6.6 Dining philosophers problem

Sunday, April 19, 2020 1:50 AM

Dette problemet illustrerer konseptene om vranglåser og *starvation*. I tillegg er problemet representativt for problemer med koordinering av delte ressurser som kan oppstå når en applikasjon executer tråder i samtidighet. Problemet går ut på at det er 5 filosofer som kun spiser spaghetti til middag – det er totalt 5 gafler på bordet, men hver filosof trenger 2 gafler når hun spiser. Figuren under illustrerer problemet:

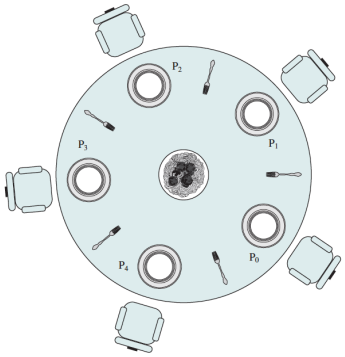

Figure 6.11 Dining Arrangement for Philosophers

Løsning med semaforer

I denne løsningen plukker alle filosofene først opp gaffelen på venstre side, og deretter gaffelen på høyre side. Problemet som kan oppstå her er at alle filosofene er sultne samtidig, og dermed plukker opp sin venstre gaffel samtidig, men så oppdager at høyre gaffel ikke er der. Dermed er det oppstått en vranglås. Det er flere måter å tilpasse denne løsningen på, og i koden under er det implementert en butler som håndhever at kun 4 filosofer får lov til å entre bordet om gangen. Dette gjør at hvert fall én filosof alltid får spist (har tilgang til to gafler). Denne løsningen er fri for vranglåser og starvation.

```
/* program diningphilosophers */
semaphore fork[5] = {1};
semaphore room = {4};
int i;
void philosopher (int  i)
{
     while (true) {
          think();
          wait (room);
          wait (fork[i]);
          wait (fork [(i+1) mod 5]);
          eat();
          signal (fork [(i+1) mod 5]);
          signal (fork[i]);
          signal (room);
     }
}
void main()
{
     parbegin  (philosopher (0), philosopher (1),
          philosopher (2), philosopher (3),
          philosopher (4));
 }
```

Figure 6.13 A Second Solution to the Dining Philosophers Problem

Løsning med monitor

Her er det definert en vektor med fem tilstander, der det er én betingelse per gaffel. Disse betingelsesvariablene lar filosofene vente på at gaffelen blir ledig. Det er også en boolsk vektor som holder på tilgjengelighetsstatusen til en gaffel (True betyr at gaffelen er ledig). Monitoren består av to prosedyrer. get_fork(*int PID*)-metoden brukes av filosofen til å gripe venstre/høyre gaffel. Dersom en av gaflene er opptatt vil filosofen plasseres i en kø på den passende betingelsesvariabelen. Dermed kan en annen filosof(prosess) entre monitoren og potensielt benytte prosedyrer. release_forks(*int PID*)-metoden gjør gaflene tilgjengelig igjen. Denne løsningen unngår vranglås ved at det kun er en prosess av gangen som kan entre monitoren (gjensidig utelukkelse). Dette betyr at den første filosofen som entrer monitoren er garantert å få to gafler før en annen filosof forsøker å gripe gaflene. Løsningnen er basically førstemann-til-mølla... Se under.

```
monitor dining_controller;
cond ForkReady[5];    /* condition variable for synchronization */
boolean fork[5] = {true}; /* availability status of each fork */

void get_forks(int pid)    /* pid is the philosopher id number */
{
    int left = pid;
    int right = (++pid) % 5;
```

```
    /*grant the left fork*/
    if (!fork[left])
       cwait(ForkReady[left]); /* queue on condition variable */
    fork(left) = false;
    /*grant the right fork*/
    if (!fork[right])
       cwait(ForkReady[right]);/* queue on condition variable */
    fork[right] = false:
    }
void release_forks(int  pid)
{
    int left = pid;
    int right = (++pid) % 5;
    /*release the left fork*/
    if (empty(ForkReady[left]) /*no one is waiting for this fork */
       fork[left] = true;
    else              /* awaken a process waiting on this fork */
       csignal(ForkReady[left]);
    /*release the right fork*/
    if (empty(ForkReady[right])/*no one is waiting for this fork */
       fork[right] = true;
    else              /* awaken a process waiting on this fork */
       csignal(ForkReady[right]);
}
```

```
void philosopher[k=0 to 4]    /* the five philosopher clients */
{
    while  (true) {
      <think>;
     get_forks(k);    /* client requests two forks via monitor */
      <eat spaghetti>;
      release_forks(k); /* client releases forks via the monitor */
    }
}
```

Figure 6.14 **A Solution to the Dining Philosophers Problem Using a Monitor**