INTRODUKTION
til

Real Time Operating System
til RC 7000

Ole Olsen

N 77 - 1 Marts 77

Indhold

Denne note indeholder dels en introduktion til RTOS og dels en formel beskrivelse af datastrukturen i RTOS og nogle af de typiske rutiner i RTOS.

Indholdsfortegnelse

1. Reeltidssystemer.

Et reeltidssystem er en kombination af materiel og programmel, der tillader behandling af information så hurtigt, at resultatet af databehandlingen er til rådighed til at påvirke den proces eller de omgivelser, der overvåges eller styres.

2. RTOS opsummering.

1. Real Time Operating System er en lille lagerbaseret monitor (supervisor) beregnet til styring af reeltidsopgaver, hvori parallelle processer forekommer.
2. Rutinerne i RTOS frigør programmøren for bekymringer angående input/output enheders tidsforhold, buffering af data, prioritering og udvælgelse af næste proces (opgave ~ proces).
3. Processer i RTOS kan desuden udføres i parallel med mulighed for kommunikations- og synkroniseringsfaciliteter.
4. Input/output operationer er standardiserede i RTOS og tilpasningen til de forskellige ydre enheder sker via håndteringsrutiner (device handlers) for disse.
   Da RTOS er modulært opbygget med reentrante rutiner, kan nye håndteringsrutiner tilføjes.

5. Brugerprogrammets kommunikation med RTOS sker via en række system- og proces-kald (kald af RTOS-rutiner). Da RTOS er en delmængde af RDOS (Disk-baseret system) kan programudvikling af afprøvning foregå under RDOS (dette er en fordel).

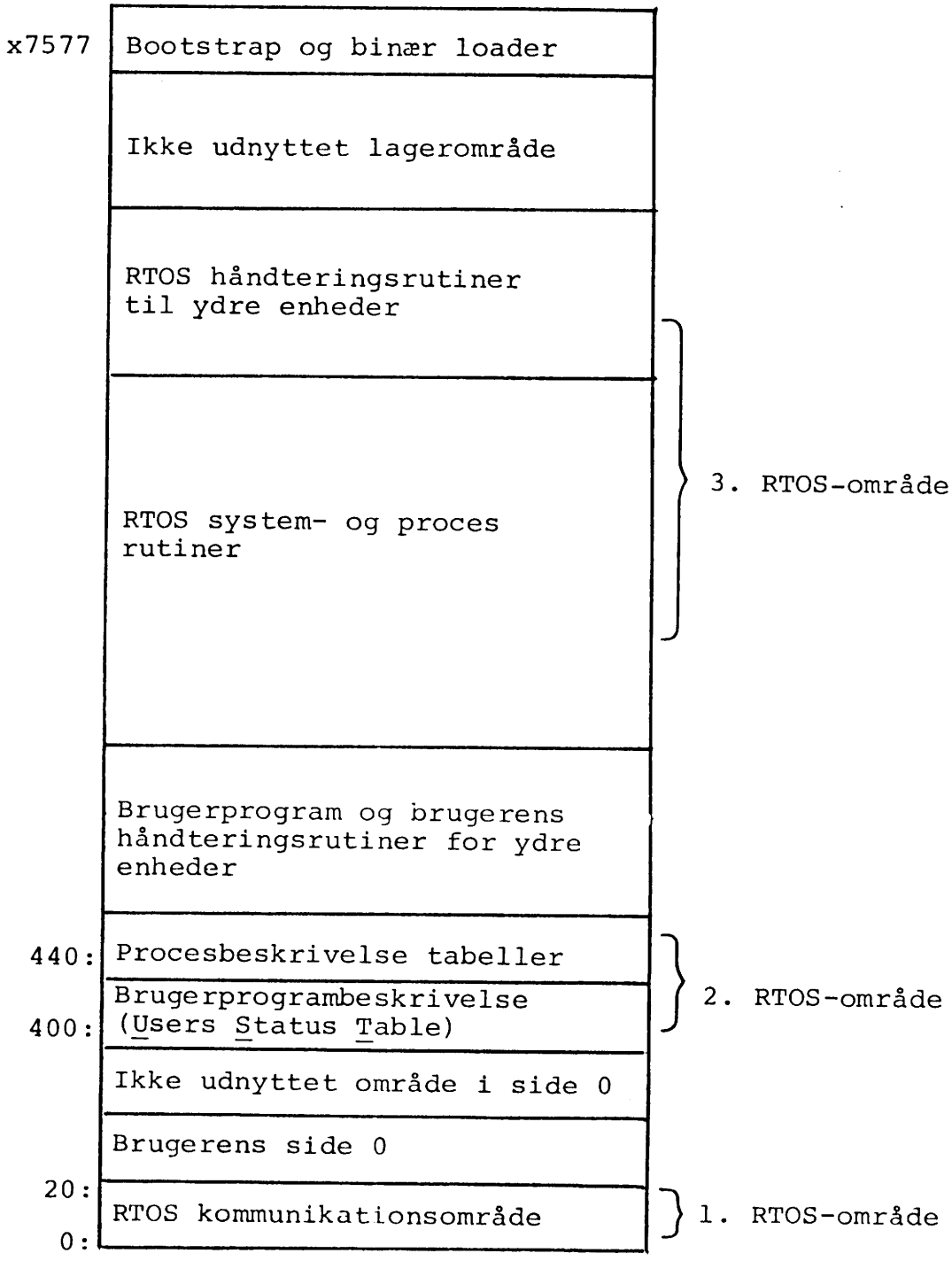


Fig. 3.1

RTOS lagerorganisation

3. RTOS organisation og lagerbenyttelse.

RTOS monitoren består af rutiner til:

1. initiering, udførsel og afslutning af input/output operationer.
2. behandling af interrupts.
3. definition af nye specielle interrupt behandlingsrutiner.
4. kommunikation med et systemur.
5. oprettelse, nedlæggelse og omprioritering af processer.
6. kommunikation og synkronisering mellem processer.

RTOS lægger beslag på tre områder i lageret. De første 20(8) lager celler anvendes til data og pegepinde for både interrupt- og proces-rutiner.

Det andet område begynder i celle 400 og indeholder en tabel over det samlede programs status (UST = User Status Table). Tabellen indeholder bl.a. programmets startadresse, dets størrelse, antal processer og I/O-kanaler, adresser på procesbeskrivelser og procestilstandskøer. Over UST findes et dataområde med en beskrivelse af hver proces og forskellige dataområder med beskrivelse af input/outputsystemet.

Brugerprogrammet(erne) og brugerens håndteringsrutiner (device handlers) for ydre enheder følger derefter og lig-

ger foran det tredie RTOS-område, der indeholder de i monitoren fundne rutiner og de nødvendige håndteringsrutiner for ydre enheder.

4. Systemgenerering.

For at kunne tilpasse et RTOS-system til en konkret anvendelse, hvori antallet af interne processer og input/output enheder er defineret, er det første trin i konstruktionen af et RTOS-program en systemgenerering. Resultatet af systemgenereringen er en binær strimmel, hvori lagerstørrelse, systemurets frekvens, antal interne processer, antal i/o-kanaler og antal og art af ydre enheder defineres.

Denne strimmel definerer størrelse og indhold af RTOS-dataområder og hvilke RTOS-rutiner og håndteringsrutiner, der skal indgå i det færdige program.

Efter systemgenerering og oversættelse af brugerprogrammet kædes disse sammen med biblioteket af RTOS-rutiner og håndteringsrutiner, og et absolut binært program opstår. Dette program kan udhulles og derefter indlæses og eksekveres på normal vis.

Et eksempel på systemgenerering er vist på fig. 4.1, og et såkaldt loader map for den ved systemgenereringen producerede programstrimmel er vist fig. 4.2.

Et loader map produceres af den relokerbare loader. Denne er et indlæseprogram, der indlæser og sammenkæder separat oversatte programmer (f.eks. hovedprogram og subrutiner). Et loader map indeholder bl.a. en liste i sorteret rækkefølge af de etiketter (labels), der er fun-

```
RTOS REV 4.00


  RTOS REV 4.00 SYSTEM GENERATION

  CORE STORAGE (IN K WORDS) 24

  RTC FREQ (0=NONE,1=10HZ,2=100HZ,3=1000HZ,4=LINE) 1

  TASKS(1-255) ? 4
  CHANNELS(1-63) ? 4

RESPOND WITH NUMBER OF UNITS

  DSK(0-1) ? 0
  DKP(0-4) ? 0
  MTA(0-8) ? 0
  CAS(0-8) ? 0
  PTR(0-2) ? 1
  PTP(0-2) ? 1
  LPT(0-2) ? 0
  CDR(0-2) ? 0
  PLT(0-2) ? 0
  QTY(0-64) ? 0
  TTYS(0-3) ? 1
  MCA(0-15) ? 0

  RESPOND WITH 0 FOR NO, 1 FOR YES

  AUTO RESTART ? 0

  HIGH PRIORITY INTERRUPTS? 0

  USER SUPPLIED DRIVERS? 0

  COMPUTER: NOVA (0) OR ECLIPSE (1) ? 0

     SUMMARY OF RTOS SYSGEN

     CODE     DCT NAME    NAME

      10        TTILC      $TTI
      11        TTODC      $TTO
      12        PTRDC      $PTR
      13        PTPDC      $PTP
      14        RTCDC

  SYSGEN OKAY? 1
  OUTPUT FILENAME ? $TTO
```

Fig. 4.1

```
SAFE =
*2
IB        045457

*2 RTOS
*6
     NMAX 000705
     ZMAX 000050
     CSZE
      EST 017413
      SST 017545

U  PTPDC 000703
U  PTRDC 000677
U  RTCDC 000600
U  RTCIS 000563
U  TTIDC 000667
U  TTODC 000673
   .CHTB 000664
   .CKDK 177777
   .CKMC 177777
   .CKMT 177777
   .CKPK 177777
   .CKQT 177777
   .CORE 057400  3.
   .ETBL 000663
   .FRTC 000001
   .GTMC 173777
   .HINT 000554
U  .INTD 000561
   .INTR 177777
   .ITBL 000564
   .NCHL 000004
   .NTSK 000004
   .RLES 177777
   .RTCF 000001  4.
   .RTCI 000012
U  .SAC3 000562
   .TCBP 000440  1.
U  .TMAX 177777
   .TPIC 177777
   .UFPT 000544  2.
*
```

Fig. 4.2

det i de indlæste programmer, og deres værdi d. v. s. adressen på den lagercelle, som det anførte symbol henviser til, eller værdien af et symbol.

I lagerområdet, der starter i celle 440, findes en række data og tabeller, som udgør en del af datastrukturen i RTOS.

1. Fra celle .TCBP = celle 440 placeres procesbeskrivelser for de interne processer. Hver procesbeskrivelse fylder 13(8) ord og da antallet af processer angives af symbolet .NTSK (no_of_tasks) = 4, kan størrelsen af lagerområdet til procesbeskrivelser beregnes.

2. Fra celle .UFPT = celle 544 placeres en tabel (User file pointer table), der fylder 2 ord for hver ydre enhed. Da antallet af ydre enheder er angivet af symbolet .NCHL (no_of_channels) fylder tabellen 8 ord. Disse og flere tabeller vil blive omtalt senere.

3. Lagerstørrelsen er angivet af symbolet .CORE og ses at være 57777(8) = 24K, som angivet ved systemgenereringen.

4. Frekvensen for reeltidsuret styres af værdien af symbolet .RTCF = 1. Frekvensen bliver 10Hz som angivet ved systemgenereringen.

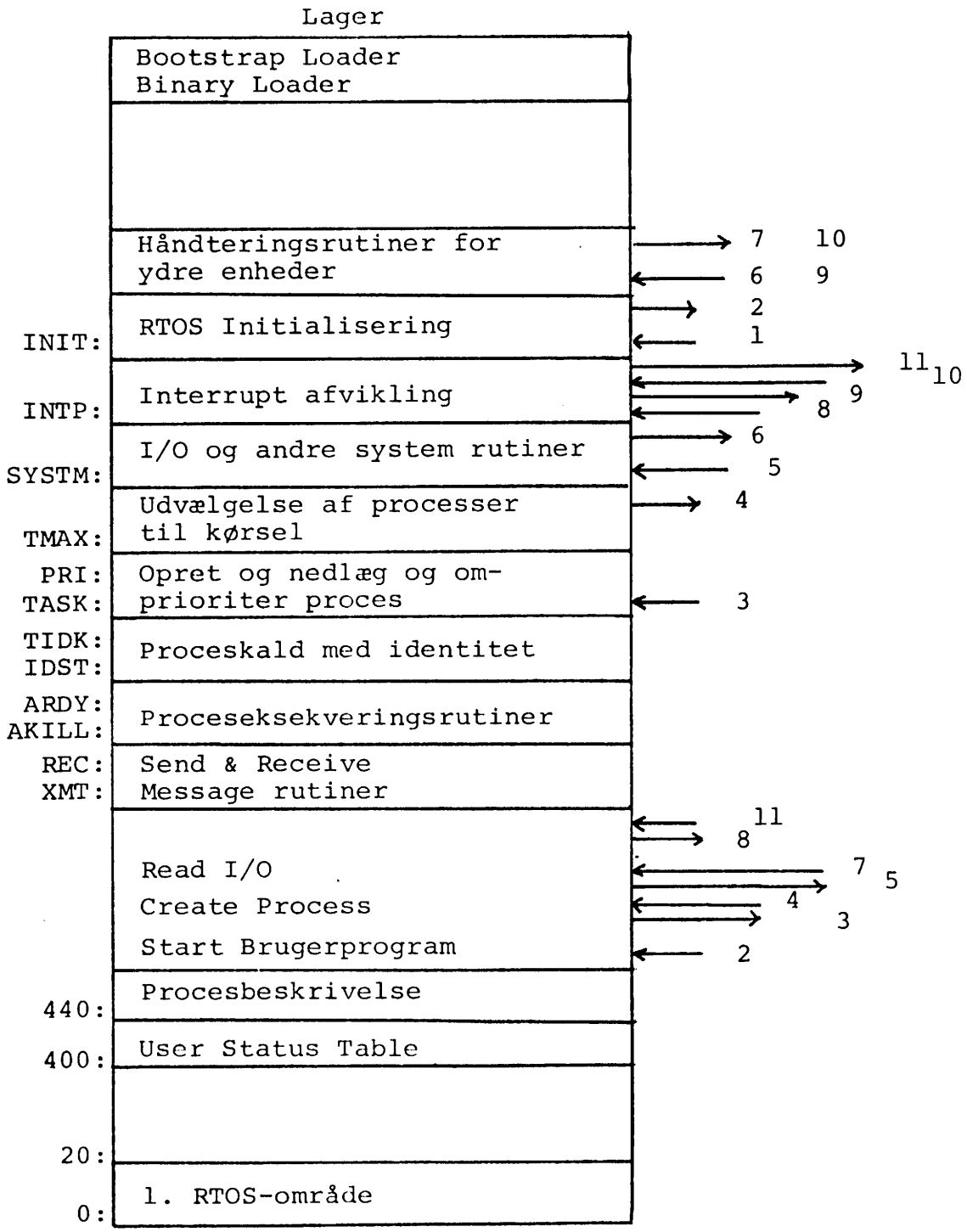

Lager
Bootstrap Loader
Binary Loader
Håndteringsrutiner for ydre enheder
RTOS Initialisering
INIT:
Interrupt afvikling
INTP:
I/O og andre system rutiner
SYSTM:
Udvælgelse af processer til kørsel
TMAX:
Opret og nedlæg og om-prioriter proces
PRI:
TASK:
Proceskald med identitet
TIDK:
IDST:
Proceseksekveringsrutiner
ARDY:
AKILL:
Send & Receive Message rutiner
REC:
XMT:
Read I/O
Create Process
Start Brugerprogram
Procesbeskrivelse
440:
User Status Table
400:
20:
1. RTOS-område
0:
7
10
6
9
2
1
11
10
9
8
6
5
4
3
11
8
7
5
4
3
2

Fig. 5.1

5. Programforløb under eksekvering.

Det mere detaljerede lagerbillede af det fuldstændige binære program er vist i fig. 5.1, og eksekveringsforløbet for et proces-rutinekald og et input/output kald er illustreret. Lagerbilledet for det i fig. 5.2 viste program og RTOS-systemet er vist i fig. 5.3.

Uden på nuværende tidspunkt at beskrive i detaljer, hvad der sker i de enkelte programmoduler, skal det principielle programforløb søges illustreret.

Den binære strimmel indlæses og startes (i celle 376). Kontrol overføres til RTOS-initialiseringsprogrammet(1), der stiller ydre enheder og tabeller i en veldefineret udgangstilstand og aktiverer interruptsystemet før kontrol overføres til brugerprogrammet(2). I brugerprogrammet findes foruden normale instruktioner, proces-kald og systemkald. Proces-kald opretter og nedlægger processer og befordrer kommunikation mellem disse, og systemkald udfører input/output operationer.

