Grenoble Universities (GrenobleINP & Université Joseph Fourier)
M1 MoSIG
Academic year: 2013-2014

Operating System Design — Midterm exam

October 23, 2013 — Duration: 90 minutes

Important instructions: All printed documents are allowed. All electronic devices are forbidden (calculators, computers, mobile phones, PDAs).
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.

Problem 1 (4 points)

For each of the questions of this problem, simply answer by indicating its number and one or several letters corresponding to the correct statement(s) (you are not required to justify your answers). Note that there are between 1 and 5 correct statements per question. 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) yields 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 1.1

Which of the following statements are true ?

(a) An operating system exports a source code-level interface (named *ABI*) to user applications.

(b) In a typical operating system (e.g., Linux), the code of standard libraries (e.g., the GNU C Library) is typically executed in privileged mode.

(c) The global variables of a C program are allocated on the stack.

(d) A *memory leak* may, for instance, occur when a process terminates without having explicitly freed (via `free`) all the memory blocks allocated via `malloc`.

(e) Timer interrupts allow the operating system to preempt any application running in user mode.

Question 1.2

Which of the following statements are true ?

(a) The termination of a process necessarily results in invoking the scheduler.

(b) A system call is typically implemented with a specific hardware instruction.

(c) A system call is required to invoke a library function such as `printf`.

(d) A library function may internally issue one or several system calls.

(e) The reason why some processes may be in *ready* state is simply because the number of (non-blocked) processes may exceed the number of available processors on the machine.

Question 1.3

Which of the following statements are true ?

(a) On a modern Intel x86 processor, all the page tables of a running process are stored in specific on-chip registers in the Memory Management Unit.

(b) Executing a write instruction (e.g., a `store` instruction) that attempts to modify a memory location within a read-only page always results in a protection fault, upon which the operating system will destroy the offending process.

(c) In a process, the user-level code cannot directly manipulate the paging structures (e.g., page tables) but can nonetheless modify them via some specific system calls.

(d) If there are multiple processes simultaneously executing the same program, the operating system may let them share the same physical page frames for their code. However, they will never share the same physical page frames for their stacks and their global variables.

(e) A memory allocator (or a garbage collector) for C programs cannot move the currently allocated memory blocks.

Question 1.4

Which of the following statements are true ?

(a) LRU is the theoretically optimal page replacement policy but it is hard to faithfully implement in practice.

(b) Before a page gets evicted from the physical memory, it must be written to the swap partition/file.

(c) The pages within an anonymous memory mapping (created via `mmap`) can never be paged out to disk.

(d) Thrashing can potentially happen when the system uses a *global* allocation strategy for page frames (but not with a *local* strategy).

(e) The clock replacement algorithm implements an approximation of the LRU page replacement policy.

Question 1.5

Among the following events occurring on an Intel processor, which ones must be managed by a software handler (as opposed to a hardware-only handler) ?

(a) a miss in the L1 cache

(b) a TLB miss

(c) a page fault

(d) a device interrupt (e.g., from a I/O device such as the hard disk drive)

(e) an arithmetic error (e.g., due to a division by zero)

Question 1.6

Which of the following actions correspond to *policies* (i.e., configurable strategies of the system), as opposed to actions that are required for correctness ?

(a) Switching the logical state of a given process from *running* to *blocked*

(b) Switching the logical state of a given process from *running* to *runnable*

(c) Reading all the code pages of a process from the swap partition when it triggers a page fault

(d) Clearing the *Present* bit in a page table entry when a page is swapped out to disk

(e) Using the same (virtual) address for the start of the heap in every process address space

Question 1.7

Which of the following statements are true ?

(a) The main motivation for *multi-level* paging structures is to speed up hardware translation.

(b) Using a *multi-level* paging structure can increase the average TLB miss time.

(c) Accessing a file via a memory mapping (set up by `mmap`) may be more efficient than using `read`, because it requires less mode switches.

(d) In most operating systems, the kernel code is mapped in the address space of every process and always in the same address range.

(e) In most operating systems, the kernel data structures are mapped in the address space of every process, but can only be accessed in read-only mode from the user-level code.

Problem 2 (7 points)

In this problem, we consider a machine with a 32-bit Intel x86 processor, with the PAE paging mode enabled (*Physical Address Extension*). An overview of the paging structures used in PAE mode is provided in Figures 1 and 2, respectively for a page size of 4 kilobytes (2^{12} bytes) and 2 Megabytes (2^{21} bytes). Note that, in any case :

- page directories and page tables are always stored on 4-kilobyte pages ;
- page directory pointer tables are always stored on 32-byte memory blocks.

