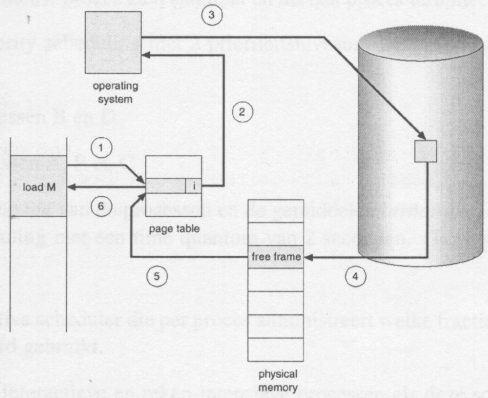
Tentamen Operating Systems

Woensdag 16 april 2008, 14.00-17.00

- 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 even veel 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.

Opgave 1: Virtueel geheugen

(a) Leg uit hoe virtueel geheugen werkt aan de hand van de 6 gemarkeerde punten.



(b) Het is mogelijk het geheugen van twee computers (die met een netwerkverbinding met elkaar verbonden zijn), virtueel één shared geheugen te laten simuleren d.m.v. virtueel geheugen technieken. Leg globaal uit hoe dit geïmplemnteerd zou kunnen worden. Wat moet je voor iedere memory page administreren?

(c) Een proces kan via `malloc` and `free` geheugen aanvragen en vrijgeven. Stel nu dat de enige actie die het besturingsysteem onderneemt het aanpassen van het valid-invalid-bit van de betreffende memory pages is. Geef commentaar op deze memory management strategie.

(d) Voor processen die veel geheugen gebruiken lijkt het handig om grote memory pages te gebruiken. Omgekeerd, voor processen die weinig geheugen gebruiken lijken kleine pages handiger. Deze redenering zou kunnen leiden tot de gedachte dat het een goed idee is om variabele pagegroottes te gebruiken. Leg uit of dit mogelijk is, en wat de voor en nadelen van een dergelijk systeem zijn.

1

Opgave 2: Scheduling

Gegeven zijn 5 processen die zich aanmelden aan het besturingsysteem op verschillende tijdstippen. Alle processen zijn volledig CPU-bound. We gaan 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
A	0	5
B	2	2
C	3	5
D	4	4
E	4	1

(a) Bepaal de volgorde van executie van de processen en de *gemiddelde wachttijd (waiting time)* voor ieder van de volgende systemen.

- Systeem met non-pre-emptive *First Come First Served* (FCFS) scheduling
- Systeem met non-pre-emptive *Shortest Job First* (SJF) scheduling
- Systeem met pre-emptive *Shortest Remaining Time Next* (SRTN) scheduling. Context-switching vindt alleen plaats als een nieuw proces zich aandient en als een proces termineert.

(b) We beshouwen nu priority scheduling met 2 prioriteitsniveaus. De bovenstaande processen hebben de volgende prioriteiten:

- hoge prioriteit: processen B en D
- lage prioriteit: processen A, E en C

Bepaal de volgorde van executie van de processen en de gemiddelde *turnaround time* voor een systeem met pre-emptive priority scheduling met een time quantum van 2 seconden. Ga er vanuit dat per klasse round robin wordt gescheduled.

(c) Beschouw een pre-emptive scheduler die per proces administreert welke fractie van zijn laatst toegekende tijdsquantum werkelijk werd gebruikt.

- Wat is het effect op interactieve en reken-intensieve processen als deze scheduler processen die een kleine fractie hebben besteed voorrang geeft?
- Wat is het effect op interactieve en reken-intensieve processen als deze scheduler processen die een grote fractie hebben besteed voorrang geeft?

Opgave 3: Page frame replacement
Ga in deze opgave uit van een virtueel memory systeem met slechts 4 page frames per proces. We beschouwen een proces dat memory pages benadert volgens de volgende "reference sequence".

ref=[1, 5, 4, 3, 1, 2, 6, 5, 2, 6, 1, 5, 3, 4, 2].

Deze sequence geeft weer dat het proces gedurende executie eerst page 1 gebruikt, vervolgens page 5, daarna page 4, etc. Een aantal pages is onderstreept. Dit speelt pas een rol in onderdeel (c).

(a) Bepaal het minimaal aantal pagefaults, m.a.w. het aantal pagefaults dat een optimaal page replacement algoritme voor dit proces genereert. Geef voor ieder van de bovenstaande 15 page-referenties aan welke pages in het fysieke geheugen aanwezig zijn. Leg uit wat de 'optimale' stategie is.

(b) Ga uit van een FIFO page replacement algoritme. Geef voor ieder van de bovenstaande 15 page-referenties aan welke pages in het fysieke geheugen aanwezig zijn. Wat is het aantal pagefaults?

(c) Veronderstel nu dat het "aging algoritme" wordt gebruikt voor page replacement (zie fig. 4-17 in het boek). De klok-tikken vinden plaats juist voordat de onderstreepte pages worden gerefereerd. We gaan uit van age-counters van 4 bits. Hoeveel pagefaults levert dit page replacement algoritme voor de bovenstaande reference sequence?

