Western Norway University of Applied Sciences

Department of Computer science, Electrical engineering and Mathematical sciences

Examination in:	DAT103 – Computers and Operating Systems
Day of examination:	12 December 2022
Time of examination:	9:00 – 13:00
Permitted aids:	Calculator
Language:	English

This problem set consists of 10 pages, excluding the appendix.
You can answer in either English or Norwegian.

Good Luck!
Dag and Violet

Problem 1 – Multiple Choice Questions (24%)

For each of the multiple choice questions below, select at most one choice.

1.1 Which one of the following is a control and status register?

(a) Memory address register
(b) Instruction register
(c) Memory buffer register
(d) All of the above
(e) None of the above

Solution: All of the above □

1.2 Which one of the following statements is **incorrect**?

(a) Multiprogramming is possible in a single processor system
(b) Multiprogramming is possible in a multiprocessor system
(c) Multiprocessing is possible in a single processor system
(d) Multiprocessing is possible in a multiprocessor system
(e) All of the above
(f) None of the above

Solution: Multiprocessing is possible in a single processor system □

1.3 Which one of the following scheduling algorithms is in general optimal for minimising average waiting time?

(a) Round robin
(b) First come first served
(c) Shortest job first
(d) All of the above
(e) None of the above

Solution: Shortest job first □

1.4 Which one of the following scheduling algorithms are *non-preemptive*?

(a) Round robin
(b) First come first served
(c) Earliest deadline first
(d) Shortest-remaining-time-first
(e) None of the above

Solution: First come first served □

1.5 A *page fault* occurs when ...

(a) A TLB miss occurs
(b) An interrupt occurs
(c) The requested page found in the main memory and is dirty
(d) The requested page found in the main memory and is clean
(e) None of the above

Solution: None of the above □

1.6 Consider a multiprocessor system that uses symmetric multiprocessing (SMP). Which one of the following is correct about multiple processor scheduling in this system?

(a) Each processor in the system is self scheduling
(b) Each processor may have its own private queue of ready processes, and has a scheduler to examine the ready queue and select a process to execute
(c) All processors may be in a common ready queue, and each processor has a scheduler to examine the ready queue and select a process to execute
(d) All of the above
(e) None of the above

Solution: All of the above □

1.7 Consider a computer system that support logical cache. Which one of the following is **correct**?

(a) The logical cache locates between the memory management unit (MMU) and main memory
(b) The logical cache store data using main memory physical address
(c) The processor has direct access to the logical cache
(d) All of the above
(e) None of the above

Solution: The processor has direct access to the logical cache □

1.8 Consider two processes, one for a web browser and one for word processor, competing for CPU time. Which scheduler will typically be responsible to schedule the two processes to the CPU?

(a) The long-term scheduler
(b) The mid-term scheduler
(c) The short-term scheduler
(d) Depends on the operating system

Solution: The short-term scheduler □

1.9 What is the most common way to utilise operating system functionality from a user program?

(a) Directly through system calls
(b) Indirectly through system libraries
(c) Implementing it from scratch without involving the operating system
(d) User programs typically do not need operating system functionality

Solution: Indirectly through system libraries □

1.10 You have developed some software in C for a Linux computer based on x86. Which other platforms will it be "easy" to port this to?

(a) Any computer based on x86
(b) Any operating system based on POSIX
(c) Any computer based on x86 running any operating system based on POSIX
(d) Any computer running any operating system

Solution: Any operating system based on POSIX □

1.11 What is meant by "programming in hardware"?

(a) Old term for loading the program from a file
(b) The program is stored on a hard disk
(c) The program is stored on a punch card
(d) The program is in the form of hardware

Solution: The program is in the form of hardware □

1.12 What is the "modern" way to transfer data inside a computer?

(a) Parallel bus point-to-point using packets
(b) Serial point-to-point connection using packets
(c) Parallel bus point-to-point without packets
(d) Serial point-to-point connection without packets
(e) None of the above

Solution: Serial point-to-point connection using packets □

Problem 2 – Miscellaneous (14%)

2.1 You are developing a software that needs to compress large sets of data in the background at the same time as the user is interacting with a GUI. This can be achieved by either starting a new thread or starting a new process to handle this task. What are the differences between either letting a thread or a process handle this?

