https://folk.ntnu.no/michaeng/tdt4186_21/
michael.engel@ntnu.no

Example Exam Questions

Notice: This exam contains 10 questions covering a wide range of the topics we discussed in this course. The real exam will also have 10 questions of similar complexity to the ones in this example exam.

1 Processes (10 points)

1.1. Process execution order (6 p.)

How many times does the following program print “Hello World”? Draw a simple tree diagram to show the parent-child hierarchy of the spawned processes.

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

int main() {
  int i;
  for (i = 0; i < 3; i++)
    fork();
  printf("Hello World\n");
  return 0;
}
```

1.2. Process execution order (4 p.)

Consider the following example program. List **all legal outputs** this program may produce when executed on a Unix system. The output consists of strings made up of multiple letters.

```c
#include <unistd.h>
#include <sys/wait.h>

// W(A) means write(1, "A", sizeof "A")
#define W(x) write(1, #x, sizeof #x)

int main() {
  W(A);
  int child = fork();
  W(B);
  if (child)
    wait(NULL);
  W(C);
}
```

2 Shell pipelines (10 points)

In a very special Unix shell, the command `A <> B` means that the standard output of A should be connected to the standard input of B and the standard input of A should be connected to the standard output of B (so this creates a bidirectional shell pipeline).
The following C code is an excerpt of a shell that should implement this behavior. It contains a number of errors.
Indicate the location of each error (give the line number), shortly describe the problem and propose fixes that realize the desired behavior.

```
1  #define WRITE  2
2  #define READ   0
3
4  int pipe1[2], pipe2[1];
5
6  pipe(pipe1);
7  pipe(pipe2);
8
9  if(fork() == 0) {
10     dup2(pipe1[WRITE], STDOUT);
11     dup2(STDIN, pipe2[READ]);
12     close(pipe1[READ]);
13     close(pipe1[WRITE]);
14     exec(A);
15 }
16
17 if(fork() != 0) {
18     dup2(pipe1[READ], STDIN);
19     dup2(pipe2[WRITE], STDOUT);
20     close(pipe1[WRITE]);
21     close(pipe2[READ]);
22     exec(B);
23 }
24
25 close(pipe1[READ]);
26 close(pipe1[WRITE]);
27 close(pipe2[READ]);
28 close(pipe2[WRITE]);
```

3 Threading (10 points)

The following code that uses two separate threads describes a famous bug in MySQL, a widely used open source SQL database server.

3.1. Explain the problem in the following code (5 p.)

```
1  // Thread 1:
2
3  if (thd->proc_info) {
4    ...
5    fputs(thd->proc_info, ...);
6    ...
7  }
8
9  // Thread 2:
10 thd->proc_info = NULL;
```

3.2. Does the following code fix the problem? Explain your answer. (5 p.)

```
1  Semaphore proc_info_lock = 1;
2
3  // Thread 1:
4
5  if (thd->proc_info) {
6      ...
7      wait(&proc_info_lock);
8      fputs(thd->proc_info, ...);
9      signal(&proc_info_lock);
10     ...
11 }
12
13 // Thread 2:
14
15 wait(&proc_info_lock);
16 thd->proc_info = NULL;
17 signal(&proc_info_lock);
```

4 Deadlocks (10 points)

Consider the following code:

