Att bedöma huruvida ett visst operativsystem är lämpligare, dvs bättre tillgodoser en användares önskemål, än ett annat är en mycket svår uppgift och förutsätter bl a att användaren kan specificera sina önskemål. Icke

---

1) Mycket talar för att detta problem uppstår vid konstruktion av vilket informationssystem som helst om man vill minska risken att systemet på grund av oförutsedda krav och inflexibilitet snabbt föråldras.

desto mindre är detta ett ofta förekommande problem vid datorsystemval. Den tekniska maskinvarustandarden hos dagens datorer är god och ungefär lika för samtliga leverantörer. Det är inom en mängd svårfixerbara områden, bl a rörande operativsystem, som skillnader inträffar. Programvaruproblemen är framför allt aktuella för reelltidssystemen, där vissa program körs interaktivt on-line och betjänar ett antal fjärrskrivmaskiner, bildskärmar, kassaapparater o dyl medan andra program körs som en lägre prioriterad "bakgrundsbelastning". Användaren står här vid systemval vanligen i den situationen att välja bland en flerfald tillverkares operativsystem, mer eller mindre vagt eller begripligt dokumenterade. Problemet innebär ofta att under en tid av några veckor bedöma huruvida de potentiella operativsystemen innehåller erforderliga och önskvärda funktioner för såväl "bakgrundsprogrammens" som reelltidsprogrammens övervakning och styrning.

Uppgiften försvåras av att vederbörande användare normalt ej vet i detalj, vilka program som skall utvecklas för det nya systemet, vad programmen skall göra och hur programmen skall samverka. Att tala om erforderliga op systemfunktioner och önskvärd kapacitet hos det tilltänkta systemet är, med ett dylikt informationsunderlag, mest en fråga om mer eller mindre kvalificerade gissningar.

När det gäller operativsystemets struktur och dess lämplighet för tilltänkta applikationsprogram torde följande funktioner och element hos systemet vara intressanta att studera för den potentiella användaren

(1) Hur administreras och fördelas vid multi- eller singelkörning resurser (processortid, primärminne, sekundärminne, system- och serviceprogram m m) mellan användarprogrammen?

(2) Vilka metoder tillämpas för och hur sker tillförsel och utmatning av jobb till/från systemet?

(3) Vilka möjligheter har en viss installation att på tillgänglig kompetensnivå göra tillägg eller ändringar till standardprinciper enl (1) och (2) ovan?

(4) Vilka principer för programkonstruktion och faciliteter för programadministration finnes? Vilka olika språk finnes tillgängliga? Kan ett program kombineras av element skrivna i olika språk? Hur? Vilka restriktioner?

(5) Hur administreras datatransporter till/från perifera enheter och terminaler? Finns standardprogramvara för samtliga tilltänkta perifera enheter och fjärrterminaler?

(6) Vilka servicefunktioner tillhandahålles för organisation, manipulering, utsökning och vård av filer?

(7) Vilka servicefunktioner tillhandahålles för att underlätta konstruktion av katastrofrutiner mot driftsstörningar?

(8) Vilka servicefunktioner för programtest och felsökning tillhandahålles?

(9) Kan styrspråket anpassas för <u>enkel</u> hantering för <u>flertalet</u> användare i avsedd miljö, med utvidgbarhet i specialsituationer?

(10) Kan systemet anpassas för <u>dedikerad</u> användning i skilda driftsmiljöer?

(11) Är operativsystemet med rimlig arbetsinsats utvidgbart? Vilka maskinella resurskrav kan då förväntas?

Var och en av dessa huvudpunkter omfattar givetvis en mängd konkreta detaljer.

