Uitwerking Tentamen Concurrency, 21 november 2000

Opgave 1. Gegeven is een aantal client processen volgens de declaratie

```
const K: int     # K > 0

process Client (self := 1 to K)
   do true ->
      NCS
      AS
   od
```

`NCS` staat als gebruikelijk voor een niet-critische sectie die niet hoeft te eindigen. `AS` staat voor "await service". Er is één proces Server dat als enige de service kan verlenen en wel door het aanroepen van

```
procedure Serve (q: int)
```

De aanroep `Serve(q)` mag alleen gedaan worden als proces `q` **at** `AS` staat. De client `q` mag `AS` pas verlaten als `Serve(q)` is uitgevoerd.

Implementeer `AS` en het proces Server met shared variables. Je mag hierbij gebruik maken van de constructie $\langle$ **await** B $\rangle$ voor boole'se expressies B en van atomaire tellers. Er dient de volgende voortgangrelatie te gelden

q **at** `AS` $\quad o\!\!\rightarrow \quad$ q **at** `NCS` .

Als alle clients **at** `NCS` zijn en blijven, dient de Server binnen een begrensd aantal ronden voor een **await** statement te gaan wachten.

Geef een zo volledig mogelijk stel invarianten om de correctheid van je oplossing te adstrueren.

Oplossing.

```
var
   nr: int := 0 # teller
   wa[1:K] := ([K] false)

process Client (self := 1 to K)
   do true ->
   0:  NCS
   1:  nr ++
   2:  wa[self] := true
   3:  < await not wa[self] >
   od

process Server
   var y: Index
   do true ->
   5:  < await nr != 0 >
   6:  do not wa[y] -> y := 1 + y mod K od
   7:  Serve (y)
   8:  wa[y] := false
   9:  nr --
   od
```

Correctheid van de aanroep van Serve volgt uit de invarianten

(J0) s **at** $\{7,8\}$ $\Rightarrow$ $\texttt{wa}[y]$,
(J1) $\texttt{wa}[q]$ $\Rightarrow$ q **at** 3 .

Dit impliceert tevens dat q `AS` niet verlaat voordat de service is uitgevoerd. De lus 6 eindigt binnen een begrensd aantal ronden wegens de invariant

(J2) s **at** 6 $\Rightarrow$ $(\exists\, q :: q$ **at** 2 $\vee$ $\texttt{wa}[q])$.

Invariantie hiervan volgt uit de invariant

(J3) $\texttt{nr} = (\#\, q :: q$ **at** 2 $\vee$ $\texttt{wa}[q]) + (\#\, s : s$ **at** 9) .

Er geldt dus dat s **at** 6 leidt tot s **at** 7. Hieruit volgt dat $\texttt{wa}[q]$ leidt tot $\neg\, \texttt{wa}[q]$ en tenslotte ook q **at** 3 $o\!\!\rightarrow$ q **at** 0 .

Opgave 2. Een asynchrone broadcast. Gegeven zijn N processen, genummerd van 1 tot N, die met elkaar communiceren dmv asynchrone boodschappen. Elk proces q kan alleen boodschappen sturen naar zijn buren, dwz naar elementen van de lijst $\texttt{buren}[q]$, die geen meervoudige elementen heeft. Er geldt voor alle q en r

q **not in** $\texttt{buren}[q]$,
r **in** $\texttt{buren}[q]$ $\equiv$ q **in** $\texttt{buren}[r]$,
$\texttt{cntburen}[q]$ $=$ *lengte* $(\texttt{buren}[q])$.

Elk proces is met elk ander proces via nul of meer tussenliggende processen verbonden.

Gevraagd wordt het volgende te implementeren. Proces *start* begint met van standaardinvoer een integer te lezen. Het zendt dit getal naar al zijn buren. Elk proces dat een getal ontvangt zendt het door naar zijn andere buren en druk het op den duur af. Zorg dat alle processen het getal precies één keer afdrukken en dat als het getal N keer is afgedrukt, alle processen geëindigd zijn en er geen boodschappen meer in omloop zijn.

Je mag gebruik maken van de globale declaraties

```
type Intlist = ptr rec (value: int; nx: Intlist)
var buren[N]: Intlist
var cntburen[N]: int
```

Oplossing. Elk proces mag pas schrijven en termineren als het alle boodschappen van zijn buren ontvangen heeft. Het is dus het makkelijkst als elk proces het getal naar al zijn buren stuurt en dan afwacht tot het van al zijn buren boodschappen ontvangen heeft. We kunnen daartoe een teller cnt gebruiken. Het startproces moet eerst lezen en dan direct gaan zenden. Alle andere processen moeten pas zenden als zij een getal ontvangen hebben. Na het zenden geldt $\texttt{lis} = \texttt{null}$.

```
op mes[1:N] (getal: int)

procedure Bcast (var p: Intlist; getal: int)
  var buur: int ;
  do p != null ->
     buur := p^.value ; p := p^.nx
     send mes[buur](getal)
  od
end Bcast

process Node (self:= 1 to N)
  var lis := buren[self]
  var cnt := cntburen[self]
```

```
  var getal : int
  var buur: int
  if self = start ->
    read(getal)
    Bcast (lis, getal)
  fi
  do cnt > 0 ->
    in mes[self] (y) ->
      if lis != null ->
        getal := y
        Bcast (lis, getal)
      cnt --
    ni
  od
  write (getal)
end Node
```

Opgave 3. Een cyclische buffer met zenders en één ontvanger, het ontvangen van boodschappen in een shared memory systeem.

