Tentamen Operating Systems II, 11 februari 2005

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 (30 %). Beschouw een systeem met $N \geq 3$ processen en drie gedeelde variabelen volgens

```
        var x : int := 0 , y : int := 0 , z : int := 0 ;
        process ThreadC (self := 0 to N − 1)
           do true  →
10:           TNS
11:           ⟨ x ++ ⟩
12:           ⟨ await x ≥ 3 ∧ y < 3 then y ++ ⟩
13:           CS
14:           ⟨ z ++ ;
                if z = 3  →  x := x − 3 ; y := 0 ; z := 0 fi ⟩
           od
        end ThreadC .
```

De commando's *TNS* en *CS* zijn gegeven, eindigen gegarandeerd en modificeren geen gedeelde variabelen.
(a: 15 %) Bedenk en bewijs enige invarianten voor dit systeem en bewijs hiermee dat het systeem niet in deadlock kan geraken.
(b: 15 %) Het systeem wordt nu geïmplementeerd met pthreads, waarbij gegeven is dat één mutex `mu` en één conditievariabele `cv` reeds gedeclareerd en geïnitialiseerd zijn. Implementeer hiermee de atomaire commando's 11 en 14 en de compound await statement 12. (Gebruik de in het diktaat ingevoerde afkortingen: `lock`, `unlock`, `wait`, `signal`, `broadcast`.)

Opgave 2 (40 %). Beschouw een systeem met één gedeelde variabele en N processen volgens

```
     var x: int := 0

     process ThreadM (self := 0 to N-1)
        var a: int := 0
        do true ->
           TNS
(*)        x := x + a
        od
     end ThreadM
```

Het systeem dient te voldoen aan de invariant

(J) $\quad$ x ≥ 0 .

Het programmafragment `TNS` is gegeven. Het eindigt gegarandeerd, en het kan de waarden van de privévariabele a wijzigen (bv. negatief maken).

(a: 10 %) Verander de regel (*) in een compound await statement, dat wacht of de betreffende toekenning bewerkstelligt, zodanig dat predicaat (J) gegarandeerd geldig blijft. Vermijd onnodig wachten.

(b: 30 %) Implementeer je oplossing van onderdeel (a) met behulp van binaire semaforen. Processen mogen elkaars privévariabelen niet inspecteren. Zorg dat deadlock alleen optreedt als geen van de processen `x` kan wijzigen zonder predicaat (J) te schenden. Argumenteer dat je oplossing daaraan inderdaad voldoet.

Opgave 3 (30 %). Gegeven zijn N processen die de knopen vormen van een ongerichte samenhangende graaf. De processen kunnen alleen communiceren middels boodschappen naar buren in de graaf. Voor proces p is $\texttt{nhb}[p]$ de lijst van buren van p. Elke ribbe tussen twee buurknopen p en q heeft een "lengte" $w(p,q)$. Er geldt altijd $w(p,q) = w(q,p) > 0$. Voorts is $w(p,q) = \infty$ als p en q geen buren zijn. Er is een knoop `root` tot wortel van de graaf aangewezen.

We definiëren de *lengte* van een pad in de graaf als de som van de lengten van de gepasseerde ribben. We definiëren de *afstand* tussen twee knopen als de lengte van een kortste pad tussen de twee knopen. We definiëren het *gewicht* van een knoop als de afstand tot `root`. Knoop p heet een *ouder* van knoop q, als er een kortste pad van `root` naar q loopt, waarin p de voorlaatste knoop is.

Gevraagd: geef programma's voor de processen in de knopen, zodanig dat elke knoop zijn gewicht en de verzameling van zijn ouders bepaalt en opslaat in privévariabelen. Je mag standaardnotaties gebruiken voor het aflopen van lijsten en het vormen van verzamelingen.

Het systeem dient te eindigen in een toestand waarin er geen boodschappen meer onderweg zijn. Het mag dan in deadlock zijn, in de zin dat de processen in een niet-eindigende loop nog wel wachten op (nooit komende) boodschappen.

