Tentamen Operating Systems II, 9 februari 2006

Tijdsduur 3 uur. Gesloten boek tentamen.

Opgave 1 (35 %). Gegeven is een systeem met gedeelde variabelen

const N : `int` $\{N > 0\}$;
const W : `real` ;
const $a[0 : N-1]$: `real` ;
var `x` : `real` .

en N processen van de vorm

process *modif*(*self* := 0 **to** $N-1$)
  do *true* →
    var y : `real` ;
    choose y **arbitrary** ;
    ⟨ `x` := y ⟩ ;
    ⟨ **await** `x` + $a[self] < W$ **then** `x` := `x` + $a[self]$ ⟩
  od end *modif* .

Implementeer de atomiciteitshaakjes en de await statements van dit systeem met (gesplitste) binaire semaphoren. Hierbij is de voortgangseis, dat de implementatie alleen in deadlock mag komen onder omstandigheden waarin ook het abstracte systeem deadlock zou hebben. Toon aan dat je oplossing hieraan voldoet.

Uitwerking 1.

```
    sem gs:= 1
    sem ba[0: N-1] := ([N] 0)
    var wa[0..N-1]: boolean := ([N] false)

Body :
    choose y ;
    P(gs) ; x := y ; wa[self] := true ; Wissel ;
    P(ba[self]) ;  wa[self] := false ; x := x+a[self] ; Wissel

Wissel : // zoek lineair naar i < N met wa[i] en a + a[i] < W
    var i := 0 ;
    do i < N and not (wa[i] and x + a[i] < W) -> i ++ od ;
    if i < N -> V(ba[i]) [] else -> V(gs) fi .
```

Als alle process buiten de PV-secties zijn, geldt `gs` $+ \sum$ `ba`$[p] = 1$. Proces p wacht bij de await ddan als `wa`$[p]$ geldt. Als de lus in Wissel een waarde $i < N$ oplevert, wordt proces i toegelaten dat wacht terwijl hij toegelaten mag worden: `wa`$[i]$ $\wedge$ `x` $+ a[i] < W$. Er ontstaat dan dus geen deadlock.

In geval van deadlock geldt dus `gs` $= 1$. We gebruiken nu de invariant `gs` $> 0 \wedge$ `wa`$[p] \Rightarrow$ `x` $+ a[p] \geq W$. Als `gs` $= 1$, dan is er geen bij await wachtend proces dat toegelaten kan worden. Wegens deadlock en `gs` $= 1$ staan dus alle processen p bij hun `await` te wachten terwijl `x` $+ a[p] \geq W$, zodat ook het abstract systeem dan in deadlock zou zijn.

Opgave 2 (30 %). Over pthreads. Gegeven is een systeem met twee soorten threads (processen).

```
const nH, nK : int   { nh > 1  ∧  nK > 1 } ;

process handelaar (zelf := 1 to nH)
   do true   →
      ⟨ handel(zelf) ⟩
      NCS
      ⟨ await voorwaarde() then ruimop(zelf) ⟩
   od end handelaar

process klant (zelf := 1 to nK)
   do true   →
      NCS
      ⟨ await behoefte() then koop(zelf) ⟩
   od end klant
```

De procedures *handel*, *ruimop* en *koop* werken op gedeelde variabelen, de boole'se functies *voorwaarde* en *behoefte* inspecteren deze gedeelde variabelen. Deze procedures en functies moeten dus atomair uitgeverd worden. *NCS* staat voor een niet-critische sectie, die niet hoeft te eindigen. De boole'se functies *voorwaarde* en *behoefte* zijn niet afhankelijk van de betreffende handelaar of klant.

Implementeer de atomiciteitshaakjes en de await statements van dit systeem met *één mutex en enkele conditievariabelen* (minder dan $nH + nK$). Het is niet toegstaan `broadcast`s te geven. Wees spaarzaam met het geven van `signal`s.

De voortgangseis is, dat de implementatie alleen in deadlock mag komen onder omstandigheden waarin ook het abstracte systeem deadlock zou hebben. Toon aan dat je oplossing hieraan voldoet.

Uitwerking 2. Nu met een mutex en conditievariabelen:

```
       var mu: mutex
       var cH, cK: condVar
       var wH, wK: int := 0

    process handelaar (zelf := 1 to nH)
       do true ->
          lock(mu) ; handel(zelf) ; wissel ;
          NCS ;
          lock(mu) ; wH ++ ;
(h)       do not voorwaarde -> wait(mu, cH) od ;
          wH -- ; ruimop(zelf) ; wissel
        od end handelaar

     process klant (zelf := 1 to nK)
        do true ->
           NCS ;
           lock(mu) ; wK ++ ;
(k)        do not behoefte -> wait od ;
           wK -- ; koop(zelf) ; wissel
         od end klant
```

```
wissel :
   if wK > 0 and behoefte -> signal(cK)
   [] wH > 0 and voorwaarde -> signal(cH)
   fi ;
   unlock(mu)
```

Er geldt de invariant dat `wH` en `wK` respectievelijk gelijk zijn aan de aantallen handelaars en klanten die staan te wachten bij (h) en (k), al dan niet in de wachtrijen `cH` of `cK`.

