Univ. Grenoble Alpes (Ensimag & UFR IM2AG)
M1 MoSIG
Academic year: 2022-2023

Principles of Operating Systems — Midterm exam #1

October 27, 2022 — Duration: 90 minutes

Important instructions: All documents are forbidden, except 1 (dual-side) handwritten A4 paper sheet. All electronic devices are forbidden (calculators, computers, mobile phones, etc.). The number of points indicated for each exercise is only provided to give you an idea of its weight. We reserve the right to change the exact number of points.

For each exercise with a multiple choice question, simply answer (on your sperate answer sheet) by indicating the question number followed by one or several letters corresponding to the correct statement(s) (**unless otherwise mentioned, 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).
Note: For each problem, all the questions are independent.

Problem 1 (2.5 points)

Question 1.1 We consider a process P with X threads (including the "main"/initial thread) running on a machine with a traditional operating system like Linux and N CPU cores, where $X > N$. For each configuration and each state below (*Ready*, *Running*, ...), indicate the maximum number of threads (within P) that can simultaneously be in that state. **For each line, choose only one of the proposed values and write it down on your separate answer sheet. You are not required to justify your answers.**

Configuration A. P uses the following threading model: preemptive, kernel-level

Thread state	Answer			
Running	1	N-1	N	More than N
Ready	1	N-1	N	More than N
Blocked	1	N-1	N	More than N

Configuration B. P uses the following threading model: cooperative, user-level

Thread state	Answer			
Running	1	N-1	N	More than N
Ready	1	N-1	N	More than N
Blocked	1	N-1	N	More than N

Question 1.2 While skimming through a textbook about operating systems, you read the following paragraph. Unfortunately, the book is damaged and there are some unreadable lines:
"CPU hardware support for asynchronous interrupts (issued by various peripheral devices such as timers, storage devices, network devices, keyboards, etc.) is an essential feature for any modern computer system, for two main reasons: (1) safety and security, and (2) performance. Let us illustrate each of these aspects with an example [... unreadable part ...]."
Provide a short piece of text to replace the missing part. (Expected answer length: 5-10 lines)

Problem 2 (1.5 points)

Let us consider the program shown in Figure 1.

```
1   /* Notes:
2   * - #include directives are omitted for simplification
3   * - fflush(stdout) forces the output of printf to be displayed immediately
4   * - fork returns 0 for the child, and the child's pid for the parent
5   */
6   int main(int argc, char **argv) {
7   int i; pid_t r;
8   for (i=0; i<2; i++) {
9           r = fork();
10          if (r == 0) {
11                  printf("%d", i+1); fflush(stdout);
12                  exit(0);
13          }
14  }
15  printf("p"); fflush(stdout);
16  exit(0);
17  }
```

Figure 1: Code listing for `prog1.c`

Question 2.1 How many processes are created during the execution of this program (including the initial process launched by the shell)?
(a) 2 (b) 3 (c) 4 (d) more than 4

Question 2.2 Among the following text strings, which one(s) may be produced by the (complete) execution of the program?
(a) `p122` (b) `p12` (c) `12p` (d) `1p2` (e) `ppp` (f) None of the other answers

Question 2.3 Let us consider the program shown in Figure 1 with the following modification: `execlp("/bin/sleep", "sleep", "2", (char*)NULL);` is inserted between lines 10 and 11 (we assume that this call completes successfully). which one(s) may be produced by the (complete) execution of the program?
(a) `p12` (b) `12p` (c) `p` (d) `ppp` (e) None of the other answers

Problem 3 (4.5 points)

Question 3.1 Among the following statements, which one(s) is (are) true?

(a) 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.

(b) To limit *internal fragmentation*, most memory allocators use small block headers (e.g., 8 bytes).

(c) A *memory leak* is a programming mistake that results in wasted (heap) memory space. However, that space is reclaimed by the operating system when a process terminates.

(d) If an operating system relies on paging (with a fixed page size, e.g., 4096 bytes) to implement the virtual memory abstraction, then the problem of external fragmentation within the memory heap of each process is completely avoided.

Question 3.2 Among the following statements regarding the (typical) currently available hardware and operating systems, which one(s) is (are) true?

(a) The clock frequency of a CPU is approximately 1 GHz (order of magnitude). In other words, the time needed to execute an elementary instruction of machine code is approximately 1 nanosecond (order of magnitude).

(b) The typical duration of a "time quantum" (or "time slice") used by the kernel process scheduler is in the range of 100 nanoseconds.

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

(d) If a single-threaded process P_1 is currently in the *"blocked"* state, a state change for P_1 can only occur through a system call invoked by another process P_2.

Question 3.3 Among the following statements, which one(s) is (are) true?

