Université Grenoble Alpes
M1 MoSIG
Academic year 2024-2025

Principles of Operating Systems and Concurrent Programming — 1nd Midterm Exam October 2024

- Duration: **1 hour**
- Authorized documents: **One handwritten A4 sheet of paper**

1 Some statements (5 points)

For each exercise with a multiple choice question, simply answer (on your separate answer sheet) by indicating the question number followed by one or several letters corresponding to the correct statement(s) — **you are not required to justify your answers**.

Note that, for each question, there is always at least one correct statement, and possibly several correct statements. A question without any answer is considered empty (no points). A single mistake in an answer (i.e., exactly one missing correct statement or one incorrect statement) voids half of the points allocated for the question. A number of mistakes greater than one voids all the points allocated for the question (there are no negative points).

1.1 Among the following statements, which one(s) is (are) true?

(a) A modern CPU chip typically contains several caches (from Level 1, very fast but very small, to the last level, bigger and slower).

(b) It is faster to search for a cache line in a 2-way associative cache compared to a 8-way associative cache.

(c) Reading a 64-bit variable from a CPU cache is approximately 10 times faster than reading the same data from a persistent disk.

(d) A CPU cache is only efficient for processes for which the total amount of memory allocated by the process is less than the size of the cache.

1.2 Among the following statements, which one(s) is (are) true?

(a) If the paging structure of a process was small enough to entirely fit in some registers of the CPU, then the concept of TLB would be useless.

(b) Using a multi-level paging structure (instead of a single-level paging structure, like a flat table) can increase the average time needed to handle a TLB hit.

(c) Using a multi-level paging structure (instead of a single-level paging structure, like a flat table) can increase the average time needed to handle a TLB miss.

(d) Using a multi-level paging structure (instead of a single-level paging structure, like a flat table) can increase the average time to handle a CPU cache miss.

1.3 Among the following statements, which one(s) is (are) true?

(a) Using fix-size blocks to implement a heap memory allocator induces external fragmentation if the blocks allocated by the program have different life times.

(b) Using fix-size blocks to implement the `malloc` function induces internal fragmentation in the virtual memory of the processes.

(c) In most cases, when an application invokes the `malloc` function to request a small block size (e.g., 10 bytes), the execution of the function completely takes place in user mode.

(d) For a heap memory allocator, using a *worst fit* strategy can be good to reduce internal fragmentation.

1.4 We consider the following code snippet in C taken from the implementation of a heap memory allocator (implementation of the `mem_alloc()` function), similar to the one implemented during lab 2. The variable `selected_block` corresponds to the block selected from the free list to handle the current memory allocation request.

```
1: mem_std_free_block_t *new_free_block=
    (mem_std_free_block_t*)((char*)selected_block + header_size
    + payload_size + footer_size);
2: new_free_block->next = selected_block->next;
```

Related to this code snippet, which of the following statement(s) is (are) true?

(a) The variable `new_free_block->next` stores the physical address of the next free block in the free list, if any.

(b) The variable `new_free_block->next` is stored in the stack of the process.

(c) On line 1, replacing the cast to `(char*)` with a cast to `(int*)` would not change the value stored in variable `new_free_block`.

(d) On line 1, replacing the cast to `(mem_std_free_block_t*)` with a cast to `(int*)` would not change the value stored in variable `new_free_block`.

1.5 Among the following statements, which one(s) is (are) true?

(a) Some switches from user mode to kernel mode are explicitly triggered by the latest executed instruction of user-level code (e.g., system calls) but some others are due to externals events (e.g., timer interrupts).

(b) The typical duration of 10 ms for the time quantum of an operating system scheduler is motivated by the fact that, on most machines, the hardware cannot generate timer interrupts with a higher frequency.

(c) Performing a context switch between processes requires at least two mode switches.

(d) In Linux, for security reasons, a system call can only be executed from trusted library code (for example, from a function like `printf` or `malloc`) and cannot be executed from user code.

2 Memory allocation (4 points)

2.1 We consider a memory allocator using a linked list to keep track of free blocks. The allocator is designed with the following constraints: each block requires a 4-byte header and there is no alignment constraint.

Answer the following question assuming that the free list contains only three blocks: 34 bytes, 20 bytes and 56 bytes (in that order, including the header bytes).

Describe a sequence of `malloc` calls that succeeds with a *best-fit* policy and fails with a *first-fit* policy.

2.2 A students makes the following claim: *"My machine has a 64-bit processor (and a 64-bit compatible operating system). As a consequence, each process has a virtual address space with a very large capacity. Furthermore, I use an operating system that implements virtual memory via paging with a single page size (e.g., all pages have a size of 4096 bytes). In such a situation, implementing an efficient heap memory allocator is not important (i.e., the allocator can introduce heavy fragmentation). Indeed, in such a case, having a highly fragmented heap does not introduce any drawback."*

This student is not correct.

- List 2 major drawbacks that could be observed with an inefficient heap allocator in this situation.
- For each drawback, provide a detailed explanation.
- Expected answer length: 5-10 lines

3 Segmentation (5 points)

In this exercise, we consider a simple machine with a MMU that implements virtual memory based on segmentation. The main specifications of this machine are the following:

- The MMU hardware has 4 pairs of (base, bounds/limit) registers (i.e., a process can at most have 4 segments).
- Virtual addresses (including the explicit segment ID) are stored on 12 bits and physical addresses are stored on 14 bits.
- For simplification, we ignore here the management of segment (read/write, user/supervisor) permissions.