Som ovan framgår talas av flera leverantörer om tillgodoseende av önskemål om flexibilitet för operativsystemen genom tillhandahållande av system i modulform. Härmed möjliggörs s k "figursyning" av system för skilda användningsmiljöer, så att en användare som ej har intresse av t ex time-sharing facilitet inte skall behöva se sin dator belagd till någon del av just dylika styrprogramvaror. Det finns emellertid även operativsystem som endast tillhandahålls i helt inflexibel form, där användaren inte har någon nämnvärd möjlighet påverka systemets driftsprinciper. Vi kan i det fallet se operativsystemet som en "black box", vars innehåll för oss är mer eller mindre okänt, men som ger sig tillkänna genom de arbetsresultat som produceras ur en given inmatning. Vi kan se dessa båda former, modulformen kontra den svarta boxen, som motpoler, och givetvis finns mellanformer tillgängliga. Både för- och nackdelar finns för de båda formerna. Här skall endast helt kort betraktas <u>en</u> faktor som bör påverka en användares ställningstagande vid val av system. För att ha reell möjlighet till någon glädje av ett modulärt system måste kvalificerad personal på systemprogrammerarnivå finnas tillgänglig i erforderlig utsträckning. I annat fall kan modulariteten blott bli en belastning som kan förorsaka ekonomiska avbräck för användaren. En leverantör av ett modulärt operativsystem talar <u>före</u> kontraktsskrivande gärna och mycket om den kvalificerade systemassistans som kontinuerligt kan tillhandahållas. I de fall kvalifikationsnivån eller kvantiteten hos denna system assistans inte tillräckligt explicit medtas som klausul i kontraktet, ett tyvärr hittills vanligt förhållande, kan det visa sig att intresset från leverantören att löpande underhålla operativsystemet (frekventa ändringar, ny-utgå-

vor etc) hastigt avtar efter leverans av systemet. Användaren har då endast att lita till sin egen personal för dessa i många fall komplicerade uppgifter. Detta har i ett flertal installationer visat sig ödesdigert. Användarens insikter i ett operativsystem är nära nog som regel otillräcklig. Följden blir att att han måste nöja sig med att ligga kvar med ett operativsystem, som ej i alla stycken är "up-to-date", och som kan medföra "förluster" som följd av mindre effektivt utnyttjande av datorsystemet och inkompabilitet mot övriga "versioner".

Uppenbarligen är ett viktigt avgörande för användaren att se till att systempersonal i tillräcklig utsträckning finns tillgänglig för den händelse han väljer ett modulärt operativsystem. För närvarande är bristen på kvalificerad systempersonal tydlig, och det vill därför synas som om fördelar för val av system i black-box-utförande icke kan försummas. Det åligger dock självfallet användaren att bedöma hur belastande resp värdefulla de olika alternativen ställer sig. Detta åstadkommes genom en detaljerad studie av i vilken utsträckning operativsystemets konstruktion, uppträdande och underhåll bidrar till uppfyllelsen av de mål för verksamheten som uppställts på förhand.

Ibland är användarens valfrihet rörande operativsystemsfunktioner ej särskilt stor. Några leverantörer tillhandahåller som nämnts operativsystem i form av "take it or leave it". Operativsystemets uppträdande får därvid ses som blott en del av hela datorsystemets, med de för- resp nackdelar som därmed kan följa.

10.4.6 Önskemål beträffande operativsystemets prestanda och effektivitet

De önskemål och andra mål som vi hittills har diskuterat har huvudsakligen berört önskvärda funktioner hos operativsystemet som skall medge ett önskvärt arbetsförfarande med systemet. Frågor om prestanda i form av "throughput" och "responstider" har dock nämnts. Dessa är givetvis delvis maskinvaruavhängiga faktorer.

Uppenbarligen spelar operativsystemet en stor roll i prestandasammanhang. Är det då omöjligt att specificera några kvantifierbara mål avseende prestanda och effektivitet för operativsystem? Vad som framför allt är av intresse är operativsystemets tidsresurskrav för utförande av diverse styrfunktioner. Dessa resurskrav kan ibland anta så dominerande former att applikationsprogrammens exekvering allvarligt degraderas (fördröjs)[1]. Det är dock knappast

1) Det har t ex inträffat vid testkörning att körning av två "relativt balanserade" program (m a p I/U och internbearbetning) i serie tog sammanlagt kortare tid än parallell exekvering av dessa program

troligt att leverantörerna inom en nära framtid kommer att ange och garantera några generella prestandamått för sina operativsystem.

