Tentamen Operating Systems

Maandag 17 juni 2013, 9:00-12:00

- **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 gebruik van een rekenmachine is wel toegestaan.
- Het tentamen bestaat uit 5 opgaven. Iedere opgave is 20 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[20pt]: File systems

(a)[5pt] We beschouwen een eenvoudig UNIX file system met slechts 16 disk blokken. Een file system checker bepaalt van ieder blok hoe vaak het in gebruik is en tevens hoe vaak het in de free-list voorkomt. De checker heeft de volgende counters bepaald:

Block Number	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
In use	1	1	0	1	0	1	2	1	1	0	0	1	1	1	0	0
Free	0	0	0	0	2	0	0	0	0	1	1	0	1	0	1	1

Zitten er fouten in dit file system? Zo ja, welke fouten zijn dit en welke acties moet de checker ondernemen om ze te repareren?

(b)[5pt] Er bestaan twee methoden voor toekenning van schijfruimte aan files: *contiguous* (aaneengesloten) en *indexed* (geïndexeerd). Vergelijk deze methoden op de volgende aspecten:

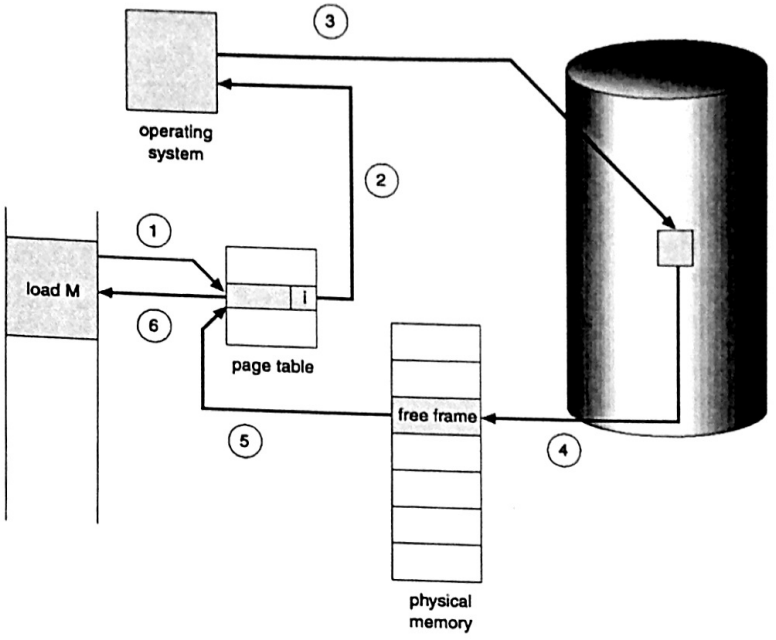
1. efficiency sequential access
2. efficiency random access
3. benodigde aantal seeks
4. disk fragmentatie
5. mogelijkheden voor file append

We gaan in de rest van deze opgave uit van een UNIX-like file system. In dit file system bevat iedere inode precies 10 entries, ieder ter grootte van 4 bytes. De block-size is 4096 bytes.

(c)[5pt] Neem aan dat alle entries van een inode pointers zijn die direct wijzen naar data blokken. Wat is de maximale file size van dit file system? Leg uit.