Når kaldet "create process" skal udføres, overføres kontrol til modulet "task" i RTOS (3), der opretter en beskrivelse af processen i en tabel, og såfremt den nys oprettede proces har den højeste prioritet, vil den blive startet. I denne proces antages det, at der findes et systemkald " Read I/O" "indlæs fra", der medfører, at kontrol overføres til modulet "system"(5), hvori kaldet af-

kodes og initieres via håndteringsrutiner for den kaldte ydre enhed (6), hvorefter der returneres til brugerprogrammet (7). Før eller senere vil den indledte I/O-operation medføre interrupt, og når dette indtræffer, vil kontrol overføres til interrupt afviklingsmodulet "intd" (8). Heri identificeres og betjenes den ydre enhed igen via håndteringsrutinerne (9), (10), hvorefter der returneres til brugerprogrammet (11).

Beskrivelse af programeksempel og loader map. (fig. 5.2)

1. Programmet har navnet TEST (. TITLE TEST).

2. Programmet starter ved etiketten TOT og TOT er anført efter programmets afslutning (. END TOT), startes programmet op, så snart initialiseringsprogrammet i RTOS er afsluttet.
   Symbolet TOT er en indgang til programmet, og der kan fra andre programmer refereres til TOT. (JMP TOT). TOT skal derfor erklæres som et entry-symbol (. ENT TOT) i det program, hvori det findes. Tilsvarende skal der i det program, hvori der refereres til TOT, erklæres et externt symbol TOT (. EXTN TOT).

3. Systemkaldet . OPEN 0 med AC 0 pegende på tekststregen $TTO og AC 1 = 0 skaber en perifer proces med navnet $TTO (dette er systemets betegnelse for Teletype out) og symbolsk kanalnummer 0.

Fremtidige referencer til kanal 0 vil af systemet blive identificeret med Teletype Output.
Den detaljerede beskrivelse af systemkald findes i RTOS reference manual.

4. Proceskaldet .TASK med AC0 = 10, AC1 = L og AC2 = 0 er identisk med proceskaldet

   create process (identity, priority, startadr, message),

   hvor identitet og prioritet overføres i AC0 (bit 0-7 er identitet og bit 8-15 er prioritet), startadresse overføres i AC1 og "message" i AC2 (AC2 = 0).

5. Der oprettes nu yderligere 2 processer med samme identitet = 0 og prioritet = 10.

6. Det hidtidigt afviklede program, der af RTOS initialiseringsprogrammet har fået tildelt højeste prioritet, får nu tildelt en ændret prioritet med proceskaldet .PRI, der er identisk med proceskaldet

   modify process (new priority)

   hvor parametren new priority overføres i AC0, som er lig med 10.
   Da den nye prioritet er den samme som de tidligere oprettede processers prioritet, vil processen blive placeret sidst i køen af kørbare processer.

7. Når de 4 aktive processer, der er fælles om den sammeprogramstump fra L til JMP L, startes op, vil den eneste forskel mellem dem være indholdet i AC2, der fastlægges i det øjeblik, processen skabes. Ved at analysere programmet yderligere, ses det, at AC2 anvendes til at hente en pegepind til en tekststreng, således at den først oprettede proces (se under 4) bruger teksten for MESS 0, den næste teksten for MESS 1 o. s. v.

8. Systemkaldet WRS 0 medfører udskrift af en tekststreng på kanal ≠ 0 (det var $TTO). En pegepind til strengen er i AC 0 og antal karakterer i AC 1.
   Resultatet af kørslen vil være en uendelig udskrift af de 4 tekststrenge i nummereret rækkefølge, indtil programmet standses.
   Det skal påpeges, at udskriften til TTO hver gang fuldendes før en ny proces initieres.
   Udfra loader map fig. 5. 3 ses det, at de i fig. 5. 1 anførte etiketter kan identificeres, og deres absolutte adresser kan bestemmes.

   1. Initialiseringsprogrammet RTIN begynder ved INIT = 2365.
   2. Brugerprogrammet TEST starter ved TOT = 705.
   3. Symbolet . TASK, der medfører skabelse af en proces, får værdien 006052.

Dette bitmønster svarer til instruktionen JRS @ 52. I celle 52 står adressen på det modul i RTOS, der udfører oprettelse af processer. (I programmodulet TCBMON findes symbolet . TASK og indholdet ses at være CTASK. CTASK er startadressen for det modul, der opretter en proces. CTASK er ikke med i loader map, da det er et internt symbol i. e. ikke erklæret som et externt symbol).

4. Interruptservicemodulet INTD begynder i celle INTP. Det ses at INTP = 2167. (Celle (1) skal forøvrigt indeholde værdien af symbolet INTP).

(Listningen af TCBMOM findes i afsnit 19, fig. 19. 7).

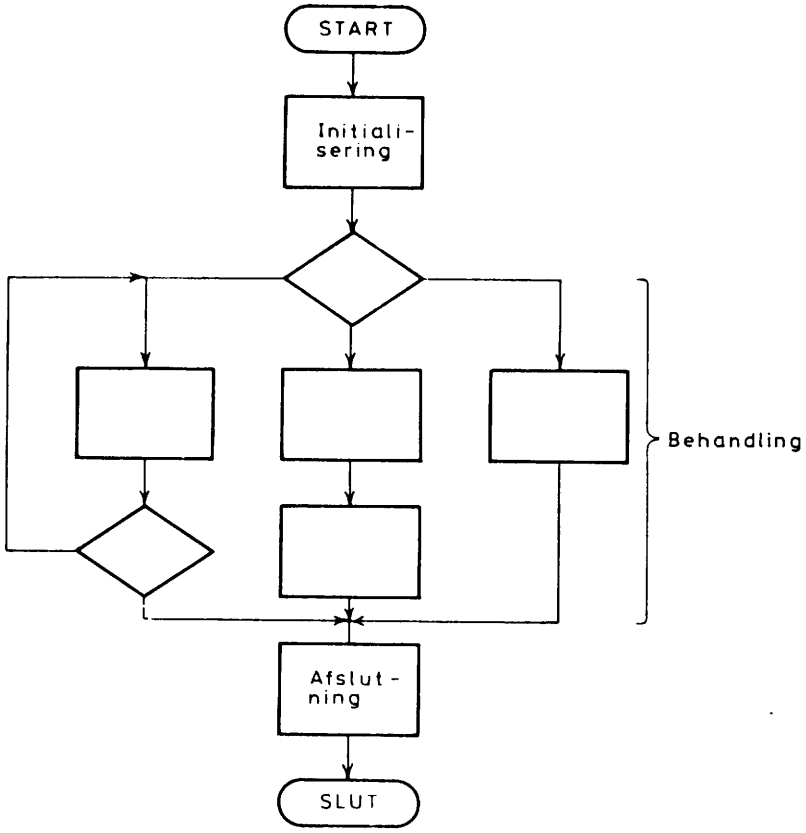
```
0001  TEST
                        .TITLE  TEST
                        .ENT    TCT
                        .EXTN   .PRI  .TASK
       000001           .TXTM   1

                        .NREL

00000'020436 TCT   :    LDA     0,.TTC  ;AC0:=POINT STC $TTC
00001'126400            SUB     1,1
00002'006017            .SYSTM
00003'014000            .OPEN   0       ;CREATE PERIPHERAL PROCES
00004'004431            JSR     ERR     ;ERROR RETURN

00005'020425            LDA     0,PRIOR
00006'024425            LDA     1,NEWTASK
00007'152400            SUB     2,2

00010'177777            .TASK           ;CREATE INTERNAL PROCES
00011'004424            JSR     ERR

00012'151400            INC     2,2
00013'000010'           .TASK           ;CREATE INTERNAL PROCES
00014'004421            JSR     ERR

00015'151400            INC     2,2
00016'000013'           .TASK           ;CREATE INTERNAL PROCES
00017'004416            JSR     ERR

00020'17777           .PRI              ;CHANGE PRIORITY
00021'151400            INC     2,2

00022'034420 L     :    LDA     3,.MESS
00023'157000            ADD     2,3
00024'021400            LDA     0,0,3
00025'024407            LDA     1,COUNT
00026'006017            .SYSTM
00027'016400            .WRS    0       ;WRITE COMMAND
00030'004405            JSR     ERR

00031'000771            JMP     L

00032'000010 PRIOR:     10
00033'000022'NEWTASK:           L
00034'000010 COUNT:     8.

00035'000400 ERR   :    JMP     .

00036'000076".TTC  :    .+1*2
                        .TXT/$TTC/
00037'022124
00040'052117
00041'000000

00042'000043'.MESS :    .+1
00043'000116"           MESS0*2
00044'000130"           MESS1*2
00045'000142"           MESS2*2
00046'000154"           MESS3*2

              MESS0:    .TXT/<15><12>TASK 1/
00047'006412
```

Fig. 5.2

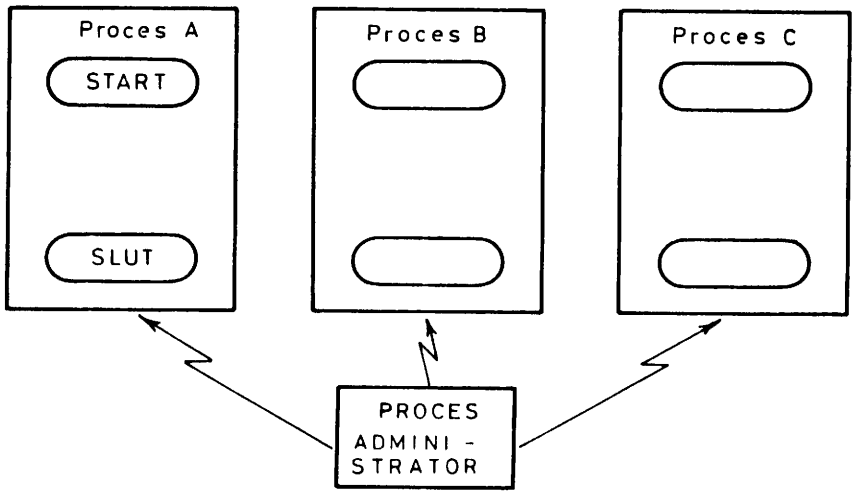
```
0002  TEST
 00050'052101
 00051'051513
 00052'020061

 00053'000000


                MESS1:   .TXT/<15><12>TASK 2/
 00054'006412
 00055'052101
 00056'051513
 00057'020062
 00060'000000


                MESS2:   .TXT/<15><12>TASK 3/
 00061'006412
 00062'052101

 00063'051513
 00064'020063
 00065'000000
                         MESS3:   .TXT/<15><12>TASK 4/
 00066'006412
 00067'052101
 00070'051513
 00071'020064

 00072'000000


        000000'          .END TCT
  0003  TEST
CCUNT   000034'
ERR     000035'
L       000022'
MESSO   000047'
MESS1   000054'
MESS2   000061'
MESS3   000066'
NEWTA   000033'
PRICR   000032'
TCT     000000'
.MESS   000042'
.PRI    000020'X
.TASK   000016'X
.TTC    000036'
*
```

```
 2
*6
     NMAX  002534
     ZMAX  000057
     CSZE
      EST  017256
      SST  017545

     INIT  002365
U    PTPDC 000703
U    PTRDC 000677
U    RTCDC 000600
U    RTCIS 000563
     SYSTM 001233
     TCT   000705
U    TTIDC 000667
U    TTCDC 000673
     .BDCT 002251
     .CHTB 000664
     .CKDK 177777
     .CKMC 177777
     .CKMT 177777
     .CKPK 177777
     .CKQT 17777
U    .CKUS 001617
U    .CLK  001354
     .CMSK 002250
     .CORE 057400
     .END1 002120
     .ETBL 000663
     .FCPN 001620
     .FRTC 000001
U    .GCHR 001674
U    .GTDY 001427
U    .GTIM 001425
     .GTMC 177777
     .HINT 000554
     .INTD 002171
     .INTP 002167
     .INTR 177777
     .ICST 001775
     .ITBL 000564
     .KILL 006051
     .NCHL 000004
     .NTSK 000004
U    .PCHR 001703
     .PRI  006050
     .RLES 177777
     .RQUE 002332
U    .RSET 001525
     .RTCF 000001
     .RTCI 000012
     .RTCS 002253
     .SAC3 002257
     .SER1 000054
     .SER2 000055
     .SER3 000056
U    .STDY 001430
U    .STIM 001426
     .SYSE 000053
     .TASK 006052
     .TCBP 000440
     .TMAX 001112
U    .TCPN 001737
     .TPIC 177777
U    .UCLI 001423
U    .UCLR 001424

     .UFPT 000544
U    .WCHR 001445
*
```

Fig. 5.3



Program, der beslaglægger CPU-tid fra start til slut ⇒ en enkelt proces i et system uden samkørsel.



Procesadministrator fordeler ressourcer (CPU-tid) til 3 processer ⇒ system med samkørsel.

Fig. 6.1

6. Systemer med samkørsel.

En proces er en eksekvering (gennemløb) fra start til slut af et brugerprogram. I systemer med samkørsel konkurerer parallelle processer om brug af cpu-tid, lagerplads og brug af ydre enheder. Tildeling af disse resourcer sker af et overordnet program (proces administrator = task scheduler) som illustreret fig. 6.1.

Processer med høj prioritet betjenes før processer med lav prioritet. Tildeling af prioritet kan være statisk eller dynamisk. Proces-administrator-modulet har ingen indflydelse på processers prioritet, undtagen ved betjening af ydre enheder, hvor en input/output operation normalt udføres fra start til slut som den højest prioriterede proces. Undtagelsen er, hvis en input/output operation afbrydes af et interrupt fra en ydre enhed med en højere materiel-prioritet.

I RTOS findes således kun en på forhånd defineret materiel prioritet. En beslutning om tildeling til og ændring af prioritet af en proces i forhold til andre processer må findes i brugerprogrammet.

7. Proces tilstande og prioritet.

Når et brugerprogram skal startes, tildeles RTOS-initieringsprogrammet højeste prioritet. For at skabe et system med parallelle processer, må brugerprogrammet oprette processer og tildele disse prioritet og identitet ved at udsende passende proceskald.

For at forhindre, at en proces egenmægtigt beslaglægger al CPU-tid, vil enhver proces under udførelse blive afbrudt af et reeltidsur med passende mellemrum, og kontrol overgår til proces-administratoren.
Et operativsystem definerer ved hjælp af en tilstandsbeskrivelse forskellige procestilstande.

I RTOS siges processer at være i en af 4 tilstande:

1. Kørende (Executing).

   Processen er under udførsel og har kontrol over centralenheden.

2. Kørbar (Ready).

   Processen venter på at opnå højeste prioritet. Dette kan ske ved at processer med højere eller samme prioritet afsluttes eller er i en tilstand kaldet

3. Udsat (Suspended)

   Hvis en proces udsender en læse/skrive ordre, kan den ikke fortsætte, før denne ordre er udført. Processen bliver derfor udsat til operationen er afsluttet.

4. Sovende (Dormant)

   Processen er endnu ikke kendt (tildelt identitet og prioritet), eller den er afsluttet og venter indtil videre.