För att en användare (köpare) överhuvudtaget skall kunna föra operativsystemeffektivitet på tal med en leverantör krävs att den avsedda driftsmiljön detaljerat och entydigt specificeras. På senare tid har sådana specifikationer för ett antal installationer legat till grund för s k "leveransprov avs operativsystem".

Önskemål beträffande leveransprov är bl a

(1) de skall vara väldefinierade och entydiga och så väl som möjligt spegla önskemålen avs det verkliga systemet[1)]

(2) de skall vara lätta att programmera och utföra (omedelbart efter installation)

(3) det skall föreligga ett minimum av möjligheter för berörda "parter" att genom olika "knep" kringgå specifikationerna

(4) de skall avspegla operativsystemets lämplighet för avsedda applikationer

(5) de såväl kvalitativa som kvantitativa resultaten skall vara lätta att mäta och registrera.

Det är av största betydelse att leveransprovets utformning så väl som sig göra låter avspeglar datorkontraktets innehåll. Den tid är numera troligen förbi då ett datorkontrakt företedde klara likheter med t ex ett diskmaskinkontrakt, dvs endast bestod av en uppräkning av maskinenhetsbeteckningar samt en klumpsumma i kronor längst ner (exempel på dylika kontrakt existerar faktiskt i svensk datorhistoria, där de kontrakterade beloppen har varit av storleksordningen 10-20 miljoner kronor). Numera göres i många fall försök att uppta prestandauppgifter rörande både maskinvara och programvara i kontrakten. För programvara innebär detta sådant som minnesbehov, behov av processor- och annan kapacitet vid olika givna belastningar, administrationstidsangivelser rörande operativsystemet, kompileringshastigheter för samtliga kompilatorer, effektivitetsmått för kompilatorgenererad objektkod osv.

---

1) Härmed avses att proven är att betrakta som en konstgjord belastning på systemet. Om operativsystemet klarar denna belastning bör även det verkliga applikationssystemet förhoppningsvis klaras. Jämför ingenjörstekniska beräkningar av konstruktioner (system) med standardbelastningar för vind, trafik, säkerhetsfaktorer m m enligt statliga föreskrifter.

Vi tar dock här inte ytterligare upp dessa problem, då en detaljredovisning med anvisningar om leveransprovs lämpliga konstruktion och utförande faller något utanför denna boks syfte.[1]

10.5 Slutsatser

Så länge som operationaliteten hos konstruktionsmålen för normalt tillgängliga operativsystem är så låg som ovan erfarits, borde leverantörerna kunna publicera och närmare diskutera sin målstruktur. Att så litet publicerats kan bero på

(1) att de inte arbetat efter någon detaljerad målstruktur eller

(2) att de haft en, men inte anser denna vara av allmänt intresse. Vi kan hoppas att detta kapitels diskussion kan påverka detta förhållande.

Önskvärt vore, sammanfattningsvis, att samtliga mål (på olika nivåer) för konstruktion och användning av operativsystem angavs detaljerat. Intressant är då dels den "vertikala" strukturern (önskemål-, målspecifikationer) för systemets egenskaper, och dels i vilken utsträckning uppfyllelse av dessa mål leder till konflikt resp samspel dem emellan, den "horisontella" strukturen. Detta kan åskådliggöras i form av en målgraf, för att åstadkomma överskådlighet.

Av stort intresse är, beträffande den horisontella målstrukturen, hur variation av systemparametrar kan påverka/förbättra eventuell konfliktssituation mellan måluppfyllelse. För ett modulärt tillhandahållet operativsystem kunde den "ideala" leverantören ge anvisning om vilka parametervärden som bör användas för olika givna, tänkbara standardmässiga användningsmiljöer, ledande till minimal måluppfyllelsekonflikt å aktuell användarmiljö.