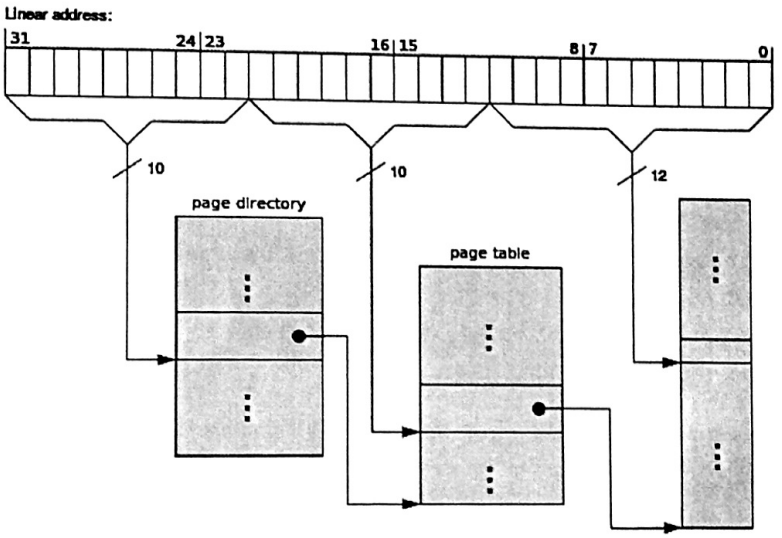
(d)[5pt] Neem nu aan dat 7 van de 10 entries direct wijzen naar data blokken. Van de overige drie entries wijst er één naar een *single indirect block*, één naar een *double indirect block* en één naar een *triple indirect block*. De indirect blokken hebben ook een grootte van 4096 bytes en worden volledig gevuld met pointers naar data blokken. Wat is nu de maximale file size van dit file system?

Opgave 2[20pt]: Virtual Memory en Page frame replacement algorithms
(a)[4pt] Leg uit hoe virtueel geheugen werkt aan de hand van de 6 gemarkeerde punten in de onderstaande figuur.



(b)[2pt] Een virtueel memory systeem gebruikt 32-bit addressing en maakt gebruik van memory frames ter grootte van 4KB. Leg uit hoe een logisch geheugenadres wordt vertaald in een fysiek geheugen adres op dit systeem als een eenvoudige (single-level) pagetable wordt gebruikt.

(c)[2pt] In de onderstaande figuur wordt gebruik gemaakt van 32-bits addressering en een two-level page table. Leg uit hoe in dit systeem een 32-bits logisch adres wordt vertaald in een fysiek adres.



Ga in de rest van deze opgave uit van een demand memory paging systeem met slechts 4 page frames per proces. We beschouwen een proces dat memory pages benadert volgens de volgende "reference sequence".

$$\text{ref}=[4, 0, 2, 3, 6, 6, 1, 3, 1, 4, 0, 5, 3, 1, 0, 2, 7, 2, 7, 5].$$

Deze sequence geeft weer dat het proces gedurende executie eerst een adres uit page 4 refereert, vervolgens een adres uit page 0, daarna page 2, etc.

(d)[4pt] Bepaal het aantal pagefaults dat een *optimaal* page replacement algoritme voor dit proces genereert. Geef na ieder van de bovenstaande 20 page-referenties aan welke pages in het fysieke geheugen aanwezig zijn. Waarom wordt in de praktijk dit optimum niet bereikt?

(e)[4pt] Ga uit van een FIFO page replacement algoritme. Geef voor ieder van de 20 page-referenties aan welke 4 pages in het fysieke geheugen aanwezig zijn. Wat is het aantal pagefaults?

(f)[4pt] Ga uit van een LRU (Least Recently Used) page replacement algoritme. Geef voor ieder van de 20 page-referenties aan welke 4 pages in het fysieke geheugen aanwezig zijn. Wat is het aantal pagefaults?

Opgave 3[20 pt]: Scheduling

(a)[3pt] Een user-level thread library maakt het mogelijk om binnen een UNIX-proces verscheidene threads te maken zonder dat de kernel dit kan zien. M.a.w. voor de kernel bestaat er slechts één proces. Leg uit wat de consequenties hiervan zijn voor een multi-threaded proces dat bestaat uit een aantal threads die CPU-bound zijn en een thread dat I/O-bound is.

(b)[3pt] Een reeks batch jobs waarvan de verwachte executietijden bekend zijn worden gelijktijdig aangeboden aan een rekencentrum. Wat is de juiste strategie voor de operateur om de *average turn-around time* te optimaliseren?

Gegeven zijn 6 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	4
B	1	3
C	5	6
D	9	4
E	11	6
F	14	2

(c)[14pt] Bepaal de volgorde van executie van de processen en de gemiddelde *turn-around time* voor ieder van de volgende systemen. Laat duidelijk zien hoe je tot jouw antwoord bent gekomen.

