Tentamen PC: Processen & Concurrency
10-05-2005

Geef bij al je antwoorden voldoende motivatie: voor alleen getallen, ja/nee antwoorden en slogans krijg je geen punten.

Neem na het tentamen even de tijd om de enquete in te vullen, want dat is voor ons erg belangrijk om volgend jaar dingen te verbeteren aan deze cursus! Lever de enquete direct in bij de surveillant of in de komende 2 weken bij het secretariaat in A6001.

Succes!!

Vraag 1 (0.5 punt)

In de discussie van user-level threads (ULTs) versus kernel-level threads (KLTs) noemt Stallings als nadeel van ULTs dat wanneer een ULT een system call uitvoert niet alleen deze thread geblokkeerd raakt maar tevens alle andere threads binnen het proces. Waarom is dit zo?

Vraag 2 (2.5 punten)

Je bent zojuist in dienst genomen door Moeder Natuur om te helpen met de chemische reactie voor het vormen van water, die ze maar niet voor elkaar weet te krijgen vanwege synchronisatieproblemen. Het probleem is om twee H atomen en een O atoom bij elkaar te krijgen op hetzelfde moment. De atomen zijn processen. Ieder H atoom roept een procedure *hReady* aan wanneer het klaar is voor een reactie, en ieder O atoom roept een procedure *oReady* aan wanneer het klaar is. De procedures dienen te wachten totdat er tenminste twee H atomen en een O atoom aanwezig zijn, waarop een van de procedures de procedure *makeWater* moet aanroepen (een triviale procedure die slechts rapporteert dat er een molecuul is gemaakt). Na afloop van de aanroep van *makeWater* dienen twee instanties van *hReady* en een instantie van *oReady* te termineren. Schrijf de code voor *hReady* en *oReady* gebruikmakend van semaforen voor het realiseren van de gewenste synchronisatie. Beargumenteer dat je oplossing zowel starvation als busy-waiting vermijdt.

Vraag 3 (1 punt)

Laat zien hoe de exchange instructie gebruikt kan worden om de semafoor operaties wait en signal te implementeren.

Vraag 4 (1 punt)

Beschouw het probleem van de dinerende filosofen. Neem aan dat er drie filosofen zijn, p_1, p_2 en p_3, die gebruik maken van 3 vorken, f_1, f_2 en f_3. De filosofen voeren de volgende code uit:

```
p1() {
while (1) {
   P(f1);
   P(f3);
   eat;
   V(f3);
   V(f1) }
}
```

```
p2() {
while (1) {
   P(f1);
   P(f2);
   eat;
   V(f2);
   V(f1) }
}
```

```
p3() {
while (1) {
   P(f3);
   P(f2);
   eat;
   V(f2);
   V(f3) }
}
```

1. Is deadlock mogelijk in dit systeem?
2. Is deadlock mogelijk indien we de volgorde van de P-operaties veranderen in proces p_1, p_2 of p_3?
3. Is deadlock mogelijk indien we de volgorde van de V-operaties veranderen in proces p_1, p_2 of p_3?

Vraag 5 (1 punt)

Welk voordeel heeft het om tijds-quanta van verschillende grootte te hebben voor de verschillende niveaus in een multi-level queuing system?

Vraag 6 (2 punten)

1. Wat is de slechtst mogelijke CPU scheduling algoritme, dat wil zeggen het algoritme dat de slechts mogelijke gemiddelde responstijd oplevert? Neem aan dat de algoritme altijd een proces moet draaien als er een beschikbaar is, en dat er geen tijd verloren gaat door context switches.
2. Kan round robin ooit de slechts mogelijke CPU scheduling algoritme zijn? Zo ja, onder welke omstandigheden? Zo neen, leg uit waarom.

3. Kan FCFS ooit de slechts mogelijke CPU scheduling algoritme zijn? Zo ja, onder welke omstandigheden? Zo neen, leg uit waarom.

Vraag 7 (0.5 punt)

Welke soorten van informatie over een task kunnen nuttig zijn bij real-time scheduling?

Vraag 8 (1.5 punt)

Welke van de volgende verzamelingen van periodieke taken zijn schedulable met het rate monotonic algoritme? En welke met het earliest-deadline-first algoritme? Leg je antwoord uit.

1. $T = \{(8, 3), (9, 3), (15, 3)\}$
2. $T = \{(8, 4), (12, 4), (20, 4)\}$
3. $T = \{(8, 4), (10, 2), (12, 3)\}$

(In een paar (p, c) geeft p de periode van de taak en c de executietijd.)