Uitwerking 1. (a) Om afwezigheid van deadlock te bewijzen, hebben we genoeg aan de volgende invarianten:

(J0) $\quad \texttt{y} \leq 3$,
(J1) $\quad 0 \leq \texttt{z} < 3$,
(J2) $\quad \texttt{y} - \texttt{z} = \#\{q \mid q \textbf{ in } \{13, 14\}\}$,
(J3) $\quad \texttt{x} - \texttt{z} = \#\{q \mid q \textbf{ in } \{12, 13, 14\}\}$.

Bewijs van de invarianten. Initieel geldt $\texttt{x} = \texttt{y} = \texttt{z} = 0$ en alle processen zijn bij 10. De predicaten gelden dus alle initieel.

De variabele `y` wordt alleen verhoogd als hij < 3 is. Verder wordt `y` alleen nog soms op 0 gezet. Dus (J0) wordt nooit geschonden. De variabele `z` wordt alleen in 14 gewijzigd, en wel volgens $0 \to 1 \to 2 \to 0$. Dus (J1) blijft geldig.

Linker en rechter lid van (J2) veranderen alleen als een proces 12 of 14 uitvoert. Als een proces, zeg p, 12 uitvoert, gaat het naar 13 en worden linker en rechter lid van (J2) beide met 1 verhoogd. Als p 14 uitvoert met preconditie $\texttt{z} \neq 2$, worden linker en rechter lid van (J2) beide met 1 verlaagd. Als p 14 uitvoert met preconditie $\texttt{z} = 2$, dan geldt wegens (J2) en (J0) dat $\texttt{y} = 3$. Ook dan worden linker en rechter lid van (J2) dus beide met 1 verlaagd. Dus (J2) is een invariant.

Linker en rechter lid van (J3) veranderen alleen als een proces 11 of 14 uitvoert. Als een proces, zeg p, 11 uitvoert, gaat het naar 12 en worden linker en rechter lid van (J3) beide met 1 verhoogd. Als p 14 uitvoert met preconditie $\texttt{z} \neq 2$, worden linker en rechter lid van (J3) beide met 1 verlaagd. Als p 14 uitvoert met preconditie $\texttt{z} = 2$, worden linker en rechter lid van (J2) ook beide met 1 verlaagd. Dus (J3) is een invariant.

Uit deze vier invarianten volgt

$$
\begin{array}{ll}
 & \textit{deadlock} \\
\equiv & \{ \text{ programma } \} \quad (\texttt{x} < 3 \ \vee \ 3 \leq \texttt{y}) \quad \wedge \quad (\forall\, q : q \textbf{ at } 12) \\
\equiv & \{ \text{ (J2) en (J3) } \} \quad (\texttt{x} < 3 \ \vee \ 3 \leq \texttt{y}) \quad \wedge \quad \texttt{x} - \texttt{z} = N \quad \wedge \quad \texttt{y} - \texttt{z} = 0 \\
\Rightarrow & \{ \text{ (J1) } \} \quad (\texttt{x} < 3 \ \vee \ 3 \leq \texttt{y}) \quad \wedge \quad \texttt{x} \geq N \quad \wedge \quad \texttt{y} < 3 \\
\equiv & \{ \text{ rekenen en } N \geq 3 \ \} \quad \textit{false} \ .
\end{array}
$$

(b) De await statement wordt (bv.)

```
12      lock(mu) ;
        while (x < 3 || 3 <= y) do wait(cv, mu) end ;
        y ++ ;
        unlock(mu) .
```