```
1  Semaphore L1=1, L2=1, L3=1;
2
3  // Thread 1:
4  wait(L1);
5  wait(L2);
6  // critical section requiring L1 and L2 locked.
7  signal(L2);
8  signal(L1);
9
10 // Thread 2:
11 wait(L3);
12 wait(L1);
13 // critical section requiring L3 and L1 locked.
14 signal(L1);
15 signal(L3);
16
17 // Thread 3:
18 wait(L2);
19 wait(L3);
20 // critical section requiring L2 and L3 locked.
21 signal(L3);
22 signal(L2);
```

Initially, all three mutexes are initialized as “not locked”. Also assume that the threads can execute in any arbitrary interleavings.

4.1. Can there be a problem when executing this multithreaded code? If yes, show an interleaving resulting in the problem. If no, explain why not. (5 p.)

4.2. If there is a problem, propose a fix (Note that each critical section requires two different locks, you cannot change this assumption). (5 p.)

5 Memory management (10 points)

5.1. **Buddy allocation** (4 p.)

Use dynamic memory allocation according to the buddy method. The second row of each table below shows the *current* allocation of memory with a total memory size of 32 MB and a block size of 2 MB. Fill in markings for the allocated blocks according to the request given along with each table.

Hint: The four examples are *independent* of each other. If a memory allocation *can not* be performed, mark the respective line accordingly.

Scenario 1: Process C requests 3 MB.

0	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30
A	A	A	A							B	B				

Scenario 2: Process D requests 12 MB.

0	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30
A															

Scenario 3: Process E requests 14 MB.

0	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30
		B	B									A	A		

Scenario 4: Process F requests 7 MB.

0	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30
A	A	A	A	B	B										

5.2. **Fragmentation** (3 p.)

The memory allocation algorithms discussed in the course can result in *external* and *internal* fragmentation. Explain the key points for both terms and name a placement strategy (but not the same strategy twice!) for which the respective fragmentation effect shows up.

5.3. **Virtual memory** (3 p.)

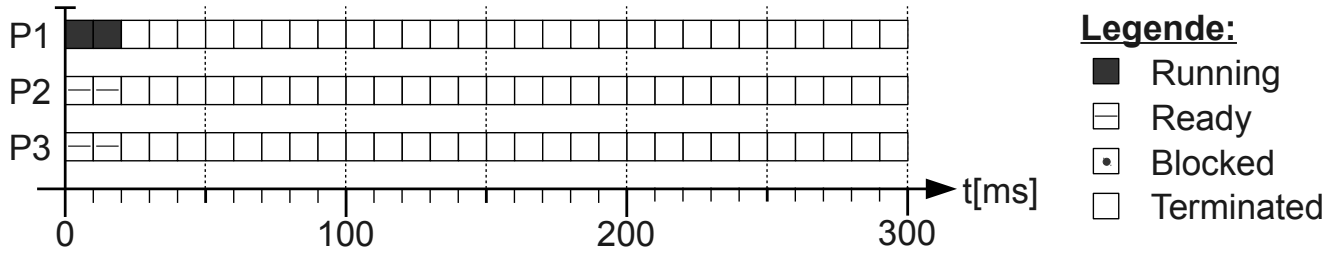
Describe shortly what a TLB is and why it is required to build efficient computers using virtual memory.

6 Scheduling (10 points)

An operating system has three processes P1, P2 and P3.
The processes are activated in the order P1, P2, P3 and are all ready to run at time $t = 0$. The compute times in milliseconds (ms) of each process and the starting time of I/O operations *relative to the compute time* are given in the following table:

Process	P1	P2	P3
Compute time	70	110	90
I/O start	20	30	60
I/O duration	80	30	50

Use the following Gantt diagram to show how the three processes P1, P2, and P3 are processed if the scheduling strategy is *round robin* with a chosen time slice of 40 ms. Each process performs exactly one I/O operation with the starting time and duration given in the table. The process switching time can be neglected. Mark the related process state for each process and time step according to the patterns shown next to the diagram.
Hint: The first two time units are already filled in.



7 File I/O (10 points)

7.1. File reads (5 p.)

Assume the initial contents of file myfile2 and the execution of the following block of code.

The initial contents of the file `myfile2` are:

```
foo#bar#baz
```

What is the output printed by the program?

```
1  int fd = open("myfile2", O_RDWR);
2
3  if (fd <= 0) {
4    printf("Error");
5    exit(-1);
6  }
7
8  if(lseek(fd, 5, SEEK_SET) < 0) {
9    printf("Error");
10   exit(-1);
11 }
12
13 char buffer[100];
14
15 if(read(fd, buffer, 3) != 3) {
16   printf("Error");
17   exit(-1);
18 }
19
20 buffer[3] = 0;
21
22 printf("%s\n", buffer);
23 close(fd);
```

7.2. File writes (5 p.)

Assume the initial contents of file `myfile` and the execution of the following block of code.

What are the final contents of myfile?

The initial contents of the file `myfile` are:

```
Tweedle
```

Assume that there are no errors with permissions and that there are no newlines.

```
1  int fd = open("myfile", O_WRONLY | O_APPEND | O_TRUNC) ;
2
3  if (fd < 0) {
4    printf("Error");
5    exit(-1);
6  }
7
8  write(fd, "dee", 3);
9  close(fd);
```

8 File systems (10 points)

A Unix filesystem has 2 kB (1 kB = 1024 bytes) blocks and 4 byte disk addresses. Each inode contains 10 direct entries, one singly-indirect entry and one doubly-indirect entry.

8.1. Calculate the maximum possible file size in this file system (3 p.)

8.2. Suppose half of all files are exactly 1.5 kB in size and the other half of all files are exactly 2 kB. What fraction of the disk space would be wasted? (Consider only the blocks used to store data) (4 p.)

8.3. Based on the same condition as in the previous item above, does it help to reduce the fraction of wasted disk space if we change the block size to 1 kB? Justify your answer. (4 p.)

9 Security (10 p.)

9.1. Unix login (5 p.)

The Unix login program checks whether or not a user has entered the correct password before taking on the user's ID and executing the user's shell.

Explain how it can check passwords this way without storing the user's actual password on the system.

9.2. System subversion (5 p.)

Suppose an attacker breaks into a Unix machine, obtains root (superuser) privileges, and manages to keep them for a long period of time (e.g., many months).

What might the attacker do to learn users' real passwords, even if they are *not* stored on the system?

10 Storage systems (10 p.)

Many RAID devices now ship with the following options:

- RAID 0 - data striped across all disks
- RAID 1 - each disk mirrored
- RAID 5 - striped parity

Given a system with 8 disks of 1 TB each, answer the following questions:

10.1. For each of the three RAID levels, how much *usable* storage does the system provide? (3 p.)

10.2. Assume a workload consisting only of small reads, evenly distributed. Assuming that no verification is performed on reads, what is the throughput of each level assuming that a single disk can perform 100 reads/sec? (2 p.)

10.3. Assume a workload consisting only of small writes, evenly distributed. Again, calculate the throughput assuming that a single disk can execute 100 writes/sec. (2 p.)

10.4. For each level, what is the minimum number of disks that may fail before data may be lost? (1.5 p.)

10.5. For each level, what is the minimum number of disks that must fail to **guarantee** data loss? (1.5 p.)