5.7 Readers/writers problem

Friday, April 17, 2020 4:01 PM

Dette er en annet vanlig problem som er en utfordring designmessig. Problemet går utpå delt dataområde mellom flere prosesser. Dataområdet kan for eksempel være en fil eller en blokk fra hovedminnet. Det er ulike typer prosesser som kun leser fra dataområdet (readers) og prosesser som kun skriver til dataområdet (writers). Betingelsene som må overholdes er:

1. Ethvert antall lesere må kunne lese fra fil samtidig.
2. Kun én skriver kan skrive til filen om gangen.
3. Imens en prosess skriver til filen, så kan ingen andre leser-prosesser lese fra filen.

Dette betyr at reader-prosesser ikke trenger å ekskludere andre prosesser, mens writer-prosesser må ekskludere alle andre prosesser. Begge de to løsningene under på dette problemet tar i bruk semaforer.

Leser-prioritet (readers have priority)

```
/* program readersandwriters */
int readcount;
semaphore x = 1,wsem = 1;
void reader()
{
    while (true){
     semWait (x);
     readcount++;
     if(readcount == 1)
         semWait (wsem);
     semSignal (x);
     READUNIT();
     semWait (x);
     readcount--;
     if(readcount == 0)
         semSignal (wsem);
     semSignal (x);
     }
}
void writer()
{
      while (true){
       semWait (wsem);
       WRITEUNIT();
       semSignal (wsem);
      }
}
void  main()
{
     readcount = 0;
     parbegin (reader,writer);
}
```

Figure 5.25 **A Solution to the Readers/Writers Problem Using Semaphore: Readers Have Priority**

Merk at denne løsningen ikke endrer seg dersom det er flere lesere og skrivere. Semaforen *wsem* over håndhever gjensidig utelukkelse. Så lenge en skriver aksesserer det delte dataområdet vil ikke noen andre skrivere eller lesere kunne aksessere dataområdet. For å muliggjøre flere lesere må den første leseren som skal lese vente på _wsem. Når det finnes en leser vil ikke nye lesere måtte vente. Den globale variabelen *readcount* brukes til å holde oversikt over antall lesere, og semaforen *x* brukes til å sikre at _readcount er oppdatert korrekt.

Skriver-prioritet (writers have priority)

I den ovennevnte løsningen er at problem at writer-prosesser kan bli utsatt for starvation. Dette vil si at en prosess ikke får komme til (og gjerne aldri får tildelt nødvendige ressurser) fordi andre prosesser prioriteres. Denne løsningen garanterer at ingen nye lesere blir tillatt aksess til dataområdet så lenge det finnes en leser som har fått aksess. For skrivere blir det lagt til følgende semaforer og variabler I implementasjon:

- En semafor *rsem* som forhindrer alle lesere aksess hvis det er én leser som har fått aksess
- En variabel *writecount* som kontrollerer innstillingene til _rsem
- En semafor *y* som kontrollerer _writecount

I tillegg legges det til en semafor *z* for lesere ettersom det ikke kan tillates å bygges en lang kø på _rsem. Det er kun lov med én leser i køen til _rsem, mens de andre leserne blir lagt i køen på semaforen z. Se løsning under.

```
/* program readersandwriters */
int readcount,writecount; semaphore x = 1, y = 1, z = 1, wsem = 1, rsem = 1;
void reader()
{
   while (true){
     semWait (z);
         semWait (rsem);
             semWait (x);
                 readcount++;
```

```
                if (readcount == 1)
                    semWait (wsem);
                semSignal (x);
            semSignal (rsem);
        semSignal (z);
        READUNIT();
        semWait (x);
            readcount--;
            if (readcount == 0) semSignal (wsem);
        semSignal (x);
    }
}
void writer ()
{
    while (true){
      semWait (y);
          writecount++;
          if (writecount == 1)
              semWait (rsem);
      semSignal (y);
      semWait (wsem);
      WRITEUNIT();
      semSignal (wsem);
      semWait (y);
          writecount--;
          if (writecount == 0) semSignal (rsem);
      semSignal (y);
    }
}
void main()
{
    readcount = writecount = 0;
    parbegin (reader, writer);
}
```

Figure 5.26 A Solution to the Readers/Writers Problem Using Semaphore: Writers Have Priority

Meldingsutveksling (writers have priority w/ message passing)

I denne løsningen har igjen skrivere prioritet. Denne løsningen implementeres med en kontroller-prosess som har aksess til det delte dataområdet. Andre prosesser som ønsker aksess hit må sende en forespørsel-melding til kontrolleren, og får aksess ved "OK" (ACK) og prosessene indikerer at de er ferdig med en "finished"-melding. Kontrolleren har tre mailbokser, én for hver type melding den kan motta. Prioriteten hos skrivere ligger i at kontrolleren tjenestegjør skrive-forespørsler før lese-forespørsler.

Variabelen *count* brukes til å håndheve gjensidig utelukkelse. Denne variabelen er initialisert med en verdi større enn maksimalt antall lesere. Det er tre tilfeller som oppstår ved verdiene til _count:

1. **count > 0:** Det er ingen skrivere venter på aksess, mens lesere kan kanskje være aktive. Tjenestegjør alle "finished"-meldinger for å stryke alle aktive lesere. Så tjenestegjør alle skrive-forespørsler etterfult av lese-forespørsler.
2. **count = 0:** Den eneste utestående forespørselen er en skrive-forespørsel. Tillat skriveren å fortsette og vent på en "finished"-melding.
3. **count < 0:** En skriver har etterspurt en forespørsel og må vente på at alle aktive lesere strykes. Derfor skal kun "finished"-meldinger håndteres.

Løsningen med meldingsutveksling:

```
void reader(int i)
{
    message rmsg;
        while (true) {
               rmsg = i;
            send (readrequest, rmsg);
            receive (mbox[i], rmsg);
            READUNIT ();
            rmsg = i;
            send (finished, rmsg);
        }
}
void writer(int j)
{
    message rmsg;
    while (true){
        rmsg = j;
        send (writerequest, rmsg);
        receive (mbox[j], rmsg);
        WRITEUNIT ();
        rmsg = j;
        send (finished, rmsg);
    }
}
```

```
void controller()
{
    while (true)
    {
        if (count > 0) {
            if (!empty (finished)) {
                receive (finished, msg);
                count++;
            }
            else if (!empty (writerequest)) {
                receive (writerequest, msg);
                writer_id = msg.id;
                count = count - 100;
            }
            else if (!empty (readrequest)) {
                receive (readrequest, msg);
                count--;
                send (msg.id, "OK");
            }
        }
        if (count == 0) {
            send (writer_id, "OK");
            receive (finished, msg);
            count = 100;
        }
        while (count < 0) {
            receive (finished, msg);
            count++;
        }
    }
}
```

Figure 5.27 A Solution to the Readers/Writers Problem Using Message Passing