(a) Nowadays, the size of the source code for the Linux kernel is in the range of 10k (ten thousand) lines (mainly written in C).

(b) Compared to a monolithic kernel, a microkernel has a much smaller code base. The main benefit is that a microkernel can be fully implemented in assembly language (rather than C) and thus achieve better performance.

(c) In a mainstream operating system such as Linux or Windows, the code of the device drivers is running in kernel mode, even if these drivers have been provided by third-party developers and/or compiled separately from the kernel binary.

(d) Unlike user-level code, the kernel code can freely modify the configuration of the MMU.

Question 3.4 Among the following statements, which one(s) is (are) true?

(a) For a heap memory allocator, sorting the list of free blocks by increasing addresses (rather than increasing block sizes) allows limiting the occurrence of external fragmentation.

(b) For a heap memory allocator, using a *first fit* strategy with a list of free blocks sorted by increasing sizes is equivalent to using a *best fit* strategy.

(c) In the heap memory allocator used by a Linux application, the memory pointers used in the linked list of free blocks correspond to *virtual memory* addresses.

(d) In the case of a mainstream operating system that uses paging (such as Linux), several free blocks (elements of the free list) and several allocated blocks of the same process heap may be stored together in the same physical page frame.

Question 3.5 Among the following statements, which one(s) is (are) true?

(a) The local variables of two concurrent threads (within the same process) are not stored on the same stack.

(b) Creating a new thread within an existing process is generally faster than creating a new process, even in the case of kernel-managed threads.

(c) In Linux, within a multi-threaded process, each thread has its own private heap.

(d) A context switch between two processes P_1 and P_2 takes a non-negligible amount of time because, among other things, the kernel must save (copy) all the contents of P_1's heap in kernel memory (in the "process control block" of P_1) and do the reverse (restore) operation for P_2.

Question 3.6 Among the following statements, which one(s) is (are) true?

(a) The (user-level) memory heap is useful to store data (necessary for the application/library code) whose size and/or lifetime cannot be determined at compilation time.

(b) The ABI (application binary interface) of an operating system defines, among other things, conventions that the machine-level code must follow regarding the management of the stack and the invocation of system calls.

(c) In the case of Linux, the system call interface consists of approximately 20 functions (such as `fork` and `exit`).

(d) For security reasons, a system call can only be triggered from trusted library code (for example, from a function like `printf` or `malloc`) and cannot be triggered from untrusted application-level code.

Problem 4 (2.5 points)

Question 4.1 We consider a memory allocator based on a single list of free blocks (with support for block splitting and coalescing). For simplification, we ignore metadata (headers and footers) as well as alignment constraints, and we assume that the requested sizes are never rounded up.
The free list contains the following blocks, in this order (from the head of the list):
100 bytes, 200 bytes, 400 bytes and 100 bytes.
Besides, let us assume that we receive the following sequence of requests:
allocate 100, allocate 150, allocate 100, allocate 350.

(a) Using a *first-fit* policy, will all requests succeed?

(b) Using a *best-fit* policy, will all requests succeed?

Briefly justify your answers:

- If all the requests succeed, describe the final state of the free list.
- If a request fails, describe which one and describe the state of the free list.

Question 4.2 One of your friends makes the following claim, comparing segmentation and paging to implement virtual memory management (here we assume a paging approach based on a unique page size of 4096 bytes, like in the lectures):
"Using segmentation brings some benefits over paging. One of these benefits is that the required MMU logic for address translation is simpler and faster. However, segmentation has some major drawbacks, which make it much less appealing than paging overall: it is less flexible for disk swapping and, above all, is more prone to external fragmentation."
Do you agree with this claim? Justify your answer (Expected answer length: about 10-15 lines.)

Problem 5 (3.5 points)

In this exercise, we consider a machine with a single CPU that implements virtual memory via segmentation, using an explicit segment identifier. The CPU architecture has the following characteristics:

- There are 4 segments per address space.
- All segments are configured for growth from the base towards higher addresses (like in the lectures).
- Each virtual address (including the segment ID) is stored on 12 bits.
- Each physical address is stored on 11 bits.
- For each segment of the currently running process, the MMU is configured via 3 registers:
  - *Base address*
  - *Size* (in bytes)
  - *U/K permissions*: a boolean indicating if the segment is accessible from user mode (0) or only form kernel mode (1). For simplification, we do not consider read-write permissions in this exercise (all the segments have read-write permissions).

We also assume that the operating system always sets up 4 segments for each process: three segments for the application and one for the kernel. We also assume that the kernel code is mapped in the same address range for all the processes. In contrast, the size of a user segment is potentially different from one process to another. Besides, the operating system does not support disk swapping (the segments always remain in main memory until the termination of a process).
Important: For this exercise, all the addresses are provided in *decimal notation*. The answers must use the same notation.