Målkonfliktproblematiken är tveklöst av intresse för potentiella användare. I enskilda mindre komplicerade situationer har lösningar presenterats som mer eller mindre allmänt har accepterats. Hit hör konflikten för en kompilator mellan åstadkommande av snabb kompilering och hög objektprogrameffektivitet. Vi har ovan noterat att detta numera ofta löses genom tillhanda-

---

1) Från Inst för Informationsbehandling, KTH, Stockholm, planeras separat publicering av arbetsgång vid val av datorutrustning, inkluderande bl a leveransprovsutförande.

hållande av skilda kompilatorer (för samma språk). Detta kan möjligen innebära ett steg i riktning mot tillhandahållande av större dedikerade system.

Många mera komplicerade konfliktsituationer existerar. Det är t ex tydligt att i många situationer allmänt sett allt mer omfattande bearbetning önskas samtidigt som arbetet önskar utfört på allt kortare tid. (Kraven ökar här snabbare än datorsystemen strukturmässigt rationaliseras och uppsnabbas internt). Två dylika motstridiga mål kan inte uppnås samtidigt, utan användaren kan anmodas att, och på ett vettigt sätt ges möjlighet att, specificera viss målprioritering. Detta kan sannolikt lösas genom parameterstyrning av måluppfyllelsen i generells system, eller genom tillhandahållande av omkopplingsbarhet mellan mer eller mindre utpräglat dedikerade system.
- Man kan ännu inte samtidigt både spara karamellen och äta upp den.

LITTERATUR

1. Langefors, B., "System för företagsstyrning", Studentlitteratur, Lund 1968.

2. Mealy, G.H., "The functional structure of OS/360", Introductory Suyrvey, IBM Syst. Journal, Vol 5, No 1, 1966.

3. ICL, Operating Systems George 1, 2 and 3. Technical Publ, No 4114 (July 68) and 4026 (april 67).

4. Ostreicher, M.D., Bailey, M.J., and Strauss, J.I., "George-3-A General Purpose Time Sharing and Operating System". CACM, Vol 10 No 11, nov 1967.

5. Univac 1108 Multiprocessor System, "EXECUTIVE- Programmers Reference Manual" UP-4144, 1966.

6. Corbato´ F.J. och Vyssotsky V.A., "Introduction and overview of the MULTICS system" in Rosen (editor), "Programming Systems and Languages", Mc Graw Hill, 1967.

7. Daley R.C. och Dennis J.B., "Virtual memory, processes and sharing in MULTICS". CACM, Vol 11, No 5. May 1968.

8. Hansen P.B. "A Dynamic Operating System Concept", Paper presented at the IFIP WG 3.2 Workshop on Systems Design in Friboug, jan 1969.

9. Johnstone J.L., "A Real-Time Executive System for Manned Spaceflight", FJCC Proceedings, 1967, p 215.

10. Burkinshaw P., "Operating Systems: Introduction to GEORGE 1 and GEORGE 2", i Cuttle och Robinson (editors), "Executive programs and operating systems", MacDonald, London, 1970.

INDEX

activity 31
administrativ typ av jobb 127
administration av
- primärminne vid blockutbyte 212
- brytsignaler 13, 163
- buffertar 37, 262
- resurser 40, 109, 182
adressering av primärminne 201
adressrum 169
- avbildning mellan - 170, 198
- fysiskt 169, 198
- linjärt 170
- logiskt 169
- symboliskt 170
- virtuellt 169
adresstransformation 198, 202
aktivt tillstånd 178
allokeringstillstånd 49
ankomstfrekvens för transaktioner 133
ankomsttider (avs jobb) 138
användarkategori 123
användarprogramnivå 12, 103
applikationsprogram 15
arbetsunderlagsstrategi 217
artificial contiguity 201
associativ relation 230
attribut 230
avbrytbar resurs 19
avslutning av jobb, deljobb 82, 95

basadress 202
bearbetningsklart tillstånd 28, 178
behovsstyrd inläsning 200
beordningstillstånd 12, 163
blockhantering 226
blockstorlek (vid tidsdelning) 211
blocktabell 203
blockutbyte 212

brytpunkt med återstart 291
brytsignal 184
brytsignalsadministration 13, 163
buffert 37, 262
buffertadministration 37, 226, 262

centraliserad drift 75
checkpoint-restart 291
cirkulär kö (vid tidsplanering) 191
closed-shop 75

