Tentamen Operating Systems

Woensdag 27 juni 2007, 9:00-12:00, Examenhal

- Lees eerst een opgave volledig door, alvorens deze te maken.
- Schrijf netjes en zorgvuldig.
- Dit tentamen is 'Open Boek', d.w.z. dat het boek "*Operating Systems, Design and Implementation*" van Tanenbaum & Woodhull gebruikt mag worden als naslagwerk. Het is niet toegestaan ander materiaal, zoals college-aantekeningen en powerpoint-slides, te raadplegen.
- Het tentamen bestaat uit 5 opgaven. Iedere opgave is 2 punten waard.
- Je hebt 3 uur de tijd, gebruik deze nuttig. Ook als je snel klaar bent, gebruik dan de resterende tijd om jouw antwoorden nog eens te controleren.
- Succes!

Opgave 1: Scheduling
Gegeven zijn 5 processen van drie gebruikers die zich aanmelden aan het besturingsysteem op verschillende tijdstippen. Ieder proces heeft een 'eigenaar', namelijk de gebruiker die het proces heeft gestart. In deze opgave gaan we uit van tijdstippen in gehele seconden. Van ieder proces is het tijdstip van aanmelden (aankomst) en de duur van de executie bekend, en weergegeven in de volgende tabel.

proces	aankomst	executietijd	eigenaar
P1	0	7	A
P2	19	1	B
P3	6	5	A
P4	5	9	C
P5	2	7	B

(a) Bepaal de volgorde van executie van de processen op systemen met de volgende scheduling methodes. Bepaal van ieder proces zijn starttijd en wachttijd (waiting time). Bereken tevens de gemiddelde wachttijd van het gehele systeem.

- Nonpreemptive First Come First Served (FCFS)
- Nonpreemptive Shortest Job First (SJF)
- Preemptive Round Robin met time quantum van 1 seconde. Ga hierbij uit van een FIFO-queue implementatie, waarbij een proces bij aankomst onmiddellijk de beschikking over de processor krijgt en voorrang krijgt boven ieder ander proces.

(b) Bepaal de volgorde van executie van de processen voor de fair-share variant van de bovenstaande preemptive Round Robin scheduler, d.w.z. een scheduler die ieder van de drie gebruikers evenveel rekentijd toekent. Bepaal opnieuw voor ieder proces zijn starttijd en wachttijd. Wat is de gemiddelde wachttijd?

Opgave 2: Unix system programming

(a) Het onderstaande programma berekent de som $\sum_{i=0}^{15}$. Leg uit hoe dit werkt.

(b) Geef een voorbeeld van een mogelijke uitvoer van het programma. Waarom kan de uitvoer per executie verschillen?

(c) Leg kort (maar duidelijk) uit wat de code van de volgende regels bewerkstelligen:

- regels 18-26
- regels 29-34 (i.h.b. regel 32)
- regels 36-40

```
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3  #include <string.h>
 4  #include <unistd.h>
 5  #include <sys/wait.h>
 6
 7  int sum(int a, int b) {
 8    int i, s=0;
 9    for (i=a; i<b; ++i) {
10      s += i;
11    }
12    return s;
13  }
14
15  int main (int argc, char *argv[]) {
16    int i, m, s, w, status;
17    int pid = getpid();
18    int a=0, b=16;
19    for (i=0; i<3; ++i) {
20      m = (b-a)/2;
21      if (fork()) {
22        b = a+m;
23      } else {
24        a = a+m;
25      }
26    }
27    printf ("%d-%d\n", a, b);
28
29    s = sum(a, b);
30    w = waitpid(-1, &status, 0);
31    while (w >= 0) {
32      s += status/256;
33      w = waitpid(-1, &status, 0);
34    };
35
36    if (pid == getpid()) {
37      printf ("sum=%d\n", s);
38      return EXIT_SUCCESS;
39    }
40    return s;
41  }
```

Opgave 3: Mutual exclusion

Beschouw een uni-processor besturingsysteem met een preemptieve scheduler.

(a) Als een proces de clock-interrupt zelf uit en aan kan schakelen, dan kan wederzijdse uitsluiting (mutual exclusion) eenvoudig door een proces zelf geïmplementeerd worden. Leg uit hoe.

(b) De bovenstaande methode wordt over het algemeen niet als een acceptabele methode beschouwd. Waarom niet?

(c) Waarom is het wel toegestaan om deze techniek te gebruiken op het niveau van de kernel?

Gegeven zijn de onderstaande twee processen `P0` en `P1`. Een gedeelde variabele (shared variabele) `turn` geeft aan welk van de twee processen zijn critische sectie (`CS0` en `CS1`) kan uitvoeren. De variabele `turn` heeft initiëel de waarde `0`.

```
P0: while (1) {                          P1: while (1) {
      while (turn!=0);                         while (turn!=1);
      CS0;  /* critical section     */         CS1;  /* critical section     */
      turn = 1;                                turn = 0;
      NCS0; /* non-critical section */         NCS1; /* non-critical section */
    }                                        }
```

(d) Neem aan dat schrijf- en leesoperaties op de variabele `turn` atomair zijn. Geef minstens 3 nadelen van de bovenstaande implementatie van mutual exclusion.

