Univ. Grenoble Alpes (Ensimag & UFR IM2AG)
M1 MoSIG
Academic year: 2023-2024

Principles of Operating Systems — Midterm exam #1

October 26, 2023 — 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.

Note : For each problem, all the questions are independent.

Problem 1 (4 points)

Question 1.1 For each of the above hardware instructions (on a typical CPU such as the ones studied in the lectures), indicate whether it must be privileged (i.e., only allowed for code running in kernel mode) or not. In your answer, write either "P" (privileged) or "NP" (not privileged) and provide a brief justification (approx. 3–5 lines for each).

(a) Hardware instruction for modifying the stack pointer.

(b) Hardware instruction for disabling/blocking interrupts.

(c) Hardware instruction for modifying the CPU register that points to the start of the paging structure of the currently running process.

(d) Hardware instruction for invoking a system call.

Question 1.2 We consider a 32-bit Intel x86 machine running Linux. The machine has a single CPU with a clock speed of 1 GHz. We assume that there is only a single process (with a single thread) running on the machine. Sort the following actions by increasing duration (from 1=*shortest* to 5=*longest*). We assume that there is no disk swapping activity involved. If necessary, consider the worst-case time for each action. No need to give details (durations) or justifications.

(a) Executing a CPU instruction that copies 4 bytes from one CPU register into another.

(b) Reading 4096 bytes from a mechanical hard disk.

(c) Executing the `getpid` system call (from user space), until completion (assuming that there is no preemption of the process in the meantime).

(d) Returning from function `func_b` (callee) into `func_a` (caller), assuming that both functions are running in user mode and that their signatures are : `void func_X()`

(e) Executing the following system call, until completion : `sleep(3)` (the sleep duration is in seconds).

Question 1.3 While skimming through a textbook about operating systems, you read the following paragraph. Unfortunately, the book is damaged and two parts of the text are unreadable. *"User-level threads (i.e., thread abstractions implemented as a user-level library) have a lower performance overhead than kernel-level threads because [... first unreadable part...] However, they cannot take advantage of multiple CPUs (it is not possible to have several threads of the same process running in parallel on distinct CPUs) : indeed, [... second unreadable part ...]"*. For each of the two unreadable parts, provide a short piece of text to replace the missing content. (Expected answer length : 5-10 lines in total)

Problem 2 (4,5 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).

Question 2.1 Among the following statements regarding a traditional operating system design (such as Linux) which one(s) is (are) true ?

(a) The operating system kernel is running as a separate process with full (administrator) privileges and a higher scheduling priority.

(b) Most of the calls to `malloc` and `free` do not trigger a CPU mode switch.

(c) For safety and security reasons, code running in user mode has a limited access to the kernel memory (read-only rather than read-write privileges).

(d) The memory address spaces of different processes are completely isolated by default. However, the OS can transparently share one or several physical pages of memory between processes (for example, similar code pages in the case of processes running the same application).

Question 2.2 In this question, we consider a traditional operating system design with kernel threads (also known as "kernel-managed threads"), such as Linux. Among the following statements, which one(s) is (are) true ?

(a) Within a given process, the kernel manages a distinct paging structure for each thread.

(b) Within a given process, the kernel creates a distinct heap for each thread.

(c) It is possible for a thread to directly transition from logical state *"running"* to state *"ready"*.

(d) It is possible for a thread to directly transition from logical state *"ready"* to state *"blocked"*.

Question 2.3 Among the following statements, which one(s) is (are) true ?

(a) On mainstream CPUs (such as Intel x86 and Arm), the set of available system calls (e.g., `open`, `fork`, `getpid`, ...) and their interface (e.g., list of arguments and their types) is imposed by the hardware specification.

(b) On an Intel x86 CPU, a L1 cache miss is automatically and fully handled by the hardware.

(c) On an Intel x86 CPU, a TLB miss is automatically and fully handled by the hardware.

(d) The introduction of a TLB inside the MMU (memory management unit) of a CPU design allows taking advantage of the *temporal* locality as well as the *spatial* locality exhibited by the sequence of memory accesses performed by the applications.

Question 2.4 In this question, we only consider single-threaded processes. Among the following statements, which one(s) is (are) true ?

(a) There are different circumstances that may lead to a context switch between processes, for instance, system calls, timer interrupts and errors (e.g., memory protection faults).