Word 0	TPC	Task's Program Counter & Carry Bit
1	TAC0	Task's AC0
2	TAC1	Task's AC1
3	TAC2	Task's AC2
4	TAC3	Task's AC3
5	TPRST	Task Priority and Status bits
6	TSYS	System Temporary
7	TLNK	Link to next TCB in chain
10	TUSP	User or FORTRAN Stack Pointer
11	TELN	Link to Extended Save Area
12	TID	Task I.D. (1-255) or 0

TASK CONTROL BLOCK LAYOUT

Fig. 8.1

8. Procesbeskrivelse.

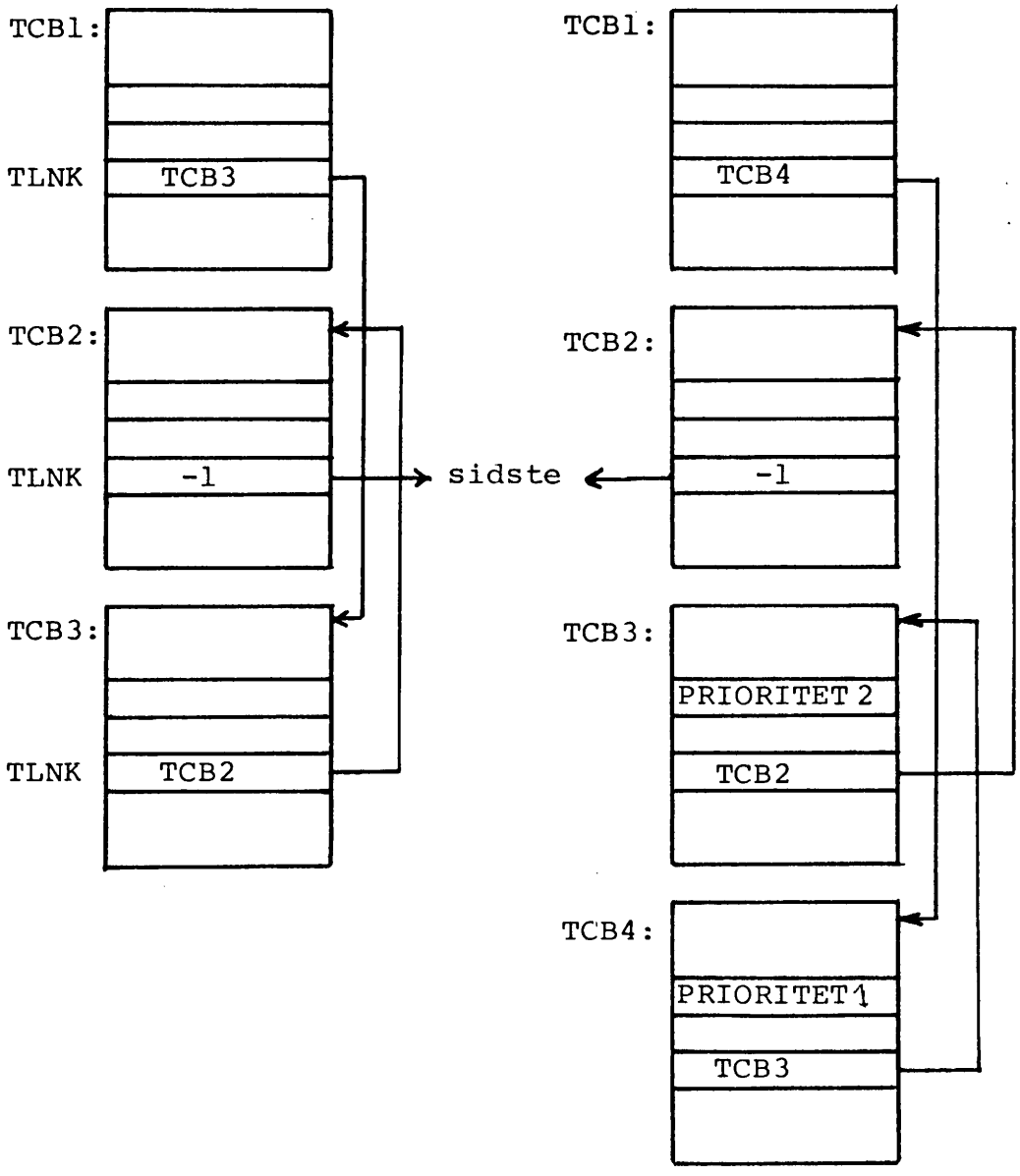
Når en proces under RTOS er i en af tilstandene kørende, kørbar eller udsat, må der i systemet findes en information om processens tilstand, for at tillade operativsystemets rutiner at betjene hver proces i overensstemmelse med deres krav til systemet. (CPU-tid-ydre enheder).

RESERVATION Ved systemgenereringen defineres det største antal samtidigt eksisterende processer, således at der altid vil være plads til en procesbeskrivelse for en aktiv proces. For en sovende proces vil der være afsat plads til procesbeskrivelsen, men denne vil være tom, da der ingen oplysninger findes om processen.

Procesbeskrivelsen for en aktiv proces er indeholdt i datastrukturen i RTOS og kaldes en Task Control Blok (TCB). Procesbeskrivelsen er illustreret fig. 8.1.

Når en proces skabes i systemet, reserveres en TCB til processen og udfyldes med oplysning om processens identitet og prioritet samt processens startadresse.

Procesbeskrivelsen for den aktive proces placeres i en kæde af procesbeskrivelser organiseret efter prioritet, således at processer med høj prioritet (højeste prioritet er 0) står først i kæden. Indsættelse i kæden af procesbeskrivelser er illustreret i fig. 8.2 A/B.



.TCBP : TCB1

Kø af aktive processer

Fig. 8.2A

.TCBP : TCB1

Tilføjelse af ny proces til køen

Fig. 8.2B

Efter indsættelsen udføres en omprioritering af de aktive processer, idet den introducerede proces kan have højest prioritet.

Når en kørende proces giver afkald på forbrug af CPU-tid, bliver dens status i.e. accumulatorer, programtæller, carry og stakpegepind (USP) gemt i dens procesbeskrivelse (TCB).

Når en kørende proces afsluttes, bliver dens TCB frigjort og bliver placeret i puljen af frie TCB'er.

Da procesbeskrivelserne af aktive eller udsatte processer er organiseret efter faldende prioritet, skal procesadministratoren (task scheduler) kun finde det øverste element i kæden af aktive processer for at finde den næste proces til kørsel.

Processer med samme prioritet afvikles cirkulært (round-robin scheduling) med lige meget CPU-tid til hver.

Når en proces introduceres i kæden af aktive processer, placeres den som sidste element indenfor sit prioritetsniveau.

Den sidste procesbeskrivelse i køen har -1 i det element, der peger på det næste led i kæden.

Ubrugte procesbeskrivelser danner en kæde i tomme beskrivelser.

9. Proceskald.

I det følgende findes en beskrivelse af nogle af de proceskald, der findes i RTOS. Der er anvendt en procedurelignende skrivemåde med angivelse af parametre til kald. I et RTOS-program vil proceskaldene have de i [ ] angivne navne, og parametre vil overføres i accumulatorer.

1. Create process(ident, priority, startadr, message) [. TASK]

   Dette kald skaber en brugerproces og beslaglægger en procesbeskrivelse i puljen af procesbeskrivelser.
   I kaldet angives identitet og prioritet, samt startadresse for det til processen knyttede program. En besked kan overføres fra den proces, der udfører oprettelse, til den oprettede proces.

2. Remove process [. KILL]

   Dette kald medfører nedlæggelse af den proces, i hvilken kaldet forekommer, og frigivelse af den til processen kørende procesbeskrivelse.

   Kontrol overføres til procesadministratoren.

3. Modify process(priority) [. PRI]

   Dette kald ændrer prioritet af den proces, der udfører kaldet.

4. Suspend process [.SUSP]

   Dette kald placerer den proces, der udfører kaldet, i tilstand udsat (suspended).

   Processen kan kun vækkes af et proceskald fra en aktiv proces.

5. Examine process (ident, result) [.IDST]

   Dette kald returnerer et resultat, der angiver tilstanden for den ved identitet angivne proces.

   Resultatet angiver, hvorvidt processen er kørbar eller udsat, og hvilke grupper af proceskald, der har sat processen i udsat tilstand.

6. Remove id process (ident) [.TIDK]

   Ready id process (ident) [.TIDR]

   Suspend id process (ident) [.TIDS]

   Modify id process (ident) [.TIDP]

   Disse proceskald udfører de angivne funktioner på den ved identitet angivne proces.

7. Remove processes (priority) [.AKILL]

   Ready processes (priority) [.ARDY]

   Suspend processes (priority) [.ASUSP]

   Disse proceskald udfører de angivne funktioner på den eller de processer, der har den angivne prioritet.

Med hensyn til yderligere detaljer f.eks. fejlreaktioner henvises der til RTOS-reference manual.

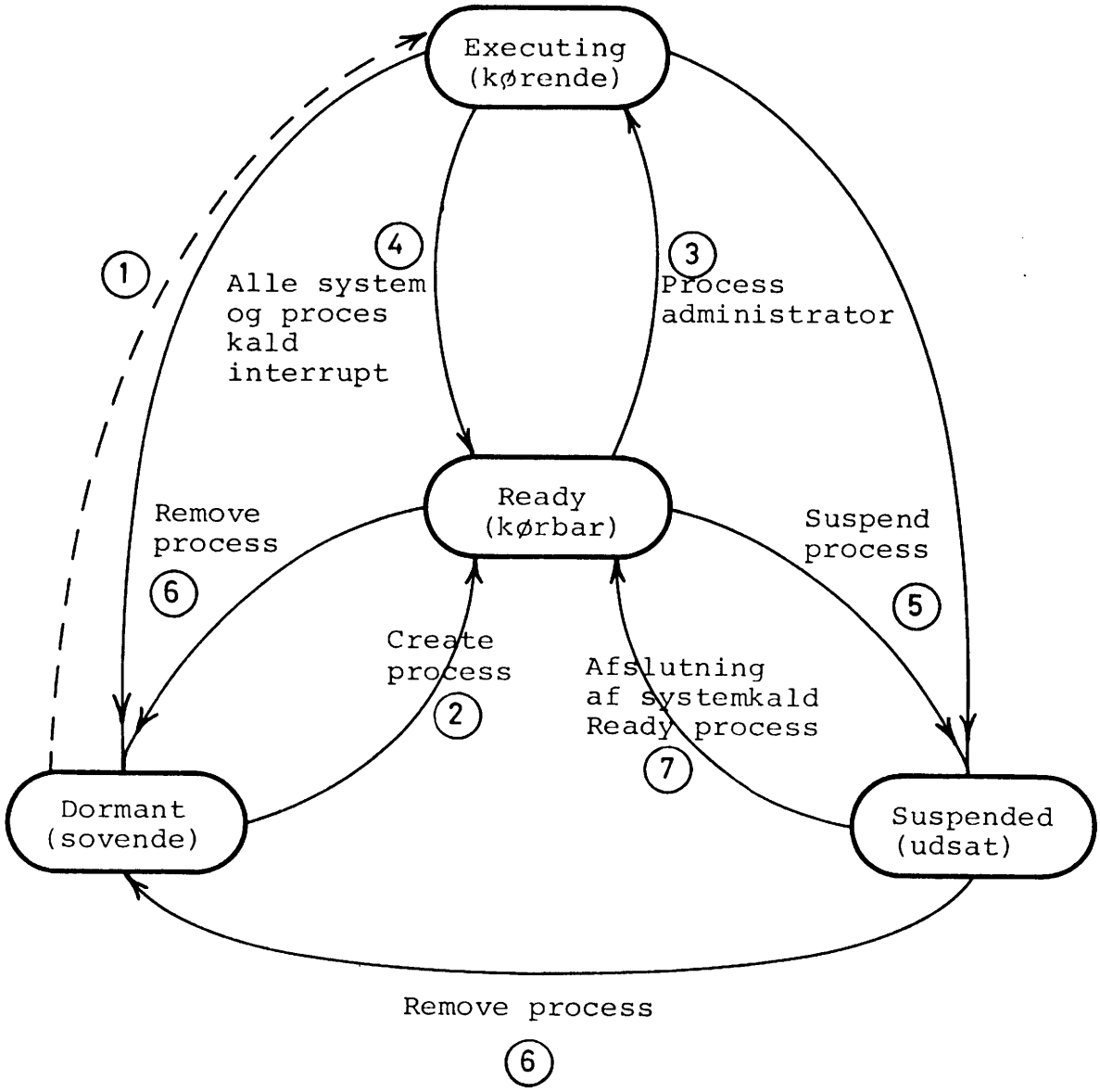


Fig. 10.1

Tilstandsskift for processer

10. Procesadministration.

I fig. 10.1 er det vist, hvorledes de hidtil anførte proceskald kan ændre tilstanden for en given proces. Figuren er ikke fuldstændig, idet ikke alle proceskald er omtalt.

Når et RTOS-program startes af RTOS-initiatiseringsrutinen, bringes processen direkte fra en sovende til en kørende tilstand (1).

Den nu kørende proces kan da skabe nye processer, og disse overgår da fra sovende til kørbar tilstand (2).

En kørbar proces vil, når den har højeste prioritet, af procesadministratoren blive udvalgt til kørsel (3).

Hvis processen under kørsel udsender et proces- eller systemkald, eller der forekommer interrupt, vil processen overgå fra kørende til kørbar tilstand (4).

En kørende eller kørbar proces kan bringes i tilstand udsat ved udsendelse af et suspend proces-kald (5).

En kørende, kørbar eller udsat proces kan bringes i sovende tilstand ved, at processen fjernes fra rækken af procesbeskrivelser af et remove proces-kald (6). Ved afslutning af systemkald eller ved ready proces-kald, kan en proces overgå fra udsat til kørbar tilstand (7).

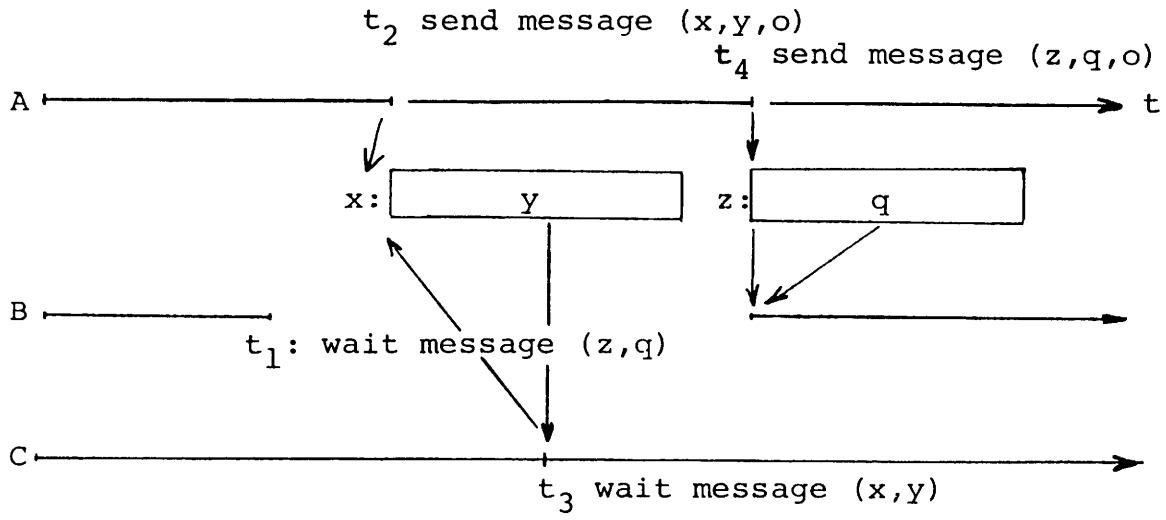


Fig. 11.1 Proces-kommunikaiton

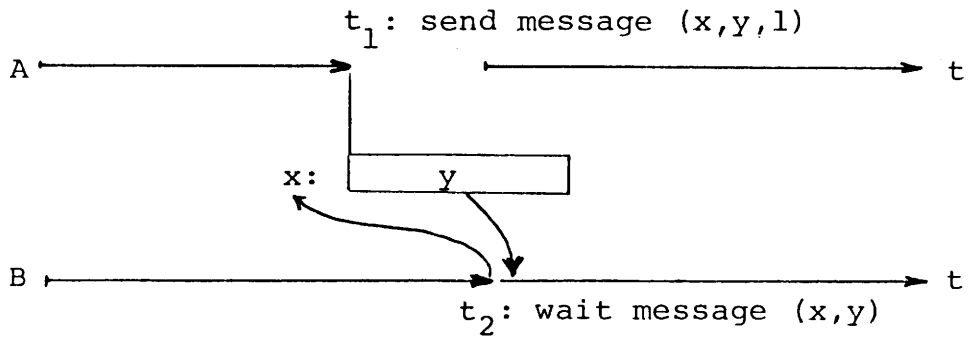
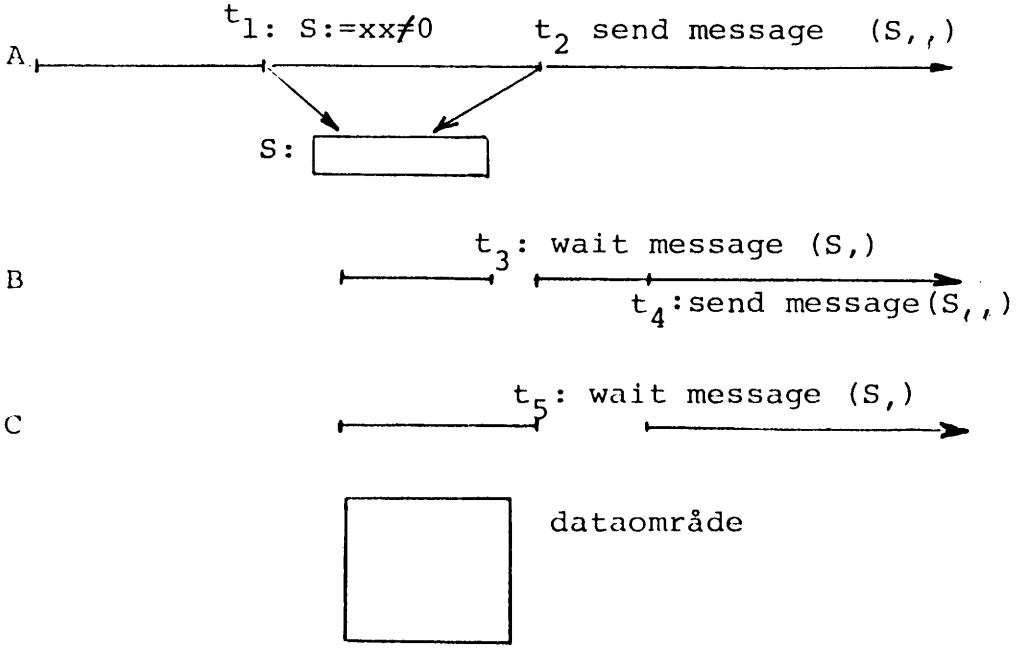


Fig. 11.2 Proces-synkronisering



11. Proceskommunikation og synkronisering.

Processer i reeltidssystemer kan forløbe asynkront, d.v.s. det kan ikke forudsættes, at et procestrin i en proces udføres før et procestrin i en anden proces. Hvis det videre forløb af en proces B er betinget af, at en anden proces A når til en bestemt tilstand, anvendes proces-kaldene "send message" og "wait message". Dette er illustreret fig. 11.1 og 11.2.