3.1 Answer the following questions (explain briefly your computation):

(a) What is the maximum size of a segment (in bytes)?

(b) What is the maximum size (in bytes) of the physical memory for this machine?

3.2 Process P_1 is currently running and the operating system has configured the MMU as shown in the table below:

Segment	**Base**	**Bound**
0	0x1C00	0x100
1	0x3140	0x250
2	0x0300	0x040
3	0x0000	0x080

Which of the following virtual memory addresses can be accessed by the running process? (For each case, explain in a few words)

(a) 0xC6F

(b) 0x320

(c) 0x61D

3.3 We consider the same setup as the one described in the previous question. Which of the following physical memory addresses can be accessed by the running process? (For each case, explain in a few words)

(a) 0x0C6F

(b) 0x1280

(c) 0x324C

4 Paging (6 points)

We consider a machine with a single Intel x86 32-bit CPU, configured to use a 2-level paging structure and a (single) page size of 4096 bytes — as studied in the lectures. The format of the main data structures is provided in Figure 1. We also assume that disk swapping is not enabled/supported by the operating system.
Additional reminders:

- Each virtual address is stored on 32 bits.
- Each physical address is stored on 32 bits.
- The size of a page directory (PD) is 4096 bytes.
- The size of a page table (PT) is 4096 bytes.
- The size of a page directory entry (PDE) is 4 bytes.
- The size of a page table entry (PTE) is 4 bytes.

4.1 For this question, we assume that 1 page over 2 in the virtual address space of the process is configured as valid (page 0 is valid, page 1 is not valid, page 2 is valid, page 3 is not valid, etc.).

Answer the following questions and justify briefly:

(a) What is the size (in number of pages) occupied by the paging structure of the process if we assume a flat table?

(b) What is the size (in number of pages) occupied by the paging structure of the process if we assume the paging structure of x86 processors described above?

4.2 For each of the following cases, what is the amount of (physical) memory needed to store the paging structure of one process? (Briefly justify your answers.)

(a) There is only a single valid address range: `0x00800000` to `0x009FFFFF`

(b) There are 3 valid address ranges:

- range 1: `0x00800000` to `0x009FFFFF`
- range 2: `0x00B00000` to `0x04FFFFFF`
- range 3: `0x64000000` to `0x70FFFFFF`

4.3 We consider a process for which the paging structure has the following characteristics:

- Only 2 PDE entries are valid:
  - Entry number 100 and entry number 101 (decimal numbers)
- In the PT pointed to by entry 100, only the second half of the pages are valid.
- In the PT pointed to by entry 101, only the first half of the pages are valid.

What is/are the valid range(s) of virtual addresses for this process? (Briefly justify your answers)

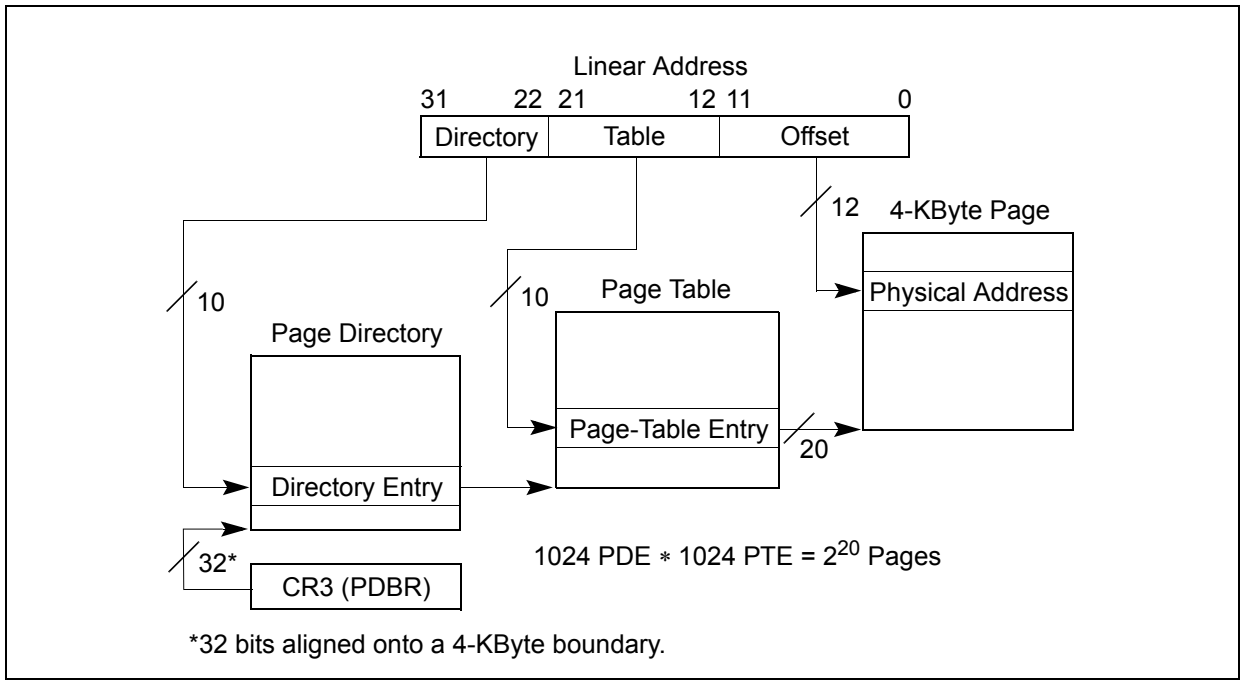

Figure 1: x86 address translation with 4-kilobyte pages (source: Intel documentation)