(b) When performing a context switch between two processes, the kernel must modify the CPU register(s) that is (are) used for the MMU configuration.

(c) 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.

(d) The time needed to perform a mode switch (from user mode to kernel mode) mainly depends on software policies (configurable by the OS designers/administrators) rather than on the implementation of the hardware mechanism.

Question 2.5 Among the following statements, which one(s) is (are) true ?

(a) In a language that supports garbage collection (for example, Java), the garbage collector is triggered upon the termination of a process in order to (automatically and transparently) reclaim the allocated blocks in the heap.

(b) On a machine with a MMU that supports paging, the size (i.e., the number of bits used in the binary format) of a virtual page number is always identical to the size of a physical page number.

(c) In the case of a virtual memory system that uses paging, two adjacent (valid) virtual addresses within a process address space are not necessarily mapped to adjacent physical addresses.

(d) One of the main advantages of processors with a software-managed TLB (for example, the MIPS architecture) is that OS kernel developers can easily modify the format (size) of the (virtual and physical) addresses, and/or the page size in order to implement various optimizations.

Question 2.6 Among the following statements, which one(s) is (are) true ?

(a) An application based on several single-threaded processes can potentially leverage parallel execution on a machine equipped with multiple CPUs.

(b) An application based on a single process with multiple cooperative threads cannot leverage parallel execution on a machine equipped with multiple CPUs.

(c) On a server machine, periodic CPU preemption is useful and sufficient to guarantee that an application will not monopolize the CPU(s).

(d) On a machine with a single CPU, compared to the communication (data transfer) between two threads in different processes via a Unix pipe, the communication between two threads within the same process can potentially be more efficient, due to several reasons, among which (1) fewer data copies and (2) fewer TLB misses.

(e) Decreasing the duration of the "time quantum" (or "time slice") used by a CPU scheduler increases the overhead of the operating system (a smaller fraction of the total CPU time is spent in the code of applications).

Question 2.7 Let us consider the program shown in Figure 1.
Among the following statements, which one(s) is (are) true ?

(a) Running this program always outputs the same number of characters and lines.

(b) The (ordered) text sequence displayed may vary from one run to another.

(c) Removing the line "`exit(0);`" does not impact the displayed output.

(d) If we replace "`printf("e\n");`" with "`printf("%d\n", getpid());`", there will never be two similar lines in the output trace of a run.

Problem 3 (3 points)

Question 3.1 For each of the following situations, indicate the main type of (memory) fragmentation that corresponds to the description. Simply write *"External"* or *"Internal"*. You are not required to justify your answer.

```
1   /* Notes:
2   * - #include directives are omitted for simplification
3   * - fork returns 0 for the child, and the child's pid for the parent
4   */
5   int main(int argc, char **argv) {
6       pid_t r;
7       r = fork();
8       if (r == 0) {
9           printf("c\n");
10      }
11      printf("e\n");
12      exit(0);
13  }
```

FIGURE 1 – Code listing for `prog1.c`

(a) An operating system uses (paging-based) virtual memory with a page size of 4096 bytes but a process needs 5000 bytes to store its code region.

(b) Several consecutive calls to `malloc` result in the allocation of adjacent blocks of very diverse sizes (some of them very small), which turn out to be freed at very different moments (some of them are released quickly and others remain allocated for a long time).

(c) In order to perform safety checks, some additional (fixed-size) metadata information is inserted in the header of each block of a heap memory allocator.

(d) An operating system uses virtual memory implemented via segmentation. The size of the heap segment is adjusted during the lifetime of a process based on its dynamic memory requirements. The placement policy of the heap allocator sometimes leaves very large chunks of free memory between two consecutively allocated blocks.

Question 3.2 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) :
30 bytes, 100 bytes, 200 bytes, 400 bytes, 100 bytes, 50 bytes, 20 bytes.
Besides, let us assume that we receive the following sequence of requests :
allocate 100, allocate 150, allocate 100, allocate 30, allocate 350, allocate 10, allocate 40.

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

(b) Using a *next-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 3.3 Your friend Bob makes the following claim : *"People that write application code using a language (like Java) or a library that supports garbage collection do not have to worry about heap fragmentation"*. Your other friend Alice then says that this claim is only correct if we are considering the case of a "compacting" (also known as "moving") garbage collector. Do you agree with Alice, Bob, or none of them ? Justify your answer (Expected answer length : about 5-10 lines.)

