Written Exam

DM510: Operating Systems (Spring 2025)

June 13, 2025

This exam consists of 7 pages in total, including this one, with 4 topics and several questions in each topic. There are 60 points in total. The number of points is given for each question. Along with your answers, also explain how you arrived there, in a clear, but concise way.

Topic 1: Concurrency (12 points)

Recall that Amdahl's law states that

$$\text{speedup} \leq \frac{1}{\mathrm{S} + \frac{1-\mathrm{S}}{\text{cores}}}$$

where S be the ratio of sequential operations to all operations.

The program `Fcast` computes weather forecasts for a Danish news channel. This computation currently takes 9 minutes on a single CPU core. Management is unsatisfied with this and suggests to decrease the time to 3 minutes by using 3 cores instead of one. You estimate that 20% of the operations are sequential.

1.a) ***[3 points]*** Explain intuitively why the suggestion by management will not suffice for their goal. Then using Amdahl's law calculate the minimum number of cores necessary to reduce the time to 3 minutes. What is the maximum speedup that could be achieved by adding more cores?

The following code implements a semaphore using the atomic instructions `compare_and_swap()` and `increment()`:

```
1  void wait(int *sem) {
2    while (true) {
3      int old = *sem;
4      if (old > 0) {
5        if (compare_and_swap(sem, old, old - 1))
6          break;
7      }
8    }
9  }
10
11 void signal(int *sem) {
12   increment(sem)
13 }
```

1.b) ***[2 points]*** For simplicity, your teammate suggests to replace lines 5-6 of the semaphore implementation by `decrement(sem)`, which atomically subtracts 1 from `*sem`. Would the solution still be correct?

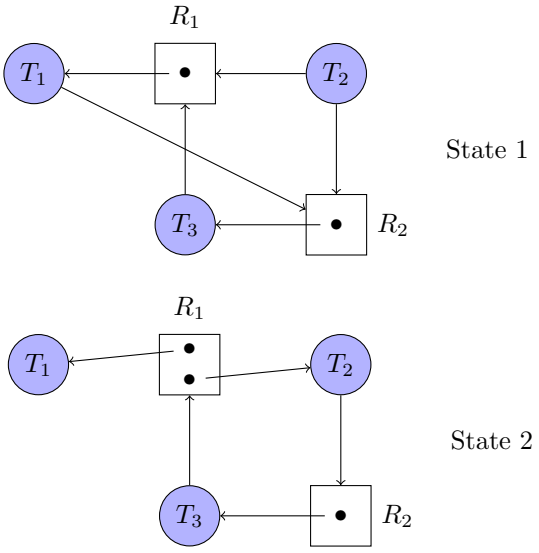
1.c) ***[2 points]*** Recall that mutex is equivalent to a semaphore that takes only values 0 and 1. Your teammate notes that in an implementation of a mutex the `release()` operation (corresponding to `signal()` here) does not need to be made atomic and suggests to replace line 12 by `*sem = *sem + 1`. Is the suggestion correct for general semaphores?

1.d) ***[2 points]*** To reduce risk of race conditions, why don't compilers automatically translate all increments of variables (for example, `i++`) into the atomic increment instructions?

1.e) ***[3 points]*** As in Project 1, critical sections are sometimes implemented by temporarily disabling interrupts. Explain this approach and state two drawbacks of it.

Topic 2: Deadlocks (16 points)

The following figure shows states of systems where threads request or hold resources. Threads are depicted as cycles and resources are depicted as rectangles, with dots representing the instances of this resource. An arrow from a thread to a resources describes that the thread requests this resource, an arrow from a resource instance to a thread describes that the instance is currently held by the thread.



2.a) ***[3 points]*** Which of the states above are deadlocked and why (or why not)?

Recall, the system call `wait()` suspends the calling process until a child process that was previously created with `fork()` terminates. In the deadlock formalism of resources and threads, we may think of each process P_i having a personal resource R_i with one instance for each child process. A child process of P_i holds one instance of R_i and releases the instance when it terminates. When P_i calls `wait()` this corresponds to waiting for R_i.

2.b) ***[3 points]*** Unless another shared resource is involved, `fork()` and `wait()` alone cannot lead to a deadlock. Explain this by showing that one of the necessary conditions for deadlocks cannot be satisfied.

2.c) ***[3 points]*** A single mutex without other shared resources involved cannot result in a deadlock. Combined with `fork()` and `wait()`, however, already a single mutex suffices to create a deadlock. Give an example in pseudocode and explain it.

To avoid ambiguities, your example should not call `fork()` while holding a mutex.

Consider the following snapshot of a system:

	Allocation			Max		
	A	B	C	A	B	C
T_1	2	1	1	2	3	2
T_2	0	0	1	3	0	1
T_3	0	1	0	0	1	1
T_4	1	3	2	2	3	3

$$\text{Available} = (0, 1, 1)$$

2.d) ***[4 points]*** Determine if the system is in a safe state using the subroutine from Banker's algorithm. If the state is safe, illustrate the order in which the threads may complete. Otherwise, illustrate why the state is unsafe.