Solution: When a new thread is is started, it will share almost everything with the parent: code, memory and files. Typically, the only a Thread Control Block and stack memory. This makes it very fast to create and destroy threads. However, when a new process is started it is completelly independent from the parent - it does not share neither memory nor any other resources. It is issentially a different "program" as far as the operating system is concerend. This also makes starting and stoping processes more costly than threads. In princliple, both can be used to compress the data in

the backgrund. Depending on which thread model is used, using a thread may have implications for whether the thread can execute truly in parallel with the parent, or only concurently. Since a thread will share memory with the parent, it will be easy to pass data between the thread and the parent. For processes on the other hand, it will be necessary to use for example shared memory or pipes to achieve this. □

2.2 In the problem above, the code for compressing data seems to have some issue that is causing it to crash from time to time. Will this influence your decision to use a thread or a process to handle this task? Explain your answer.

Solution: Since a thread will share "all" resources with the parent, it also means that a critical bug in the thread can typically not only crash the thread, but also the parent. A process on the other hand will be much more immune to this, as it is completelly separated from the parent. If a child process crashes, it may be restared from the parent (asuming it implements methods to handle such situations). Although a thread is cheaper to create than a process, a process may still be advantageous if fault protection is needed. □

2.3 Assume 10 multithreaded programs are running concurrently on a computer system with 16 cores. Assume further that the multithreaded programs are written using *many-to-one* threading model, and in total have 100 user threads. What is the maximum number of cores that will be utilised to run the 10 programs?

Solution: 10 □

2.4 Which one(s) of the following can be part of a CPU?

(a) Program counter
(b) I/O controller
(c) Direct memory access (DMA)
(d) L1 cache
(e) Heap
(f) Stack pointer

Solution:
Program counter
L1 cache
Stack pointer
□

2.5 Let ℓ_1, ℓ_2 and ℓ_3 be locks, and `x` a shared `int` variable initialised to `10`. Consider three processes P_1, P_2 and P_3 that concurrently run the following pieces of code:

```
1  // Run by P1 //
2  acquire(ℓ1);
3  x++;
4  release(ℓ1);
```

```
5  // Run by P2 //
6  acquire(ℓ2);
7  x++;
8  release(ℓ2);
```

```
9  // Run by P3 //
10 acquire(ℓ3);
11 x++;
12 release(ℓ3);
```

List all possible values that `x` could have when all three processes have finished.

Solution: 11, 12, 13 □

Problem 3 – Paging (11%)

3.1 Consider the following table:

Page number (i)	Frame number (j)	Frame starting address (k)
0	12	49152
1	3	12288
2	7	28672
3	10	40960
4	2	8192
5	1	4096
6	0	0
7	9	36864

Each entry of the table represents a mapping from Page i to Frame j, where Frame j starts from address k in the physical memory. For example, the first entry refers to Page 0 is mapped to Frame 12 that starts from address 49152 in the physical memory.

Assume the page size is 4 KB[1]. Write down, in decimal, the *page number*, the *offset* and the *physical address* for the logical addresses:

(a) 20515

Solution: page number: 5, offset: 35, physical address: 4131 (4096+35) □

(b) 3812

Solution: page number: 0, offset: 3812, physical address: 52964 (49152+3812) □

3.2 Consider a computer system with a logical address space of 65536 pages, which is mapped onto a physical memory of 2048 frames. Assume the page size is 2 KB.

(a) What is the size of a frame?

Solution: 2KB □

(b) How many bits should the logical address at least have?

Solution: 27 bits: 65536 pages = 2^{16} pages, 2KB = 2^{11} bytes □

(c) Assume further that each table entry is 32 bits long. What is the maximum size of the physical memory the system can address?

