NTNU
Kunnskap for en bedre verden

KANDIDAT

10058

PRØVE

TDT4186 1 Operativsystemer

Emnekode	TDT4186
Vurderingsform	Hjemmeeksamen
Starttid	21.05.2022 07:00
Sluttid	21.05.2022 11:00
Sensurfrist	15.06.2022 21:59
PDF opprettet	15.08.2022 13:37

Title page

Oppgave	Tittel	Oppgavetype
ℹ	Title page	Informasjon eller ressurser

Processes

Oppgave	Tittel	Oppgavetype
1	Unix processes (4 points)	Tekstfelt
2	Fork (4 points)	Tekstfelt
3	Behavior of fork (4 points)	Tekstfelt

System calls and the Unix shell

Oppgave	Tittel	Oppgavetype
4	System calls (4 points)	Tekstfelt
5	File offsets (4 points)	Tekstfelt
6	Pipelines (4 points)	Tekstfelt

Synchronization

Oppgave	Tittel	Oppgavetype
7	Synchronization (6 points)	Tekstfelt
8	Semaphore implementation (4 points)	Tekstfelt

Memory allocation

Oppgave	Tittel	Oppgavetype
9	Buddy algorithm (4 points)	Tekstfelt

10	Fragmentation (4 points)	Tekstfelt

Virtual memory

Oppgave	Tittel	Oppgavetype
11	Page tables (4 points)	Tekstfelt
12	Address translation (4 points)	Tekstfelt
13	Translation lookaside buffer (6 points)	Tekstfelt

Processor scheduling

Oppgave	Tittel	Oppgavetype
14	Virtual Round Robin (4 points)	Tekstfelt
15	Scheduling theory (6 points)	Tekstfelt

Disk and I/O scheduling

Oppgave	Tittel	Oppgavetype
16	Disk scheduling (4 points)	Tekstfelt

File systems

Oppgave	Tittel	Oppgavetype
17	Inodes (3 points)	Tekstfelt
18	Free space management (3 points)	Tekstfelt
19	Optimization of lseek (4 points)	Tekstfelt

1 Unix processes (4 points)

The following program is given. What is the output that is printed when the line of code marked with **/* OUTPUT LINE */** is executed? Explain your answer! (Assume that all required header files are included)

```
int value = 23;
int main()
{
    pid_t pid;
    pid = fork();

    if (pid > 0) {
        wait(NULL);
        printf("value = %d\n", val); /* OUTPUT LINE */
        return 0;
    } else if (pid == 0) {
        value += 42;
        return 0;
    }
}
```

Fill in your answer here

Assuming there is a typo in the task, such that val=value. The program wil ouput 23. This is because processes dont share memory (they are duplicates, but with different a different address space), and any changes done in the child process, will not change the value in the parent.

2 Fork (4 points)

Including the initial parent process, how many processes are created by the following program? Assume that all calls to fork() succeed. Explain your answer!

```
#include <stdio.h>
#include <unistd.h>

int main()
{
   /* fork a child process */
   fork();
   /* fork some more */
   if ( ! fork() ) fork();
   return 0;
}
```

Fill in your answer here

First the program forks, such that we have two processes. !fork() will be caleed in both, and it will only spawn a new processes if it is in the child. This means that we spawn a total of 5 processes.

3 Behavior of fork (4 points)

When a process returns **fork**, the system call returns twice – once in the parent process that called fork and once in the newly generated child process. The only difference is the value returned by fork.

Explain how the operating system can achieve returning different values to identical copies of a program.

Hint: think about how resources are shared between parent and child process.

Fill in your answer here

4 System calls (4 points)

Consider the system call
count = write(fd, buffer, nbytes);

Can this system call return any value in **count** other than **nbytes**?

If yes, give two possible reasons.
If no, explain why not.

Fill in your answer here

Yes.
1)
It can return -1 which indicates an error.

Otherwise write only returns the amount of bytes written, which can be less than nbytes. This can happen if write could not write all the bytes to the requested file, f.e. there was no more space in the medium.

