Hertentamen PC/A2 (deel a) 24-08-2004

Geef (liefst beknopte en heldere) motivatie bij je antwoorden; dus niet enkel 'ja' of 'nee' antwoorden, maar ook waarom. Geef van berekeningen niet alleen het eindresultaat, maar ook hoe je hieraan komt. Succes!

Vraag 1 (2 punten)

Stel we hebben operating system dat 3 toestanden van processen onderscheidt, namelijk Ready, Running, en Blocked, zoals beschreven in Stallings.

a. In welke toestand denk je dat een shell (bijv. de minishell die je op het practicum gemaakt hebt) meestal zit? Motiveer je antwoord.

b. We garanderen mutual exclusion tussen kritieke secties in twee processen P en Q met semaforen die als primitieve op een of ander operating system aanwezig zijn en dat op een machine met 1 processor draait. Stel dat P in zijn kritieke sectie zit, en dat Q op het punt staat zijn kritieke sectie binnen te gaan. In welke van de 3 toestanden kan P_1 in dat geval zijn? En P_2? Motiveer je antwoord, en geef expliciet aan welke verdere aannamen je eventueel maakt.

Vraag 2 (1 punt)

Stel je gebruikt threads bij het implementeren van een web-browser en je laat het afbeelden van verschillende plaatjes etc., door verschillende threads doen.

a. Zou je hiervoor liever user level threads of kernel level threads gebruiken?

b. Heeft de minder gunstige van de twee – user level threads of kernel level threads – nog wel voordelen boven helemaal geen threads gebruiken?

Motiveer je antwoorden.

Vraag 3 (2 punten)

Op een multi-user systeem met round-robin scheduling draaien 3 interactieve processen: $P1$, $P2$ en $P3$. $P1$ heeft steeds CPU-bursts van 50 msec, $P2$ van 70 msec, en $P3$ van 300 msec.

a. Welke waarde van het quantum q denk je dat het beste is om de gemiddelde response-tijd zo klein mogelijk wilt krijgen? Motiveer je antwoord. Waarom verwacht je dat bij een kleinere waarde van q de response-tijd langer zal zijn? En waarom bij een grotere?

b. De operator vindt response-tijd niet van belang en wil enkel de bezettingsgraad van de CPU (de fractie van de tijd dat de CPU met nuttig werk – dus niet context switches – bezig is) maximaal houden. Om de overhead van context switches te vermijden neemt hij q=1000 msec. Kan het zo zijn dat een kleiner quantum toch een hogere bezettingsgraad van de CPU geeft? Motiveer je antwoord.

Vraag 4 (2 punten)

Beschouw de volgende pseudo-code voor een proces dat uit twee threads bestaat:

```
Shared variables
-----------------
 data buffer[10];   /* array of buffers to hold produced data */
 semaphore sem = 0; /* general semafore */

Producer thread                          Consumer thread
---------------                          ---------------
for (i=0; i<10;i++) {                    for (i=0; i<10;i++) {
  read from file into buffer[i];           wait(sem);
  signal(sem)                              process contents of buffer[i];
}                                        }
```

a. Stel de producer thread blokkeert telkens t_p seconden bij het lezen uit de file, en de consumer thread heeft steeds t_c seconden nodig om de data te verwerken. Neem aan dat de consumer niet blokkeert tijdens deze t_c seconden. Neem aan dat de resterende delen van de code voor producer en consumer een verwaarloosbare hoeveelheid CPU tijd vergen. Hoeveel tijd kost het voordat beide threads klaar zijn als het user-level threads zijn? Je antwoord zijn dient te bestaan uit een enkele expressie met daarin t_p en/of t_c.

b. Idem, maar nu aannemend dat de threads kernel-level threads zijn.

Vraag 5 (3 punten)

Een proces bestaat uit 6 threads: P_1, P_2, P_3, P_4, Q, R.
De code van de threads P_i is
$\{\mathtt{s}_1;\ \mathtt{s}_2;\ \mathtt{s}_3;\ \mathtt{x} = \mathtt{x} + 1;\}$
De code van Q is
$\{\mathtt{s}_4;\ \mathtt{s}_5;\ \mathtt{s}_6;\ \mathtt{x} = \mathtt{x} + 1;\}$
De code van R is
$\{\mathtt{s}_7;\ \mathtt{x} = \mathtt{x} + 1;\}$
Hier is `x` een shared variable die alle threads gebruiken en die op 0 geïnitialiseerd is. Deze variabele houdt bij hoeveel threads klaar zijn, en moet dus de waarde 6 hebben als alle threads klaar zijn.

a. We willen de volgende synchronisatie tussen deze threads regelen:

   - Er mogen maximaal 2 van de threads P_1 t/m P_4 in het code-fragment $\mathtt{s}_2$ zitten.
   - Thread Q mag pas met $\mathtt{s}_5$ beginnen als twee van de threads P_1 t/m P_4 klaar zijn met hun $\mathtt{s}_2$.
   - Thread R mag pas starten zodra alle vier de P_i threads klaar zijn.
   - Verder mag access aan de shared variabele `x` geen problemen opleveren.

   Introduceer semaforen en/of shared variables en breidt de code van de threads uit zodanig dat deze condities gegarandeerd worden. Vergeet niet te vermelden wat de initiële waardes van semaforen en shared variable moeten zijn, en of semaforen binaire of algemene (general) semaforen zijn. Je oplossing mag (natuurlijk) geen busy waiting gebruiken.

b. Kunnen de bovenstaande synchronisatiecondities deadlock veroorzaken? Zoja, schets een scenario waarbij deadlock optreedt. Zonee, geef overtuigende redenen waarom deadlock niet kan optreden.