[2pt] Systeem met non-pre-emptive *First Come First Served* (FCFS) scheduling

[2pt] Systeem met non-pre-emptive *Shortest Job First* (SJF) scheduling

[5pt] Systeem met pre-emptive *Shortest Remaining Time Next* (SRTN) scheduling met een time quantum van 3 seconden.

[5pt] Systeem met pre-emptive *Round-Robin Scheduling* met een time quantum van 3 seconden.

Opgave 4[20pt]: Proces Synchronisatie
(a)[2pt] Waarom worden vaak *spinlocks* gebruikt in plaats van normale *mutexes*?

(b)[2pt] Leg uit wat het verschil is tussen een *semafoor* en een *mutex*.

(c)[4pt] Implementeer zelf semaforen (in Java, Python, C++, C of pseudocode) waarbij je gebruik mag maken van een routine `TSL(rx,lock)`. Deze routine implementeert de z.g.n. *Test and Set Lock* operatie. Je mag gebruik maken van busy-waiting.

(d)[4×3pt] We beschouwen verschillende implementaties van een eindige FIFO-buffer met één producer en één consumer. De functie `producer()` is gegeven:

```
#define N 100 /* number of slots in the buffer */
semaphore mutex = 1, empty = N, full = 0;

void producer() {
  int item;
  while (1) {
    item = produce_item();
    down(&empty);
    down(&mutex);
    insert_item(item);
    up(&mutex);
    up(&full);
  }
}
```

We beschouwen nu de onderstaande 4 implementaties voor de consumer. Beoordeel de verschillende implementaties. Geef van iedere implementatie aan of deze wel of niet correct is. Leg uit waarom wel/niet.

```
void consumer1() {
  while (1) {
    down(&mutex);
    down(&full);
    item=remove_item();
    up(&mutex);
    up(&empty);
    consume_item(item);
  }
}
```

```
void consumer2() {
  while (1) {
    down(&mutex);
    down(&full);
    item=remove_item();
    up(&empty);
    up(&mutex);
    consume_item(item);
  }
}
```

```
void consumer3() {
  while (1) {
    down(&full);
    down(&mutex);
    item=remove_item();
    up(&mutex);
    up(&empty);
    consume_item(item);
  }
}
```

```
void consumer4() {
  while (1) {
    down(&full);
    down(&mutex);
    item=remove_item();
    up(&empty);
    up(&mutex);
    consume_item(item);
  }
}
```

Opgave 5[20pt]: Unix system programming

(a)[2pt] De uitvoer van het onderstaande programma zal bij iedere executie waarschijnlijk verschillend zijn. Leg uit waarom de uitvoer per executie kan verschillen. Geef twee voorbeelden van mogelijke uitvoeren.

(b)[6×3pt] Leg kort (maar duidelijk) uit wat de code van de volgende regels bewerkstelligen:

- regel 13
- regels 18-21
- regel 23
- regel 26
- regel 30 (i.h.b. hoe kan `n<1` gelden?)
- regel 38

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <string.h>
4   #include <unistd.h>
5   #include <sys/wait.h>
6   #include <fcntl.h>
7
8   int main (int argc, char *argv[]) {
9     int fd[2], status;
10    int flags, n;
11    char msg[256];
12
13    if (pipe(fd) < 0) {
14      fprintf (stderr, "Could not make pipe\n");
15      return (EXIT_FAILURE);
16    }
17
18    flags = fcntl (fd[0], F_GETFL);
19    fcntl (fd[0], F_SETFL, flags | O_NONBLOCK);
20    flags = fcntl (fd[1], F_GETFL);
21    fcntl (fd[1], F_SETFL, flags | O_NONBLOCK);
22
23    if (fork() != 0) {
24      close(fd[0]);  /* fd[0] is used for reading */
25      sprintf (msg, "Hello world!\n");
26      write (fd[1], msg, 256); /* fd[1] is used for writing */
27    } else {
28      close(fd[1]);
29      do {
30        n = read (fd[0], msg, 256);
31        if (n<1) {
32          putchar ('#');
33          fflush(stdout);
34        }
35      } while (n<1);
36      printf (msg);
37    }
38    waitpid(-1, &status, 0);
39    return EXIT_SUCCESS;
40  }
```