Herentamen A2: Bedrijfssystemen, 4 maart 2003

Vraag 2 (1.5 punt)

Op een operating systeem met pre-emptive round-robin scheduling draaien 3 processen, die elke uit threads bestaan:

process	threads waaruit dit process bestaat
P_1	T_{11}, T_{12}, T_{13}
P_2	T_{21}, T_{22}
P_3	T_{31}

1. Stel T_{11} draait en blokkeert halverwege zijn quantum, op tijdstip t terwijl z'n quantum pas afloopt of tijdstip $t + \frac{1}{2}q$. Welke threads kunnen er nu aan de beurt komen, als de threads user level threads zijn? En welke als het kernel level threads zijn? Zeg er ook bij op welk tijdstip, t of $t + \frac{1}{2}q$, de volgende thread de CPU krijgt.

2. Stel thread T_{11} draait en z'n quantum loopt af. Welke threads zouden er nu aan de beurt kunnen komen, als de threads user level threads zijn? En welke als het kernel level threads zijn?

Geef een korte motivatie bij je antwoorden.

Vraag 4 (3 punten)

Verschillende processen, die van type `lezer` of `schrijver` kunnen zijn, delen een globale lijst met data. Processen van type `schrijver` produceren data en zetten dit achteraan in een globale lijst, processen van type `lezer` halen data uit het begin van deze globale lijst en verwerken het.

Voor de globale lijst worden twee pointers bijgehouden, `first` en `last`, die naar het eerste en het laatste element in de rij wijzen. Voor elke pointer p in de lijst is `p->inhoud` een of ander data element en `p->next` de pointer naar het volgende element in de lijst. De (pseudo)code voor een `lezer` is

```
while TRUE
  { y = first->inhoud;
    "verwerk data die in de lokale variabele y staat";
    first = first->next;
  }
```

en die voor een `schrijver` is

```
while TRUE
  { "produceer data in een lokale variabele x";
    alloc(new_last);
    /* Er is nu geheugen gealloceerd voor een nieuw element in */
    /* de lijst, en new_last wijst naar dit verse stuk geheugen */
    last->next = new_last;
    last = new_last;
    last->inhoud = x;
  }
```

We maken ons niet druk over het dealloceren van geheugen.

1. Er kan het een en ander misgaan met deze code. Voeg semafoor-operaties toe om er voor te zorgen dat dit niet meer kan gebeuren.

   Zeg duidelijk welke semaforen je gebruikt en wat hun initiële waarden zijn. Je mag enkel semafoor-operaties in de bovenstaande code toevoegen, de volgorde van de statements in de code hierboven mag niet veranderen.

2. Kun je de code van lezers of schrijvers veranderen om meer concurrency mogelijk te maken? Zoja, leg uit hoe.

3. Zowel schrijvers als lezers moeten in iedere slag van hun repetitie totaal r seconden rekenen. Tijdens het produceren van de data is een schrijver gemiddeld b seconden geblokkeerd (bijv. omdat-ie invoer moet doen), en tijdens het verwerken van de data is een lezer gemiddeld b seconden geblokkeerd (bijv. omdat-ie uitvoer moet doen). Je mag aannemen dat b een stuk groter is dan r.

   Stel we meten de tijd t_1 die nodig is voor het verwerken van 100 data-elementen met één lezer en één schrijver die draaien op een machine met één processor, en daarna de tijd t_2 die hiervoor nodig is met één lezer en één schrijver op een machine met twee processoren.

   Geef een zo nauwkeurig mogelijke ondergrens voor t_2. **100 r + 100 b**

   Wat denk je dat het maximale verschil tussen t_1 en t_2 zou kunnen zijn ? **100 r**

   Motiveer je antwoorden!