CISC 324, Winter 2018, Assignment 4

This assignment is due Thursday, March 8 in lecture

Readings

Course reader — Pages 50-55. Sections 1-4 of *Overview of Memory Management* as well as the first few paragraphs of section 5 that introduce *paging combined with segmentation*.

Textbook — Memory management receives extensive coverage in chapters 8 and 9. Textbook readings for this assignment are given with the problems below. Assignment 5 adds more readings in these two chapters.

Terminology for Memory Management

kilobyte — 2^{10} bytes = 1024 bytes. Usually *kilo* means exactly 1000 as in *kilometer* or *kilowatt*. For memories, *kilo* means 1024 because memory size is a power-of-two. Ten address bits are needed to access a one kilobyte memory.

megabyte — 2^{20} bytes = 1024 kilobytes = approximately 1 million bytes (1,048,576 bytes). Twenty address bits are needed to access a one megabyte memory.

gigabyte — 2^{30} bytes = 1024 megabytes = approximately 1 billion bytes (1,073,741,824 bytes). Thirty address bits are needed to access a one megabyte memory.

millisecond — 10^{-3} seconds = 1/1000 of a second. The prefix *milli* means exactly 1/1000, as in *millimeter* or *milliliter*.

microsecond — 10^{-6} seconds = 1/1000 of a millisecond.

nanosecond — 10^{-9} seconds = 1/1000 of a microsecond.

Virtual address and *logical address* are synonymous. Address translation hardware converts a virtual address to a physical address, accessing segment tables and page tables as needed to do this. See textbook section 8.1.3 for an introduction to logical and physical addresses.

Questions

1) This question is about the memory pyramid I discussed in lecture: a small amount of fast, expensive memory at the top of the pyramid and large amounts of slow, cheap memory at the bottom of the pyramid. Data migrates up and down the pyramid, moving toward the top when it is accessed and shifting back down when it hasn't been accessed for a while. Briefly describe how each of the following three types of data movement occur; indicate whether the compiler, the operating system and/or the hardware is involved. (I use *main memory* to refer to the physical memory that is accessible to the CPU via the memory bus. This is also referred to as the RAM, or simply as the *memory*.)
   - data moves between main memory and general-purpose registers
   - data moves between disk and main memory
   - data from main memory is stored in a cache

   See textbook section 8.1.1 for introductory information about registers, cache etc.

2) (a) Briefly describe the difference between absolute code and relocatable code (textbook section 8.1.2).
   (b) Briefly describe dynamic loading and dynamically linked libraries (sections 8.1.4 and 8.1.5).
   Optional: read wikipedia "Dependency Injection", a flexible way to provide libraries and other services to a client.

3) Refer to the introductory paging example, Section 3 in course reader pages 51-54. [Study Section 3 carefully, particularly if paging is new to you.] Translate the following virtual addresses to physical addresses, using the page table in section 3.4 on page 53. To get started, look at the examples of address translation in section 3.5 on page 53.
   (a) virtual address is $\texttt{2FFF}_{16}$ (b) virtual address is $\texttt{30A0}_{16}$

4) Consider a (tiny) computer system in which the virtual memory consists of 8 pages of 1024 bytes each, mapped onto a physical memory of 32 page frames. How many bits are there in a virtual address? How many bits are there in a physical address? Assume that memory is byte-addressed, meaning that every byte of memory has its own address. (The alternative used on some computers is a word-addressed memory, where every word of memory has its own address.)

5a) Locality of addressing: describe three reasons why a process that accesses virtual memory address L is likely to access L or a nearby address in the near future. [Refer to course reader page 50.]

b) A virtual memory address is divided up as follows, using the most significant bits as the page number: [page offset]. Why is this convention used, rather than [offset page]? Hint: relate your answer to the locality of addressing in (a).

Two questions on segmentation with paging

In question 6 you trace the steps in address translation: access a segment table, use those results to find the page table, access the page table to find the frame number, combine the frame number and page offset to form the physical address.

In question 7 you compute the sizes of segment tables and page tables.

Readings on this topic

Introduction to segmentation: textbook section 8.4.

Introduction to paging: textbook section 8.5 up to the fourth paragraph of 8.5.2. (The rest of 8.5.2 is about translation look-aside buffer; this is covered in assignment 5.)