Proces A sender til tiden t_2 en besked (kaldet message = y) til celle x (kaldet message address) med kaldet "send message (x, y, 0)!" A fortsætter efter at have udført kaldet. Til tiden t_4 udsender A kaldet "send message (z, q, 0)" til processen C og fortsætter.

Processen B udsendte til tiden t_1 kaldet "wait message (z, q)!" Da z var tom (=0), går B i en ventetilstand, der varer indtil tiden t_4, hvor B genaktiveres og får overført beskeden q.

Processen C udsender til tiden t_3 kaldet "wait message (x, y)". Da A til tiden t_1 har placeret beskeden y i x, overføres y til C, og C kan fortsætte. Denne form for proceskommunikation anvendes f. eks. til at sikre, at A er færdig med at behandle de data, som B og C skal arbejde videre med. Det er også muligt at udsende et proceskald "send message" fra en interrupt servicerutine og på denne måde aktivere en proces, når et externt interrupt indtræffer.

To processer kan synkroniseres som vist i fig. 11.2. Processen A udsender til tiden t_1 kaldet "send message (x, y, 1)", og går i en ventetilstand, indtil processen B til tiden t_2 udsender kaldet "wait message (x, y)". Beskeden bliver overført til B, og A og B fortsætter.

Proceskaldene "send message" og "wait message" kan anvendes til låse og åbne adgangen til et dataområde, som anvendes af flere processer, således at kun en proces ad gangen har adgang til dataområdet.

Fremgangsmåden består i at definere en synkroniseringsadresse (message address), hvorfra andre processer forsøger at modtage en besked. Hvis den proces, der først har adgang til dataområdet, udsender kaldet "send message (s, ,)", hvor s er synkroniseringsadressen, vil den første proces, der udsender kaldet "wait message (s,)", få adgang til dataområdet. Da "wait message (s,)" samtidig nulstiller indholdet af s, vil de følgende processer, der forsøger sig med "wait message (s,)", blive udsat, indtil der udsendes et nyt kald "send message (s, ,)" o. s. v.

Da en vilkårlig proces kan knyttes til synkroniseringsordet ved at udsende "wait message (s,)", hvis dette er tomt før det første "send message (s, ,)" udsendes, skal indholdet af s være forskellig fra nul før det første "send message (s, ,)". Hvis (s) ikke er nul, vil der

komme en fejlretur fra "send message (s, ,)", men denne skal da blot ignoreres. (Disse detaljer fremgår af den nærmere beskrivelse af "send og wait message").

Eksemplet er medtaget her, da samtidig adgang fra flere processer til fælles data på en tidsmæssigt helt vilkårlig måde, er en af de oftest forekommende årsager til fejl i reeltidssystemer. Da fejlen ikke er reproducerbar p.g.a. processernes asynkrone natur, er den næsten umulig at spore, idet de involverede processer hver for sig fungerer fuldstændig korrekt.

1. Procedure delay (antal rtc intervaller)

2. Procedure get_time (time, min, sek)
   Procedure set_time (time, min, sek)

3. Procedure duser_clock (antal, rutine)

Rutiner til tidsstyring

Fig. 12.1

12. Tidsstyring af processer.

RTOS gør det muligt for en proces, at

1. gå i ventetilstand i et specificeret tidsinterval, der er et multiplum af reeltidsurets interval.
2. udføre givne funktioner på et givet tidspunkt ved at få oplysning om data og tidspunkt for systemuret. Dette kan initialiseres med et systemkald, og vil da blive ajourført, sålænge reeltidsuret giver interrupts.
3. definere et proces-intervalur for en given proces. Hver gang reeltidsuret når til det definerede interval, vil den tilknyttede proces aktiveres som en del af interrupt service rutinen for reeltidsuret. Denne facilitet er velegnet til at udføre korte processer f.eks. indsamling af analoge data med millisekunders interval i modsætning til den under 2. nævnte mulighed, hvor intervallet er af størrelsesordenen sekunder.

13. Input/output administration.

Den bedst mulige udnyttelse af de begrænsede resourcer i et datasystem var det egentlige motiv til konstruktionen af et operativsystem.

Da input/output enheder normalt er langsomme sammenlignet med datamatens interne hastighed, gælder det om at udnytte ventetiden mellem i/o-operationer til afvikling af processer, hvis dette er muligt, for at forøge udnyttelsesgraden for centralenheden.

I/O-rutinerne i RTOS skal foruden at indeholde standardiserede i/o-kald til forskellige ydre enheder sørge for at betjene enhederne hurtigst muligt. Dette opnås ved at udnytte det materielle prioritetssystem, således at hurtige ydre enheder kan betjenes, selvom langsomme enheder er i funktion.

RTOS skal udføre en sammensat input- eller output-operation (f. eks. udskrift af en tekststreng) som en udelelig operation, da kun en proces ad gangen kan kommunikere med en ydre enhed.

Hvis en ydre enhed er inaktiv eller optaget, skal RTOS oprette en kø af henvendelser til den ydre enhed og afvikle disse, efterhånden som enheden bliver ledig, uden at de processer, der har bedt om input/output, skal beskæftige sig med den konkrete afvikling af input/output operationen.

Kun hvis en proces-kombination er i/o-begrænset d. v. s. at processerne bruger mere tid til input/output end til beregning, vil processerne gå i ventetilstand.

Alle i/o-operationer i RTOS bør foregå med standardiserede i/o-kald, såfremt der findes håndteringsrutiner til den aktuelle ydre enhed.

Brugeren kan tilføje håndteringsrutiner for specielle ydre enheder ved at følge de retningslinier, der er beskrevet i manualen om "User device driver implementation in the real time operating system".

0. Procedure open (navn, kanal)

1. Procedure get_ch (ch)
   procedure put_ch (ch)
   procedure wait_ch (ch, device kode)

2. Procedure write_line(byte pointer, kanal)
   procedure read_line (byte pointer, kanal)

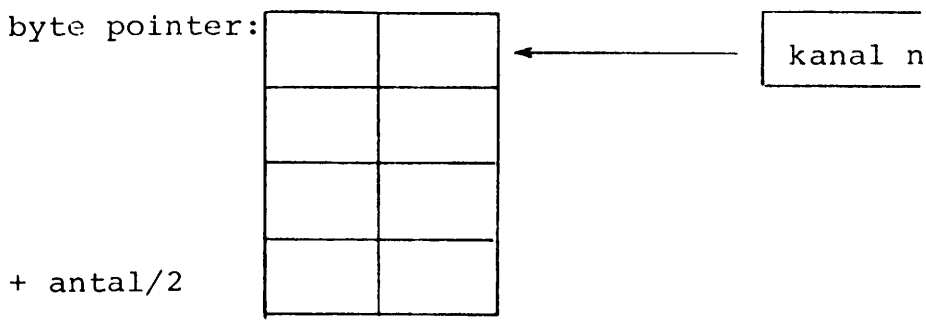
3. Procedure write_seq (byte pointer, kanal, antal)
   procedure read_seq l(byte pointer, kanal, antal)

4. Procedure write_block(adr, blok nr, antal, kanal)
   procedure read_block (adr, blok nr, antal, kanal)

Procedurer til ind- og udlæsning

Fig. 14.1

read_seq (byte pointer, antal, kanal)



Indlæsning fra kanal til lager

Fig. 14.2

14. Input/output kommandoer.

RTOS indeholder 6 muligheder for at læse eller skrive data:

1. Karakter modus: En karakter transporteres mellem konsollen og ACO (9..15). En speciel kommando "wait_ch" medfører, at en proces udsættes, indtil den angivne karakter indlæses fra en ydre enhed.

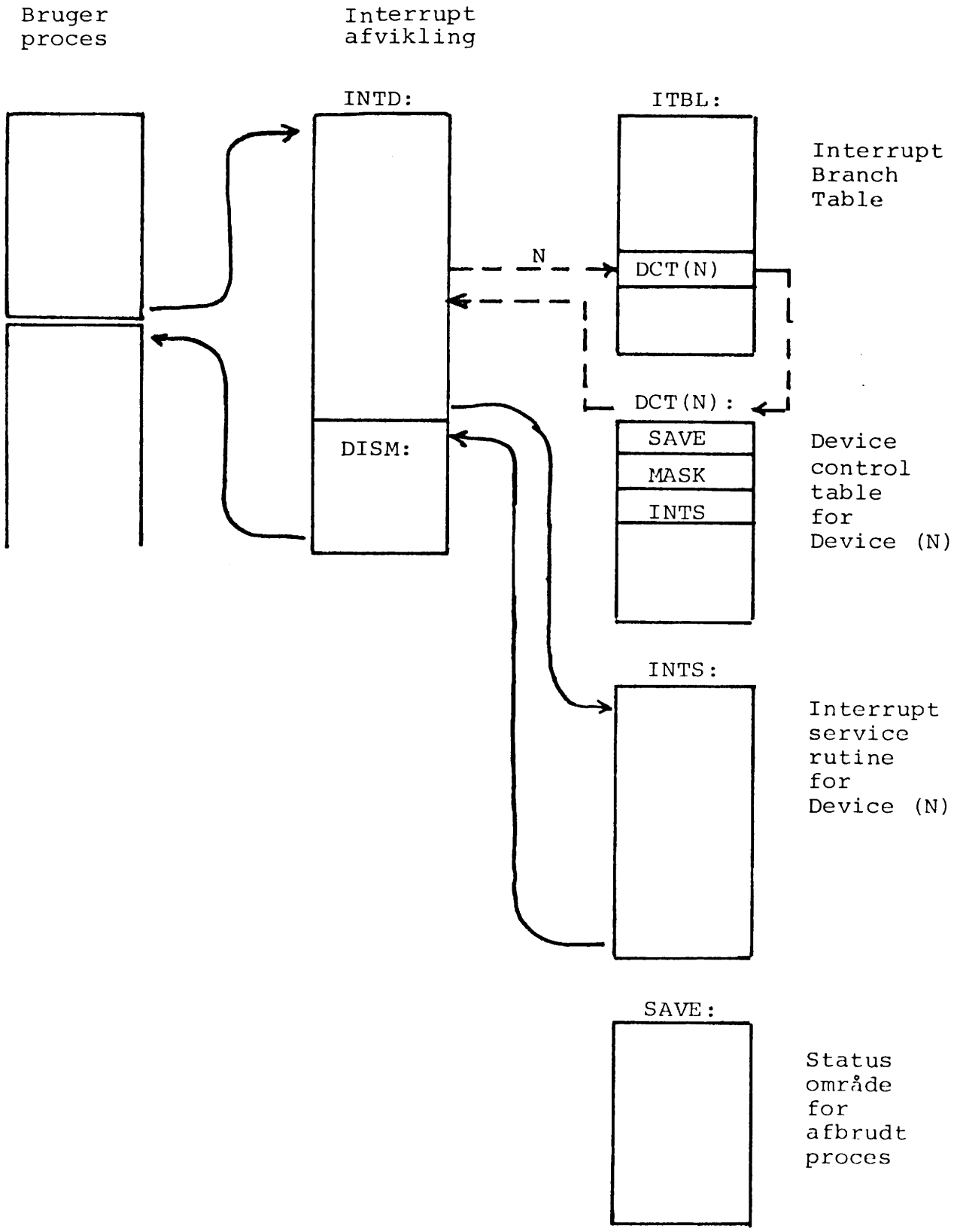
2. Linie modus: Data antages at bestå af ASCII-karakterstrenge, der afsluttes med vognretur (CR) sideskift (FF) eller en nul-karakter. Læsning eller skrivning fortsættes, indtil en af de nævnte karakterer forekommer. Denne type kommando kræver derfor ingen angivelse af antallet af karakterer, idet alene afslutningskarakteren bestemmer, hvornår kommandoen afsluttes.

3. Sekventiel modus: Et givet antal ord (16 bit) eller bytes (8 bit) overføres, som de er, mellem et dataområde og den ydre enhed. Bytes kan være karakterer eller binære data.

4. Direkte blok modus: Binære data overføres mellem pladelager og et område i arbejdslageret. Data er organiseret i blokke af 256 ord, og en eller flere blokke kan overføres ad gangen. Ved systemgenereringen defineres hvilke områder på pladelageret, der skal anvendes, og hvilke navne disse områder skal have. Ved

en OPEN-kommando defineres kanalnummeret for den ved "navn" angivne fil.

5/6. Desuden findes der rutiner beregnet til behandling af magnetbånd og båndkasetter, og endelig kan brugeren tilføje sine egne rutiner til ind- og udlæsning af data.



Programforløb ved afvikling af interrupt

Fig. 15.1

15. Interrupt service program.

Når et interrupt indtræffer, afbrydes den kørende proces, og kontrol overtages af interrupt service programmet INTD, der er et programmodul i RTOS. INTD henter den afbrydende enheds devicekode (0-76) og anvender denne som index til at hente en adresse i en tabel ITBL (interrupt branch table), som er en del af datastrukturen i RTOS. ITBL indeholder adresser på device kontrol tabeller (DCT) for de ydre enheder, der er en del af det aktuelle input/output system. DCT'en for en ydre enhed har et standardiseret format og anvendes ved alle input/output operationer på den ydre enhed.

De tre første ord i DCT'en indeholder:

Ord	Navn	Indhold
0	DCTSV	Adressen på et status område
1	DCTMS	Interrupt service maske
2	DCTIS	Adresse på enhedens interrupt service rutine.

INTD henter adressen på status området SAVE i DCTSV og i SAVE gemmes status for den afbrudte proces. En ny interrupt mask (MASK) hentes i DCTMS og sendes ud til de ydre enheder. Hermed bestemmes det hvilke ydre enheder, der kan afbryde den nys påbegyndte interrupt service rutine d. v. s. den materielle prioritet fastlægges.

INTD henter adressen på interrupt service rutinen INTS i DCTIS og overfører kontrol til denne.

Når INTS er afsluttet, overføres kontrol til interrupt - afviklingsdelen af INTD ved adressen DISMIS og nu genetableres status for den afbrudte proces, før denne igen aktiveres.

16. RTOS beskrivelse.

I de foregående afsnit er de grundlæggende ideer og begreber i et operativsystem søgt beskrevet.

For at uddybe forståelsen for hvorledes funktionerne i et operativsystem realiseres, følger en beskrivelse af datastrukturen og en verbal beskrivelse af nogle af de væsentligste procedurer (rutiner) i RTOS.

Der følger til slut en formel beskrivelse af de samme procedurer.

Den formelle beskrivelse anvender en modificeret algol-syntaks til beskrivelse af procedurer, og de anvendte variable og data repræsenterer henholdsvis ord (16-bit), bytes (8 bit) og bits. Beskrivelsen er konstrueret udfra den oversatte programlistning for RTOS. Det har vist sig problematisk dels at beskrive den adresseringsmekanisme og at overføre den kontrolstruktur, der anvendes i symbolsk maskinsprog, til en pæn algol-syntaks.

Ved at frigøre sig fra kontrolstruktur og adresseringsmekanik, skulle det være muligt at fremstille en mere indbydende og indholdsmæssigt korrekt beskrivelse af operativsystemets rutiner.