databas 224, 271
datadefinition: se deklaration av datamängd
datafil: se datamängd, fil
datamäng 144, 149
dataorganisation 238
dataprocess: se process
datatransmission 270
datatransportbunden process 193
datatransportövervakning 222, 258
dataresurs 19
datastruktur 228, 231, 272
deadlock: se låsning
deadly-embrace: se låsning
decentraliserad drift 75
dedikerat system 68
deklaration av datamängd 149
delad tillgång 23
deljobb 57, 80, 109, 144
-avslutning av 82
- initiering av 82, 92
delmål 301
delprocess 33
diagnostik 288
dialog: se även konversation, interaktion
dispatcher: se processortidstilldelning, tidsplanering
driftsfilosofi, driftsform 57, 75, 77

egenskaper
-avs jobb 114
- avs processer 167
- vid utsökning 255
elementarmeddelande 231
elementarpost 239
etablerad process 174, 178

ett-jobb-i-taget bearbetning 59
exekverande tillstånd 28, 178
exklusivt användbar resurs 18, 33
exklusiv tillgång 23

facilitet
- utrymmesberoende 81
- tidsberoende 81
felsökning 288
feltillstånd 139, 185
fetch-strategy 200
fjärrterminaler 56
FIFO (first-in-first-out) 191
fil 222
- beskrivning (se även deklaration av datamängd) 271
- datafil: se datamängd
- generation 225, 265
- indexfil 252, 254
- inverterat 255
- katalogisering 225, 263
- låsning: se skydd
- mosaikfil 111
- multilist 255
- nyckelfil 251
- organisation 224
- rekonstruktion 294
- tabell 259
- övervakning 222
fregmentering 207, 209
fysiskt adressrum 169, 198

generation (avs filer) 225, 265
generationsgrupp 225, 265
generering (av operativsystemet) 297
godhetstal för system 188

heuristisk metod vid tidsplanering 192
hierarkisk relation 230
huvudrutin (vid multiprogr) 63

indexerad utsökning 252
indexfil 252, 254
informationsresurs 19
initiering av jobb, deljobb 82, 92
initiering av systemet 297

inloggning 156
integritetsproblem 266
interaktion människa-maskin 68, 88, 131
intern process 58
inverterad fil 255

JA: se jobbadministratör
JD: se jobbdisponent
jobb 57, 109
- administrativ typ av jobb 128
- administratör 92
- av beräkningstyp 119
- av reelltidstyp 131
- disponent 92
- egenskaper hos jobb 114
- klass 90
- kö 82, 9, 108
- namn 146
- non-set-up-jobb 96
- planering av jobb 83, 100, 113, 134, 137
- prioritetsstyrd selektering av jobb 81, 90
- profil 113, 118
- selektering av jobb 81, 90
- ström (el flöde) 61, 82
- tidsbehov för jobb 119
- övervakare 80, 88, 104, 108
JÖ: se jobbövervakare

katalogisering (av filer) 225, 265
klass
- av poster 222
- av processer 26, 31, 168
kommandon för tidsdelningssystem 157
kommunikationsprogram 65
kompilering 144
konsolidering av filer 239
kontrakt 326
konversation vid tidsdelning 158
kritisk fas (avs process) 34
kvantum 72, 107, 191
köravbrott 291
körningsstatistis (se även logg) 136
kövis bearbetning 64, 114

laddare 104
laddmodul (laddningsbart program) 276
leveransprov 326
linjärt adressrum 170
logg 136
logiskt adressrum 169
låsning av fil: se skydd
låsningsproblemet 40, 44, 109
låsningstillstånd 54
ländkad lagring 242, 248
länkning 144, 277
länkningsinstruktion 281
läsare 92, 97, 104

manuell resurs 19
maskinell resurs 19
maskinrepresentation av datafiler 241
master mode 12
mediaoberoende 152
mediaomvandling 62
minnesadministration 197
minnesdispotion 110, 196, 201, 204
minneskomprimering 209
minnesutnyttjande 209
montering (av sekundärminnen mm) 96
mosaikfil 111
multianvändbar resurs 18, 171
multilist fil 255
multithread 70
mål för operativsystem 300
människa-maskininteraktion: se interaktion

