Tentamen Concurrency, 7 februari 2008

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.

Als het tentamen is nagekeken, kun je het komen inzien (en desgewenst bespreken) bij Wim H. Hesselink, Bernoulliborg kamer 374.

Opgave 1 (25 %). Gegeven is een systeem met twee processen

```
      var  active[0: 1]: bool := ([2] false) ;
      var  turn: int := 0 ;

      process Partner (self := 0 to 1)
         do true ->
10:          NCS
11:          active[self] := true
12:          if active[1-self] ->
13:              turn := 1-self
14:              await (turn = self or not active[1-self])
             fi
15:          CS
16:          active[self] := false
         od
      end Partner
```

NCS is de niet-critische sectie; dit commando hoeft niet te eindigen.
CS is de critische sectie, die altijd gegarandeerd eindigt en die alleen onder wederzijdse uitsluiting uitgevoerd mag worden.
(a) Geef een formele specificatie van wederzijdse uitsluiting voor dit systeem.
(b) Bewijs dat dit systeem aan wederzijdse uitsluiting voldoet, volgens je specificatie van onderdeel (a). Geef alle benodigde invarianten en bewijs hun invariantie.

Opgave 2 (30 %). Beschouw een systeem van N processen volgens

```
process Member(self := 0 to N − 1)
   var item : Item
   do true →
      NCS
      swapItems
   od end Member .
```

De noncritical sectie *NCS* is gegeven; het mag de privévariabele *item* wijzigen. Het fragment *swapItems* moet telkens door twee processen samen uitgevoerd worden. Als twee processen dit samen uitvoeren, dient dat tot effect te hebben, dat zij de waarden van hun *items* uitwisselen. Precieser: als processen p en q samen *swapItems* uitvoeren met vooraf $item.p = A$ en $item.q = B$, dan geldt $item.p = B$ na uitvoering van *swapItems* door p, en $item.q = A$ na uitvoering van *swapItems* door q, en de items van de andere processen zijn ongewijzigd.

De voortgangseis is, dat *swapItems* uitgevoerd zal worden indien tenminste twee processen daarvoor klaar staan.

(a) Implementeer *swapItems* met gedeelde variabelen en eventuele privévariabelen, atomiciteitshaakjes en samengestelde **await** statements. Houd het aantal gedeelde variabelen beperkt (bv. geen arrays). Geef aan waarom je oplossing correct is en waarom er bv. geen interferentie door een derde proces kan plaats hebben.

(b) Geef een implementatie van de atomiciteitshaakjes en de **await** statements met binaire semaforen.

Opgave 3 (20 %). Beschouw een systeem van concurrent processen. We maken hiervoor een begrensde stapel met twee procedures

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

(I0) $\quad 0 \leq i < comp \;\Rightarrow\; \mathtt{b}[i] = \mathtt{a}[i]$.

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.