(d) Veronderstel een page-replacement strategie waarbij voor elke pagereferentie de tijd van deze referentie wordt bijgehouden. Op het moment dat een 'victim' aangewezen moet worden voor replacement zal de page met de laagste tijd ('oudste' page) gekozen worden. Waarin verschilt dit van FIFO-replacement? Wat zijn de voordelen/nadelen van zo'n systeem?

(e) Stel dat een systeem probeert het aantal pagefaults te minimaliseren via de volgende strategie. Zodra een proces een memory-page refereert die niet aanwezig is, wordt het proces door de scheduler in de blocked state geplaatst en krijgt een ander proces de beschikking over de CPU. Als ten gevolge van deze strategie alle processen in de blocked state zijn geraakt, zal een willekeurig proces gekozen worden voor een 'doorstart' (uiteraard vind er dan wel gewoon page-replacement plaats). Geef commentaar op dit 'lazy' page-replacement algoritme.

Opgave 4: Deadlock preventie
In deze opgave beschouwen we een systeem met 3 resources (R_0, R_1 en R_2) en 3 processen (P_0, P_1, P_2). Van iedere resource is slechts een beperkte hoeveelheid beschikbaar, namelijk 4 eenheden van R_0, 3 eenheden van R_1 en 5 eenheden van R_2.

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

Resource	Totaal	Beschikbaar	Aanvraag			Toegekend		
R_i	aantal	aantal	P_0	P_1	P_2	P_0	P_1	P_2
0	4	2	3	1	1	1	0	1
1	3	1	3	2	1	2	0	0
2	5	0	2	3	4	2	3	0

De bovenstaande tabel dient als volgt gelezen te worden. Uit de eerste regel blijkt dat er 4 eenheden van resource R_0 in het systeem aanwezig zijn, waarvan op dit moment nog 2 eenheden beschikbaar. De reeds toegekende 2 eenheden zijn gelijk verdeeld over de processen P_0 en P_2 (m.a.w. ieder 1 eenheid). Proces P_0 heeft maximaal 3 eenheden van resource R_0 aangevraagd, en kan in de toekomst dus nog ten hoogste 2 eenheden aanvragen. Proces P_2 zal geen eenheden van resource R_0 meer aanvragen.

(a) Een toestand heet *veilig* als het mogelijk is voor alle processen om te termineren (zonder deadlock). Leg uit waarom de bovenstaande toestand veilig is.

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

(c) Leg uit hoe de routine uit onderdeel (b) gebruikt kan worden door het besturingsysteem om te bepalen of aanvragen voor resources door de processen gehonoreerd kunnen worden.

(d) Ga uit van de situatie in de bovenstaande tabel. Leg uit waarom een aanvraag van proces P_2 voor 1 eenheid van resource R_1 wel of niet gehonoreerd kan worden.

(e) Schets een variant van het banker's algoritme waarbij op ieder moment zich nieuwe processen kunnen aandienen (dus niet alle informatie is vooraf bekend). Een proces moet nog steeds bij aanvang aangeven hoeveel het van ieder resource nodig heeft. Je hoeft dit niet uit te programmeren. Een globale uitleg is voldoende.

Opgave 5: Unix system programming
(a) Wat is de uitvoer van het onderstaande programma? Is de uitvoer altijd hetzelfde?
(b) Leg kort de structuur van het programma uit. Maak een eenvoudige schets van de processen en geef de communicatiepatronen aan.
(d) Leg kort uit wat de volgende programmaregels bewerkstelligen:

- regels 30-31
- regels 32-34
- regel 19
- regel 21-22
- regels 13, 24, en 38-39

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <unistd.h>
 4 #include <sys/wait.h>
 5
 6 void procPing(int fromPong, int toPong) {
 7    int msg = 0;
 8    while (msg < 5) {
 9       printf ("Ping\n");
10       write(toPong, &msg, sizeof(int));
11       read(fromPong, &msg, sizeof(int));
12    }
13    exit(EXIT_SUCCESS);
14 }
15
16 void procPong(int fromPing, int toPing) {
17    int msg;
18    do {
19       read(fromPing, &msg, sizeof(int));
20       printf ("...Pong\n");
21       msg++;
22       write(toPing, &msg, sizeof(int));
23    } while (msg<5);
24    exit(EXIT_SUCCESS);
25 }
26
27 int main(int argc, char *argv[]) {
28   int chanPing[2], chanPong[2], status;
29
30   pipe(chanPing);
31   pipe(chanPong);
32   if (fork() == 0) {
33    procPing(chanPong[0], chanPing[1]);
34   }
35   if (fork() == 0) {
36    procPong(chanPing[0], chanPong[1]);
37   }
38   waitpid(-1, &status, 0);
39   waitpid(-1, &status, 0);
40   return EXIT_SUCCESS;
41 }
```


5