Solution: $2^{32} \times 2^{11} = 2^{43}$ bytes $(8TB)$ □

Problem 4 – Assembly and Addressing Modes (8%)

4.1 Consider the assembly code fragment in Listing 1.

```
1 push dword 4
2 push dword 3
3 push dword 2
4 mov eax,[esp+4]
5 mov ebx,[esp+8]
6 mov ecx,[esp]
7 add eax, ebx
```

Listing 1

Assume all the instructions in Listing 1 have been executed, and the stack grows towards lower addresses.

[1] 1 KB is 1024 bytes

(a) What is the value at the address stored in `esp`?
Solution: 2 ☐

(b) What is the value at the address stored in `esp+8`?
Solution: 4 ☐

(c) Does register `eax` now hold a 32-bit `esp+12`. If not, what value does it hold?
Solution: No, 7 ☐

(d) Does register `ebx` now hold a 32-bit `esp+8`? If not, what value does it hold?
Solution: No, 4 ☐

(e) Does register `ecx` now hold a 32-bit `esp`? If not, what value does it hold?
Solution: No, 2 ☐

4.2 Assume each instruction below has already been fetched from the main memory and placed in one of the registers. What is the *total number* of memory accesses that are required to fetch the operands for each of the instructions?

(a) `add eax, ebx` **Solution:** 0 ☐
(b) `mov ebx, [esp]` **Solution:** 1: for `[esp]` ☐
(c) `mov [esp + 12], eax` **Solution:** 0: `[esp + 12]` is the operand for *storing* the result. ☐

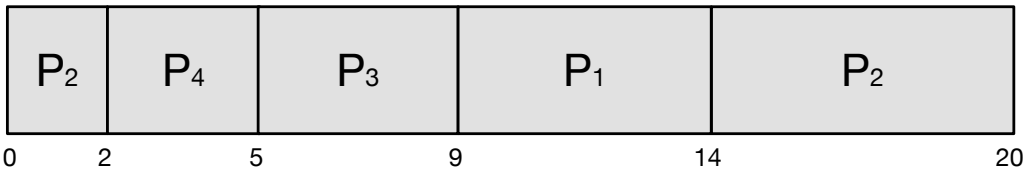
Problem 5 – CPU Scheduling (11%)

5.1 Consider the following table of burst time and arrival time for four processes P_1, P_2, P_3 and P_4:

Process	Burst Time	Arrival Time
P_1	5	3
P_2	8	0
P_3	4	5
P_4	3	2

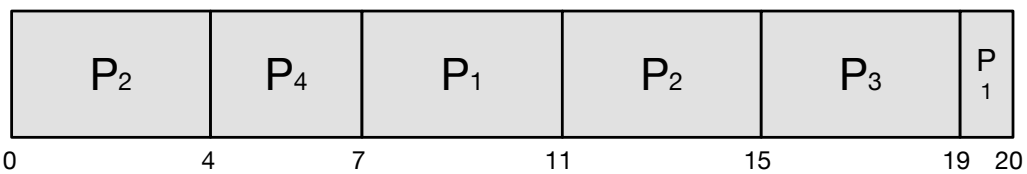
Use gantt charts to justify your answers to the following:

(a) Assume the *shortest-remaining-time-first* scheduling algorithm is used, and scheduling is carried out only at arrival or completion of processes. What is the completion order of the four processes?
Solution: P_4, P_3, P_1, P_2



☐

(b) Assume *round robin* scheduling algorithm with time quantum 4 is used. What is the completion order of the four processes?
Solution: P_4, P_2, P_3, P_1



☐

(c) Which scheduling algorithm gives a lower average waiting time of the four processes, shortest-remaining-time-first or round robin? Justify your answer by showing the calculation of the average waiting time for each scheduling algorithm.

Solution: Shortest-remaining-time-first

Shortest-remaining-time-first: $\big((9\text{-}3)+(0+14\text{-}2)+0+0\big)/4 = 4.5$

Round robin with tq=4: $\Big(\ \big((7\text{-}3)+(19\text{-}11)\big)+\ \big(11\text{-}4\big)+\ \big(15\text{-}5\big)+\ \big(4\text{-}2\big)\ \Big)\ /\ 4 = 7.75$ □

Problem 6 – Memory Access Time (8%)

6.1 Consider a computer system that does not support paging, and assume the processor in the system needs 5 ns to access the cache memory and 50 ns to access the main memory. Assume the cache hit ratio is 95%, and each cache lookup takes no time. What is the average memory access time of the processor?

