3. Proceskoordinering

CS

1. Kritisk region (tilgår delt data). Flere processer.

Semaphorer

1. Semaphorer (binær / tælle)
   a. V(s): Lægger 1 til s.
   b. P(s) = Trækker 1 fra s, hvis s er større end 0.
   c. V(s) og P(s) resulterer i Petersons, da de sikrer mutual exclusion.

Test-and-set, spin locks, interrupts

1. Test and set: Det gælder om at kunne lave test and set som et atomart skridt. TS(R,X) sætter R til X, og sætter derefter X = 0. Når R er 1 er der adgang, og hvis R er 0 er der ikke adgang til resursen.
   a. Binær semaphor (spin locks – busy wait):

```
Pb(sb): do TS(R,sb) while (!R); /* wait loop */
Vb(sb): sb = 1;
```

   Ikke særlig effektivt, da CPU ofte skal tjekke R.
2. Tællesemafor kan impl. Vha 2 binære semaforer (problem med prioriteter – kan risikere at den ene låser mutex_s, og derefter skiftes over til den anden - deadlock)
3. Inhibit interrupts. Down(s) kan komme til at stå og vente på at delay_s bliver frigivet.
4. Droppe delay_s, og bloker processen selv, og kald scheduleren til at finde næste proces (context switch). Vi busy-waiter kun når en anden proces er i P eller V.

s	>= 0 værdien af tællesemafor, < 0 antal ventende processer
mutex_s	binær semafor til beskyttelse af tælleren s
delay_s	binær semafor til at få processer (tråde) til at vente

Initialisering: `mutex_s = 1; delay_s = 0;`

```
down(s,mutex_s,delay_s) {
  lock(mutex_s);
  s = s - 1;
  if (s < 0) {
       unlock(mutex_s);
       lock(delay_s);
  }
  unlock(mutex_s)
}
```

```
up(s,mutex_s,delay_s) {
  lock(mutex_s);
  s = s + 1;
  if (s <= 0)
    unlock(delay_s);
  else
    unlock(mutex_s);
}
```

```
down(s) {
  Inhibit_Interrupt;
  lock(mutex_s);
  s = s - 1;
  if (s < 0) {
       Block(self, Ls);
       unlock(mutex_s);
       Enable_Interrupt;
       Scheduler();
  }
  else {
       unlock(mutex_s);
       Enable_Interrupt;
  }
}
```

```
up(s) {
  Inhibit_Interrupt;
  lock(mutex_s);
  s = s + 1;
  if (s <= 0) {
       Unblock(q, Ls);
       unlock(mutex_s);
       Enable_Interrupt;
       Scheduler();
  }
  else {
       unlock(mutex_s);
       Enable_Interrupt;
  }
}
```

```
down(s) {
  Inhibit_Interrupts;
  lock(mutex_s);
  s = s - 1;
  if (s < 0) {
       unlock(mutex_s);
       Ena  In  upts;
       lock(delay_s)
  }
  unlock(mut  s);
  Enable_Interrupts;
}
```

```
up(s) {
  Inhibit_Interrupts;
  lock(mutex_s);
  s = s + 1;
  if (s <= 0)
       unlock(delay_s);
  else
       unlock(mutex_s);
  Enable_Interrupts;
}
```

busy waiting ønskes reduceret