NOTES
for
MIPS/TS
Course

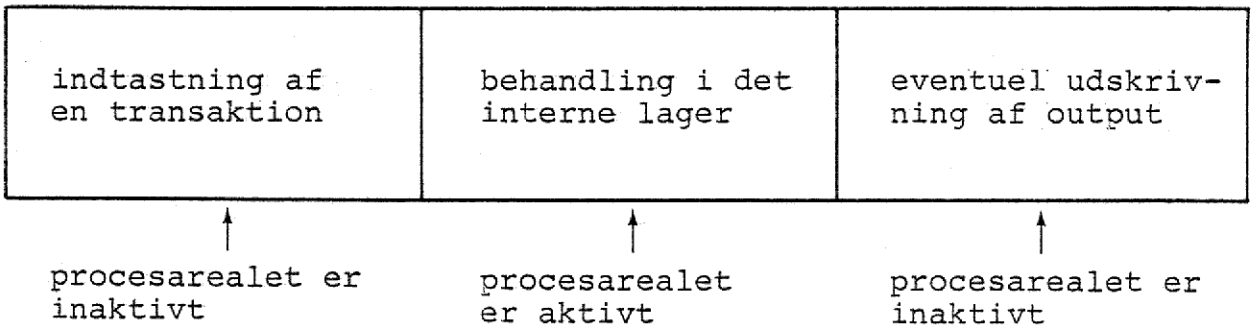
MIPS/TS

1. Indledning

Navnet MIPS/TS kan fortolkes som Modular Information Processing System/Terminal System. Formålet med udviklingen af MIPS/TS har været at skabe et effektivt operativsystem til afvikling af flere samtidige interaktive multiterminalprogrammer i det samme interne lager.

Hidtil har RC ladet interaktive programmer afvikle under det simple operativsystem "s". Ulempen ved "s" er ud fra et ressourceudnyttelsessynspunkt, at et procesareal, som afsættes til afviklingen af et program i det interne lager, først frigives til andre anvendelser, når operatøren fjerner processen. Det er i denne forbindelse uden betydning, at det tildelte procesareal er inaktivt. Ved inaktivt skal i denne forbindelse forstås, at der ikke udføres instruktioner fra procesarealet. Da afviklingen af interaktive programmer netop er karakteriseret ved mange relativt lange tidsperioder, hvor programmet blot venter på terminaloperatørens indtastning af transaktioner, vil udnyttelsen af det interne lager være urimeligt dårligt, hvis det ikke kan frigives til andre anvendelser i sådanne perioder.

Til illustration af den totale tid, som medgår til gennemførelsen af en transaktion, kan følgende figur anvendes



Det ressourcespild, som et inaktivt procesareal er udtryk for, er en væsentlig del af baggrunden for udviklingen af MIPS/TS.

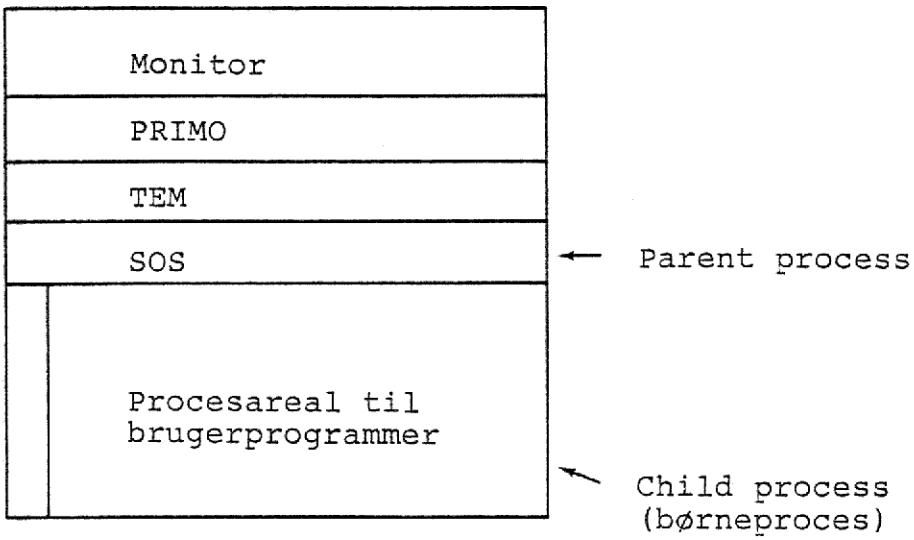
2. Implementeringsmetode

MIPS/TS består af tre selvstændige moduler, nemlig:

- SOS Swopping Operating System
- TEM Terminal Access Module
- PRIMO Printer Module

I drift vil de tre moduler ligge som adskilte processer i det interne lager.