namnrum 169, 198
nivå för deljobb 58, 60
nyckelfil 251

objektsystem 228
omloppstid 66
open-shop 75
operationella mål 301
operativa mål 301
operativsystem def 11
operatörsarbete 141
operatörskommando 92
operatörsmeddelande 105
optimal blockstorlek (vi blockbyte) 211

overhead 194
overheadreducerande metod vid tidsplanering 194

P-operation 34
paging (se även blockutbyte) 212
parallella processer 30
partition 90
passivt tillstånd 178
placeringsstrategi 200, 209
planering av jobb 83, 100, 113, 134, 137
post, postbeskrivning 222, 240, 271
posthantering 226
postorganisation 241
precedensmatris 52
precedensstruktur för jobb, deljobb 58, 127, 139
prestnadastatistik 111
primärminne
- administration 196, 212
- adressering 201
- behov 95, 122
- bokföring 205
- disposition 201
- region 83, 103
- relokering 110
- tilldelning 83, 105, 196
- utnyttjande 100, 209

primärmål 301
prioritet 90, 105, 108, 172, 189
prioritetsstyrd selektering av jobb 81, 90
priviligerad programnivå 12
priviligierat tillstånd 103
problem 89
problempartition 97
process 18, 23, 57, 163, 176, 171, 181, 193
- egenskaper hos process 167
- interaktion: se samverkan
- parallella processer 30
- processortidsbunden 193
- samverkan med styrprogrammet 171
- styrning av process 167
- tillstånd 27, 178
profil (avs jobb) 113
prognosstyrd inläsning av block 200

program
- kommunikationsprogram 65
- dataoberoende 271
- fel 139
- nivå 12, 103
- konstruktion 275
- segment 201, 203
- serviceprogram 295
- struktur 276
- programvaruresurs 19
- testning 290
protektion (se även skydd) 21, 265
protektionsgrad 22
pseudoadressering 251
PSI (processens styrinformation) 167, 182

randomiserad adressering 251
realiserbart allokeringstillstånd 50
reelltidsapplikationer 85
reelltidsbearbetning 67, 116, 131
reelltidsorienterat operativsystem 85
reelltidssystem 83
reetablering vid köravbrott 291, 293
referenssträng 201
register 222
relation 229
replacement strategy 200
resident läsare 97
resident styrprogram 165
responstid 70, 84, 188, 211
resurs
- administration 40, 109, 182
- avbrytbar resurs 19
- behov 26, 42, 52, 70, 135, 167
- delning 17
- exklusivt användbar resurs 18, 33
- informationsresurs 19
- konflikt 171
- kö 180, 183
- manuell resurs 19
- maskinell resurs 19
- multianvändbar resurs 18, 171
- mänd 18, 41
- programvaruresurs 19
- tilldelning (se även "tilldelning") 17

samverkan mellan processer 36, 174, 181
satellitdator 61
satsvis bearbetning 61, 114
segment (avs program) 201, 203
segmentering 282
segmenttabell 205
sekretess (avs filer) 266
selsektering av jobb 81, 90
seriellt återanvändbart program 287
serviceprogram 295
selekteringsprioritet 90
semafor 34
SIMULA 31
single thread 70
självmodifierande program 287
skrivare 65, 92, 97
skydd 21, 152, 265
skyddstillstånd 12, 163, 278
slave mode 12
statusinformation (tillståndinf) 163
styrinformation: se styrinstrktuioner, styrspråk
styrinstruktioner 61, 80, 104, 143, 148
- sekvenser 153
- sekvenskontroll 149
- för tidsdelningssystem 156
styrkod: se styrinstruktioner, styrpråk
styrkodsinterpretator 146
styrning av blockutbyte 215
styrning av process 167
styrprogram 12, 86, 162, 165, 171
styrspråk 143
subprodess (delprocess) 180
supervisor call 163
swapping: se blockutbyte
symboliskt adressrum 170
synkroniserng av processer 36
syntaxkontroll 289
system (allm def) 230
systeminitiering 297
systemgenerering 297
systemmeddelande till operatör 94, 105
systemprogram 12
systemprogrammering 16
systemtillstånd 184
säkerhetsproblem (avs filer) 266
säkert tillstånd 45, 50

