Homework #3 – 24 pts
Submit your homework to TEACH by Fri. 10/26/2018, at 11:59pm

Your submission should be comprised of one item: a .pdf file containing answers to the questions. Legible, handwritten solutions are acceptable for undergraduate students (472). The handwritten solutions must be electronically scanned and submitted as a PDF file. Graduate students (572) must compose their solutions in MS Word, LaTeX, or some other word processor and submit the results as a PDF file.

Clarification (10/24/18): As discussed in class on Tuesday, when I refer to 4KB or 16KB pages, I'm actually referring to 4KiB (4,096 bytes) and 16KiB (16,384 bytes). This will make the math easier.

If you are wondering why I bothered including the following problems on the homework, it's to illustrate the importance of a large page size when there is a large amount of addressable memory. With a small page size, there is a multitude of page table entries that must potentially be maintained for each individual program running in the operating system. In many cases the memory might be better managed using a larger page table size.

1. (2 pts) For a 4K~~B~~ page, and a 32 bit address space, calculate the amount of memory needed to store a process's page tables. Assume each entry in the page table requires 12 bytes. Show all calculations.

   32 bit address space = 2^{32} bytes of addressable memory

   4K = 2^{12} bytes of memory per page

   $\frac{2^{32}}{2^{12}} = 2^{20}$ page table entries

   2^{20} x 12 (bytes per entry) = 12582912 bytes for the table

2. (2 pts) For a 4K~~B~~ page, and a 64 bit address space, calculate the amount of memory needed to store a process's page tables. Assume each entry in the page table requires 12 bytes. Show all calculations.

   64 bit address space = 2^{64} bytes of addressable memory

   4K = 2^{12} bytes of memory per page

   $\frac{2^{64}}{2^{12}} = 2^{52}$ page table entries

   2^{52} x 12 (bytes per entry) = 54043195528445952 bytes for the table!!

3. (2 pts) For a 16K~~B~~ page, and a 32 bit address space, calculate the amount of memory needed to store a process's page tables. Assume each entry in the page table requires 12 bytes. Show all calculations.

   32 bit address space = 2^{32} bytes of addressable memory

   16K = 2^{14} bytes of memory per page

   $\frac{2^{32}}{2^{14}} = 2^{18}$ page table entries

   2^{18} x 12 (bytes per entry) = 3145728 bytes for the table

4. (2 pts) For a 16K~~B~~ page, and a 64 bit address space, calculate the amount of memory needed to store a process's page tables. Assume each entry in the page table requires 12 bytes. Show all calculations.

64 bit address space = 2^{64} bytes of addressable memory
16K = 2^{14} bytes of memory per page
$\frac{2^{64}}{2^{14}} = 2^{50}$ page table entries
2^{50} x 12 (bytes per entry) = 13510798882111488 bytes for the table!

Note: This next question is intended to give students a chance to demonstrate their understanding of caches. If you worked on homework 2 and understood the material, this should be a very quick question to complete. You can reuse a lot of your existing work.

5. (6 pts) Suppose the existence of an **associative cache design** with **N = 2.** Assume that the main memory uses a 32-bit address (and that each address maps to a single byte). If the cache scheme uses the following breakdown of the address bits:

Tag	Index	Offset
31-6	5-3	2-0

   a. What is the cache block size (in words)?
      3 bits allocated to the byte offset. 2^3 = 8 bytes → 8/4 = 2 words
   b. How many block indices does the cache have?
      3 bits allocated to the index. 2^3 = 8 entries (i.e. indices)
   c. Including space for the valid bits, tags, and actual block data, how many bits would be required to actually implement this hypothetical cache?
      For each row index (since there are N=2 block entries for each index):
      2 * 8 bytes of data = 128 bits
      2 * 1 valid bit for each entry = 2 bits
      2 * 26 bits for each tag = 52 bits
      ➔ Adds up to 182 bits per index
      182 bits * 8 entries = 1456 total bits required
      In real life there could actually be additional requirements such as an LRU bit, dirty bit, etc. For this problem you didn't have to take those into consideration.

6. (10 pts) Using the cache design from the previous problem, assume the following byte-addressed cache references are recorded. Assume that the cache is initially empty. Also assume that accesses occurred from left to right (e.g. address 0 was requested, then address 4, followed by address 16, etc).

Byte Address											
0	4	16	132	232	160	1024	30	140	3100	180	2180

Byte Address	Block address $\text{floor}\left(\frac{byte\ address}{block\ size\ in\ bytes}\right)$	Block Index Block address % num_of_indices	Tag	Hit ?
0	0	0	00000000 00000000 00000000 00	M
4	0	0	00000000 00000000 00000000 00	H
16	2	2	00000000 00000000 00000000 00	M
132	16	0	00000000 00000000 00000000 10	M
232	29	5	00000000 00000000 00000000 11	M
160	20	4	00000000 00000000 00000000 10	M
1024	128	0	00000000 00000000 00000100 00	M
30	3	3	00000000 00000000 00000000 00	M
140	17	1	00000000 00000000 00000000 10	M
3100	387	3	00000000 00000000 00001100 00	M
180	22	6	00000000 00000000 00000000 10	M
2180	272	0	00000000 00000000 00001000 10	M

a. What was the number of cache hits?
1

b. Create a table to show the final state of the cache including the value of each index, the valid bits, and the tags.

This depends on the cache replacement policy that you used. For this example, let's assume that an LRU policy is used. If you used a randomized replacement scheme then your tag entries for index 0 might look different.

Index	V	Tag	V	Tag
0	1	00000000 00000000 00000100 00	1	00000000 00000000 00001000 10
1	1	00000000 00000000 00000000 10	0	?
2	1	00000000 00000000 00000000 00	0	?
3	1	00000000 00000000 00000000 00	1	00000000 00000000 00001100 00
4	1	00000000 00000000 00000000 10	0	?
5	1	00000000 00000000 00000000 11	0	?
6	1	00000000 00000000 00000000 10	0	?
7	0	?	0	?