5 File offsets (4 points)

Consider a file that was opened in a Unix process and has the file descriptor *fd*.
This file contains the following sequence of bytes:
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31
The following system calls are executed:
lseek(fd, 3, SEEK_SET);
read(fd, &buffer, 4);
Assume that buffer is an array of 10 bytes (uint8_t) with its contents initially set to all null (0) bytes.

Which contents does *buffer* have after the call to read returns?

Fill in your answer here

lseek, sets the file fileoffset of fd to absolute byte 3
offset is now on the byte which represents 7 (offsets starts at index 0)

read, reads 4 bytes from fd into the buffer (from the offset)

buffer = [7,11,13,17,null,null,null,null,null,null]

6 Pipelines (4 points)

A Unix shell implements **pipelines** to implement sequences of commands such as
cat file | grep pattern | sort

Explain how the Unix shell builds pipelines internally by giving the sequence of system calls executed and indicating over which communication channel the communication between two (or more) processes in a pipeline works.

Fill in your answer here

7 Synchronization (6 points)

The following three functions of a program run in separate threads each and print some prime numbers. All three threads are ready to run at the same time.

Use synchronization using the semaphores S1, S2 and S3 and wait/signal operations on the semaphores to ensure that the program outputs all characters in **reverse alphabetic order**.

Insert appropriate wait or signal operations in the code lines indicated with // SYNC and give the correct initial values for the semaphores. Give the completed functions and the initial values for the semaphores.

Hint: Note that it might not be required to add a wait or signal operations in all of the places indicated.

```
Semaphore S1 = ___; Semaphore S2 = ___; Semaphore S3 = ___;
f1() {
  // SYNC
  printf("e");
  // SYNC
  printf("c");
  // SYNC
}

f2() {
  // SYNC
  printf("d");
  // SYNC
  printf("b");
  // SYNC
}

f3() {
  // SYNC
  printf("f");
  // SYNC
  printf("a");
  // SYNC
}
```

Fill in your answer here

```
Semaphore S1 = 1;
Semaphore S2 = 0;
Semaphore S3 = 0;

f1() {
  wait(S2);
  printf("e");

  signal(S3);
  signal(S2);
```

```
  wait(S2);

  printf("c");
  signal(S3);

}



f2() {
  wait(S3)
  printf("d");

  signal(S2)
  signal(S3)

  wait(S3);

  printf("b");
  signal(S1);
}



f3() {
  wait(S1)
  printf("f");

  signal(S1);
  signal(S2);
  wait(S1);

  printf("a");

}
```

8 Semaphore implementation (4 points)

Johnny Hacker has not bothered to watch all videos of TDT4186 during his studies at NTNU.
In his side job working for the famous operating systems company Simocroft, he now has to implement semaphores for their new operating system Dinwows 13.
Johnny implements the **wait** function of the OS in C as follows:

```
wait(Semaphore *S) {
   while (S->value == 0);  /* wait until the value of the semaphore is > 0 */
   S->value--;
}
```

Unfortunately, after his supervisor examined the code, Johnny was fired and now has to program web apps in Javascript. You are the person now responsible for fixing the problem(s) in Johnny's code.
Explain the problem(s) of this **wait** implementation Johnny wrote and why a problem with this implementation may be hard to find and debug. Assume that a pointer to a valid semaphore structure is passed to the wait function.

Fill in your answer here

A correct implementation will first lock the mutex with pthread_mutex_lock, then use the pthread_cond_wait on its condition, and finally unlock with pthread_mutex_unlock before returning.

9 Buddy algorithm (4 points)

A computer has 16 MB of memory. Its operating system allocates memory according to the **Buddy algorithm** with the smallest possible block size of 1 MB.

Initially, the memory is all unused.

The following allocation and free requests are made:

1. Allocation A of 2 MB
2. Allocation B of 6.5 MB
3. Allocation C of 1.3 MB
4. Allocation D of 42 kB
5. Allocation E of 1.8 MB
6. Free C
7. Free D
8. Allocation F of 3.5 MB
9. Allocation G of 512 kB

If an allocation cannot be fulfilled, it is rejected and no changes to the overall memory allocation will be performed.

