Written Exam

DM510: Operating Systems (Spring 2026)

June 4th, 2026

This exam consists of 8 pages in total, including this one, with 5 topics and several questions in each topic. There are 66 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: I/O Systems (13 points)

Linux exposes a device `/dev/mem`, which you have used in the user mode hack of the first project. It gives users direct access to the raw memory via the standard file API: When you open `/dev/mem` and read the nth byte, you will get the value at **physical** address n of the main memory. Writing to the device works accordingly.

1.a) ***[2 points]*** Explain what the following C code does:

```
1 char* a = (char*)42;
2 *a = 5;
```

Compare to writing to the 42th byte of `/dev/mem`. Would the result be the same? Justify your answer.

Solution: The code writes byte value 5 to the logical address 42. The page table would map this to some physical address that the user program has no control over. Therefore the actual physical address that is being written to is not 42. Therefore the result would be different from writing to the 42th byte of `/dev/mem`.

1.b) ***[3 points]*** There is a driver that implements the functionality of `/dev/mem`.

1. Explain in 2-3 sentences what a driver is.
2. Sketch how the driver implementation for `/dev/mem` would look like.
3. Could a similar functionality be provided by a shared library purely in user space instead of a driver?

Solution:

1. A driver provides an interface to hardware, for example, by handling the protocol for interacting with device controllers. A driver executes in kernel mode and thus has access to priviledges instructions, which user programs do not.

2. There is a `read` and `write` callback function in the driver `/dev/mem`, which takes a physical address, a buffer, a size parameter. It would remap the physical address to the kernel's address space and then write the buffer contents to it or vice versa.

3. No. A user space library cannot map an arbitrary physical address to the user program's address space. This is a priviledges operation.

1.c) ***[3 points]*** For `/dev/mem` suggest sensible permissions:

1. Specify who the owner of the device should be.
2. Specify access rights in string form (for example, `rwx`) and numerical (for example, 7) for the **owner** and for **other users** (neither owner nor owner group).
3. Name a vulnerability if `/dev/mem` had inappropriate access rights.

Solution: The owner should be the superuser (root). The access rights for the owner should be `rw-` (6) and for other users `---` (0). If other users could read or write to `/dev/mem`, this would allow them to read the data of other processes or even modify their data, allowing code injections, sniffing and similar attacks.

In the following, consider a serial output pin that transmits data by changing between high and low signal at a rate in the order of 10-100 KHz. Assume that the pin signal can be set to high or low using memory-mapped I/O.

1.d) ***[3 points]*** For the serial output pin:

1. Explain what memory-mapped I/O is and how it would work in this example.
2. One could implement the transmission using a user program that writes to `/dev/mem` or in a sophisticated driver. Which alternative could achieve a higher transmission rate? Justify your answer. *Hint: Consider system calls.*

Solution: In memory-mapped I/O, specific physical memory addresses are mapped to registers of I/O devices. By reading or writing to these addresses, the CPU can communicate with device controllers, for example sending commands or polling the status. In this example, the CPU would change the serial pins signal by writing to the a specific memory address.

Writing to `/dev/mem` incurs significant overhead, since one does not directly write to the physical memory address, but instead a file system call is invoked, a context switch to the kernel occurs and the kernel driver then performs the

memory instruction. If the entire implementation is in kernel space then this overhead is avoided and the implementation could likely achieve a higher transmission rate.

1.e) ***[2 points]*** Another technical solution for the serial output pin would be a sophisticated device controller managing the output pin together with direct-memory access (DMA). Explain what DMA is and explain its advantage compared to previous solutions.

Solution: DMA works by giving the device controller direct access to main memory. The CPU can then for example give the controller a pointer to data that it should transmit. The controller can then transmit all data by itself (without interaction from the CPU from there on). This allows the CPU to work on other tasks instead, whereas in programmed I/O the CPU would be busy throughout the transmission.

Topic 2: Scheduling (18 points)

Consider the following CPU bursts by processes P1-P5.