Fordeling af det interne lager:



2.1 Beskrivelse af SOS

2.1.1 Afviklingsstrategi

I et RC 8000 datamatsystem kan en intern proces fungere som operativsystem ved at oprette og kontrollere børneprocesser. Dette betyder, at alle hjælpeprogrammer (utilities) såsom oversættere, editor, kataloghåndteringsprogrammer etc. kan benyttes på samme måde under alle operativsystemer. Operativsystemets kontrol med udførelsen af børneprocesser foregår i henhold til en veldefineret strategi. Som nævnt er SOS et swoppende operativsystem, som giver en række programmer mulighed for at deles om det samme interne lager. Når man taler om afviklingsstrategi, er det centrale spørgsmål, som rejser sig, hvorledes denne række af programmer deles om det interne lager, i.e.

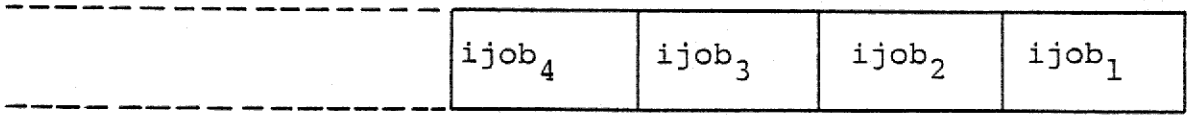
- hvornår vil et program i det interne lager blive afbrudt og swoppet ud på baggrundslageret ?

- hvornår vil et program, som er swoppet ud på baggrundslageret blive overført til det interne lager og aktiveret igen ?

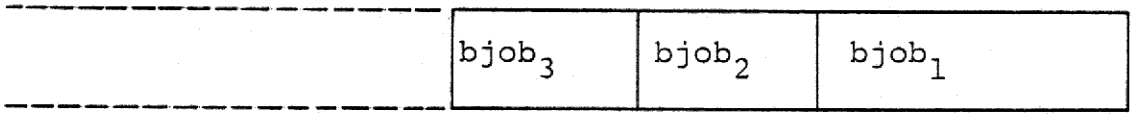
Operativsystemets administration af den knappe ressource er baseret på en køstruktur. De enkelte jobs, som bliver sat i kø, får tildelt den knappe ressource i henhold til en prioriteringsalgoritme. I operativsystemet SOS udgøres køstrukturen af tre køer.

Skønt SOS primært er udviklet med afviklingen af interaktive programmer for øje, kan SOS ligeledes håndtere baggrunddsjobs. Jobafviklingsstrategien bygger således på, at der oprettes en kø for hver jobtype, i.e. en kø med interaktive jobs og en kø med baggrundsjobs.

Interaktiv kø:

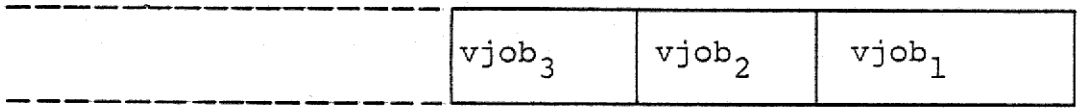


Baggrunds jobkø:



Disse to køer indeholder kun jobs, som er parate til at blive aktiveret. Derudover oprettes der en kø for de jobs, som venter på eksterne hændelser fx terminalinput.

Ventekø:



Når den eksterne hændelse, som et job i ventekøen afventer "indtræffer", vil jobbet blive flyttet tilbage til enten den interaktive kø eller baggrundsjobkøen.