Question 2.1

The goal of this question is to determine the required space to store the paging data structures of a process. For each of the following cases, indicate the corresponding size and justify your answer.

(a) There is only one valid address range (0x00000000 to 0x00000FFF) in the process address space, whose mapping uses exclusively 4-kilobyte pages.

(b) There is only one valid address range (0x00000000 to 0x001FFFFF) in the process address space, whose mapping uses exclusively 4-kilobyte pages.

(c) There is only one valid address range (0x00000000 to 0x001FFFFF) in the process address space, whose mapping uses exclusively 2-Megabyte pages.

(d) All the addresses in the process address space are valid and the mappings use exclusively 4-kilobyte pages.

(e) All the addresses in the process address space are valid and the mappings use exclusively 2-Megabyte pages.

Question 2.2

Explain in which circumstances it may be worthwhile to enable PAE paging. Expected answer length : approximately 5 to 10 lines.

Question 2.3

We consider a process whose address space is configured as depicted on Figure 3 (note that, for simplification purposes, some memory regions such as `.bss` and the dynamic shared libraries are not present). For instance, the process stack is located between addresses `0xCFC00000` (included) and `0xCFCFFFFF` (included). We assume that the operating system has enabled PAE paging, and uses 2-Megabyte pages

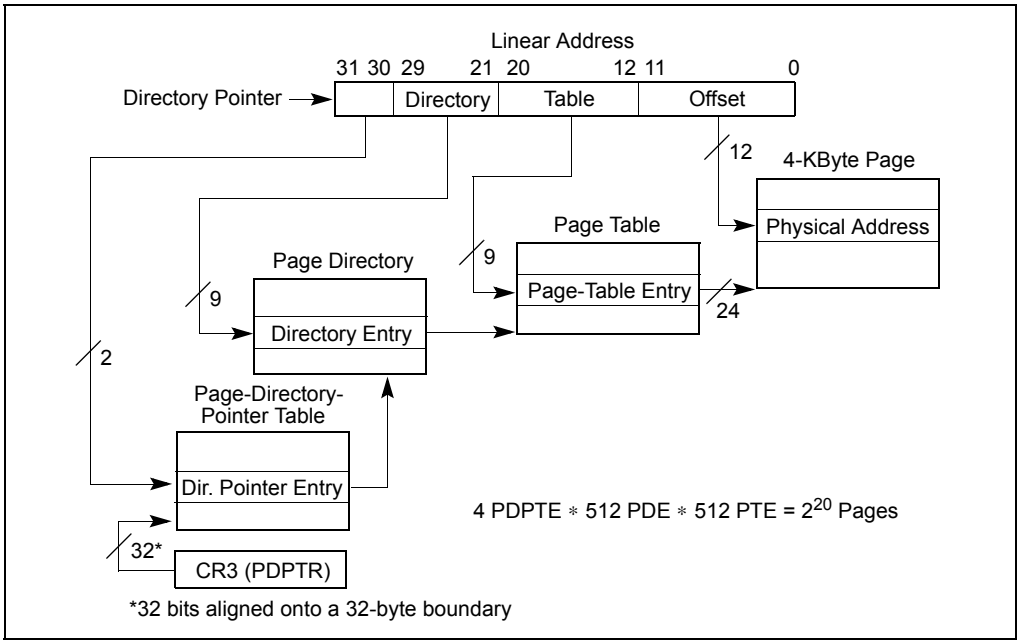

FIGURE 1 – Address translation with PAE enabled and 4-kilobyte pages (source : Intel documentation)

