Tentamen Concurrency, 24 augustus 2007

Tijdsduur 3 uur. Gesloten boek tentamen.

Voorzie alle in te leveren bladen van je naam, en nummer ze. Schrijf op het eerste blad het aantal ingeleverde bladen. Werk netjes, formuleer scherp en zorgvuldig. Schrijf duidelijk leesbaar.

Opgave 1 (25 %). Beschouw een begrensde stapel voor een systeem van concurrent processen met twee procedures

procedure *push*(x : *Item*) ;
procedure *pop*() : *Item* .

De aanroep *push*(x) plaatst x bovenop de stapel. De aanroep *pop*() haalt het bovenste element van de stapel en levert dit op. Als de stapel leeg is, moet de procedure *pop* wachten tot een ander process iets gepusht heeft. De stapel kan N items bevatten. Als de stapel N items bevat, moet *push* wachten tot een ander process iets gepopt heeft. Implementeer deze concurrente datastructuur met mutexen en conditievariabelen.

Opgave 2 (25 %). Gegeven zijn natuurlijke getallen N en K. Er zijn N processen volgens

process *Member*(*self* := 0 **to** $N-1$)
  var $st := 0$
  do *true* → **syn** ; $st := (st+1)$ **mod** K ; **ann** ; *Com* **od**
end *Member* .

De privévariabele $st.p$ van proces p geeft het *stadium* van p aan, dwz. hoever proces p gevorderd is. De commando's *Com* zijn gegeven. De synchronisatiecommando's **syn** and **ann** moeten zo geïmplementeerd worden, dat alle processen onderling voldoende *afstand houden* volgens

(*) $\quad \forall\, p,q :\ 0 < st.p \leq st.q \ \wedge\ p \neq q \ \Rightarrow\ st.p + 2 \leq st.q$.

Merk op, dat stadium 0 altijd toegestaan is.

Implementeer deze conditie met behulp van gedeelde variabelen, atomiciteitshaakjes en (enkelvoudige of samengestelde) **await** statements. Houd het zo eenvoudig en zo nondeterministisch mogelijk.

Opgave 3 (25 %). We beschouwen een client-server systeem met N clients volgens:

var `request`[0 : $N-1$] : *bool* := ([N] *false*) ;

process *client*(i := 0 **to** $N-1$)
  do *true* →
    NCS ;
    `request`[i] := *true* ;
    ⟨ **await** ¬`request`[i] ⟩
  od end *client* .

Z.O.Z.

Deze clients moeten worden geholpen door K servers volgens

```
process server(p := 0 to K − 1)
   do true  →
      zoek een index i met request[i] ;
      bedien(i) ;
      request[i] := false
   od end server .
```

De commando's *NCS* en *bedien*(i) zijn gegeven en kunnen willekeurig lang duren.

Implementeer de servers en hun synchronisatie met semaforen. Elke request moet door precies één server behandeld worden. Er mag alleen *busy waiting* van servers optreden als alle openstaande requests in behandeling zijn.

Opgave 4 (25 %). Het sliding-window protocol met onbegrensde volgnummers kan in SR beschreven worden volgens:

```
op mess (i: int, v: item)
op ack (i: int)

process Sender
   const a[int] // is given somehow
   var down := 0, j := 0
   do true ->
      in ack(k) ->
         if down < k -> down := k ; j := down fi
      [] else ->
         send mess(j, a[j])
         j := down + (j-down+1) mod W
      ni
   od end Sender

process Receiver
   var b[int] := ([int] undef)
   var comp := 0
   do true ->
      in mess(k,x) ->
         if b[k] = undef ->
            b[k] := x
            if comp = k ->
               do b[comp] != undef -> comp ++ od
               send ack(comp)
            fi
         [] k < comp -> send ack(comp)
         fi
      ni
   od end Receiver
```

(a) Dit is een zg. fault tolerant algoritme. Beschrijf de aannamen over het medium dat de boodschappen van de Sender naar de Receiver kan overbrengen. Welke soorten *faults* mogen optreden?
(b) De safety conditie van het algoritme is

(10) $\quad 0 \leq i < comp \;\; \Rightarrow \;\; \mathtt{b}[i] = \mathtt{a}[i]$.

Bewijs, dat het algoritme onder de aannamen van onderdeel (a) hieraan voldoet.
(c) De voortgangsconditie is, dat *comp* op den duur willekeurig groot wordt. Bewijs, dat het algoritme onder de aannamen van onderdeel (a) hieraan voldoet.