Process	Burst Time	Arrival Time
P1	11	2
P2	2	1
P3	3	4
P4	5	10
P5	12	5

2.a) ***[4 points]*** For each of the following algorithms produce a schedule for the given CPU bursts and compute their turnaround times:

1. First-Come-First-Serve
2. Shortest-Job-First
3. Round-Robin with time quantum $q = 2$

Solution: FCFS:

Process	Execution	Completion Time	Turnaround
P2	1-3	3	2
P1	3-14	14	12
P3	14-17	17	13
P5	17-29	29	24
P4	29-34	34	24

SJF:

Process	Execution	Completion Time	Turnaround
P2	1-3	3	2
P1	3-14	14	12
P3	14-17	17	13
P4	17-22	22	12
P5	22-34	34	29

RR:

Process	Execution	Completion Time	Turnaround
P2	1-3	3	2
P1	3-5		
P3	5-7		
P5	7-9		
P1	9-11		
P3	11-12	12	8
P5	12-14		
P4	14-16		
P1	16-18		
P5	18-20		
P4	20-22		
P1	22-24		
P5	24-26		
P4	26-27	27	17
P1	27-29		
P5	29-31		
P1	31-32	32	30
P5	32-34	34	29

2.b) ***[2 points]*** Name an advantage of First-Come-First-Serve over Shortest-Job-First.

Solution: FCFS cannot lead to starvation of jobs.
Other advantages possible: for example (arguably) fairness.

2.c) ***[2 points]*** Suppose that a process is in the CPU scheduler's waiting queue due to a blocking I/O operation. From the operating systems view, what happens from when the I/O operation finishes until the process resumes execution?

Solution: When the I/O operation finishes, the kernel is notified for example through an interrupt. The kernel then moves the process from the waiting queue to the ready queue. At some point later, the CPU scheduler will allow the process to execute on the CPU. When that happens, it will perform a context switch, recovering the state (register values, etc.) that the CPU had the last time the process has been executing.

2.d) ***[5 points]*** Consider a single core system running the following two functions in two seperate threads.

1. Explain in your own words what the program does.
2. Explain three different potential behaviors of the program depending on the CPU scheduler.
3. Explain how the program could be changed to guarantee correct and predictable behavior.

```
1  int* buf = NULL;
2  int i = 0;
3
4  void thread1() {
5    int j = 0;
6    buf = malloc(128 * sizeof(int));
7    while (buf != NULL) {
8      if (i > j) {
9        printf("%d", buf[j]);
10       j++;
11     }
12   }
13 }
14
15 void thread2() {
16   while (buf == NULL)
17     ;
18   while (i < 128) {
19     buf[i] = i;
20     i++;
21   }
22   free(buf);
23   buf = NULL;
24 }
```

Solution:

Program description. thread1 allocates space for a buffer. Once this happens, thread2 starts to write the numbers 1-128 into the buffer. Afterwards, thread2 frees the buffer. thread1 waits for the inputs of thread1 and prints them.

Behavior. One possible behavior is that the numbers 1-128 are printed to stdout and both threads exit normally. This happens if between line 19 (with $i = 127$) and line 22 a context switch from thread2 to thread1 occurs and thread1 then prints all remaining numbers until 127.

Another possible behavior is that only some of the numbers are printed to stdout and both threads exit normally. This happens if thread2 executes `free(buf)` and/or `buf = NULL` before thread1 prints out all numbers and when context switching to thread1, the thread is after reading `buf[j]`, but before the next check of the condition of the while loop.

The third possible behavior is a memory error leading to abnormal termination of the process. This happens if thread2 executes `free(buf)` and/or `buf = NULL` before all numbers are printed and when context switching to thread1, the thread is after checking the condition of the while loop at line 7, but before reading `buf[j]`. The thread will then dereference the memory address $\text{NULL} + j \cdot \text{sizeof(int)}$.

Fixes. One way of fixing the problem is to make thread1 (instead of thread2) call `free(buf)` and `buf = NULL` after exiting its while loop. The condition for the while loop of thread1 would also need to be changed to `while (j < 128)`.

Another fix would be to use a semaphore where thread1 signals after it exits its while loop and thread2 waits before line 22.

2.e) ***[3 points]*** For each of the following scheduler requests, name one advantage (or reason for issuing them) and one disadvantage (or risk):

1. Increasing the time quantum
2. Disabling interrupts
3. Aging a process (increasing its priority)

Solution: Increasing time quantum: Reason could be to reduce the number of context switches and thus increase the CPU utilization. A disadvantage would be that very small CPU burst (finished in one time quantum) need to wait longer.

Disabling interrupts: This could be a synchronization mechanism (implementing a critical section). Disadvantages include lower responsiveness of the system.