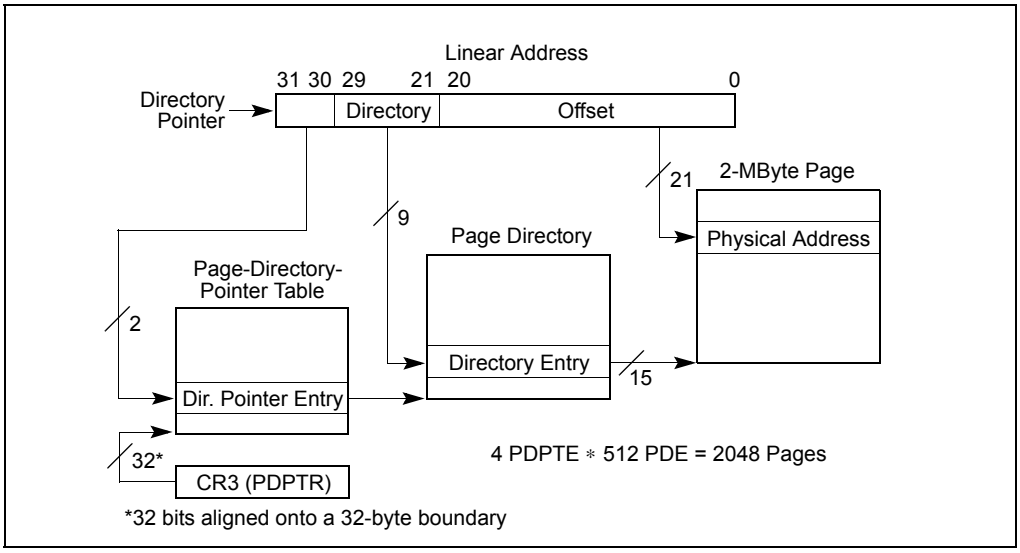

FIGURE 2 – Address translation with PAE enabled and 2-Megabyte pages (source : Intel documentation)

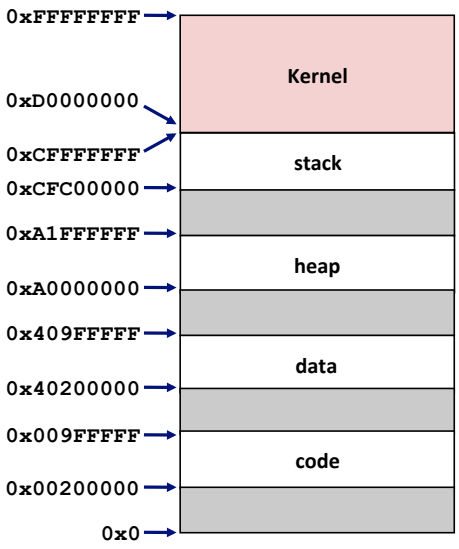

FIGURE 3 – Process address space. Note that the figure is not drawn to scale.

for the code and data regions and 4-kilobyte pages for the other regions.

What is the space required (in bytes) by the paging data structures for the heap region ? Justify your answer.

Question 2.4

Discuss the pros and cons of using 2-Megabyte pages instead of 4-kilobyte pages. Expected answer length : approximately 10 lines.

Problem 3 (3 points)

Let us consider the following program :

```
1   /* #include directives omitted for simplification */
2   pid_t pid = 0;
3   void func1(), func2(), func3();
4
5   void handler(int sig) {
6       if (pid > 0) {
7           printf("yes\n");
8       } else {
9           printf("no\n");
10      }
11      exit(0);
12  }
13
14  int main() {
15      func1();
```

```
16      Signal(SIGCHLD, handler);
17      if ((pid = fork()) == 0) {
18          func2();
19          exit(0);
20      }
21      func3();
22      while(1);
23      exit(2);
24  }
```

Question 3.1

When the bodies of `func1`, `func2`, `func3` are empty, a given execution of the program may either print `yes` or `no`. Explain why (in a few lines).

Question 3.2

Is it possible to write `func1`, `func2` and `func3` in such a way that the program execution always prints `yes` ? If so, describe the corresponding code for these functions [1]. Otherwise, explain why.

Problem 4 (6 points)

Question 4.1

Describe three general patterns for streams of dynamic memory allocation/deallocation requests that do not cause any external fragmentation. Expected answer length : 5 to 10 lines.

Question 4.2

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 must be a multiple of 4 bytes in size (including the header bytes). Answer the following questions assuming that the free list contains only two blocks : 28 bytes and 16 bytes (in that order).

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

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

(c) Describe a sequence of `malloc` calls that succeeds with a *first-fit* policy and fails with a *next-fit* policy.

Question 4.3

Is it possible for a TLB to contain (at the same point in time) several entries that have the same physical address ? Justify your answer. Expected answer length : 5 to 10 lines.

1. Notes :
   - You are not required to provide the exact C code. A clear description (for instance, using pseudo-code) is sufficient.
   - You are allowed to introduce additional global variables if necessary.

Question 4.4

We consider a set of applications running on an operating system, which implements paging and uses a swap partition on the hard disk drive. We observe that the workload suffers from thrashing. For each of the following actions, discuss whether it is likely to improve the situation (in any case, briefly justify your answer) :

(a) Increasing the size of the swap partition on disk

(b) Replacing the disk with a faster one

(c) Replacing the CPU with a faster one (higher clock frequency)

(d) Replacing the CPU with another one, very similar but with bigger caches

(e) Increasing the capacity of the physical RAM