15-213: Introduction to Computer Systems
Written Assignment #6

This written homework covers virtual memory.

Directions

Complete the question(s) on the following pages with single paragraph answers. These questions are not meant to be particularly long! Once you are done, submit this assignment on Canvas.

Below is an example question and answer.

Q: Please describe benefits of two's-complement signed integers versus other approaches.

A: Other representations of signed integers (ones-complement and sign-and-magnitude) have two representations of zero (+0 and −0), which makes testing for a zero result more difficult. Also, addition and subtraction of two's complement signed numbers are done exactly the same as addition and subtraction of unsigned numbers (with wraparound on overflow), which means a CPU can use the same hardware and machine instructions for both.

Grading

Assignments are graded via *peer review:*

1. Three other students will each provide short, constructive feedback on your assignment, and a score on a scale of 1 to 20. You will receive the maximum of the three scores.
1. You, in turn, will provide feedback and a score for three other students (not the same ones as in part 1). We will provide a *rubric*, a document describing what good answers to each question look like, to assist you. You receive five additional points for completing all of your peer reviews.

Due Date

This assignment is due on July 20 by 11:59pm Pittsburgh time (currently UTC−4). Remember to convert this time to the timezone you currently reside in.

Peer reviews will be assigned roughly 12 hours later, and are due a week after that.

Question #1

Virtual addressing is one of the most innovative ideas in computer science. How does virtual memory keep address spaces separate, and how can it share information between address spaces?

Virtual memory keeps address spaces separate by ensuring that each process has its own page table that maps virtual addresses to physical addresses. This is done inside the operating system. Therefore, multiple processes can access the same virtual address simultaneously--this is possible because the virtual address within each process would map to a different physical page in physical memory.

The OS can share information between address spaces by mapping virtual pages in each of the page tables to the same physical page of memory. This is most useful for code libraries--if multiple processes use the same code library, any process that needs the library can map

Question #2

Assume a System that has

- 1MB of virtual memory
- 4KB page size
- 512KB of physical memory
- A two way set associative TLB with 8 total entries

How many bits are needed to represent the virtual address space? How many bits are needed to represent the physical address space? How many bits are needed to represent a page offset? How many bits are needed to represent the TLBI (TLB index)?

There are 20 bits needed to represent the virtual address space.

There are 19 bits needed to represent the physical address space.

There are 12 bits needed for the page offset.

There are 2 bits needed to represent the TLBI.

Below is the TLB of this system:

Index	Tag	PPN	Valid
0	0x06	0x15	1
	0x18	0x58	0
1	0x3F	0x27	0
	0x2A	0x72	1
2	0x1B	0x35	1
	0x28	0x4B	0
3	0x0D	0x15	1
	0x1D	0x5F	0

Use the TLB to translate the virtual address 0x18613. What is the VPN? What is the TLB index? What is the TLB tag? Is it a TLB hit? If so, what is the physical address?

The VPN is 0x18.

The TLB index is 0x0.

The TLB tag is 0x6.

It is a TLB hit.

The physical address is 0x15613.

Use the TLB to translate the virtual address 0x77513. What is the VPN? What is the TLB index? What is the TLB tag? Is it a TLB hit? If so, what is the physical address?

The VPN is 0x77.

The TLB index is 0x3.

The TLB tag is 0x1D.

It is not a TLB hit.

Question #3

You are tasked with the design and implementation of a virtual addressing system. The system has a 64-bit address space and 2KB page tables. Assume the linux default 4096 byte page size. Each page table entry is 64 bits. How much memory would be necessary for a single-level page table using the aforementioned parameters? Comparing this to the 2KB page tables, which scheme would you use if you wanted to optimize for space and why? However, what is one advantage in using a single-level page table over multi-level page tables? How can we offset this disadvantage of the multi-level page tables? (Hint: what other pieces of hardware are available to speed up page table entry look-ups?)

Assuming 2KB byte page size and 2^64 total bytes to address, then there is 2^64 total bytes / 2048 bytes per page = 2^53 total pages. Since each page table entry requires 8 bytes, then the page table size should be 2^53 * 8 = 2^56 bytes total.

We should use multi-level page tables because it will decrease the total amount of space needed for page tables.

In using a single-level page table, you only need to do one lookup versus k lookups for a k-level page table.

We can offset the disadvantages using TLBs to boost the performance of the VM system by caching PTEs from page tables at different levels.