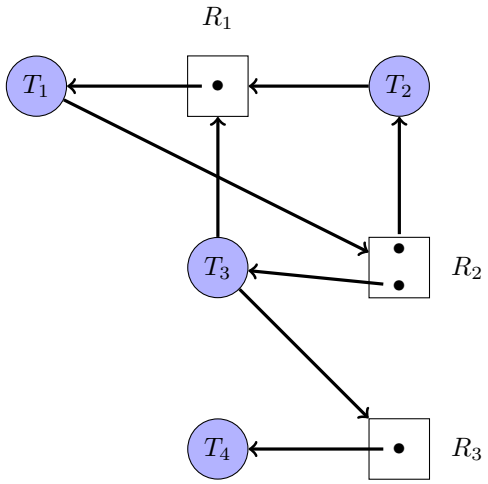
Aging a process: This is done to avoid starvation when a process has been waiting very long in a low priority queue. A disadvantage is that higher priority processes may then be delayed by this (less important) process.

2.f) ***[2 points]*** Why are the SCAN and C-SCAN algorithms used for scheduling requests to hard disk drives (HDDs), but not for non-volatile memory devices (NVMs) or in CPU scheduling?

Solution: SCAN and C-SCAN are intended to minimize the physical movement of the read-write head in HDDs (as distance of cylinders). An NVM has no cylinders, no moving parts and does not benefit from locality of accesses. Similarly in CPU scheduling there is no notion of locality or cylinders so it does not make sense to use SCAN or C-SCAN.

Topic 3: Deadlocks (16 points)

The following figure shows the state of a system 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.



3.a) ***[3 points]*** Is the state above in a deadlock? Argue why or why not.

Solution: It is deadlocked. None of the threads T_1, T_2, T_3 can continue before an instance of R_1 or R_2 is released. However, all these instances are held by T_1, T_2, T_3 themselves.

3.b) ***[2 points]*** What action could be taken if a system is in a deadlock?

Solution: To recover from a deadlock, one could manually terminate threads or processes and release their resources until the system is no longer in a deadlock.

Consider the following snapshot of a system:

	Allocation			Max		
	A	B	C	A	B	C
T_1	2	3	1	2	3	2
T_2	0	0	1	0	4	1
T_3	0	1	0	5	2	1
T_4	1	1	2	5	3	3

$$\text{Available} = (4, 1, 1)$$

3.c) ***[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.

Solution: Initialization:

	Need		
	A	B	C
T_1	0	0	1
T_2	0	4	0
T_3	5	1	1
T_4	4	2	1

Available = $(4, 1, 1)$

T_1 can complete:

	Need		
	A	B	C
T_2	0	4	0
T_3	5	1	1
T_4	4	2	1

Available = $(6, 4, 2)$

T_2 can complete:

	Need		
	A	B	C
T_3	5	1	1
T_4	4	2	1

Available = $(6, 4, 3)$

T_3 can complete:

	Need		
	A	B	C
T_4	4	2	1

Available = $(6, 5, 3)$

Finally, T_4 can also complete. This means the state is safe.

3.d) ***[3 points]*** Suppose that in the state above T_4 requests $(4, 0, 1)$. Determine with Banker's algorithm whether this request can be granted.

Solution: After granting the resource, the state would change to

	Allocation			Max		
	A	B	C	A	B	C
T_1	2	3	1	2	3	2
T_2	0	0	1	0	4	1
T_3	0	1	0	5	2	1
T_4	5	1	3	5	3	3

Available = $(0, 1, 0)$

We test if this is a safe state as before. Initialization:

	Need		
	A	B	C
T_1	0	0	1
T_2	0	4	0
T_3	5	1	1
T_4	0	2	0

Available $= (0, 1, 0)$

`Available` is not large enough to guarantee that any of the threads can complete. Thus the state is unsafe. Therefore, the request cannot be granted.

3.e) ***[4 points]*** Consider the following program code. There are threads $T_0, \ldots, T_{n-1}$ which run function `thread` with parameter `id` equal to $0, 1, \ldots, n-1$. There is a global array `msg_queues`, which contains one message queue for each thread. Each message queue has a mutex that can be locked and unlocked.

```
1  void thread(int id) {
2    while (true) {
3      lock(&msg_queues[id]);
4      if (has_message(&msg_queues[id])) {
5        // a message has been received
6        msg message;
7        int id2;
8        // writes to id2 the id of the sender
9        // and to message the message received
10       receive_msg(&msg_queues[id], &id2, &message);
11       msg answer = produce_answer(id2, msg);
12       lock(&msg_queues[id2]);
13       send_msg(&msg_queues[id2], answer);
14       unlock(&msg_queues[id2]);
15     } else {
16       msg message;
17       int id2;
18       produce_msg(&message,&id2);
19       lock(&msg_queues[id2]);
20       send_msg(&msg_queues[id2], message);
21       unlock(&msg_queues[id2]);
22     }
23     unlock(&msg_queues[id]);
24   }
25 }
```

1. Explain how the program can get stuck in a deadlock.
2. Modify the code in a way that has the same functionality, but cannot get into a deadlock. Explain why your variant is deadlock free.