term 222
terminal 68
terminalanvändares "tänketid" 133
testutskrifter 290
throughput 189, 210
tidsberoende facilitet 81
tidsdelning 72, 88, 116, 131, 156, 194
tidsplanering 175, 187, 189, 192
tidsplaneringsalgoritm 191
tilldelning
- av perifera enheter 96, 104, 110
- av primärminne 104, 185, 196
- av processortid 90, 185
tillstånd
- aktivt 178
- allokeringstillstånd 49
- bearbetningsklart 28, 178
- beordrings 12, 163
- exekverande 28, 178
- låsninstillstånd 54
- passivt 178
- priviligerat 103
- realiserbart allokeringstillstånd 50
- säkert 45, 50
- väntande 28, 178, 185
tillståndsförändring 42, 179, 186
time-sharing: se tidsdelning
time-slicing: se kvantum
transaktioner, ankomstfrekvens 132
transient läsare 97
transient (del av) styrprogram 165
transport: se datatransport

ultimärmål 301
utdataklass 99
utmatningsfil 104
utnyttjande av datorsystemet 100, 213
utnyttjande av primärminne 100, 209
utrullning: se roll-out
utrymmesberoende facilitet 81
utrymmeskrav (direktminne för buffring) 67
utsökning (av filer, databas) 224, 240, 250, 255

V-operation 34
verkningsgrad vid blockutbyte 214

virtuellt adressrum 169
väntande tillstånd 28, 178, 185
working-set strategy (vid blockutbyte) 214

återstart 291
åtkomst avs datorkapacitet 75, 77
åtkomst avs datafiler 224, 250, 268

STUDENTLITTERATURS UTGIVNING I INFORMATIONSBEHANDLING
ETT URVAL

Författare	Titel
S Allén - J Thavenius	Språklig databehandling
C Andersen - M Arentzen - - A Petersen	Systembeskrivning
H E Andersin - R Sulonen	Simuleringsteknik
I Asplund (Editor)	Management Control
B-E Bengtsson	ALGOL - övningsuppgifter
G Birtwistle - O Dahl - - B Myhrhaug - K Nygaard	SIMULA BEGIN
O Björner - K Holm	Grunderna i COBOL
R Brandinger	ADB - Datorn
R Brandinger	ADB - Systemarbete
R Brandinger	Systemrationalisering
J Bubenko jr	Databehandlingsteknik
J Bubenko jr - O Dopping	Databehandlingens xyz
J Bubenko jr - O Källhammar - - B Langefors - M Lundeberg - - A Sølvberg	Systemering 70
J Bubenko jr - B Langefors - - A Sølvberg (Editors)	Computer-Aided Information Systems Analysis and Design
J Bubenko jr - T Ohlin	Introduktion till operativsystem
P G Cassel	Statistisk databehandling
K Cervin - K-E Hillander	Data för datorer
P-E Danielsson	Digitalteknik
T Danielsson - C Mellner - - S Parkholm	J5-handbok
O Dopping	Datamaskiner och databehandling
O Dopping	Kort och brett om ADB
O Dopping	Sätt rätt lätt
Edb-rådet	Training Course for Datamaticians
Edb-rådet	Training Course for Programmers
T Ekman - C-E Fröberg	Data för skolan
T Ekman - C-E Fröberg	Lärobok i ALGOL
N Enlund - R Sulonen - - M Syrjänen	Interactive Computer Graphics
L Ewald - E Roupé - - B Wahlqvist	Lärobok i BASIC
U Flodström - A Ullberg	Problemlösning med FORTRAN
S Frenkel - S Persson	Elementär FORTRAN
M Glader	ADB-lösningar på redovisningsproblem
E Gustafsson	Handbok i optisk läsning

