DIKUrevy 1986

# Kernesketch

skrevet af JC, hvem det så end er

Status: Færdig

($n$ minutter)

## Roller:

| | |
|---|---|
| **L** (Vilmar) | Lærer som leverer al talen |
| **P1** (Mia) | En CPU (som i senere versioner er en proces) |
| **P2** (Lotte) | En anden proces |
| **SE** (Karen) | Syngeenhed |
| **NE** (Lise) | Nummerproducerende enhed |
| **Tom** (Karoline) | Semafor |
| **Fuld** (Inger) | Anden semafor |

---

## Rekvisitter:

**To store vækkeure** ()
**Et passende antal eksemplarer af "lystige viser"** ()
**To bakker, hvoraf den ene skal ødelægges** ()
**Overhead eller flip-over** ()
**Sandsynligvis div. skilte og kasser** ()

---

*En dat1forelæsning om kerner og multiprogrammering.*

*Syngeenhedens funktion: SE får en opslået "lystige viser" fra cpu, synger et vers og dropper bogen på gulvet.*

*Nummerproducerende enheds funktion: Leverer cpu en lap pair med et nummer på.*

*Semaforoperationer:*

**Vent** *Vil have en "lystige viser". Hvis der ikke er nogen sover processen, dvs. at semaforen "grabber" processen.*

**Signaler** *Aflever en "lystige viser". Hvis der er en proces, der sover, vækkes den (vha vækkeuret) og får overleveret bogen.*

**L** : Ja, godaften og velkommen, jeg ser at dormitoriet er godt fyldt, men augusteksamen nærmer sig jo også. Jeg har i samråd med instruktorerne besluttet at anvende denne ekstraordinære lejlighed til at forelæse

lidt om multiprogrammering. både rapportopgave– og eksamensbesvarelserne viser også, at det er tiltrængt.

Se, emnet for aftenens forelæsning er **samtidig afvikling af parallelle processer.** Hvordan kan det nu lade sig gøre ? JO. I moderne datamater kan de ydre enheder, f.eks. skrivere eller læsere arbejde uafhængigt af og samtidig med centralenheden – den såkaldte "cpu" eller "værket" som det også kaldes i fagkredse. Hver ydre enhed indeholder en lille styreenhed, som styrer enheden, deraf navnet. Ser vi på en typisk ydre enhed – en sangenhed – udfører styreenheden en algoritme noget a la:

fig.1:
```
Repeat
    Klar := TRUE;
    Repeat Until Bog_Modtages;
    Klar := FALSE;
    Syng_Sang;
Forever;
```

**L** : Lad mig demonstrere.

*SE ind, L demonstrerer.*

**L** : En anden typisk enhed – en sidenummerproducerende enhed – vil f.eks. udføre:

fig.2:
```
Repeat
    Klar := TRUE;
    Repeat Until Der_bedes_om_et_nummer;
    Klar := FALSE;
    Aflever_nummer;
Forever;
```

**L** : Bemærk at enheden har uendelig mange numre til sin rådighed. Det vil jeg dog ikke demonstrere.

Hvis centralenheden skal bruge f.eks. syngeenheden, må der finde en vis synkronisering sted, centralenheden må nemlig vente på, at syngeenheden er klar. Dette kan f.eks. foregå som:

fig.3:
```
Repeat
    Repeat Until Klar;
    Aflever_opslået_bog;
Forever;
```

**L** : Vi ser her, at variablen klar anvendes af begge enheder; Centralenheden aflæser og venter, og styreenheden sætter klar når den er klar. Klart, ikke ? Tilsvarende bruges bogen til at indikere, at centralenheden er klar.

Hvis vi kalder centralenhedens lille algoritme for "syng" og dens venten på en sidenummerproducerende enhed for "hent_nummer", har

vi da følgende lille algoritme:

*NE og P1 ind*

fig.4:
```
Repeat
     Hent_nummer;              (* fra NE til P1      *)
     slå_op_i_bog;             (* P1 slår op              *)
     Syng;                     (* SE får bog, synger *)
Until ikke_flere_numre;
```

**L** : Bemærk slutbetingelsen, vi har jo ikke hele dagen.

*Demonstration. Først langsomt, så hurtigere*

**L** : Det kører jo helt fint og uproblematisk, men vi bemærker at centralenheden **venter** det meste af tiden, idet de ydre enheder er mange gange langsommere. Derfor har man fundet på, at centralenheden jo bare kan lave noget andet i den tid, den ellers skulle have ventet. Det fører os over til kernen i stoffet, nemlig parellelle processer. Vi reformulerer vores løsning fra før, men denne gang som to processer, som vi forestiller os kører samtidigt – virtuelt parallelt naturligvis.

fig.5:
```
Repeat  (* P1 *)                Repeat   (* P2 *)
     Hent_nummer;                 While p1_har_bakke do;
     While p2_har_bakke do;       p2_har_bakke := TRUE
     p1_har_bakke := TRUE         tag_bakke;
     tag_bakke;                   tag_bogen_i_bakken;
     slå_op_i_bog;                aflever_bakken;
     læg_bog_i_bakken;            p2_har_bakke := FALSE
     aflever_bakken;              syng;
     p1_har_bakken := FALSE
Until ikke_flere_numre;              Until ikke flere numre;
```

**L** : Vi ser her, at de to processer kommunikerer ved hjælp af en fælles variabel kaldet en bakke, som kun een af dem må have adgang til ad gangen.

*(prøver, det kører og cpu'en udnyttes meget bedre)*

**L** : Glimrende, strålende. Det er jo ganske tydeligt, at ikke alene udnyttes centralenheden bedre, de ydre enheder kører også meget bedre.

Løsningen er imidlertid ikke uproblematisk. Betragter vi – af overskuelighedsgrunde – hver af processerne som selvstænigt kørende, *(P2 ind)* vil vi få følgende situation, hvis de på et tidspunkt "kommer i trit":

*(kører lige hurtigt, slås om bakken, som går midt over)*

**L** : Problemet er, ar den fælles variabel, bakken, ikke opdateres udeleligt, vi så lige hvordan bakken blev delt i 2. For at skire udelelig adgang indfører vi en særlig type variable, kaldet semaforer *(TOM og FULD*

*ind)* med de tilhørende operationer "vent" og "signaler".
*(demonstrerer i følge beskrivelsen af operationerne)*

**L** : Kalder vi denne semafor "tom" og denne for "fuld", kan vi opskrive vores løsning således, at "fuld" har nul "lystige viser" og "tom" har een "lystige viser". Vi ser her, at semaforerne ikke alene styrer tilgangen til bakken, de overflødiggør den også.

fig.6:

```
Repeat                          Repeat
     Hent_nummer;                 vent (fuld);
     vent (tom);                  syng;
     slå_op_i_bog;                signaler (tom);
     signaler (fuld);
Until ikke_flere_numre;         Forever;
```

**L** : Og mere når vi desværre ikke idag, men næste gang vil jeg komme ind på begrebet "kerner". Tak for i dag.