2.e) ***[3 points]*** Suppose that in the state above T_1 requests $(0, 0, 1)$. Determine with Banker's algorithm whether this request can be granted.

Topic 3: Main Memory (18 points)

We consider a CPU architecture with 16-bit addresses and a naive page table implementation. The first 3 bits of the address specify the page or frame number. The current page table of a process contains the following entries (in binary):

page	frame
000	010
001	110
010	-
011	111
100	100
101	-
110	-
111	101

3.a) ***[2 points]*** How large is a page or frame in the architecture above?

3.b) ***[2 points]*** Based on the page table above, translate the following logical addresses into physical addresses:

0110111010110000
0011010000010110
1111111110000000

3.c) ***[4 points]*** Describe the following terms in 2-3 sentences each: page fault, least-recently-used replacement (LRU), inverted page table, translation look-aside buffer

In the following example, process A wants to create a process B and pass a large chunk of raw data as a parameter. Since it would not be feasible to pass it as a command line argument, process A instead wants to write the data into memory, which process B then should read.

The following shows an implementation using the `fork()` system call.

```
char* data;
...
pid = fork();
if (pid == 0)
{ /* process B */
  consume(data);
  ...
}
else
{ /* process A */
  free(data);
  ...
}
```

3.d) ***[2 points]*** Describe in 2-3 sentences what the system call `fork()` does using the example above.

3.e) ***[2 points]*** Discuss race conditions: does the behavior of the program depend on whether `consume(data)` executes before `free(data)`?

A pointer can be cast into an integer (describing the raw logical address). Below is an attempt at an alternative solution for the previous problem. In this solution we write two entirely separate programs for process A and B and process B is not a child process of A. Instead, process A is started from the terminal and prints out the address of `data`:

```
char* data;
...
printf("%lld\n", (long long int)data);
...
```

The output of this may for example be 106934719226528. Then concurrently to process A, we start process B from another terminal and pass the previous number (106934719226528) as a command line argument. Process B then executes the following code:

```
char* data;
int main(int argc, char** argv) {
  /* atoll converts string to long long int */
  data = (char*) atoll(argv[1]);
  consume(data);
  ...
}
```

3.f) ***[4 points]*** The approach above does not work as intended. Explain this using logical and physical addresses and page tables.

3.g) ***[2 points]*** Describe what shared memory is and (on a high level) how this could be used to repair the example above.

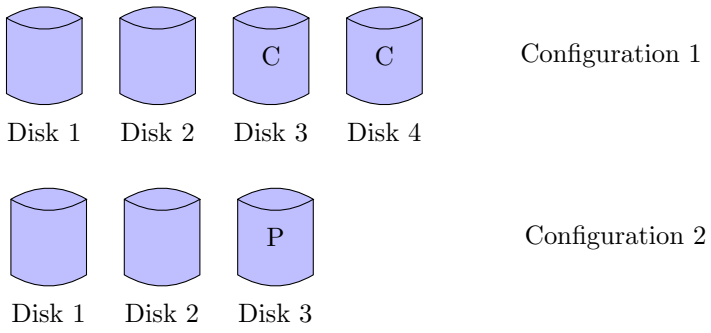
Topic 4: Storage and File Systems (14 points)

Consider a disk drive with cylinders numbered from 0 to 1999. The drive is currently serving a request at cylinder 401 and the previous served request was at cylinder 350. The queue of pending requests in order of arrival is: 30, 1910, 500, 37, 1400, 783.

4.a) ***[2 points]*** In which order would the SCAN and FCFS algorithms serve the requests? What distance (in number of cylinders) would the disk arm move in each algorithm?

4.b) ***[2 points]*** Explain why SCAN and C-SCAN are used with hard disk drives (HDDs), but not with flash drives (NVMs/SSDs).

The following picture shows two RAID configurations: Configuration 1 uses RAID 1 (mirrored disks), where Disk 3 mirrors Disk 1 and Disk 4 mirrors Disk 2. Configuration 2 uses RAID 4 (block-interleaved parity), where Disk 3 is the parity disk.



4.c) ***[2 points]*** For each configuration: list all combinations of disks that would lead to data loss if they fail simultaneously.

4.d) ***[3 points]*** Consider the basic implementation of a file system via FUSE as in Project 3 (without potential extensions). Name three ways in which this implementation differs from practical file systems.

4.e) ***[3 points]*** Describe in 2-3 sentences what contiguous allocation is in the context of a file system. Name one advantage and one disadvantage.

4.f) ***[2 points]*** In 2-3 sentences, describe an alternative to contiguous allocation.