Zowel 11 als 14 kunnen de await-conditie opheffen. Als de await-conditie false is mogen alle wachtende processen pogen toegang te krijgen (hoewel er slechts 3 zullen slagen). We kunnen dus met broadcasts toe:

```
11      lock(mu) ;
        x ++ ;
        if x >= 3 && y < 3 -> broadcast(cv) fi ;
        unlock(mu)

14      lock(mu) ;
        z ++ ;
        if z = 3 -> x := x-3 ; y := 0 ; z := 0 ;
                    if x >= 3 -> broadcast(cv) fi ;
        fi ;
        unlock(mu) .
```

De condities vóór de broadcasts zijn niet nodig (een kwestie van smaak). Het is niet correct hier signals te geven, omdat als $\mathtt{x} \geq 3$ wordt of $\mathtt{y} = 0$ wordt, er drie processen verder moeten kunnen gaan.

Uitwerking 2.

```
(a)     < await x+a >= 0 then x := x + a >
```

(b) We maken gebruik van een hulparray `aa` waarin de processen negatieve waarden van hun a kenbaar maken. Verder gebruiken we een gesplitste binaire semafoor.

```
      var aa[0: N-1]: int := ([N] 0)
      var next: int ;
      sem gs := 1
      sem wa[0: N-1] := 0

      process ThreaM (self := 0 to N-1)
        var a: int
        do true ->
           TNS
           if a >= 0  ->
              P(gs)
              x := x + a
              Wissel
           [] a < 0  ->
(*)           P(gs)
              aa[self] := a
              Wissel
(**)          P(wa[self])
              x := x + a
              aa[self] := 0    # t.b.v invariant (J1) hieronder
              Wissel
           fi
        od
```

```
Wissel =
  next := 0
  do next < N && (aa[next] >= 0 || x + aa[next] < 0) ->
     next := next + 1
  od
  if next < N  -> V(wa[next])
  [] else      -> V(gs)
  fi
```

Buiten de PV-secties gelden de volgende invarianten:

(J0) $\mathtt{gs} + (\sum\ q : \mathtt{wa}[q]) = 1$,
(J1) $\mathtt{aa}[q] < 0 \ \equiv \ q$ **at (**)** ,
(J2) $\mathtt{wa}[q] = 1 \ \Rightarrow \ \mathtt{aa}[q] < 0 \ \wedge \ \mathtt{x} + \mathtt{aa}[q] \geq 0$,
(J3) $\mathtt{gs} = 1 \ \Rightarrow \ (\forall\ q : \mathtt{aa}[q] \geq 0 \ \vee \ \mathtt{x} + \mathtt{aa}[q] < 0)$,
(J4) $\mathtt{aa}[q] < 0 \ \Rightarrow \ \mathtt{aa}[q] = a.q$.

Wegens (J1) en (J2) kan er geen deadlock optreden als $\mathtt{wa}[q] = 1$. Dus deadlock impliceert $\mathtt{gs} = 1$ wegens (J0). Er wachten dan kennelijk geen procesen bij `(*)`. Dus alle processen wachten bij `(**)`. Wegens (J1) en (J3) geldt dus $\mathtt{aa}[q] < 0$ en $\mathtt{x} + \mathtt{aa}[q] < 0$ voor alle q. Wegens (J4) is dan $\mathtt{x} + a.q < 0$ voor alle q. Deadlock impliceert dus $\mathtt{x} + a.q < 0$ voor alle q.

Uitwerking 3.

```
op mess [1: N](sender: int, we: int)

process Node (self := 1 to N)
  var ouders := empty
  var gewicht := oneindig
  if self = root ->
     gewicht := 0
     lis := nhb[self]
     do lis != null ->
         send mess[lis^.nr](self, gewicht)
         lis := lis^.nx
     od
  fi
  do true ->
     in mess[self](sender, ww) ->
        int nw := ww + w(sender, self)
        if nw < gewicht ->
            gewicht := nw
            ouders := new set(sender)
            lis := nhb[self]
            do lis != null ->
                send mess[lis^.nr](self, gewicht)
                lis := lis^.nx
            od
        [] nw = gewicht -> ouders.add(sender)
        fi
     ni
  od
end Node
```