E Gustafsson	HUPOL
E Gustafsson	Optisk handskrift - självstudiekurs
IAG	Management Information Systems
C Jennel - S-E Wallin	Datorteknik och programmering
S Kallin	Lärobok i FORTRAN
S Kallin - C Odén	Lärobok i PL/I
R Kurki-Suonio	Computability and Formal Languages
B Langefors	Problem, algoritm, datamaskin
B Langefors	System för företagsstyrning
B Langefors	Theoretical Analysis of Information Systems
Ö Leringe	Programmering av datorer - en inledning
N Lindecrantz	Datamaskinförmedlad undervisning
N Lindecrantz	Dokumentation och datamaskiner
A Lysegård	Assembler för UNIVAC 1108
A Lysegård	Lärobok i COBOL
T Matre	Prosesstyring med datamaskin - en introduksjon
P Naur	Datamaskinerna och samhället
P Naur	Planer og idéer for datalogisk institut ved Københavns universitet
P Naur	Twelve Exercises in Algorithmic Analysis
T Ohlin	Knuth´s förslag till in/utmatning i ALGOL
A Olerup	Introduktion till systemalgebra
J Palme	FORTRAN för den som kan ALGOL
J Palme	Programmeringsspråk
S Persson	Beslutstabeller
S Persson	Elementär datamatik
Regnecentralen, Köpenhamn	Datamatik
Y Sundblad	Algoritmteori
L-E Thorelli	BIM och BAS - en orientering om maskinnära programmering.
H van Tongeren - J Bubenko jr	Administrativ rationalisering - ADB systemarbete
H van Tongeren - G Lekberg	Databehandlingens och programmeringens grunder
H van Tongeren - P Övernäs	COBOL - övningsuppgifter
S-E Wallin	Datamaskinteknik
L Wettermark	ADB från början
L Wettermark	Systemarbete pågår
P Östling (red)	Projektering av reelltidssystem - en introduktion

Janis Bubenko jr - Tomas Ohlin

Introduktion till operativsystem

Ett operativsystem är ett system av datormekanismer och program som har till syfte att dels underlätta konstruktion och implementering av applikationsrutine dels styra dessa rutiners datorbearbetning.

Del 1 beskriver den maskin- och programvarumässiga utvecklingen fram till dagens operativsystem.

Del 2 belyser de viktigare funktionerna hos operativsystem och diskuterar även dessas roll i systemeringssammanhang.

Del 2 och vissa avsnitt i del 1 förutsätter kunskaper om datorers uppbyggnadsprinciper och programmering.

Dataserien — ett urval

S Allén–J Thavenius (red)	Språklig databehandling
C Andersen–M Arentzen–A Petersen	Systembeskrivning
H E Andersin–R Sulonen	Simuleringsteknik
G Birtwistle–O J Dahl–B Myhrhaug–K Nygaard	SIMULA BEGIN
O Björner–K Holm	Grunderna i COBOL
R Brandinger	Systemrationalisering
J Bubenko jr–O Dopping	Databehandlingens xyz
J Bubenko jr–T Ohlin	Introduktion till operativsystem
P G Cassel	Statistisk databehandling
O Dopping	Kort och brett om ADB
T Ekman–C E Fröberg	Lärobok i ALGOL
L Ewald–E Roupé–B Wahlqvist	Lärobok i BASIC
U Flodström–A Ullberg	Problemlösning med FORTRAN
M Glader	ADB-lösningar på redovisningsproblem
E Gustafsson	Handbok i optisk läsning
S Kallin	Lärobok i FORTRAN
S Kallin–C Odén	Lärobok i PL/I
R Kurki-Suonio	Computability and Formal Languages
B Langefors	System för företagsstyrning
B Langefors	Theoretical Analysis of Information Systems
Ö Leringe	Programmering av datorer
N Lindecrantz	Datamaskinförmedlad undervisning
P Naur	Datamaskinerna och samhället
A Olerup	Introduktion till systemalgebra
J Palme	Programmeringsspråk
S Persson	Beslutstabeller
L Wettermark	ADB från början
L Wettermark	Systemarbete pågår
P Östling (red)	Projektering av reelltidssystem

Rekvirera gärna vår katalog över databöcker!

STUDENTLITTERATUR AB FACK 221 01 LUND 1