Also refer to course reader page 55: the first few paragraphs of section 5. (The rest of section 5 provides details including the translation look-aside buffer; this is covered in assignment 5.)

6) This question is given on the next page.

7) Using segmentation with paging, this table show three methods of dividing up virtual addresses. For each case, calculate maximum sizes for a segment, a segment table, and a segment's page table. You may state your answers in terms of bytes, kilobytes, megabytes, gigabytes, or using a format like "2^{15} bytes". (Assume the memory is byte-addressed.)

	Bits in virtual address	Page size	Size of entries in segment table and page table	Maximum number of segments	Maximum size of a segment	Maximum size of a segment table	Maximum size of a segments' page table
(i)	16 bits	128 bytes	8 bytes	64 (2^6)			
(ii)	32 bits	1024 bytes	8 bytes	4096 (2^{12})			
(iii)	32 bits	8192 bytes	8 bytes	65536 (2^{16})			

Here are hints to get you started on problem 7.

- The "Maximum number of segments" tells you how many bits (of the virtual address) are used to encode the segment number. All the remaining bits in the virtual address are used as "offset within the segment": use this information to figure out the maximum size of a segment.
- To find the maximum size of a segment table, you need to multiply "Max number of entries in a segment table" * "number of bytes per entry". Column 4 tells you the first value, column 3 tells you the second value.
- To find the maximum size of a segment's page table, you need to multiply "Max number of entries in a page table" * "number of bytes per entry". Again, column 3 tells you the second value. To find the first value (max number of entries in a page table) you need to divide the "max size of a segment" by the page size.

6) **Print this page and write your answers directly on it**.

A computer uses segmentation with paging. A virtual address consists of six hexadecimal digits, SSPPEE, where SS is a segment number and PPEE is the offset within the segment. The segment is paged, with PP as the page number and EE as the page offset. (The page size is $100_{16} = 256_{10}$.) Physical addresses are four hexadecimal digits.

The figure below shows selected parts of the current memory contents. All numbers are hexadecimal. Segment and page tables are indexed starting at 0. Two-digit frame numbers are stored in the segment tables, in the page tables, and in STBR (Segment Table Base Register). These frame numbers are converted to physical addresses by appending 00_{16}. For example, the entry 0F in a segment table means that the segment's page table is located at address 0F00.

A * means the page is on disk. The *segment table* and *page table* labels are given here as a hint; the operating system does not use such labels during address translation.

```
                 STBR value is 06    List of free page frames: 1A, 08, 21, 16, 0E

0200 | 0F |  segment            0A00 | *  |  page table         3D00 | 5C |  page table
     | 13 |  table with              | 57 |                          | 1E |
     | 09 |  5 entries               | *  |                          | 5A |
     | *  |                          | *  |                          | 59 |
     | 4E |
                                0F00 | *  |  page table         4C00 | *  |  page table
0600 | 09 |  segment                 | 25 |                          | 3B |
     | 0A |  table with                                              | *  |
     | 4C |  4 entries          1300 | 18 |  page table              | 20 |
     | 3D |
                                3000 | 0F |  segment            4E00 | 11 |  page table
0900 | 04 |  page table              | 09 |  table                   | *  |
                                     | 13 |                          |    |
                                     | 4C |                          |    |
```

a) The following sequence of virtual memory addresses are generated by the current process. What are the corresponding physical addresses? In answering this question, assume that new pages are brought into main memory using the given list of free page frames (1A, 08, 21, 16, 0E). Write changes to the segment or page tables into the above figure. To get you started: the first address is segment 0, page 0, offset B2. Use STBR to find the segment table; at index 0 this table contains 09; find the page table at 0900; at index 0 this page table contains 04; append B2 to form the main memory address 04B2.

Virtual Memory Address	Physical Address	Is There a Page Fault?
0000B2	____________	YES NO
020216	____________	YES NO
02015E	____________	YES NO
0202A7	____________	YES NO

b) Following the four accesses performed in part (a), the system switches execution to another process. The value "02" is loaded into the STBR. What segments, if any, are shared between this new process and the previous process?

c) Describe what happens when the new process (STBR value 02) references address 030142. Keep using the list of free page frames from part (a). Write changes to the segment or page tables into the above figure. Can you determine what main-memory address results after address translation? If so, state the address. If not, state what additional information is needed.