Under SOS er der kun et job, som kan være aktivt ad gangen. Det job, som skal aktiveres, findes efter følgende regler:

> Hvis den interaktive kø ikke er tom, udvælg da det job, som har den højeste prioritet.
>
> Såfremt den interaktive kø er tom, men baggrundsjobkøen ikke er det, tag da det første job i baggrundsjobkøen.
>
> Hvis begge køer er tomme, vil systemet vente på en hændelse, som gør et job parat til at blive udført.

Det fremgår umiddelbart af jobudvælgelsesstrategien, at et baggrundsjob kun vil blive udført, når der ikke er nogen interaktive jobs, som umiddelbart kan udføres.

Den tidsperiode, som et aktiveret job vil få lov til at køre, afhænger af jobtypen.

Fire hændelser vil bringe et interaktivt job til standsning:

1. Når det beder om terminalinput
2. Når jobbets terminaloutput overskrider spoolingskapaciteten
3. Ved udløbet af en timeslice
4. Ved jobbets afslutning

Ligeledes findes der fire hændelser, som vil stoppe udførelsen af et baggrundsjob, nemlig:

1. Når jobbet beder om terminalinput
2. Når jobbets terminaloutput overskrider spooling-kapaciteten
3. Jobbets afslutning
4. Et interaktivt job bliver parat til at blive udført.

Baggrundsjobkøen afvikles i henhold til FIFO-princippet. Operatøren har dog mulighed for at manipulere med rækkefølgen i køen. Afviklingen af interaktive jobs finder sted i henhold til prioriteter, som ændres dynamisk.

Den prioritet, som et interaktivt job vil have på et givet tidspunkt, vil afhænge af jobbets terminalaktivitet. Et job, som hyppigt lader sig swoppe ud, fordi det beder om terminalinput, vil opnå en høj prioritet. Jobs, som bliver swoppet ud, fordi de opbruger deres timeslices, vil efterhånden opnå en lav prioritet.

Rimeligheden i denne prioriteringsmekanisme fremgår af det forhold, at jobs med en relativt omfattende terminalaktivitet sædvanligvis opfattes som mere kritiske i henseende til responsetiderne end de cpu-bundne jobs.

Prioriteringen af jobs i den interaktive kø foretages i henhold til nedenstående regler:

a) Et interaktivt job afbrydes, fordi det beder om terminalinput:

```
if priority _ class + class _ gain > 0

then priority _ class: = 0

else priority _ class: = priority _ class + class _ gain;

priority: = priority _ class
```

Når et interaktivt job således lader sig swoppe ud af det interne lager, vil dets prioritet blive forøget med den trimbare parameter "classgain". Det bemærkes, at den maksimale prioritet er 0 (nul).

b) Et interaktivt job afbrydes ved udløbet af en timeslice:

```
if priority _ class - class _ loss < min _ prio

then abort _ job

else priority _ class: = priority _ class - class _ loss;

priority: = priority _ class
```

Et interaktivt job, som opbruger hele sin timeslice, vil få sin prioritet nedtalt med den trimbare parameter "classloss". SOS tillader dog ikke, at prioriteten for et job underskrider parameteren "min_prio", der kan beregnes som - ("cpulimit"/ "class_loss"), hvor "cpu_limit" ligeledes er en trimbar parameter. Når prioriteten kommer under "min_prio", vil jobbet blive fjernet fra jobkøen.

c) Ved udvælgelsen af det "bedste" job i den interaktive kø anvender SOS følgende fremgangsmåde:

```
job= queue. first;

while job. priority < max _ priority do

job. priority: = job.  priority + prio_gain;

put _ back _ in _ queue (job);

job: = queue. first;

end;
```

