Winter 2017 Mid Term Solution

Solution for CS 143A course at University of California, Irvine

Chen Wang[*]

11/8/2019

Contents

1 Basic page tables.

1.1 Address Translation through an example

(5 points) Illustrate organization of the x86, 4K, 32bit 2-level page tables through a simple example. Assume that the hardware translates the virtual address '0xc04005' (binary 0b1100 0000 0100 0000 0000 0101) into the physical address '0x55005'. The physical addresses of the page table directory and the page table (Level 2) involved in the translation of this virtual address are 0x8000 and 0x2000. Draw a diagram, provide a short explanation.

Reference Solution:

Well, it should be simple. Just follow the diagram on page 65 of the slide Lecture 10 - Kernel Page Table. Fill in the actual address number and it should be OK.

2 Shell

This question is not covered in this midterm

3 OS isolation and protection

3.1 User vs. kernel isolation

Reference Solution:

Please refer to the Problem 3.1 of Midterm Fall 2017.

[*]Undergraduate in Computer Engineering, Samueli School of Engineering, University of California, Irvine. (chenw23@uci.edu)

3.2 Page Flag Design

(5 points) Imagine you have hardware that is identical to x86, but does not have a user bit in the page tables. What changes need to be made to xv6 to ensure isolation of the kernel from user-processes?

Reference Solution:

We need to modify the MMU part. In this part, rather than directly looking up in the directory, we will check whether the address is greater than 2GB. If the address is greater than 2GB, then it must be from a system call. Otherwise, the system can raise a fault.

4 OS organization.

Imagine you want to optimize xv6 to run a large number of very small processes. A realistic example can be a web server that implements a Facebook's login page—you have to isolate each user in its own process, otherwise a single exploit from any user reveals accounts of all users going through the login page, but at the same time each process is very small (it just sends out a simple HTML page). Entire logic of such web server program can fitin 2-3K of memory.

4.1 Page count

(10 points) You start by analyzing the overheads involved into creating a process. How many pages of memory are allocated when xv6 creates a smallest process? Count both user-level and kernel resources.

Reference Solution:

Since the process is small, we assume each part of the user-level program can fit in one page.

For user-level: One for text, one for data, one for heap, one for stack, and 1 for the Page table directory, 1 for the Page table. Totally at least 6. For kernel-level: 0 ~ 234MB, and we know that a page maps 4MB of space, that is it needs `234MB/4MB` pages

```
234 / 4 + 6
```

```
## [1] 64.5
```

Therefore, in total there should be 65 pages.

4.2 Optimizing Suggestion

(10 points) Suggest a set of changes to xv6 aimed at minimizing the number of pages that are required for creating very small processes, e.g., the once that are 1K in size.

Reference Solution:

Carefully analyze what system calls are used by these web page processes. Then in the mapped kernel space, omit out the functions that are not used by these small programs. It should be common that the small programs should only use limited functions of the system - otherwise, they won't be small.