Problem 4 (3 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 two pairs of (base, bounds/limit) registers (i.e., a process can at most have two segments).
- Virtual addresses (including the explicit segment ID) are stored on 10 bits and physical addresses are stored on 16 bits.
- The machine is equipped with a capacity of 64 kB of physical memory (RAM).
- For simplification, we ignore here the management of segment (read/write, user/supervisor) permissions.

We consider a process with two segments, for which the MMU configuration is as follows (the values are given in hexadecimal notation) :

- **segment 0 :**
  - base = 0x8400
  - limit (size) = 0x100 bytes
- **segment 1 :**
  - base = 0x0c00
  - limit (size) = 0 bytes

Unfortunately, it appears that the above MMU configuration is incorrect because the values stored in some of the MMU registers have been corrupted due to some hardware defects. More precisely, the wrong values suffer from exactly one flipped bit : i.e., compared to the originally written (good) value, there is exactly one bit that has been accidentally modified (from 0 to 1 or vice-versa).

Question 4.1 Let us assume that we also know for sure that :

- virtual address 0x300 should cause a segmentation violation (invalid address) ;
- virtual address 0x2ff is a valid address.

(a) Which of the segmentation registers (among seg. 0 base, seg. 0 limit, seg. 1 base, seg. 1 limit) determines whether these two addresses are valid or not ? Justify your answer.

(b) What should the correct value be for the register identified in the previous answer ? Justify your answer.

(c) Given the correct value identified in the previous answer (and assuming that there is no other error for this segment), what physical address should virtual address 0x2ff translate to ? Justify your answer.

Problem 5 (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 Figures 2, 3 and 4. 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.

Question 5.1 Regarding the format of the entries in the paging structure of an Intel x86 processor : we have seen that a PDE (page directory entry) and a PTE (page table entry) both contain fields for read/write and user/supervisor permissions. Suggest one reason that motivates this design choice (in comparison with a design in which the fields for permissions are only stored in PTEs). (Expected answer length : approx. 5 lines)

Question 5.2 Let us consider a process whose address space has 4 distinct valid virtual memory regions (named R_1, ..., R_4), each with a size of 2^{20} bytes (i.e., 1 Megabyte). We also assume that there are no particular constraints regarding the virtual address range chosen for each region. In such a situation, what is the smallest possible size (in number of pages) for the complete paging structure of the process ? Justify your answer and describe the chosen address range for each region.

Question 5.3 We consider a process address space in which there are only two valid regions, each region accounting for 25% of the maximum capacity the virtual address space. The first region corresponds to the lowest possible addresses and the second region corresponds to the highest possible addresses.

(a) What is the size of each region (in bytes or pages) ? Briefly justify your answer.

(b) What are the virtual address ranges (in hexadecimal notation) of the two regions. Briefly justify your answer.

(c) What is the total size (in bytes or pages) of the paging structure required for the process ?

Question 5.4

(a) On an Intel x86 CPU, every write to the CR3 register (also known as the "Page Directory Base Register" — see Figure 2) automatically triggers a flush of the entries of the TLB. Briefly describe a reason that motivates this design choice.

(b) The above-described flush does not occur when a TLB entry corresponds to a PTE in which the "G" boolean flag (see Figure 4) is set to 1. Briefly describe one example of virtual memory region for which setting up this flag may be useful. If you do not have any idea, describe one example of virtual memory region that must absolutely not be configured with the G flag. In any case, briefly justify your answer.