Er geldt de invariant:

$$\text{(J)} \qquad \begin{array}{l} \texttt{mu} = \bot \ \wedge\ ((\texttt{wH} > 0 \ \wedge\ \texttt{voorwaarde}) \ \vee\ (\texttt{wK} > 0 \ \wedge\ \texttt{behoefte})) \ \Rightarrow \\ \quad (\texttt{wH} > 0 \ \wedge\ \texttt{voorwaarde} \ \wedge\ \exists\, p : p \ \textbf{at}\ (h) \ \wedge\ p \notin \texttt{cH}) \\ \quad \vee\ \ (\texttt{wK} > 0 \ \wedge\ \texttt{behoefte} \ \wedge\ \exists\, p : p \ \textbf{at}\ (k) \ \wedge\ p \notin \texttt{cK})\ . \end{array}$$

Initieel geldt $\texttt{wH} = \texttt{wK} = 0$ zodat (J) geldt. Het predicaat (J) kan alleen false gemaakt worden doordat een proces $\texttt{mu} := \bot$ uitvoert. Dit gebeurt in de wissel en in de `wait` statement. Als de wissel een signal geeft, maakt dat (J) waar. Als de wissel geen signal geeft, blijft (J) waar omdat de antecedent van (J) onwaar is. Als een proces $\texttt{mu} := \bot$ uitvoert in `wait`, kan het (J) niet onwaar maken, want als het een handelaar is geldt $\neg\,\texttt{voorwaarde}$ en als het een klant is geldt $\neg\,\texttt{behoefte}$.

Stel nu dat er deadlock optreedt. Dan geldt $\texttt{mu} = \bot$. Omdat er deadlock optreedt en $\texttt{mu} = \bot$ geldt, zijn alle handelaars en klanten bij (h) en (k). Dus $\texttt{wH} = \texttt{nH} > 0$ en $\texttt{wK} = \texttt{nK} > 0$. Omdat er deadlock is, is de consequent van (J) false. De antecedent van (J) is dus false. Hieruit volgt dat *voorwaarde* en *behoefte* beide onwaar zijn, terwijl alle handelaars en klanten voor hun **await** staan. Onder deze omstandigheden is ook het abstracte systeem in deadlock.

Opgave 3 (35 %). Gegeven is een systeem met N processes van de vorm:

```
process Member(self:= 0 to N-1)
   do true ->
      NCS
      Enter
      Chamber
      CS
      Exit
   od end Member
```

De commando's `NCS` en `CS` zijn ook gegeven; `NCS` hoeft niet te eindigen; `CS` eindigt altijd. Het kritische gebied `CRIT` bestaat uit de samenstelling van de drie commando's `Chamber`, `CS` en `Exit`. Gegeven is een constante M met $1 \leq M \leq N$.

Implementeer de commando's `Enter`, `Chamber` en `Exit` met behulp van hulpvariabelen, atomaire commando's en compound await statements zodanig dat
(1) Als er een proces in `CS`, dan zijn er precies M processen in `CRIT`, dwz. vóór of in `Chamber`, `CS` of `Exit`.
(2) Een toestand met M processen in `CRIT` leidt binnen begrensde vertraging (bounded delay) tot een toestand waarin `CRIT` leeg is.
(3) Er wordt niet overbodig gewacht.

Maak puntsgewijs aannemelijk, dat je oplossing aan de condities (1), (2) en (3) voldoet.

Uitwerking 3. Er zijn veel verschillende mogelijkheden, maar twee tellers en een boolean lijken noodzakelijk.

```
var n := 0, k := 0
var b := false

Enter:    < await not b then
              n ++ ;
              if n = M -> b := true fi >
Chamber:  < await b >
Exit:     < k ++ >
          < await k = n then
              k -- ; n -- ;
              if n = 0 -> b := false fi >
```

`n` is het aantal processen in `CRIT`; `k` is het aantal processen dat `Exit` is binnengegaan maar nog niet verlaten heeft. Derhalve betekent $\texttt{k} = \texttt{n}$ dat `CS` leeg is. Er zijn verder de invarianten

$$q \textbf{ in } \texttt{CS} \;\Rightarrow\; \texttt{b}\,,$$
$$\texttt{k} < \texttt{n} \;\wedge\; \texttt{b} \;\Rightarrow\; \texttt{n} = M\,.$$

Hieruit volgt (1).

Als er M processen in `CRIT` zijn, geldt dus $\texttt{n} = M$. Er is ook de invariant $\texttt{n} = M \;\Rightarrow\; \texttt{b}$. Binnen twee rondes geldt dus $\texttt{k} = \texttt{n} = M$. In de volgende ronde wordt $\texttt{k} = \texttt{n} \;\wedge\; \neg\texttt{b}$ bewerkstelligd, zodat `CRIT` leeg is. Hieruit volgt (2).

We bewijzen nu ook, dat een toestand met M processen bij `Enter` terwijl `CRIT` leeg is, binnen begrensde vertraging leidt tot een toestand met M processen in `CRIT`. We hebben nu de invariant nodig: $\texttt{CRIT} = \emptyset \;\Rightarrow\; \neg\,\texttt{b}$. Als er M processen bij `Enter` staan en $\neg\,\texttt{b}$ geldt, dan geldt binnen éń ronde $\texttt{n} = M \;\wedge\; \texttt{b}$.

(3) Wordt er overbodig gewacht? Als er meer dan M processen in `Chamber` zijn, forceert conditie (1) deadlock. Dus er moet in `Enter` gewacht worden. Als er minder dan M processen bij `Chamber` zijn, moet er volgens (1) gewacht worden; dus ook bij `Chamber` moet gewcht worden. Als er processen in `CS` zijn, mogen geen processen `CRIT` verlaten. Er moet dus in `Exit` gewacht worden.

Er is een oplossing denkbaar waarin `Enter` al nieuwe processen toelaat, als `CS` is leeggelopen terwijl `Exit` nog niet leeg is. Deze processen moeten dan toch bij `Chamber` wachten, of het leeglopen van `Exit` moet onderbroken worden. De eerste optie geeft evenveel wachten en een minder net programma. De tweede optie blokkeert processen die `CS` doorlopen hebben onnodig en brengt eis (2) in gevaar.