Solution: A deadlock similar to the dining philosophers can occur when two threads i and j first acquire their own message queue (line 3) and then try to acquire each other's message queue (line 12 or 19). Neither thread will release their message queue so both are stuck. The same example can happen with more than two threads in a cycle.

An easy fix is to release ones own message queue before acquiring the other. Then the hold-and-wait condition is not satisfied and therefore there cannot be a deadlock:

```
void thread(int id) {
  while (true) {
    lock(&msg_queues[id]);
    if (has_message(&msg_queues[id])) {
      // a message has been received
      msg message;
      int id2;
      // writes to id2 the id of the sender
      // and to message the message received
      receive_msg(&msg_queues[id], &id2, message);
      unlock(&msg_queues[id]);

      msg answer = produce_answer(id2, msg);
      lock(&msg_queues[id2]);
      send_msg(&msg_queues[id2], answer);
      unlock(&msg_queues[id2]);
    } else {
      unlock(&msg_queues[id]);

      msg message;
      int id2;
      produce_msg(&message,&id2);
      lock(&msg_queues[id2]);
      send_msg(&msg_queues[id2], message);
      unlock(&msg_queues[id2]);
    }
  }
}
```

Topic 4: File Systems (11 points)

4.a) ***[4 points]*** Explain the following terminology (each in 1-3 sentences) and their relationship:

1. disk drive
2. partition
3. mounting
4. file system

Solution: A disk drive is a device that contains data. The data on it is divided into partitions. A partition can contain a file system, which structures its data in files, directories, etc. To use the file system, the partition needs to be mounted to a specific directory of the root file system.

4.b) ***[2 points]*** In principle, one could replace every file system implementation in Linux by a FUSE file system.

1. Compared to the usual kernel space implementation in Linux, would this resemble a microkernel or a monolithic approach in operating system architecture?
2. Name an advantage that such a design would have.

Solution: In a microkernel architecture, as much functionality as possible is placed in user space rather than kernel space. In a monolithic kernel, it is the opposite. Since a file system in FUSE is in user space, this corresponds to the microkernel approach.

Advantages of implementing the file system in user space are easier development and debugging, as well as less severe consequences in case of bugs or security flaws.

4.c) ***[2 points]*** In Project 3 you were allowed to restrict your file system to only files of the same size. How do variable size files complicate the implementation of a file system?

Solution: Space allocation for files of uniform size is very easy and there is no external fragmentation. With variable size files, one needs to use more sophisticated allocation methods and data structures, for example, linked allocation, indexed allocation, etc.

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

Solution: In indexed allocation a file has an index block that contains pointers to the data blocks that contain the data of the file.

Advantage: A file can have variable size. Random access is fairly efficient.

Disadvantage: The size of the file is still bounded by the number of pointers a block can hold. Through the index block there is an added indirection, which results in overhead.

Topic 5: Virtual Machines (8 points)

5.a) *[2 points]* Name two reasons for using virtual machines.

Solution: Virtual machines can be used by cloud providers to achieve high server utilization by running several users in virtual machines on the same physical machines. Virtual machines can be used for compatibility reasons to run applications that were developed for an operating system different from the host.
There are other sensible answers as well.

5.b) *[3 points]* Describe the role of interrupts in the implementation of a type 2 hypervisor.

Solution: Type 2 hypervisors use trap-and-emulate: When a guest tries to execute a priviledged instruction, this results in an interrupt, since the guest is not running in kernel mode. The interrupt is handled by the hypervisor, which then emulates the priviledged instruction.

5.c) *[3 points]* In which way is a Docker container similar to a virtual machine? In which way does it differ?

Solution: A docker container provides similar functionality to a virtual machine in the sense that it can isolates different containers from each other and from the host system and that it runs them within an environment different from the host. A docker container does not run a guest kernel and therefore is not considered a virtual machines. It uses Linux features (cgroups, namespaces) to implement isolation.