Solution: $0.95 \times 5 + 0.05 \times (5 + 50) = 7.5$ ns

We also accept $0.95 \times 5 + 0.05 \times (0 + 50) = 7.25$ ns □

6.2 Now consider a computer system that supports paging and has a paging hardware with a translation look-aside buffer (TLB). Assume that the page table and *all the pages* a process requests are in the physical memory. Suppose the percentage of times that the page number of interest is found in the TLB[2] is 90%, each TLB lookup takes no time, and the memory access time for each access is 100 ns.

(a) What is the effective access time (EAT)?

Solution: $0.9 \times 100 + 0.1 \times (2 \times 100) = 110$ ns □

(b) Now assume that *two level paging* algorithm is used to structure the page table, and there is a TLB miss when the process requests a page. How many memory accesses are required to fetch this page?

Solution: 3 □

Problem 7 – Cache (12%)

7.1 Consider B_0, B_2, B_4, B_6 and B_8 are mapped to C_0, and B_1, B_3, B_5, B_7 and B_9 are mapped to C_1 where B_i refers to the i-th block in the main memory and C_j refers to the j-th cache line. Assume that B_3 and B_8 are in the cache. What are the number of cache misses if B_8, B_2, B_8, B_9, B_8 are accessed in the shown order? Justify your answer by identifying which accesses lead to cache misses.

Solution: B_8: hit; B_2: miss; B_8: miss (overwritten by B_2); B_9: miss; B_8: hit

Hits: 2; misses: 3 □

7.2 As a programmer, you can utilise the CPU registers by refering to their names, and you can utilise the main memory by refering to the addresses. But how do you utilise the cache? Explain your answer.

Solution: In contrast to CPU resgisters and main memory, cache can not be used explicitly by the programmer. Rather, the memory management hardware is keeping track of which main memory segments should be mapped to which cache segments, based on some algorithm that takes into account the usage pattern of the main memory. Thus, the programmer can not directly influence

[2] It is also called TLB hit ratio.

the content of the cache memory. However, the programmer may try to make the best use of the cache by deliberately crafting code and data usage size that will fit inside the cache memory, thus reducing the number of expensive (time consuming) cache misses. □

7.3 A multithreaded software is tested on two computers with similar specifications (e.g., frequency, cache size, etc.), except one has one physical CPU with two cores, while the other has two physical CPUs with one core each. The software runs significantly faster on the computer with one CPU with two cores. It is suggested that the difference is related to cache. Provide an explanation for this. Are the threads likely to be extensively exchanging data or not? Explain your answer.

Solution: Typically, if a CPU has two cores, they will share level 3 cache. This means that if threads running on different cores need to exchange data, this can be done at the level 3 cache. However, if the system rather has two physical CPUs with one core each, they will not have shared level 3 cache. This means that if threads running on different cores need to exchange data, this has to be done through the much slower system bus and main memory. The effect of this difference will be much more noticable if the threads actually are extensively exchanging data. Thus, it is likely that this is the case. □

7.4 Somebody is suggesting that the cache size does not matter if the computer has a lot of main memory (i.e., the cache size only matters for a computer with little main memory). Do you agree or disagree? Explain your answer.

Solution: While it in general is true that a computer with more main memory will usually perform better than a computer with less main memory, this is not to say that it renders the cache size irrelevant. Although the main memory is much faster than hard disk drives and even solid state drives, it is still very slow compared to the cache memory. Every time there is a cache miss, the requested data will have to be obtained from the main memory at a huge (for the CPU) time penalty. Obviously, a computer with larger cache memory will have fewer cache misses than a computer with less cache memory. Thus, the cache size still matters for a computer with lots of main memory. □

Problem 8 – Process synchronisation (12%)

8.1 Somebody is saying they do not understand why process synchronisation is a problem in the first place. They say: if one process has written data to some variable or buffer in the main memory, how can some other process not know this? Provide an explanation how this can happen (on the CPU-register–level). What is the term that is used to describe this?