Navn	Rel adr	Indhold
USTPC	0	
USTZM	1	Laveste ubrugte celle i side 0 (ZREL)
USTSS	2	Startadresse på symboltabel
USTES	3	Slutadresse på symboladresse
USTNM	4	Laveste ubrugte celle ~~over~~ side ~~0~~ 1 (NREL)
USTSA	5	Brugerprogrammets startadresse
USTDA	6	Debuggerens startadresse
USTHU	7	Laveste ubrugte celle før RTOS og bruger-program startes
USTCS	10	
USTIT	11	
USTBR	12	
USTCH	13	Antal processer (tasks) og antal kanaler
USTCT	14	Adresse på procesbeskrivelsen (TCB) for den proces, der startes efter RTOS-initialisering
USTAC	15	Startadresse for køen af kørbare (Ready) procesbeskrivelser
USTFC	16	Startadresse for køen af sovende (Dormant) eller frie procesbeskrivelser
USTIN	17	Begyndelsesværdi af brugerprogrammet kode over side 0 (NREL)
USTOD	20	
USTSV	21	
USTSQ	22	Startadresse for køen af udsatte (Suspended) procesbeskrivelser
USTXQ	23	Startadresse for køen af ventende (Delayed) procesbeskrivelser
USTRL	24	
USTEL	25	

Users Status Table
Fig. 17.1

17. RTOS datastruktur.

Datastrukturen i RTOS omfatter bruger status tabellen (UST) i celle 400-426, procesbeskrivelserne fra celle 440 og de tabeller, der følger derefter, samt kommunikationsområdet i side 0.

1. Bruger status tabel (UST).

   I fig. 17.1 er denne tabels indhold beskrevet, således som den ser ud, når indlæsning og sammenkædning af et brugerprogram er udført.

   Oplysningerne i UST kan f.eks. udnyttes som følger:

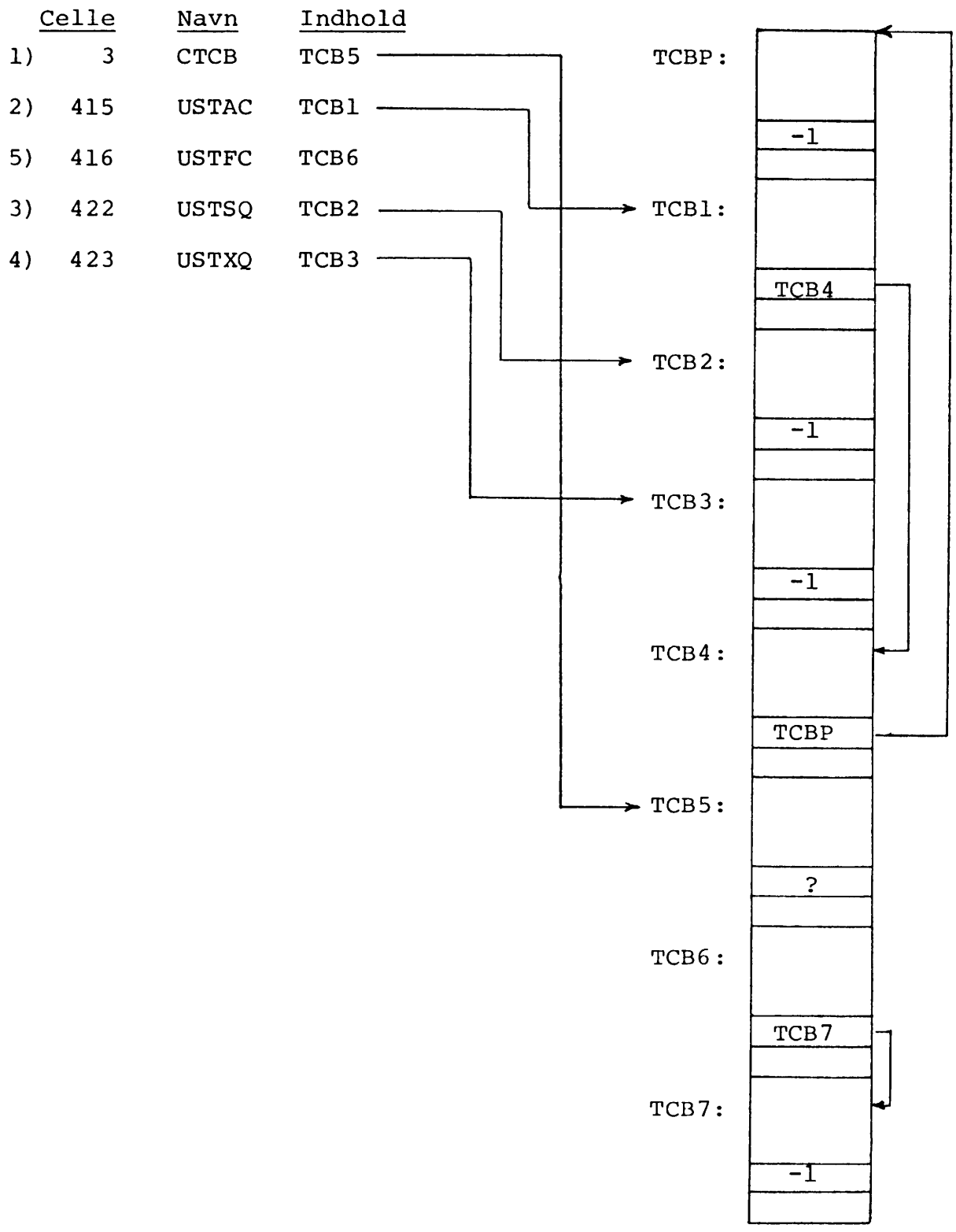
   1.1. Frit lagerområde til data bestemmes af (USTNM) og værdien af .CORE (fig. 4.2).

   1.2. Proces-rutiner i RTOS refererer til (USTAC), (USTFC), (USTSQ) og (USTXQ), der indeholder begyndelsesadresser på de fire køer af procesbeskrivelser, som RTOS opretter og vedligeholder (se fig. 17.2)

   Det er normalt kun proces- og systemrutiner, der anvender oplysningerne i UST.

2. Pulje af procesbeskrivelser.

   Ved systemgenereringen (fig. 4.1) oprettes en pulje af procesbeskrivelser startende ved .TCBP (= 440 se fig. 4.2).



Procesbeskrivelseskæder og adressereferencer

Fig. 17.2

Hver procesbeskrivelse blev omtalt i fig. 8 og sammenkædningen af procesbeskrivelser i fig. 8. 2.

RTOS har som nævnt 4 kæder af procesbeskrivelser samt en procesbeskrivelse med en særlig status (procesbeskrivelsen for den kørende proces).

Referencer til adresser på de forskellige kæder er organiseret som vist fig. 17.2.

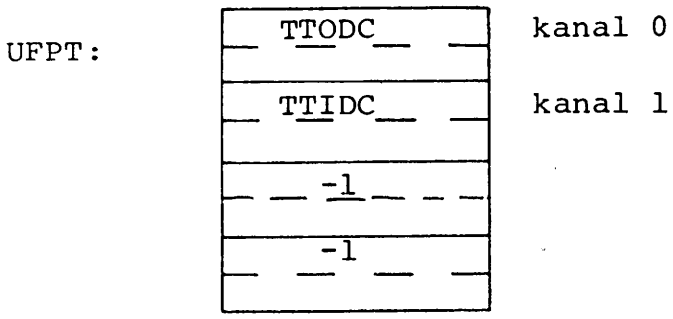
2.1. Adressen på procesbeskrivelsen for den kørende proces er altid at finde i RTOS-kommunikationsområdet i side 0. Adressen står i celle (3), der benævnes CTCB (current task control block).

2.2. Adressen på første element i kæden af kørbare procesbeskrivelser står i celle 415 = UST+USTAC.

Det ses, at kæden i eksemplet starter i TCB1 og slutter i .TCBP, hvor adressen på det næste element i kæden er -1.

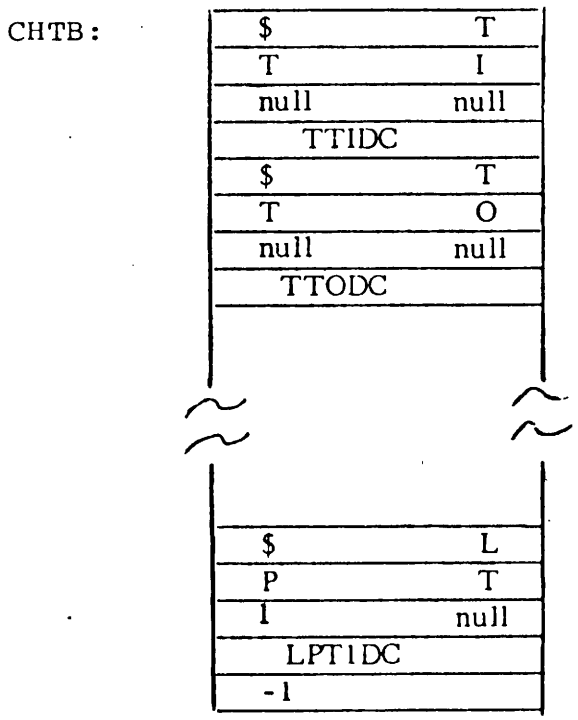
2.3. Adressen på første element i kæden af udsatte procesbeskrivelser står i celle 422=UST+USTSQ. Der er kun en udsat proces.

2.4. Adressen på kæden af ventende procesbeskrivelser (processer, der venter på at en besked (message) bliver afsendt eller modtaget) står i celle 423 = UST+USTXQ. Der er kun en ventende proces.



User file pointer table

Fig. 17.3



Standard Device Name Table

Fig. 17.4

The format of the high priority interrupt dispatcher is as follows:

```
HINT:   SKPDZ CPU          ; CHECK FOR POWER FAIL INTERRUPT
        JMP @ A            ; YES. GO TO POWER FAIL INTERRUPT SERVICE.
        STA 3@ B           ; OTHERWISE, SAVE 3
        INTA 3             ; AND GET INTERRUPT DEVICE CODE.
        SKPDZ RTC          ; WAS IT THE REAL TIME CLOCK?
        JMP @ C            ; YES. GO TO RTC INTERRUPT SERVICE
        SKPDZ device1      ; WAS IT DEVICE 1?
        JMP @ D            ; YES. GO TO DEVICE 1 INTERRUPT SERVICE
           :               ; ETC.
           :
        SKPDZ devicen

        JMP @ N            ; GO TO DEVICE N INTERRUPT SERVICE, BUT IF
        JMP @ .+1          ; NO HIGH PRIORITY DEVICE INTERRUPT
        .INTD              ; GO TO ORDINARY INTERRUPT DISPATCH ROUTINE
B:      .SAC3
A:      PWRIS              ; POWER FAIL INTERRUPT SERVICE ADDRESS
C:      RTCIS              ; RTC INTERRUPT SERVICE ADDRESS
D:      DV1IS              ; DEVICE 1 INTERRUPT SERVICE
           :
           :
N:      DVNIS              ; DEVICE N INTERRUPT SERVICE.
```

High priority interrupt dispatcher

Fig. 17.5

2.5. Adressen på kæden af sovende (frie) procesbeskrivelser findes i celle 416 = UST+USTFC. Kæden består af procesbeskrivelserne ved TCB6 og TCB7.

3. Tabeller over ydre enheder.

Efter procesbeskrivelserne følger en række tabeller, der anvendes til beskrivelse af I/O-systemet.

3.1. User File Pointer Table .UFPT.

Denne tabel anvendes til at etablere forbindelsen mellem det logiske kanalnummer, som anvendes i input/output ordrer og den fysiske enhed.

UFPT-tabellen består af blokke af 2 ord for hver kanal, som defineret ved systemgenereringen (.NCHL i fig. 4.2).

Det ene ord i blokken indeholder altid adressen på Device Control Tabellen for den fysiske enhed. Det andet ord i blokken anvendes ikke for almindelige ydre enheder, der kun indeholder 1 dokument (single file), som f.eks. TTI, PTP osv. I fig. 17.3 er tabellens udseende vist for det tilfælde, at kanal 0 er åbnet til TTO, kanal 1 til TTI. De to øvrige kanaler er endnu ikke åbnet. Ved systemgenereringen er antallet af kanaler sat til 4.

3.2. Andre tabeller.

Efter .UFPT kan der forekomme tabeller for f. eks. pladelagre, kommunikationsenheder o. s. v. Disse omtales ikke her.

3.3. High priority interrupt table .HINT .

I interruptsystemet har 2 standardenheder en speciel status nemlig overvågningsenheden for netspændingen (power fail monitor) og reeltidsuret (real time clock).

Ved systemgenereringen kan brugeren definere specielle enheder med høj prioritet. Adressen på disse enheders interruptservicerutine vil da blive inkluderet i .HINT .

Den rutine, der residerer i .HINT udfører følgende funktion, der gennemløbes som den første del af en interruptrutine:

> Det undersøges om interrupt skyldes netudfald, realtimeclock eller en bruger defineret enhed. Når enheden er identificeret, hoppes til dennes interrupt service rutine.
> Hvis interrupt ikke skyldes en af enhederne hoppes til den normale interrupt afvikler .INTD .

Interrupt service rutinen i .HINT er vist fig. 17.5.

INTD er beskrevet i afsnit 15.

3.4. Interrupt branch table .ITBL .

.ITBL, der altid findes, indeholder et ord for hver ydre enhed. Den initialiseres af RTOS-initialiseringsprogrammet RTIN, som omtales senere. RTIN vil placere adressen på enhedskontroltabellen DCT for definerede ydre enheder i den celle i .ITBL, der svarer til enhedens kode, d.v.s. ord (N) indeholder adressen på DCT for enheden med devicekode (N). For eks. indeholder ord (10) adressen TTIDC, devicekontroltabellen for TTI.

3.5. Standard Device Name Table .CHTB .

Den sidste RTOS tabel, der indlæses, er .CHTB. Tabellen indeholder oplysninger om enheder, der indeholder 1 dokument. Tabellen skabes ved systemgenereringen og indeholder blokke af 4 ord for hver defineret enhed.

Tabellen er organiseret efter stigende devicekode og indeholder dels den karakterstreng, der identificerer den ydre enhed, og dels adressen på enhedens DCT. Tabellen afsluttes med -1. Se iøvrigt fig. 17.4.

18. RTOS initialiseringsrutine RTIN .

Verbal beskrivelse:

RTIN starter i INIT.

I/O-systemet resettes, og interrupt branch tabellen undersøges. ITBL indeholder adresser på device kontrol tabellen for definerede ydre enheder. Hvis et element i ITBL indeholder -1, er enheden ikke defineret ved systemgenereringen. Hvis et element i ITBL indeholder et 1-tal i bit 0, er enheden en systemenhed, og dens DCT initialiseres. Hvis bit 0 er 0, er enheden en brugerdefineret enhed, og brugeren skal selv initialisere DCT'en. DCT'ens indhold er beskrevet i "Device driver implementation in RTOS" (litt. ref. 3). Derefter nulstilles .CMSK (celle 10), .SYS (celle 5) og .RQUE (en interrupt-type indikator).

I brugerstatustabellen (UST) sættes adressen for køen af procesbeskrivelser for udsatte og ventende processer (USTSQ og USTXQ) til -1, idet der ingen ventende eller udsatte processer er, når systemet startes. Da der endnu ikke er åbnet nogen I/O-kanaler, initialiseres hvert andet ord i UFPT-tabellen til -1.

USTCH initialiseres med antallet af processer (NTCB) og antallet af kanaler (NCHL). Adressen på kæden af aktive procesbeskrivelser (USTAC) og på kæden af tomme procesbeskrivelser (USTFC) initialiseres til -1.

CTCB (Current task control block pointer) (celle 3) indeholder adressen på procesbeskrivelsen for den kørende proces og initialiseres til .TCBP, der er adressen på det første element i den første procesbeskrivelse. Det samme sker med USTCT i brugerstatustabellen (UST).

Procesprioritet og -identitet (TPRST og TID) for den proces, der skal startes efter initialiseringen, sættes til 0 (højeste prioritet).

De resterende procesbeskrivelser initialiseres, idet procesidentitet sættes til -1, og adressen på det første element i den næste procesbeskrivelse indsættes i hver af de resterende procesbeskrivelser. Se iøvrigt fig. 18.2.

Den oprindelige værdi af USTNM (se fig. 17.1) opbevares i INMAX, og INMAX initialiseres eller USTNM genetableres.

En HALT-instruktion anbringes i celle (0), og interruptsystemet aktiveres.

Hvis brugerprogrammet har en startadresse, står denne i USTSA, og der hoppes da til denne.