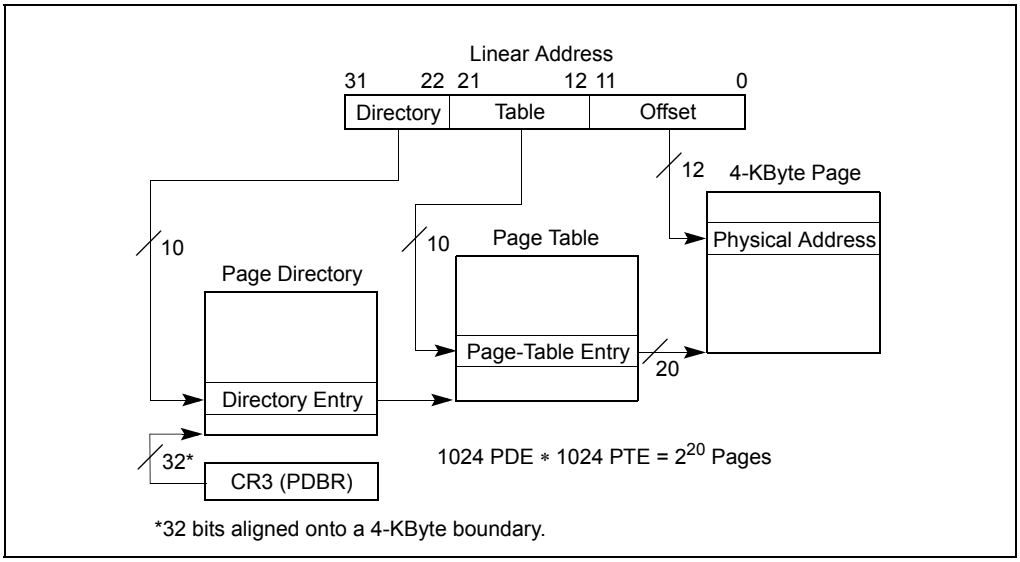

FIGURE 2 – x86 address translation with 4-kilobyte pages (source : Intel documentation)

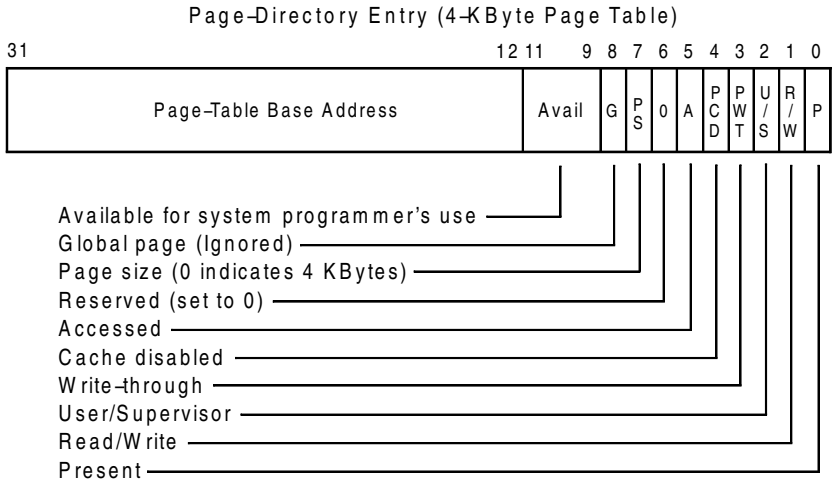

FIGURE 3 – Format of an x86 page directory entry (source : Intel documentation)

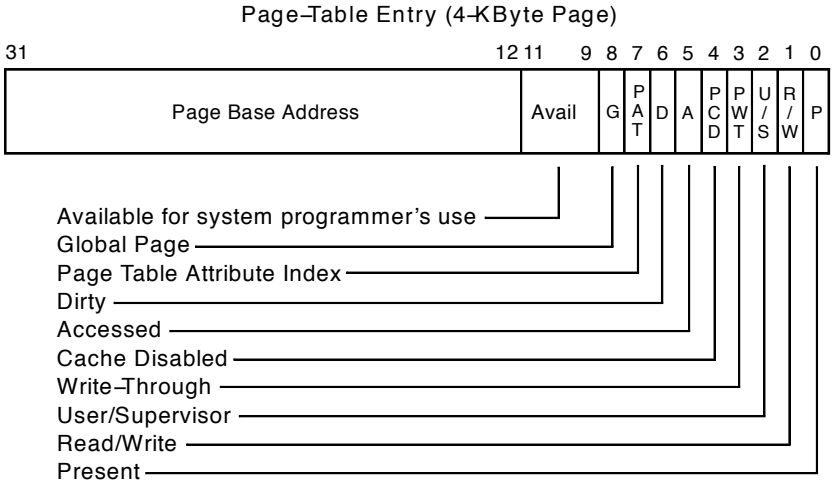

FIGURE 4 – Format of an x86 page table entry (source : Intel documentation)