Solution: The problem of process synchronisation is related to how computers actually update a value in the system memory at a low level. The content of some variable stored in main memory can not be modified direcly on the memory itself. Rather, the value of this variable has to be read (copied) from its location in main memory to a CPU register. At this point it can be modified (e.g. incremented) by means of a CPU instrction working on this register. Afterwards, the updated value for the variable may be written back to its original location on the main memory. The problem occurs if two (concurrent with context shift or parallel) processes try to perform such update at the same time. They may both read the original value of the variable from the main memory, and they will both perform their own desirable modification (e.g. one might try to increment it while the other might try to decrement it) on their own local copies in CPU registers. Finally they will both write their modified values back to the same main memory location of the variable. Obviously, the last process to update main memory location will overwrite the write of the former. This is called a race condition. □

8.2 Explain the The Dining Philosophers' Problem. In which circumstances will there be a deadlock? Propose a modification to the problem so that deadlocks may be avoided. Explain your answer.

Solution: We imagine a number of philosophers sitting around a (round) table. These philosophers at any time perform one of two actions: philosophising or eating. When they philosophise, they do not need any chopsticks. When they eat, they need two chopsticks. However, there is only one chop stick place between two neighbouring philosophers. Thus, a philosopher may only eat if he has obtained the chopsticks on both of his sides. Further, we assume all philosophers will always try to pick up the chopstick on the same side first, i.e. either right or left, then the other one. A deadlock will occur if all philosophers decide to start eating at the same time, so that they will all pick up the chopstick at the same side at the same time. In this case they will all find the chopstick on the other side already taken by the neighbouring philosopher, and all of them will be left with just one chopstick, waiting to also obtain the other forever. One possible solution to this problem can be to let every other philosopher pick up alternating chopsticks first, i.e. one picks up the chopstick to the right, the next the chopstick to the left, then the right again, and so on. This means every other philosopher should be able to eat (and let the waiting philosopher eat when he/she is finished and the chopsticks have been released again). □

8.3 Using the code below for semaphores, implement your proposed solution to the Dining Philosophers' Problem.

```
1  wait(int *semaphore){
2    while(*semaphore <= 0) {} ; // busy waiting //
3    *semaphore --;
4  }
5
6  signal(int *semaphore){
7    *semaphore++;
8  }
```

Solution: The structure of philosopher i of total n below. The shared data are "semaphore chopstick[n];" where all the elements of chopstick are initialized to 1.

```
1  while(true){
2    if(i%2){
3      wait(chopstick[i]);
4      wait(chopstick[(i+1) % n]);
5    }
6    else{
7      wait(chopstick[(i+1) % n]);
8      wait(chopstick[i]);
9    }
10   ...
11   /* eat for a while */
12   ...
13   if(i%2){
14     signal(chopstick[i]);
15     signal(chopstick[(i+1) % n]);
16   }
17   else{
18     signal(chopstick[(i+1) % n]);
19     signal(chopstick[i]);
20   }
21   ...
22   /* think for awhile */
23   ...
24 }
```

□

8.4 Besides semaphores, locks is another popular construct utilised for process synchronisation. Compare semaphores and locks. What are the differences? Give examples of situations when is one is

preferable over the other.

Solution: While semaphores can have a number of states where it will allow (>0) access to some protected section, locks only have one state where access is allowed. Thus, one may view locks as a binary version of semaphores. Or to put it in a different way: in general semaphores can be used instead of locks, but locks can not be used instead of semaphores. If one for example needs to protect the access to a single variable, locks would be sufficient (i.e. semaphores would add no extra value beyond increased complexity). Semaphores, on the other hand, may be useful in case one needs to protect a buffer with more than one slot that can be accessed sequentially, thus providing a mechanism to indicate the location of the last used buffer slot. For this a richer state than just binary is needed, and locks are impractical. □