RTIN er nærmere beskrevet i fig. 18.1.

```
RTOS initialisering rutine RTIN
ust    : array (0..27) of words
itbl   : array (0..77) of words
dct    : array (1..n,0..27) of words      n is # of channels
save   : array (1..n,0..6) of words
ufpt   : array (1..2*n) of words
tcb    : array (1..t,1..tln) of words     t is # of tasks.
comment itbl is the interrupt branch table
          dct is the device control table
          save is the save area for program
          status during interrupt
          ufpt is the user file pointer table
          tcb is the task control block
Init: begin reset I/O
      for k=0 to 100(8) do
        begin
          element:= itbl (k)
          if element ≠ -1 ∧ element < 0 then do
            begin
              dct(element,dctlk):= -1
              dct(element,dctql):= -1
              dct(element,dctqp):= -1
              dct(element,dctbs):= -1
              initialise dct(element,dctin)
            end
        end
      .cmsk:= 0; .sys:= 0; .rque:= 0;
      ust(ustxq):= -1; ust(ustsq):= -1
      ust(ustrl):= ust(ustel):= "wait instr"
```

Fig. 18.1a

```
        for k = 0 to 2 * nchl do
                Kedj
          ufpt(k):= -1
        ust(ustch):= ntcb * 2** 8 + nchl
        ust(ustac):= -1; ust(ustfc):= -1
        ctcb:= .tcbp; ust(ustct):= .tcbp
        comment .tcbp = adr (tcb(1,0)) wanneer hij end 0 : tcb
                .tcbp contains the address of the 1.st word
                in the 1.st procesdescription
      tcb(1,tprst):= 0; tcb(1,tid):= 0;
      n:= ntcb;
      if n = 1 goto init 7
      ust(ustfc):= adr(tcb(2,1))
      comment ust(ustfc) contains the address of the free
                chain of procesdescriptions
      for t = 2 to n do
        begin
          tcb(t,tid):= -1
          if t = n then
            begin
              tcb(tlnk):= -1
              goto init 7
            end
          tcb(t,tlnk):= adr(tcb(t+1,1))
        end
init 7: if inmax = 0 then inmax:= ust(ustnm)
                     else ust(ustnm):= inmax
      adr(0):= halt; enable interrupt;
      if ust(ustsa):= 0 then halt else adr:= ust(ustsa)
                                    begin
      goto adr;                     goto adr
      end                           end
    end; end of RTIN
```

Celle	Indhold	Funktion
17	System	Indgang til systemkald rutine
Tlnk	. Tlnk	- - link proces rutine
Rschd	Rshed	- - omprioritering
Sched	. Tmax	- - procesadministrator
Tsave	. Qblk	- - save proces rutine
. Task	Ctask	- - create proces
. Pri	Tpri	- - modify proces
. Kill	Kill	- - remove proces
		Hovedprogram
System		afvikling af systemkald
Ctask		create proces rutine
Tpri		modify - -
Kill		remove - -
. Tmax		procesadministrator
Rshed		omprioritering
Setup		opstart af højest prioriterede proces
. Qblk		gemmer procestilstand i TCB
. Tlnk		indsætter TCB i kæde af procesbeskrivelser

Opbygning af RTOS-monitor program.

Fig. 19.1

19. RTOS rutiner og administration.

Når initialiseringsprogrammet RTIN overfører kontrol til brugerprogrammet, er det brugerprogrammet, der råder over samtlige systemets resourcer. Brugerprogrammet kan nu oprette og nedlægge processer, ændre prioritet, udsætte processer o. s. v. ved at udsende kald af rutiner i RTOS, der udfører de ønskede funktioner.

Det er således brugerprogrammet, der bestemmer, hvorledes afviklingen af processer skal foregå ved at fastlægge prioriteten processerne imellem, mens den faktiske afvikling udføres af RTOS. Heraf udledes, at RTOS ikke er et program, men et sæt rutiner (en kerne ~ system nucleus) til hjælp ved implementering af et egentligt operativsystem, der indeholder en strategi for afvikling af processer.

1. RTOS-rutiner i TCBMON .

RTOS-modulet TCBMON (TCB Monitor) indeholder følgende rutiner:

create proces	[. TASK]
remove proces	[. KILL]
modify proces	[. PRI]

som beskrevet i afsnit 9.

```
program rescheduling
  begin
    link_proces (ctcb)                          [. TLNK]
scheduler: if adr (active chain)= -1 goto scheduler
setup    : ctcb:= adr (active chain)
           adr (active chain):= adr (next active tcb)
           restore (ac's, pc, carry, usp)
           goto pc
  end
```

Procesadministrator

Fig. 19.2

TCB-monitor indeholder de nævnte rutiner og de side 0-adresser, hvorigennem rutinerne kaldes, samt fælles underprogrammer og selve procesadministratoren for RTOS. Hovedprogrammets bestanddele er vist i fig. 19.1.

De fælles underprogrammer består af:

save current proces (ctcb) [.QBLK]

og link proces (tcb) [.TLNK]

der gemmer status for en proces i procesbeskrivelsen eller placerer en procesbeskrivelse i køen af aktive eller udsatte processer.

Procesadministratoren har 2 indgange

Rshed, hvor den kørende proces omplaceres i køen af aktive processer, og

.Tmax, hvor den højest prioriterede proces udvælges til kørsel.

Dette sker ved, at interruptsystemet aktiveres og adressen på køen af kørbare procesbeskrivelser undersøges.

Hvis der ingen kørbare processer (adressen indeholder -1) findes, hoppes tilbage til .Tmax, og adressen undersøges igen. Da interruptsystemet er aktiveret, kan en proces f.eks. vækkes af et interrupt, når en I/O-operation afsluttes. Når der findes en kørbar proces, hoppes til Setup, hvor interruptsystemet afbrydes, mens procestilstanden retableres med data fra procesbeskrivelsen for den første kørbare proces. Pege-

```
          procedure create_proces (ident, priority,
                                   startadr, mess, error)

          ; erklæringer ....

          begin error:= 0

            save_current_proces (ctcb)          [.QBLK]

            adr:= adr (free tcb's)

            if adr= -1 then

              begin

                error:= xx; goto exit;

              end

            if ident =0 goto getfree

            n:= no_of_tasks

            while n ≠ 0 do

              begin

                search_tcb's (ident, match)

                if match then

                  begin

                    error:= yy; goto exit;

                  end

                m:= n - 1

              end

getfree:  adr (free tcb's):= adr (next free tcb)

          create_proces_description          [Fig. 19.7]
          ctcb := adr
          link_proces (ctcb)                 [.TLNK]

          goto rescheduling

exit    : end;
```

Proces_kald, create proces

Fig. 19.3

pindende til den kørende proces (CTCB) og den næste kørbare proces opdateres (USTAC). Interruptsystemet aktiveres og der hoppes til startadressen for den kørende proces.

Principielt udføres alle proceskald med interruptsystemet deaktiveret, og der er kun redegjort for interruptsystemets tilstand i detaljer under omprioritering og udvælgelse til kørsel.
Procesadministratoren er beskrevet i fig. 19.2.

2. RTOS rutiner til oprettelse af en proces.

En proces i RTOS er karakteriseret ved dens identitet og prioritet samt startadressen for det til processen hørende program. Ved oprettelse af en proces kan der endvidere sendes en message til den oprettede proces. Rutinen har da følgende format:

create_proces(ident, priority, stadr, mess, error)

Fejlparametren returneres $\neq 0$, hvis der ikke er flere frie procesbeskrivelser, eller hvis ident er $\neq 0$ og allerede anvendt.

Create proces er nærmere beskrevet i fig. 19.3 og verbalt som følger.

Procesbeskrivelsen for den hidtil kørende proces udfyldes med processens øjeblikkelige status (dvs. registre og programtæller). Adressen på puljen af frie

```
        procedure link proces (adr)                    [.TLNK]
        ; a new proces description will be linked to the
        ; chain of ready or suspended procesdesrip-
        ; tions depending on bit 1 in the priority sta-
        ; tus word (TPRST). The position in the ready chain
        ; will be according to priority with highest
        ; priority first in chain.

          begin
            newpri:= adr(tprst)
            clear newpri(0,2)
            adr(tprst):= newpri
            if newpri (1)= 1 then
              begin        ; in suspend chain
                prevadr:= ust(ustsq)
                nextadr:= prevadr(tlnk)
                goto here
              end
            prevadr:= ust(ustac); adr of ready chain
tlnxt:      nextadr:= prevadr(tlnk)
            if nextadr ≠ -1 then
              begin
                oldpri = nextadr(tprst)
                if oldpri > newpri goto here
                prevadr:= nextadr
                goto tlnxt
              end
here:       adr(tlnk):= nextadr
            ; next procesdescription follows new p.d.
            ; new p.d. follows previous p.d.
          end
```

Fig. 19.4a

procesbeskrivelser undersøges. Hvis den er lig med -1, er der ikke flere frie procesbeskrivelser, og error-parametren sættes. Hvis ident er $\neq 0$ undersøges samtlige procesbeskrivelser, og såfremt der findes en procesbeskrivelse med samme ident, sættes error-parametren. Hvis samme ident ikke findes eller hvis ident = 0, udfyldes en procesbeskrivelse med parametrene og fjernes fra puljen af frie procesbeskrivelser. Den nye procesbeskrivelse introduceres i kæden af aktive procesbeskrivelser, og der udføres en omprioritering af de aktive processer, idet den nye proces kan have højest prioritet.
Hjælperutinerne save_current_proces [. QBLK] og link_proces [. TLNK] er beskrevet i fig. 19.4.

I save_current_proces (ctcb) gemmes procestilstanden for den kørende proces i procesbeskrivelsen. Dennes adresse står i CTCB. Indholdet af procesbeskrivelsen er vist i fig. 8.

I link_proces (tcb) indplaceres en proces i kæden af kørbare eller udsatte processer som illustreret fig. 8.2.A og B. Hvis TPRST i procesbeskrivelsen har bit 0 sat, er processen i tilstand udsat, og startadressen for udsatte processer (eller rettere deres procesbeskrivelser) står i UST+USTSQ. (Fig. 17.2).
Hvis bit 0 i TPRST er nul, findes startadressen for kørbare processer i UST+USTAC, (Fig. 17.2)

I fig. 19.6 er indholdet af den i fig. 19.2 anvendte procedure create procesdescription anført. Procesbeskrivelsen for den nye proces udfyldes med de parametre, som indgår i kaldet create proces, dog med den undtagelse, at såfremt prioriteten er nul, erstattes den med den kaldende proces' prioritet.

Sluttelig findes en programlistning af modulet TCBMON samt en del af de prodefinerede symboler, der anvendes i RTOS i fig. 19.7.

Den principielle funktion af de hidtil beskrevne proceskald kan hermed verificeres.

```
procedure save current proces (ctcb) [.QBLK]
; the status of the calling proces (caller)
; is saved in the proces description
  begin
    adr:= (ctcb) ; adr of tcb
    adr (tac0):= ac0
    adr (tac1):= ac1
    adr (tac2):= ac2
    adr (tnsp):= usp; user stack pointer
    adr (tac3):= usp; as return ac3
    adr (tpc):= pc*2 + carry;
    ; pc is return adr of user proces
    adr (tlnk):= -1
    ; procesdescription (tcb) is not yet
    ; linked in any queue
  end
```

Fig. 19.4A

```
procedure modify proces (priority)
; erklæringer
  begin
    save_current_proces (ctcb) [.QBLK]
    if priority = 0 then priority:= 1
    goto rescheduling
  end


procedure remove proces
; erklæringer
  begin
    return_tcb (ctcb)
    goto scheduler
  end
```

Fig. 19.5

```
                ;************************************************
                ;
                ;       REAL TIME OPERATING SYSTEM (RTOS)
                ;
                ;       1) SYSTEM CALL SETUP LOGIC
                ;       2) CREATE TASK (.TASK)
                ;       3) TERMINATE TASK (.KILL)
                ;       4) CHANGE TASK PRIORITY (.PRI)
                ;
                ;************************************************

                        .TITL TCBMON    ; RTOS MULTI-TASK MONITOR

                        .ENT .TMAX              ;FOR RTOS.RB

      000000            .IFE F4SW
                        .ENT .TASK .KILL .PRI   ; JSR-THRU-ZREL SYMBOLS
                        .ENDC


      000017            .LOC 17
00017 000000'           SYSTEM          ; SYSTEM CALL ENTRY

      000013            .LOC TLINK
00013 000170'           .TLNK           ; ENTRY TO LINK IN READY TCB

      000014            .LOC RSCHD
00014 000122'           RSHED           ; ENTRY TO RESCHEDULE SYSTEM

      000004            .LOC SCHED
00004 000112'           .TMAX           ; SCHEDULER ENTRY POINT

      000015            .LOC .TSAVE
00015 000150'           .QBLK           ; RESTORE CALLING TASK STATE TO HIS TCE

      000000            .IFE F4SW
                        .ZREL
      006000-.PRI=      JSR @.
00000-000073'           TPRI
      006001-.KILL=     JSR @.
00001-000102'           KILL
      006002-.TASK=     JSR @.
00002-000011'           CTASK
                        .ENDC

                        .NREL


                ; SYSTEM CALL LOGIC

00000'175400 SYSTEM:    INC 3 3         ; SETUP FOR NORMAL RETURN
00001'175400            INC 3 3
00002'054005            STA 3 .SYS.     ; INDICATE SYSTEM MODE
00003'006015            JSR @.TSAV      ; SETUP TCB FOR CURRENT TASK
00004'034005            LDA 3 .SYS.     ; PICKUP RETURN ADDRESS+2
00005'021776            LDA 0 -2 3      ; GET SYSTEM COMMAND WORD
00006'041006            STA 0 TSYS 2    ; SAVE IN TCB
00007'002401            JMP @.+1        ; FINISH SYSTEM CALL PROCESSING
00010'177777            SYSTM
```

Fig. 19.7a

```
                ;***********************************************
                ;
                ;       ENTRY POINT FOR .TASK SYSTEM CALL
                ; FORMAT:
                ;       LDA 0 <(TASK ID),(TASK PRIORITY).
                ;       LDA 1 <TASK ADDRESS>
                ;       LDA 2 <ONE WORD MESSAGE>
                ;       .TASK
                ;     * <ERROR RETURN>  ;NO TCB'S LEFT
                ;       <RETURN>
                ;
                ;***********************************************

00011'175400 CTASK:  INC 3 3          ; SETUP FOR NORMAL RETURN
00012'054005         STA 3 .SYS.      ; INDICATE SYSTEM MODE
00013'004535         JSR .QBLK        ; SETUP QUEUE BLOCK FOR OLD TASK
00014'060277         INTDS
00015'036503         LDA 3 @TFREE     ; START OF FREE TCB CHAIN
00016'174015         COM# 3 3 SNR     ; ALL GONE ?
00017'000452         JMP TSKER        ; YES
00020'024501         LDA 1 PRMSK      ; GET PRIORITY MASK
00021'125100         MOVS 1 1         ; SWITCH INTO TASK ID MASK
00022'123705         ANDS 1 0 SNR     ; RETAIN ONLY TASK ID
00023'000413         JMP GFCEN        ; ID IS ZERO--IGNORE ID COMPARISON
00024'034440         LDA 3 TCBPL      ; GET START OF TCB POOL
00025'030440         LDA 2 NTASK      ; GET NUMBER OF TASKS
00026'050006         STA 2 RLOC       ; SAVE AS COUNTER
00027'030437         LDA 2 TCBSZ      ; GET SIZE OF A TCB ENTRY
00030'025412 CKLP:   LDA 1 TID 3      ; GET TCB'S CURRENT ID
00031'106415         SUB# 0 1 SNR     ; CHECK IF A MATCH IN TABLE
00032'000435         JMP TIDER        ; YES--TAKE ERROR RETURN
00033'157000         ADD 2 3          ; MOVE TO NEXT TCB
00034'014006         DSZ RLOC         ; DECREMENT COUNTER
00035'000773         JMP CKLP         ; CONTINUE LOOKING
00036'036462 GFCEN:  LDA 3 @TFREE     ; GET ENTRY OFF FREE CHAIN
00037'174015         COM# 3 3 SNR     ; CHECK IF END OF TABLE
00040'000431         JMP TSKER        ; YES--TAKE ERROR RETURN
00041'025407         LDA 1 TLNK 3     ; NO--GOT ONE ALRIGHT
00042'046456         STA 1 @TFREE     ; RESTORE FREE TCB POINTER
00043'041412         STA 0 TID 3      ; SAVE AS TASK ID
00044'030003         LDA 2 CTCB       ; GET CURRENT TCB ADDRESS AGAIN
00045'025003         LDA 1 TAC2 2     ; GET MESSAGE (CONTENTS OF AC2)
00046'045403         STA 1 TAC2 3     ; PASS TO NEW TASK
00047'024452         LDA 1 PRMSK      ; GET PRIORITY MASK
00050'021001         LDA 0 TAC0 2     ; GET DESIRED PRIORITY
00051'123405         AND 1 0 SNR      ; RETAIN ONLY PRIORITY--IS IT 0?
00052'021005         LDA 0 TPRST 2    ; YES--USE OLD PRIORITY
00053'123400         AND 1 0          ; MASK OUT STATUS BITS, KEEP AC0
00054'025002         LDA 1 TAC1 2     ; GET PC FOR NEW TASK, HOLD IN AC1
00055'054003         STA 3 CTCB       ; PUT NEW TCB ADDR AS CURRENT TCB
00056'004512         JSR .TLNK        ; PUT OLD TCB INTO CHAIN BY PRIORITY
00057'030003         LDA 2 CTCB       ; NEW TCB ADDR TO AC2

      000000         .IFE F4SW
00060'125120         MOVZL 1 1        ; SHIFT PC & INCLUDE SHINY NEW CARRY
00061'045000         STA 1 TPC 2      ; PUT IN TCB
00062'041005         STA 0 TPRST 2    ; PUT PRIORITY IN NEW TCB
                     .ENDC

00063'002014         JMP @RSCHD       ; LINK NEW TCB AND SCHEDULE


00064'177777 TCBPL:  .TCBP            ; START OF TCB POOL
00065'177777 NTASK:  .NTSK            ; NUMBER OF DEFINED TASKS
00066'000013 TCBSZ:  TLN              ; SIZE OF TASK CONTROL BLOCK
00067'006001#TIDER:  JSR @.SYSER
00070'000061         ERTID            ; TASK ID ERROR CODE
00071'006001#TSKER:  JSR @.SYSER
00072'000042         ERNOT
```