Indicate the memory allocation after each of the given allocation/free requests as follows:

Example: 8 MB memory, minimal block size = 1 MB

———— Initial state without allocations
AAAA|—— Allocation A: 4 MB
AAAA|B|–|— Allocation B: 1 MB
AAAA|B|–|— Allocation C: 4 MB: not possible
——|B|–|— Free A

Fill in your answer here

0. ———— Initial state without allocations
1. AA|—|——|———— Allocation A: 2MB
2. AA|—|——|BBBBBBB– Allocation B: 7MB
3. AA|CC|——|BBBBBBB– Allocation C: 2MB
4. AA|CC|D|–|—|BBBBBBB– Allocation D: 1MB
5. AA|—|D|–|—|BBBBBBB– Free C
6. AA|—|——|BBBBBBB– Free D
7. AA|—|FFFF|BBBBBBB– Allocation F: 4MB
7. AA|G|–|FFFF|BBBBBBB– Allocation G: 1MB

10 Fragmentation (4 points)

Explain the difference between internal fragmentation and external fragmentation in two to three sentences.

Which one occurs in paging systems?

Which one occurs in systems using segmentation?

Give an example for the kind of fragmentation in systems using paging and in systems using segmentation.

Fill in your answer here

Interal fragmentation happens when we split memory into fixed size blocks, where the requested memory is less than the memory allocated. External fragmentation happens when we split memory into varying size blocks, and empty blocks are scattered all over.

Interal fragmentation happens in paging systems.
External fragmentation happens in systems when segmentation is used.

11 Page tables (4 points)

A computer system has 32 bit logical addresses and a 4 kB (4096 byte) page size.

The system supports up to 512 MB (512*1024*1024 bytes) of physical memory.

How many entries are there in each of the following? Give your calculation.

1. A conventional, single-level page table
2. An inverted page table

Fill in your answer here

1.
Adresses / page size = page table entires

32 bits / 4kb = x
2^32 / 2^12 = 2^20 entries

2.
physical adress space / page size = inverted page table entires
512MB / 4kB = x
2^29 / 2^12 = 2^17 entries

12 Address translation (4 points)

Consider a computer system using a page table with 12 bit virtual and physical addresses and 256 byte page size.

The page table for this system is given below. A dash (–) indicated that there is no current mapping to a page frame for the given page.

Convert the following virtual memory addresses (*given in hexadecimal*) to their equivalent physical memory addresses if possible. If not possible, indicate a page fault.

1. 9EF
2. 111
3. 700
4. 0FF

Page table contents

Page	Page frame
0	1
1	2
2	C
3	A
4	–
5	4
6	3
7	–
8	B
9	0

Fill in your answer here

1. 0x0EF
2. 0x211
3. Page fault (Page 7 has no mappings)
4. 0x1FF

13 Translation lookaside buffer (6 points)

The following piece of code multiplies two matrices:

```
int a[1024][1024], b[1024][1024], c[1024][1024];
multiply()
{
  unsigned int i, j, k;
  for(i = 0; i < 1024; i++)
     for(j = 0; j < 1024; j++)
        for(k = 0; k < 1024; k++)
           c[i][j] += a[i,k] * b[k,j];
}
```

Assume the following:

- the executable code for this function fits in one memory page
- the stack also fits in one memory page
- The size of an **int** variable is 4 bytes

Compute the number of translation lookaside buffer (TLB) misses if the page size is 4096 bytes and the TLB has 8 entries and uses LRU (least recently used) as replacement policy.

Explain your calculation, giving only a number is not sufficient.

Fill in your answer here

14 Virtual Round Robin (4 points)

The following four processes are added to the ready list of an operating system one after the other (arrival time 0 for all processes).

Assume that the CPU- and I/O-bursts are of identical length and known to the scheduler.