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

Segment	Base	Size	U/K
0	96	32	0
1	896	128	0
2	384	384	0
3	1536	512	1

(a) If the user-level code of P_1 performs a memory access to virtual address 25, will it be successful? And if so, what will be the corresponding physical address? In any case, briefly justify your answer.

(b) For each segment of P_1, give the range of valid virtual addresses and the corresponding range of physical addresses. No need to justify your answer.

5.2 Let us now assume that the operating system must launch a new process P_2, while P_1 is still running. The size requirements for the user segments of P_2 are as follows:

- Segment 0: 512 bytes
- Segment 1: 224 bytes
- Segment 2: 112 bytes

Explain in which physical address range each segment of P_2 must be mapped (without impacting P_1). Also provide a table (like the one above) to summarize how the MMU registers must be configured for running P_2. Justify your answer.

Problem 6 (5.5 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 2.

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 4046 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.

6.1 If all the virtual addresses of a process are configured as valid:

(a) What is the storage capacity (in number of pages) of the virtual address space? Briefly justify your answer.

(b) What is the size (in number of pages) occupied by the paging structure of the process? Briefly justify your answer.

6.2 What is the minimum number of valid pages in the virtual address space of a process such that the paging structure reaches the maximum size? Briefly justify your answer.

6.3 In this part, we make the following assumptions:

- The operating system does not use swapping (i.e., each valid virtual page is always stored in main memory).
- We consider a process with a virtual memory address space that contains only 3 valid virtual address ranges:
  - Range 1: [`0x5AC00000` ; `0x5AC01FFF`]
  - Range 2: [`0x5AE00000` ; `0x5AFFFFFF`]
  - Range 3: [`0x80000000` ; `0xFFFFFFFF`]

(a) Within one of the valid virtual address ranges, give an example of two adjacent virtual address that are not necessarily mapped to two adjacent physical addresses. (No need to justify your answer.)

(b) How much memory (in number of pages) is required to store the whole paging structure of the process? Justify your answer.

6.4 Considering the process in question 6.3, let us assume that all the bytes of Range 1 are used to store an array of integers (each integer has a size of 32 bits) and that the application code uses a loop to iterate over the array in order to compute the sum of all of these numbers. For simplification, we assume that the process is never preempted during the execution of the loop. We also assume that the MMU has two distinct TLBs: one for code instructions (I-TLB) and another one for data (D-TLB). In the worst case, how many D-TLB misses will be caused by all the memory accesses to Range 1 performed in the loop? Briefly justify your answer.

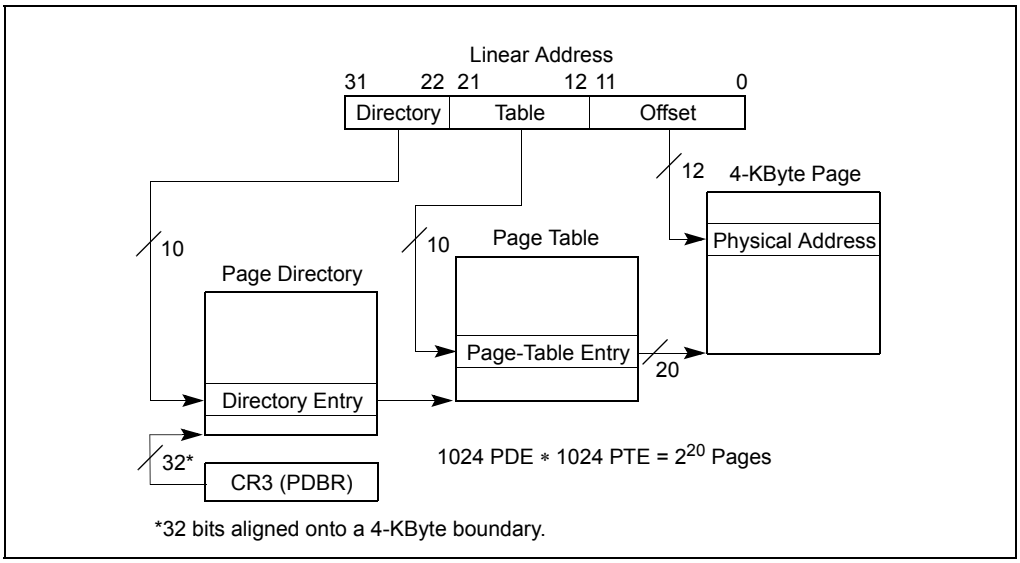

Figure 2: x86 address translation with 4-kilobyte pages (source: Intel documentation)