Stel, dat een process (*Consumer*) boodschappen moet kunnen ontvangen van een aantal andere processen (Producers). We voorzien Consumer daartoe van een circulaire buffer `buf` met K posities. We gebruiken de operator $\oplus$ voor optelling modulo K.

We gebruiken gedeelde variabelen `nr` voor het aantal bezette posities in de buffer, `free` voor de index waar de eerstvolgende boodschap geplaatst zal worden, `read` voor de index waar de eerstvolgende boodschap gelezen zal worden, en een boole's array `wr` om aan te geven dat een boodschap geplaatst is.

Alle processen hebben een privévariabele *item* van het type boodschap en x of y voor een positie.

many Producers :
  do *true* $\rightarrow$
    produce item ;
    $\langle$ **await** `nr` $\neq K$ **then**
      $x :=$ `free` ; `free` := `free` $\oplus 1$;
      `nr` := `nr` $+ 1$ $\rangle$;
    `buf`$[x]$:= *item* ;
  od .

a single Consumer :
  do *true* $\rightarrow$
    $\langle$ **await** `wr`[`read`] **then**
      $y :=$ `read` ; `read` := `read` $\oplus 1$;
      `wr`$[y]$:= *false* $\rangle$;
    item := `buf`$[y]$;
    $\langle$ `nr` := `nr` $- 1$ $\rangle$;
    consume item
  od .

Opdracht (a) In het bovenstaande programma is één toekenning weggelaten. Voeg deze op de juiste plaats toe.

Opdracht (b) Bepaal geschikte initiële waarden voor de gedeelde variabelen.

Men zou nu de correctheid kunnen bewijzen, maar dat wordt niet gevraagd.

Opdracht (c) Geef een correcte implementatie van het systeem volgens onderdeel (a), met behulp van twee gesplitste binaire semaforen. Zorg dat de body van de **await** opdracht van de consumer gelijktijdig uitgevoerd kan worden met de body van de **await** opdracht van elk van de producers. Dit kan als `nr` en `free` bewaakt worden door de ene gesplitste binaire semafoor en `read` en `wr` door de andere.

Oplossing. (a) De producers moet `wr`$[x]$ true maken om aan te geven dat er geschreven is.

Producers :
do *true* →
produce item ;
⟨ **await** `nr` $\neq K$ **then**
$x :=$ `free` ; `free` := `free` $\oplus 1$;
`nr` := `nr` $+ 1$ ⟩ ;
`buf`$[x]$:= *item* ;
`wr`$[x]$:= *true* ;
od .

(b) Initieel zijn alle gehele variabelen zijn 0; de booleans zijn initieel false. De waarden van `buf` zijn irrelevant.

(c) We introduren twee tellers: `np` en `nc` voor de aantallen wachtende producers en consumers, beide initieel 0. We introduceren twee binaire semaforen `gp` en `gc`, beide initieel 1 (open), en twee binaire semaforen `wp` en `wc`, beide initieel 0 (dicht); `gp` en `wp` vormen een gesplitste binaire semafoor ter bewaking van `np`, `free`, `nr`; `gc` en `wc` vormen een gesplitste binaire semafoor ter bewaking van `nc`, `read`, `wr`.

Producers :
do *true* →
produce item ;
P(`gp`) ; `np ++` ; *VP* ;
P(`wp`) ;
$x :=$ `free` ; `free` := `free` $\oplus 1$;
`nr ++` ; `np --` ;
VP ;
`buf`$[x]$:= *item* ;
P(`gc`) ; `wr`$[x]$:= *true* ; *VC* ;
od .

a single Consumer :
do *true* →
P(`gc`) ; `nc ++` ; *VC* ;
P(`wc`) ;
`nc --` ; `wr`[`read`] := *false* ;
$y :=$ `read` ; `read` := `read` $\oplus 1$;
VC ;
item := `buf`$[y]$;
P(`gp`) ; `nr--` ; *VP* ;
consume item
od .

VP = **if** `np` $> 0 \wedge$ `nr` $\neq K$ **then** V(`wp`) **else** V(`gp`) **fi** ,
VC = **if** `nc` $> 0 \wedge$ `wr`[`read`] **then** V(`wc`) **else** V(`gc`) **fi** .