Tentamen Processen & Concurrency, 15 mei 2007

Geef bij al je antwoorden voldoende motivatie: voor alleen getallen en ja/nee antwoorden krijg je geen punten. Succes!!

Vraag 1 (1 punt)

Wat is het verschil tussen een *suspended* en een *blocked* proces? Heeft het zin om zowel een suspended toestand als een blocked toestand te hebben? Geef vier mogelijke redenen om een proces in suspended toestand te brengen.

Vraag 2 (2.5 punten)

Beschouw het mutual exclusion algoritme, afgebeeld in Figuur 1, dat is bedacht door James Burns. Het algoritme werkt voor `n>0` processen en

```
type flagtype = {down, up};
shared var Flag: array [0..n-1] of flagtype initially {down,..,down};

program Process_i;
  local var j:0..n-1;
begin
  while true do begin
    remainder;                                    {*Remainder region*}
    repeat                                      {*Trying region entry*}
      Flag[i] := down;
      repeat until for all j in {0,..,i-1} Flag[j] = down;
      Flag[i] := up
    until for all j in {0,..,i-1} Flag[j] = down;
    repeat until for all j in {i+1,..,n-1} Flag[j] = down;
    critical                                       {*Critical region*}
    Flag[i] := down;                                   {*Exit region*}
  end
```

Figure 1: Mutual exclusion algoritme van Burns.

maakt gebruik van `n` gedeelde binaire variabelen die initieel allemaal op `down` staan. Voor iedere `i` in het interval `[0..n-1]` voert proces `i` de code uit Figuur 1 uit. Bewijs dat het algoritme mutual exclusion garandeert. Hint: maak gebruik van een bewijs uit het ongerijmde. Neem aan dat er een executie bestaat waarin zowel proces `i` als proces `j` de kritieke sectie binnenkomen. Zonder verlies van algemeenheid mag je aannemen dat proces `i` als eerste zijn vlag op `up` zet. Leidt nu een tegenspraak af.

Vraag 3 (2.5 punten)

Trams in Amsterdam passeren tussen het Leidseplein en het Koningsplein diverse bruggen over kanalen. Op de bruggen zijn er twee sporen, één voor iedere richting, en haltes waar passagiers kunnen in- en uitstappen. In de winkelstraat tussen de bruggen is er slechts ruimte voor een enkel spoor. De situatie wordt schematisch weergegeven in Figuur 2. Ontwerp met behulp

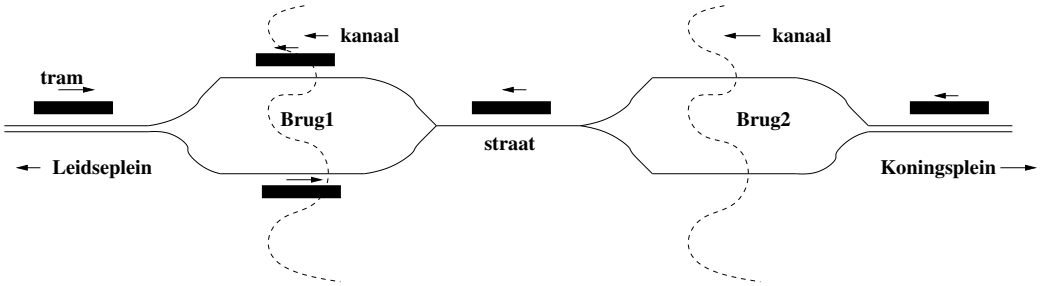

Figure 2: Tramverkeer in de Leidsestraat.

van semaforen een besturing voor dit systeem (waarbij er voor iedere tram een aparte thread is). Let daarbij op de volgende zaken:

- De bruggen zijn zo kort dat er per richting slechts plaats is voor 1 tram.
- Op het enkele spoor tussen de bruggen passen precies 2 trams.
- Trams kunnen niet van richting veranderen.

Je oplossing moet voldoen aan de volgende voorwaarden:

- Afwezigheid van deadlock.
- Afwezigheid van starvation: iedere tram die binnenkomt moet uiteindelijk het systeem verlaten.

Vraag 4 (1.5 punt)

Beschouw een systeem bestaande uit processen $P_1, \ldots, P_n$, die ieder een unieke prioriteit hebben. Schrijf een monitor die drie identieke printers toewijst aan deze processen, en gebruik maakt van de prioriteiten om de volgorde van toewijzing te bepalen.

Vraag 5 (1 punt)

Beschouw een systeem bestaande uit m identieke hulpbronnen (resources) die worden gedeeld door n processen. Processen moeten hulpbronnen één voor één aanvragen en vrijgeven. Toon aan dat er geen deadlock kan optreden indien voldaan wordt aan de volgende twee voorwaarden:

- Ieder proces heeft minimaal 1 en maximaal m hulpbronnen nodig.
- De som van de aantallen hulpbronnen die ieder proces maximaal nodig heeft is kleiner dan $m + n$.

Vraag 6 (1.5 punten)

Een OS gebruikt round robin (RR) scheduling, waarbij processen verschillende quanta kunnen hebben. Initiëel krijgt elk proces een vast quantum q. Als een proces zijn quantum helemaal opmaakt zonder te blokkeren wordt zijn quantum verdubbeld. Als een proces blokkeert wordt zijn quantum weer teruggezet op q.

a) Wat zou een motivatie kunnen zijn om deze variant van RR te gebruiken, ipv. gewone RR waarbij alle processen altijd hetzelfde quantum hebben?

b) Kan er starvation optreden?

We passen het scheduling algoritme aan met prioriteiten: processen met een kleiner quantum krijgen een hogere prioriteit. Dwz. de scheduler kijkt wat het kleinste quantum is dat in de ready-wachtrij voorkomt en kiest het eerste process in de wachtrij dat dit quantum heeft.

c) Wat zou een motivatie kunnen zijn voor deze aanpassing?

d) Kan er starvation optreden?

Motiveer je antwoorden!