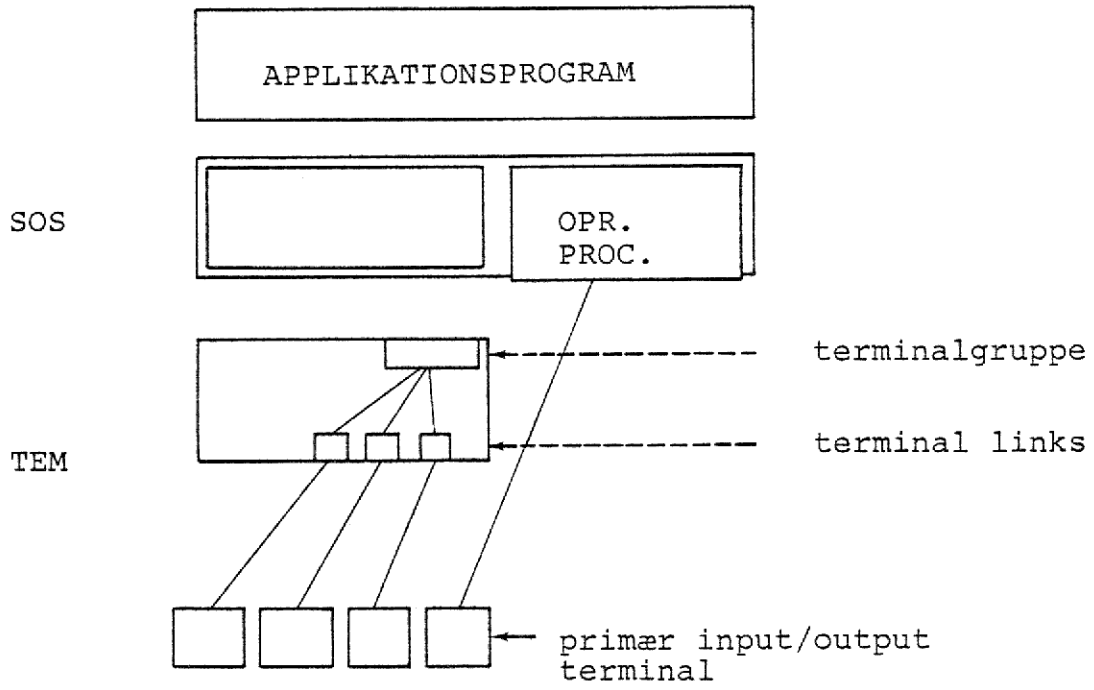
Den interaktive kø gennemløbes, indtil der findes et job, som har "max_priority" (=0). Ved gennemløbet af køen vil de jobs, som bliver sat bag i køen, få opdateret deres prioritet med den trimbare parameter "prio_gain".

Af de skitserede prioriteringsalgoritmer fremgår det, at mulighederne for tilpasning af operativsystemets jobafviklingsstrategi til en given brugerinstallations behov er ganske betydelige. Tilpasningen foretages ved at tillægge parametrene "class_gain", "class_loss" og "prio_gain" passende værdier ved systemets trimning. Længden af en timeslice, som er en trimbar variabel, er ligeledes af central betydning. En stor værdi af "class-loss" vil bevirke, at jobbet hurtigt bliver fjernet med "time exeeded". En stor værdi af "class-gain" vil bevirke, at jobs hurtigt glemmer, at de har haft en cpu-bundet periode. En stor værdi af "prio-gain" vil bevirke, at svartiden bliver proportional med cpu-tiden.

2.1.2 Terminalaccess

SOS-jobs vil kunne kommunikere med to terminaltyper, nemlig den primære input/output terminal, hvorfra jobbet er blevet oprettet, samt et antal terminaler, som er koblet op i en terminalgruppe i TEM.

Forbindelsen mellem et applikationsprogram og de terminaler, som kører det pågældende program under MIPS/TS kan illustreres på følgende måde:



Oprettelsen af et link mellem et applikationsprogram og en terminal kan foretages på programmets foranledning eller af SOS. Såfremt SOS opretter et link, vil terminalbrugernes legimitet automatisk blive kontrolleret, idet den angivne brugeridentifikation ved loginkommandoen afstemmes med SOS-brugerkatalogets indhold. Denne kontrol vil ikke blive gennemført, når applikationsprogrammet selv sørger for oprettelsen af terminallinks.

2.1.3 Behandling af ydre enheder

Da det primære formål med SOS er at understøtte interaktiv terminalbehandling bør direkte brug af ydre enheder minimaliseres. Undtagelse er baggrundslageret.