(e) Los het bovenstaande mutual exclusion probleem op zonder busy waiting door gebruik te maken van semaforen. Jouw oplossing dient geen last te hebben van de nadelen die je genoemd hebt bij onderdeel (d).

Opgave 4: Page replacement algoritmen

Ga in deze opgave uit van een virtueel memory systeem met slechts 5 pages per proces. We beschouwen een proces dat memory pages benaderd volgens de volgende "reference sequence".

ref=[R1, W2, R3, R4, W5, R7, R3, R2, W1, W4, R5, R7, W6, R1, W3, W2].

De notatie Rx geeft aan dat het proces page x leest (Read), terwijl Wx betekent dat het proces page x schrijft (Write). De sequence geeft dus weer dat het proces gedurende executie eerst page 1 leest, vervolgens page 2 schrijft, page 3 leest, etc.

(a) Wat is het aantal pagefaults als gebruik wordt gemaakt van een (theoretisch) optimaal pagefault algoritme? Leg uit hoe je aan dit antwoord komt. Geef voor ieder van de bovenstaande 16 page-referenties aan welke 5 pages in het fysieke geheugen aanwezig zijn.

(b) Ga uit van een FIFO page replacement algoritme. Geef voor ieder van de bovenstaande 16 page-referenties aan welke 5 pages in het fysieke geheugen aanwezig zijn. Wat is het aantal pagefaults?

(c) Ga uit van een LRU (Least Recently Used) page replacement algoritme. Geef voor ieder van de bovenstaande 16 page-referenties aan welke 5 pages in het fysieke geheugen aanwezig zijn. Wat is het aantal pagefaults?

(d) We beschouwen nu een variant van het NRU (Not Recently Used) page replacement algoritme. Veronderstel een virtueel geheugensysteem met een R(eferenced)- en een M(odified)-bit per memory page. Initieel zijn alle bits 0. Als een proces pagina x refereert (d.m.v. Rx of Wx) dan worden de beide bits op 1 gezet. Alle R-bits worden gereset (op 0 gezet) na elke

vijfde memory page referentie (m.a.w. bij W5, W4 en W3). In tegenstelling tot een standaard NRU-algoritme worden de R-bits dus niet gereset t.g.v. de clock interrupt. Geef voor ieder van de bovenstaande 16 page-referenties aan welke 5 pages in het fysieke geheugen aanwezig zijn. Wat is het aantal pagefaults?

Opgave 5: Deadlock preventie

In deze opgave beschouwen we een systeem met 1 resource en 4 processen (A, B, C en D). Van de resource is slechts een beperkte hoeveelheid beschikbaar, namelijk 22 eenheden.

Stel nu dat ieder proces bij aanvang (start van het proces) aan het besturingsysteem aangeeft hoeveel eenheden het van deze resource maximaal nodig zal hebben. Tevens is bekend dat een proces dat eenmaal alle gevraagde eenheden toegekend heeft gekregen op den duur zal termineren en zijn verkregen eenheden van de resource zal vrijgeven (teruggave aan het besturingsysteem).

In een besturingsysteem waarin dit het geval is kan deadlock voorkomen worden met behulp van het zgn. *Banker's algorithm*. Veronderstel dat de toestand uit de volgende tabel is bereikt.

Beschikbaar 6

Proces	Aanvraag	Toegekend
A	12	2
B	10	2
C	8	4
D	14	8

De bovenstaande tabel dient als volgt gelezen te worden. Er zijn nog 6 eenheden van de resource beschikbaar. Uit de eerste regel blijkt dat er 2 eenheden aan proces A zijn toegekend, terwijl dit proces maximaal 12 eenheden heeft aangevraagd.

(a) Een toestand heet *veilig* als het mogelijk is voor alle processen om te termineren (zonder deadlock). Laat zien dat de bovenstaande toestand veilig is.

(b) Ga uit van de situatie in de bovenstaande tabel. Leg uit dat een aanvraag van proces D voor 2 eenheden van de resource gehonoreerd kan worden. Geef de nieuwe toestand van het systeem na deze honorering.

(c) Ga opnieuw uit van de situatie in de bovenstaande tabel. Leg uit dat een aanvraag van proces A voor 3 eenheden van de resource niet gehonoreerd kan worden.

(d) Schrijf een C of Java routine dat bepaalt of een gegeven toestand veilig is. Kies zelf een geschikte datarepresentatie voor de bovenstaande tabel.

(e) Leg uit hoe de routine uit onderdeel (d) gebruikt kan worden door het besturingsysteem om te bepalen of aanvragen voor eenheden van de resource door de processen gehonoreerd kunnen worden. Schrijf in C of Java de bijbehorende routine.

(f) Het banker's algoritme gaat er vanuit dat a priori bekend is hoeveel processen meedoen aan de competitie om de gedeelde resource. Geef aan hoe het algoritme aangepast kan worden voor een situatie waarin deze veronderstelling niet langer geldt.