Fig. 19.7b

```
                ;*****************************************************
                ;
                ;       ENTRY POINT FOR .PRI INSTRUCTION CALL
                ; FORMAT:
                ;       LDA 0 <TASK PRIORITY>   ; 1-255
                ;       .PRI
                ;       <RETURN>        ;NORMAL RETURN
                ;
                ;*****************************************************

00073'054005 TPRI:    STA 3 .SYS.       ; INDICATE SYSTEM MODE
00074'004454          JSR .@BLK
00075'024424          LDA 1 PRMSK       ; GET PRIORITY MASK
00076'123405          AND 1 0 SNR       ; ZERO NOT ALLOWED TO BE USED
00077'102520          SUBZL 0 0         ; MAKE IT A 1

00100'041005          STA 0 TPRST 2     ; SET NEW PRIORITY
00101'002014          JMP @RSCHD        ; EXIT TO SCHEDULER
                                        ; TO ACTIVATE JOB

                ;*****************************************************
                ;
                ;       ENTRY POINT TO KILL CALLING TASK (.KILL)
                ;
                ;*****************************************************

00102'054005 KILL:    STA 3 .SYS.       ; INDICATE SYSTEM MODE
00103'030003          LDA 2 CTCB        ; GET ADDRESS CURRENT TCB
00104'060277          INTDS             ; DISABLE MOMENTARILY
00105'026413          LDA 1 @TFREE      ; GET ADDR OF FREE TCB
00106'045007          STA 1 TLNK 2      ; LINK NEW FIRST TO OLD FIRST
00107'126000          ADC 1 1                   ; RESET TASK ID POINTER
00110'045012          STA 1 TID 2       ; TO INDICATE IT IS AVAILABLE
00111'052407          STA 2 @TFREE      ; NEW FIRST TCB IN POOL

                ;*****************************************************
                ;
                ;       JOB SCHEDULER FOR THE REAL TIME OPERATING SYSTEM
                ;
                ;*****************************************************

00112'060177 .TMAX:   INTEN             ; TURN ON THE WORLD
00113'032404          LDA 2 @ATCB       ; START OF ACTIVE CHAIN
00114'150015          COM# 2 2 SNR      ; ANY TASKS ACTIVE ?
00115'000776          JMP .TMAX+1       ; NO--BUT KEEP LOOKING
00116'000407          JMP SETUP         ; YES--SET HIM UP

00117'000415 ATCB:    UST+USTAC         ; ADDR OF FIRST TCB IN ACTIVE CHAIN
00120'000416 TFREE:   UST+USTFC         ; ADDR OF FIRST FREE TCB
00121'000377 PRMSK:   377               ; PRIORITY MASK

                ;*****************************************************
                ;
                ;       RESTORE CURRENT TCB TO READY TCB CHAIN
                ;       AND ACTIVATE THE HIGHEST PRIORITY READY TASK
                ;
                ;*****************************************************

                TMAX2:                  ; SYMBOL FOR RDOS COMPATIBILITY
00122'030003 RSHED:   LDA 2 CTCB        ; GET CURRENT TCB ADDRESS
00123'006013          JSR @TLINK        ; LINK INTO READY CHAIN
00124'000766          JMP .TMAX         ; ACTIVE HIGHEST PRIORITY TASK
```

Fig. 19.7c

```
                ;*********************************************
                ;
                ;       ACTIVATE JOB OF HIGHEST PRIORITY
                ;
                ;*********************************************

00125'050277 SETUP:  INTDS            ; HOLD IT WORLD WHILE I SETUP
00126'032771         LDA 2 @ATCB      ; GET TOP OF CHAIN AGAIN TO INSURE
                                      ; NOBODY HAS BEEN PUT ON WHILE COMING F
00127'050003         STA 2 CTCB       ; SET TASK AS CURRENT
00130'102400         SUB 0 0          ; SET .SYS. FLAG TO INDICATE
00131'040005         STA 0 .SYS.      ; IN USER MODE
00132'021007         LDA 0 TLNK 2     ; GET ADDRESS OF NEXT TCB
00133'042764         STA 0 @ATCB      ; SAVE ON ACTIVE TCB CHAIN PNTR
00134'021001         LDA 0 TAC0 2     ; RESTORE AC0
00135'025002         LDA 1 TAC1 2     ; RESORE AC1
00136'035000         LDA 3 TPC 2      ; PICKUP PROGRAM COUNTER AND CARRY
00137'175220         MOVZR 3 3        ; RESTORE CARRY
00140'054407         STA 3 START      ; SAVE PC IN LOCATION START
00141'035010         LDA 3 TUSP 2     ; RESTORE USP
00142'054016         STA 3 USP
00143'035004         LDA 3 TAC3 2     ; RESTORE AC3 VALUE
00144'031003         LDA 2 TAC2 2     ; RESTORE AC2
00145'060177         INTEN            ; RE-ENABLE THE INTERRUPT
00146'002401         JMP @START       ; EXIT TO JOB VIA PC IN LOC "START"

00147'000000 START:  0                ; TASK STARTING ADDRESS

                ;*************************************************************
                ;
                ; A GENERAL NON REENTRANT ROUTINE TO SETUP A QUEUE BLOCK
                ;
                ; USED ONLY BY SYSTEM/TASK CALLS (NO REENTRANCY REQUIRED)
                ;
                ; CALLING SEQUENCE:
                ;       JSR .QBLK   OR   JSR @.TSAVE
                ;       <RETURN>          ;(AC2)=ADDR OF CURRENT TCB BLOCK
                ;
                ;*************************************************************

00150'054417 .QBLK:  STA 3 QRTN       ; SAVE RETURN ADDRESS
00151'155000         MOV 2 3          ; SAVE AC2 TEMPORARILY
00152'030003         LDA 2 CTCB       ; GET ADDR OF CURRENT TCB
00153'041001         STA 0 TAC0 2     ; ENTER AC0 IN QUEUE BLOCK
00154'045002         STA 1 TAC1 2     ; ENTER AC1 IN QUEUE BLOCK
00155'055003         STA 3 TAC2 2     ; SAVE AC2 IN QUEUE BLOCK
00156'034016         LDA 3 USP        ; SAVE USP IN TCB
00157'055010         STA 3 TUSP 2
00160'055004         STA 3 TAC3 2     ; SAVE USP VALUE AS RETURN AC3
00161'034005         LDA 3 .SYS.      ; GET RETURN ADDRESS
00162'175100         MOVL 3 3         ; SHIFT WITH CARRY ADDED
00163'055000         STA 3 TPC 2      ; ENTER IN QUEUE BLOCK
00164'176000         ADC 3 3          ; SET LINK WORD TO -1
00165'055007         STA 3 TLNK 2

00166'002401 TSAVX:  JMP @QRTN        ; EXIT

00167'000000 QRTN:   0                ; SUBROUTINE RETURN ADDRESS
```

Fig. 19.7d

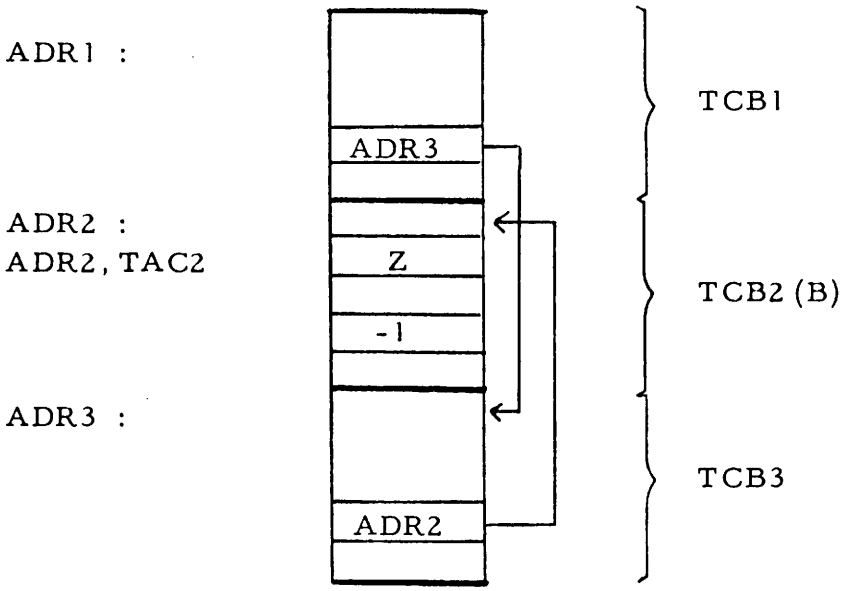
```
                ;*****************************************************
                ;
                ;       LINK A TCB INTO READY OR SUSPEND CHAIN
                ;
                ; AC2=TCB ADDRESS
                ;       TCB WILL BE LINKED IN AT END OF READY TCB'S
                ;      'OF THE SAME OR HIGHER PRIORITY, UNLESS BIT 1
                ;       OF TPRST IS SET, IN WHICH CASE IT WILL GO INTO
                ;       THE SUSPEND CHAIN.  BITS 0 AND 2 OF TPRST ARE
                ;       CLEARED.
                ;
                ;*****************************************************

00170'055006 .TLNK:   STA 3 TSYS 2     ; SAVE RETURN ADDRESS
00171'050006          STA 2 RLOC       ; SAVE TCB ADDRESS
00172'060277          INTDS
00173'040433          STA 0 TLAC0      ; SAVE AC0 AND AC1
00174'044433          STA 1 TLAC1
00175'021005          LDA 0 TPRST 2    ; PICKUP TCB PRIORITY
00176'034432          LDA 3 TLMSK      ; GET MASK TO CLEAR FLAGS
00177'163400          AND 3 0          ; RETAIN ONLY SUSPEND FLAG, PRIORITY
00200'041005          STA 0 TPRST 2    ; PUT BACK IN TCB
00201'103112          ADDL# 0 0 SZC    ; SUSPEND FLAG SET?
00202'000421          JMP TLSUS        ; YES - PUT AT HEAD OF SUSPEND CHAIN
00203'034426          LDA 3 ACTCB      ; NO - START OF "ACTIVE" CHAIN
00204'031407 TLNXT:   LDA 2 TLNK 3     ; GET NEXT TCB ADDRESS
00205'150015          COM# 2 2 SNR     ; CHECK IF END OF CHAIN
00206'000406          JMP HERE         ; YES--INSERT
00207'025005          LDA 1 TPRST 2    ; GET PRIORITY
00210'106032          ADCZ# 0 1 SZC    ; FOUND SPOT ?
00211'000403          JMP HERE         ; YES--INSERT
00212'155000          MOV 2 3          ; NO--UPDATE QUEUE POINTER
00213'000771          JMP TLNXT        ; AND CONTINUE

00214'145000 HERE:    MOV 2 1          ; SAVE NEXT TCB ADDRESS
00215'030006          LDA 2 RLOC       ; GET NEW TCB ADDRESS
00216'045007          STA 1 TLNK 2     ; SET FORWARD LINK
00217'020407          LDA 0 TLAC0      ; RESTORE AC0
00220'024407          LDA 1 TLAC1      ; RESTORE AC1
00221'051407          STA 2 TLNK 3     ; FWD LINK OF PREVIOUS TASK
00222'003006          JMP @TSYS 2      ; RETURN TO CALLER
                                       ; WITH THE INTERRUPT DISABLED

00223'034407 TLSUS:   LDA 3 SUSQ       ; DUMMY HEAD TCB IS PREVIOUS TCB
00224'031407          LDA 2 TLNK 3     ; FIRST ACTUAL TCB IS NEXT TCB
00225'000767          JMP HERE         ; LINK CUSTOMER BETWEEN THEM

00226'000000 TLAC0:   0
00227'000000 TLAC1:   0
00230'040377 TLMSK:   40377            ;SUSPEND BIT, PRIORITY MASK
                                       ;BASE ADDRESSES OF DUMMY TCB'S:
00231'000406 ACTCB:   UST+USTAC-TLNK   ;;   -> READY CHAIN HEADER
00232'000413 SUSQ:    UST+USTSQ-TLNK   ;;   -> SUSPEND CHAIN HEADER

                      .END
```

Fig. 19.7e

```
ACTCB   000231'
ATCB    000117'
CKLP    000030'
CTASK   000011'
F4CMN   000000U
F40I2   000000U
F40IO   000000U
F4SVL   000000U
F4SW    000000
FRTLP   000000U
FRTX    000000U
GFCEN   000036'
HERE    000214'
ITCB    000000U
KILL    000102'
NTASK   000065'
PRMSK   000121'
QRTN    000167'
RSHED   000122'
SETUP   000125'
START   000147'
SUSQ    000232'
SVVAR   000000U
SYSTE   000000'
SYSTM   000010'X
TCBPL   000064'
TCBSZ   000066'
TFREE   000120'
TIDER   000067'
TLAC0   000226'
TLAC1   000227'
TLENN   000000U
TLMSK   000230'
TLNXT   000204'
TLSUS   000223'
TMAN2   000122'
TNDSP   000000U
TPRI    000073'
TSAVX   000166'
TSKER   000071'
TSP     000000U
TVR     000000U
VPENT   000000U
VRSTR   000000U
.ITCB   000000U
.KILL   006001-
.NTSK   000065'X
.PRI    006000-
.QBLK   000150'
.SYSE   000001#X
.TASK   006002-
.TCBP   000064'X
.TLNK   000170'
.TMAX   000112'
```

Fig. 19.7f

USTXQ : ADR 1



Z :

Kæde af ventende processer

Fig. 20.1

20. Synkronisering i RTOS.

I afsnit 11 er formålet med synkronisering og de principielle funktioner i RTOS beskrevet.

Synkroniseringen udføres som beskrevet v.hj.a. proceskaldene "send message" og "wait message".

Det er tidligere omtalt, at RTOS indeholder dels en kæde af procesbeskrivelser for kørbare processer organiseret efter prioritet og dels en kæde af procesbeskrivelser for udsatte processer.

For at administrere processer, der er ventende efter at have udsendt et synkroniseringskald, findes også en kæde af procesbeskrivelser for ventende processer, delay-kæden.

Adressen på den første procesbeskrivelse i "delay-kæden" findes i brugerstatus tabellen UST i elementet USTXQ (se fig. 17.1). Hvis en proces B med tilhørende procesbeskrivelse TCB2 har udsendt kaldet "wait message" til celle(Z), vil adresse Z findes i TCB2 (i elementet TAC2).

Når en proces A udsender kaldet "send message" til celle (Z), kan delay-kæden undersøges, og de processer, hvis procesbeskrivelser indeholder adressen Z i TAC2, vil blive overført til køen af kørbare processer. Dette er illustreret i fig. 20.1.

Proceskaldene "send og wait message" beskrives nærmere i det følgende, idet den verbale beskrivelse svarer til den

```
        procedure send message(adr, mess, mode, result)

        begin
          result:= 0
          save current proces(ctcb);   [.QBLK]
          if (adr) < > 0 then
            begin
              result:= -1;
              goto error;
            end
          if mess = 0 then
            begin
              result:= result -2;
              goto error;
            end
          search_delay_queue(adr, match);
          if ¬ match then
            begin
              (adr):= mess;
              if mode = ↑ then "delay caller";
            end else
            while match do
            begin
              "transmit message";
              "ready reciever";
              search_delay_queue(adr, match);
            end
error:    goto rescheduling;

        end
```

Fig. 20.2

formelle programbeskrivelse i fig. 20.2 og fig. 20.3.

Proceskaldet "send message" har parametrene adr, der udpeger den lagercelle , hvori parametren message afleveres, hvis ingen anden proces i delay-køen venter på en besked i adr. Parametren mode angiver, hvorvidt den kaldene proces skal udsættes eller ej efter proceskaldet er udført. Parametren result angiver, hvorvidt de aktuelle parametre var fejlbehæftede.