Da baggrundsjobs aldrig vil blokere interaktive jobs, kan tilgang til magnetbånd og discetter rimeligt opnås herfra. Al anden tilgang til ydre enheder bør foregå via PRIMO.

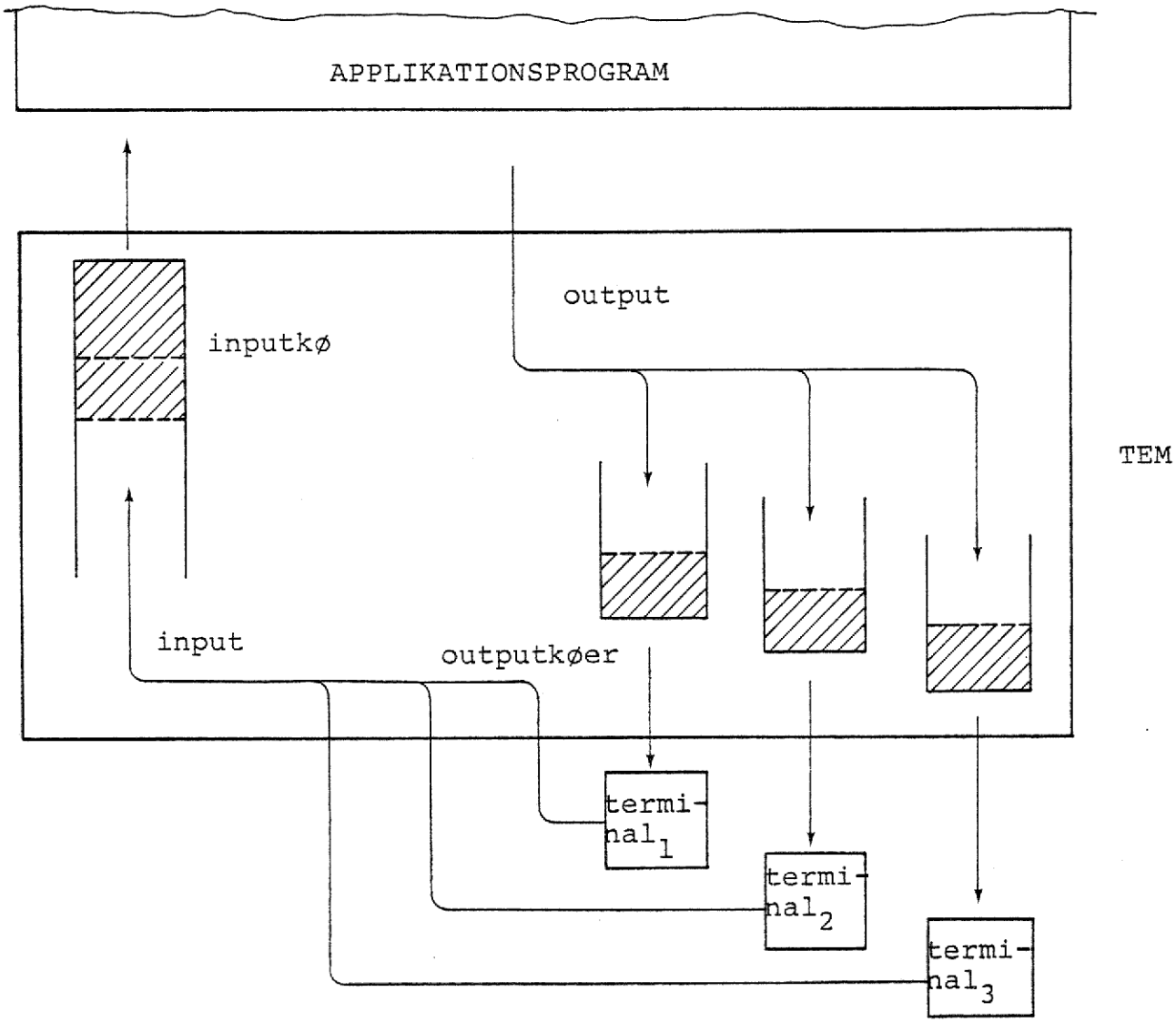
2.1.4 Brugerkatalog

Ethvert job, der udføres under SOS, skal være beskrevet i brugerkataloget. Dette katalog indeholder oplysninger om ressourcekrav, filtilgangsrettigheder, password, evt. opstartkommandoer samt beskrivelse af de terminalbrugere, der har tilgang til det pågældende job.