Each process first executes a CPU-burst and then an I/O-burst. The base time unit is 1 ms.

Process	CPU burst	I/O burst
A	7	2
B	2	2
C	7	5
D	2	5

Apply the **Virtual Round Robin (VRR)** scheduling algorithm to schedule the four processes. The scheduler assigns a *time slice* of 3 ms to each process.

Give the CPU and I/O allocation for the first 30 ms.

To simplify the task, assume the following:

- process switches take 0 ms and can thus be ignored
- several I/O bursts can run in parallel

Give your solution as in the following **example**:

"C" = process uses the CPU, "I" = process performs I/O, "–" = process is ready

A: CCCEE---CCC
B: ——CCEEE—— ...
C: ———CCC——

Fill in your answer here

1. A---------
2. AB--------
3. ABC-------
4. ABCD------

15 Scheduling theory (6 points)

1. Explain the important advantage that Virtual Round Robin scheduling (VRR) has compared to the regular Round Robin scheduling approach. Which implementation difference between the two scheduling algorithms is responsible for this advantage?
2. If a too long time slice is chosen for Round Robin scheduling, which other scheduling algorithm is approximated? Explain why.

Fill in your answer here

16 Disk scheduling (4 points)

Assume a magnetic disk with 8 tracks. After each second read request (starting from L1), the I/O scheduler receives additional read requests which are grouped (requested) together (L2 and finally L3).

Initially, the read/write head of the disk is at track 0.

1. Give the I/O scheduling order that would be performed according to the **SSTF (shortest seek time first)** algorithm for the following request sequence:
L1 = {2,3,1,1}, L2 = {6,5}, L3 = {0,7}

2. Assume the same magnetic disk as above. The read requests in L1 are already known to the disk scheduler. After three completed requests, the requests in L2 arrive, and after three additional requests (i.e., after the sixth request), the requests in L3 arrive.
Initially, the read/write head of the disk is at track 0.
Give the I/O scheduling order that would be performed according to the **elevator algorithm**:
L1 = {1,7,4,2}, L2 = {4,0,6}, L3 = {5,2}

Fill in your answer here

1.
{0,1,1,2,3,5,6,7}

2.
{1,7,4,4,5,6,2,2,0}

17 Inodes (3 points)

A small Unix system uses a file system with a variant of inodes called *minodes* (minimal inodes).

Assume that the size of a disk block is 4 kB (4096 bytes) and each disk block can hold 2048 disk block addresses.

A *minode* has the following structure:

- Entries 0–6 are direct disk block pointers
- Entry 7 is a single indirect pointer
- Entry 8 is a double indirect pointer
- Entry 9 is a triple indirect pointer

What is the maximum possible size of a file (in bytes) on this file system using *minode*s as described above? Explain your answer!

Fill in your answer here

File size = block_size * (direct_pointers + (entries of single indirect blocks) + (entries of double indirect blocks)^2 + (entries of triple indirect blocks)^3)

Direct pointers = 4kB * 7 = 28 kB = 0,028672 MB
Indirect single pointers = (2^12) * 2^11= 2^23 = 8,388608 MB
Indirect double pointers = (2^12) * (2^22) = 2^34 = 17 179,8692 MB
Indirect triple pointers = (2^12) * 2^33 = 2^45 = 35 184 372,1 MB

Maximum file size = 35201560.3865 MB = 35.2015604 TB

18 Free space management (3 points)

Most file systems store information about unallocated (free) data blocks on a disk in a special data structure, such as a free space bitmap or linked list.

Is it really necessary to store information about the unallocated disk blocks in a free list? Explain why or why not.

Fill in your answer here

19 Optimization of lseek (4 points)

The **lseek** system call positions the pointer inside a file at which the following read or write operation will take place.

Consider the file systems, the inode-based Unix System V file sytem and MS-DOS FAT16.

Which of the two allows for a more efficient implementation of the lseek system call? Explain your answer!

Hint: Examine the required blocks the operating system needs to read on the disk to find the block which corresponds to the position given by lseek!

Fill in your answer here