Når proceskaldet udføres, sker der følgende:

Resultatparametren nulstilles.

Procesbeskrivelsen for den kaldende proces udfyldes med processens status.

Hvis indholdet af den udpegede lagercelle ikke er 0, sættes resultatet til -1, og en fejlrutine aktiveres.

Hvis den overførte besked er 0, sættes resultatet til -2, og en fejlrutine aktiveres. Fejlrutinen er beskrevet senere.

Derefter undersøges det v. hj. a. proceduren "search_delay_queue", hvorvidt der findes en modtage-proces, der venter på en besked i adr.

Hvis der ingen proces findes (hvis match er false), overføres message til adr. Hvis mode er 1, placeres den kaldende proces, der udsendte kaldet, i delay-kæden ("delay caller").

Hvis der findes en ventende proces overføres message, og den ventende proces overføres til kæden af kørbare processer ("ready reciever"), og det undersøges, hvorvidt flere processer venter på besked i adr. "Ready reciever" er den tidligere omtalte procedure "link proces" [. TLNK].

Når samtlige ventende processer er undersøgt, udføres en omprioritering, hvori den højst prioriterede proces udvælges til kørsel.

Fejlrutinen består blot af en formel omprioritering, idet resultat-parametren angiver, hvorvidt de øvrige parametre var korrekte.

"Send message" er beskrevet i fig. 20.2.

```
procedure wait message(adr, mess);

begin
  save current proces(ctcb)        [.QBLK]
  if (adr) < > 0 then
    begin
      mess:= (adr); (adr):= 0;
      search delay queue(adr, match)
      if match then "ready sender"
    end
  else "delay caller";
  goto rescheduling
end;
```

Fig. 20.3

Proceskaldet "wait message" har parametrene adr, der udpeger en lagercelle, hvorfra en besked hentes, og message, der modtager indholdet af adr.

Når proceskaldet udføres, sker der følgende:

Procesbeskrivelsen for den kaldende proces udfyldes med processens status.

Hvis indholdet af adr ikke er 0, overføres dette til message, og indholdet af adr nulstilles. Delay-køen undersøges, og hvis en sender proces venter på, at en besked bliver modtaget, aktiveres sender processen ("ready sender").

Hvis indholdet af adr er 0, overføres kalde processen til kæden af ventende processer ("delay caller").

Der sluttes af med en omprioritering.

"Wait message" er beskrevet i fig. 20.3.

21. Interruptsystemet i RTOS.

I afsnit 15 er interruptserviceprogrammet INTD principielle funktion beskrevet. I afsnit 22 vil device kontrol tabellen (DCT) blive nærmere omtalt. DCT'en kan betragtes som datastrukturen for den proces (i.e. interruptserviceprogrammet), der aktiveres, når et interrupt indtræffer.

Interruptserviceprogrammet INTD aktiverer det materielle interruptsystem efter initialiseringsdelen, således at nye interrupts kan betjenes. Før interruptserviceprogrammet INTD afsluttes, deaktiveres interruptsystemet igen. Dele af INTD udføres altså som udelelige operationer.

INTD kan betragtes som den proces i RTOS, der har den højeste prioritet. Følgelig kan de interne processer ikke konkurrere med INTD, og det eneste, der kan afbryde INTD, er et nyt materielt interrupt fra en enhed, der har en højere materiel prioritet end den enhed, hvis interrupt er under behandling. Når en intern proces anmoder om udførelsen af en I/O-operation, sker dette ved hjælp af et systemkald, der specificerer hvilke data, der skal transporteres og hvor hen (til hvilken ydre enhed) data skal transporteres. I/O-rutinerne i RTOS overtager derefter udførelsen af I/O-operationen, og den interne proces kan da fortsætte, indtil den får brug for flere data fra en ydre enhed (ved en input-operation) eller skal aflevere flere data til en ydre enhed (output-operation). Hvis den fore-

Beskrivelse af Interrupt Dispatch Program.

```
;intd, program til afvikling af interrupts
itbl:   array(0..77) of words; itbl indeholder adresser på device
                               kontrol tabellen ordnet efter enhe-
                               dens kode.
dct:    array(1..n,0..27) of words; device kontrol tabellens ind-
                               hold er beskrevet i litt. 3.
save:   array(1..n,0..6) of words ; save anvendes til at opbevare
                               program status ved interrupt.
                               se litt. 3 s. 1-5.

        begin
intp:     n:= devicekode
intd:     if itbl(n) = -1 goto nodev
          ; enheden er ikke defineret
          save:= dct(n,0)
          save_state(save)
          ; gem ac's, carry, .cmsk. rloc og oldpc i
          ; dct'en status-område.
          if .cmsk = 0 then bdct:= itbl(n)
          ; gem dctadr for den først afbrydende enhed
          ; cmsk:= .cmsk .OR. dct(n,1) ; ny maske
          mem(0):= halt        ; halt i celle(0)
          maskout(-cmsk)       ; udsend ny maske
          interrupt_enable     ; aktiver interrupt
          interrupt_service(dct(n,3))
          ; interrupt service rutinens adresse findes
          ; i dct'en.
disn:       restore_state(save)     ; retabler status for afbrudt
                                      program
            maskout (.cmsk)         ; gl. maske
            interrupt_disable       ; deaktiver interrupt
            if .cmsk = 0 then
              begin
                if .rque = 0 goto disno
                ; omprioritering kan være nødvendig
                create procesdescription (ctcb)
                ; den afbrudte proces' tilstand gemmes
                ; i procesbeskrivelsen.
```

[Fig. 19.6]

gående I/O-operation ikke er afsluttet på dette tidspunkt (hvor den interne proces skal aktivere en ny I/O-operation), vil den interne proces blive udsat, indtil den foregående I/O-operation er afsluttet. Herved opnås, at interne processer kan forløbe parallelt med egne I/O-operationer, hvorved centralenheden kan udnyttets bedre. Heraf følger, at når en I/O-operation afsluttes, dvs. når en buffer bliver tom (eller fuld), skal I/O-rutinerne kunne aktivere den interne proces, der eventuelt venter på afslutningen af I/O-operationen. Der findes til dette formål en reference (pegepind) i DCT'en for en arbejdende ydre enhed til procesbeskrivelsen for den interne proces, der kommunikerer med enheden. Da to interne processer ikke samtidigt kan anvende en og samme ydre enhed, må I/O-rutinerne kunne administrere en kø af henvendelser til en ydre enhed og kunne afvikle disse i ankomstrækkefølge.

I fig. 21.1 findes en formel beskrivelse af INTD, der ligger til grund for det i afsnit 15 beskrevne forløb.

I fig. 21.2 er forløbet af INTD illustreret for forskellige betingelser, idet første interrupt antages at komme fra PTP og det andet fra LPT.

1. Hvis det element i interrupt branch tabellen, der udpeges af den afbrydende enhedskæde, er -1, er enheden udefineret (ved systemgenereringen) og der retur-

```
            .rque:= 0     ;
            .sys := 0     ; angiv system mode
              goto rescheduling ; omprioritering
          end else
          begin
            interrupt_enable
            goto oldpc              ; fortsæt afbrudt program
          end

nodev:    clear_device
          mem(0):= halt
          interrupt_enable
          goto oldpc                ; fortsæt afbrudt program
        end
```

Fig. 21.1

neres til det afbrudte program.

2. Der forekommer interrupt fra LPT efter den egentlige interruptservicerutine COSER (common output service) er påbegyndt for PTP.

   Det ses, at INTD startes forfra, og da både PTP og LPT anvender COSER til behandling af output, aktiveres COSER igen. Denne rutine skal derfor være reentrant.

   Under afvikling af det andet interrupt ses det, at der ikke udføres omprioritering, idet dette sker, når det første interrupt afvikles.

   Omprioritering kan udføres, idet afslutning af en I/O-transport kan medføre, at en udsat proces skal bringes i kørbar tilstand, og den kan dermed konkurrere med den afbrudte proces om adgang til cpu-tid. En global variabel i RTOS (.RQUE) anvendes til at indikere, at der skal foretages en omprioritering.

   Fig. 21.2 angiver desuden interruptsystemets tilstand under programafvikling.

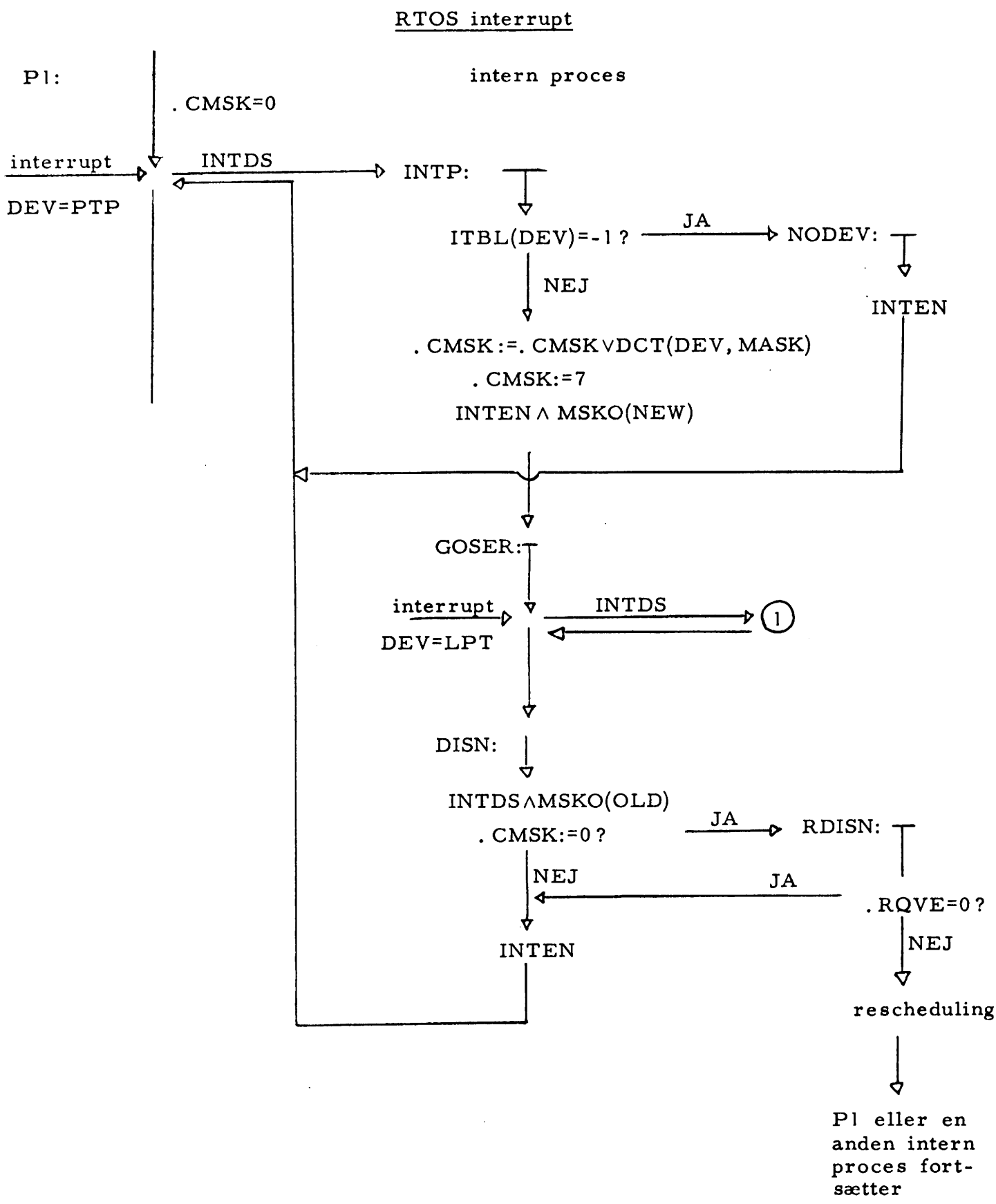

RTOS interrupt
P1:
intern proces
. CMSK=0
interrupt
INTDS
INTP:
DEV=PTP
ITBL(DEV)=-1 ?
JA
NODEV:
NEJ
INTEN
. CMSK:=. CMSK∨DCT(DEV, MASK)
. CMSK:=7
INTEN ∧ MSKO(NEW)
GOSER:
interrupt
INTDS
1
DEV=LPT
DISN:
INTDS∧MSKO(OLD)
. CMSK:=0 ?
JA
RDISN:
NEJ
JA
. RQVE=0 ?
INTEN
NEJ
rescheduling
P1 eller en anden intern proces fortsætter

Fig. 21.2A

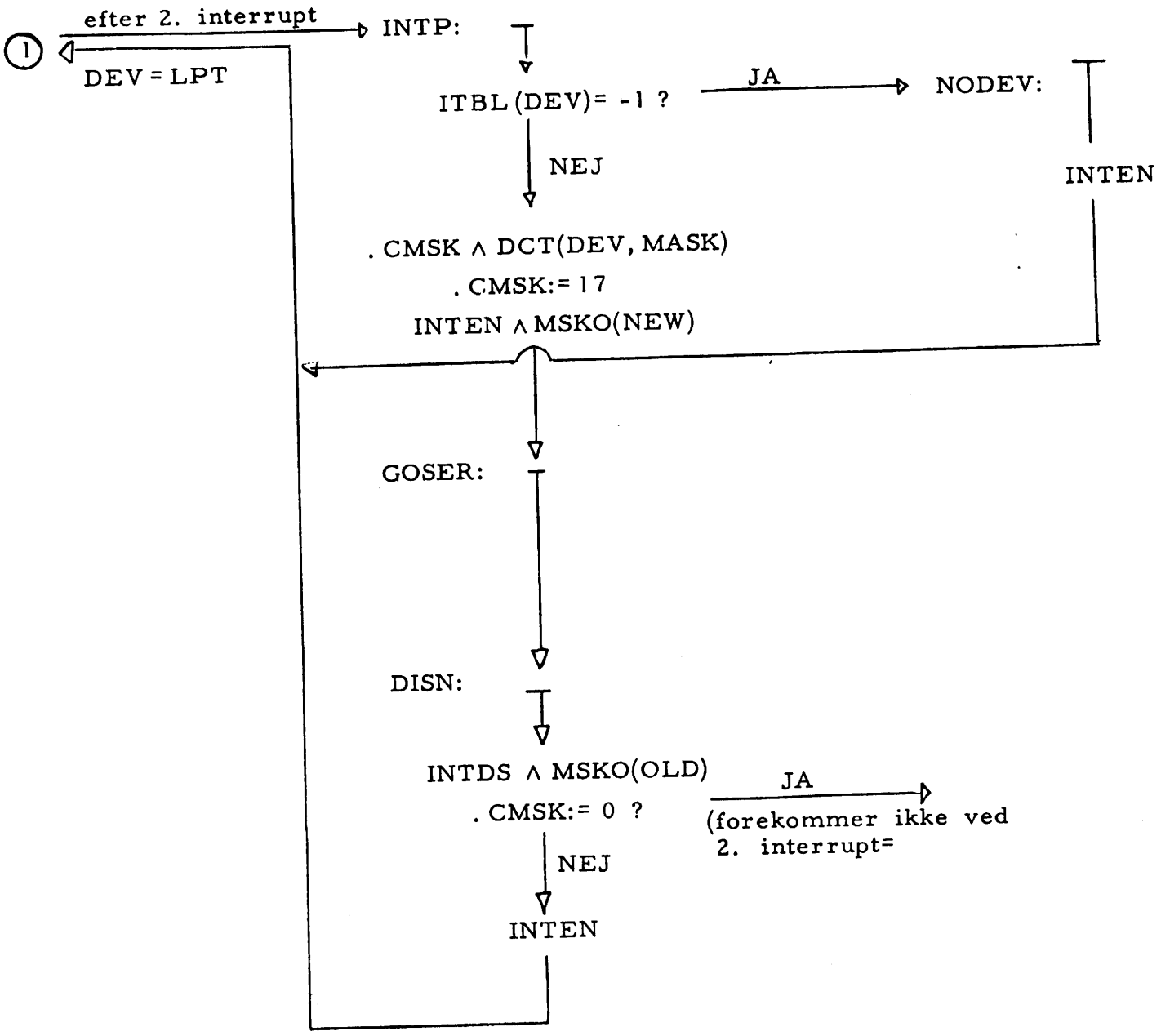

efter 2. interrupt
1
DEV = LPT
INTP:
ITBL(DEV)= -1 ?
JA
NODEV:
INTEN
NEJ
. CMSK ∧ DCT(DEV, MASK)
. CMSK:= 17
INTEN ∧ MSKO(NEW)
GOSER:
DISN:
INTDS ∧ MSKO(OLD)
. CMSK:= 0 ?
JA
(forekommer ikke ved
2. interrupt=
NEJ
INTEN

Fig. 21.2B