2.2 Beskrivelse af TEM

2.2.1 Multiplexing og spooling

Det specifikke formål med TEM er at opnå en forbedring og forenkling af applikationsprogrammers terminalaccess. TEM's virkemåde over for et applikationsprogram og et antal terminaler kan i princippet sammenlignes med en multiplexer. For hver terminalgruppe, som oprettes i TEM, bliver der automatisk etableret en kø, hvor alt terminalinput til terminalgruppens applikationsprogram samles til en fælles inputstrøm. TEM forsyner alle ankomne inputtransaktioner med terminalidentifikation, således at applikationsprogrammet kan adskille de enkelte terminalers transaktioner. Outputdata, som et applikationsprogram genererer til en terminal, afleveres i en kø, der er specifik for den pågældende terminal. Det er applikationsprogrammets ansvar, at outputdata er forsynet med entydig terminalidentifikation. Den beskrevne køstruktur er illustreret i omstående figur:



Den køstruktur, som TEM gør brug af, indebærer umiddelbart, at TEM tilbyder spooling af terminalernes input til applikationsprogrammer henholdsvis applikationsprogrammernes output til terminaler. Herved opnås en udligning af hastighedsforskellene mellem et applikationsprogram og relativt langsomme ydre enheder som fx terminaler. Et applikationsprogram vil således opfatte TEM som en hurtig terminal. Da TEM kan modtage terminalinput og afsende terminaloutput parallelt med applikationsprogrammernes transaktionsbehandling, opnår man i realtidssystemer med mange terminaler en hurtigere behandling af de enkelte terminaler end hvis applikationsprogrammerne selv varetager den nødvendige terminalaccess.

2.2.2 Terminalgruppen og terminal links

Et applikationsprogram, som skal have access til en række terminaler via TEM, må først oprette en terminalgruppe i TEM. Desuden skal applikationen etablere links mellem terminalgruppen og de terminaler, som der ønskes access til. De links, som herved oprettes mellem et applikationsprogram og en række terminaler, kan ændres dynamisk, således at en terminal kan skifte mellem forskellige applikationer. Applikationsprogrammer, som afvikles under SOS, kan overlade oprettelser respektivt nedlæggelser af såvel terminalgrupper som terminal links til operativsystemet.

2.3 Beskrivelse af PRIMO

PRIMO er et servicemodul, som kan anvendes til transport af filer mellem følgende ydre enheder:

baggrundslager	→	printer
baggrundlager	→	strimmelhuller
strimmellæser	→	baggrundslager
kortlæser	→	baggrundslager
asr-teletype	→	baggrundslager

Applikationsprogrammer, som beder PRIMO foretage en transport, kan umiddelbart fortsætte programafviklingen. Det er i denne forbindelse underordnet, hvilket operativsystem applikationen afvikles under. Enhver transport PRIMO foretager forsynes med en entydig identifikation i form af et transportnummer. Til hver transport knytter PRIMO en transportbeskrivelse, der bl.a. indeholder transportstatus. Med transportnummeret som indgangsnøgle kan en applikation til enhver tid få denne transportbeskrivelse udleveret fra PRIMO. En terminalbruger kan derved løbende få information om den aktuelle status på en defineret transport.

Ved definitionen af en transport til en printer er det ligeledes muligt at angive specielle papirtyper. En anmodning om montering af en specificeret papirtype vil da blive udskrevet på operatørkonsollen. Trykning af en fil vil først påbegynde, når operatøren giver besked om, at den ønskede papirtype er monteret i printeren.

Såfremt det ikke findes hensigtsmæssigt, at terminalbrugernes kommunikation med PRIMO finder sted via et applikationsprogram, kan to FP-utility-programmer tages i anvendelse. Utility-programmerne filexfer og fileenq giver mulighed for direkte anvendelse af de fleste af PRIMO's funktioner.