Homework 6

CST 334: Operating Systems
Dr. Glenn Bruns

Homework - Week 6

Please note: All homework is to be done completely on your own.

Note: Problem numbering seems to be broken when this document is viewed using FIrefox. It is correct when viewing with Chrome.

1. **Reading: Address Translation**. Please read OSTEP chapters 15, and answer the following questions by editing file hw6.txt. The address translation method discussed in chapter 15 is known as "dynamic relocation", or, "base and bounds" addressing.

    a. Suppose we have a simple non-segmented virtual address space, where virtual addresses are 5 bits long. How big is the virtual address space? In other words, how many bytes can be addressed?

    b. Suppose we have a simple non-segmented virtual address space of 16K bytes. How many bits in a virtual address (assuming each byte is addressable) ?

    c. (T/F) In dynamic relocation, the base and bounds values are part of the process state.

    d. (Y/N) In base and bounds addressing, can the base and bounds values for a process change during the lifetime of a process?

    e. Suppose we are using base and bounds addressing. If the virtual address is 5321, the base register value is 100, and the bounds register value is 4500, then what is the corresponding physical address. (The values given are decimal values.)

    f. (T/F) A user process can change its own base and bounds values.

    g. If a process attempts to access a memory location outside its own virtual address space, what is the result? 1) a trap 2) an exception

2. **Reading: Segmentation**. Read read OSTEP chapter 16, and answer the following questions by editing hw6.txt.

    a. Suppose we have a segmented virtual address space, with 2 segments. If a virtual address uses 1 bit for the segment, and 7 bits for the offset, how big is the virtual address space? (As in problem 1a, please give your answer as a integer without units.)

    b. (T/F) Virtual addresses from two different processes must never translate to the same physical address.

    c. (T/F) With segmentation, the number of segments used is never more than 3.

    d. (T/F) As a programmer, it is helpful to know the order of the code, heap, and stack segments in physical memory.

e. (T/F) Segmentation helps in the problem of sharing code between processes.

f. (T/F) The address space of each segment must be the same size.

Suppose we are using segmentation as an extension of simple base-and-bounds address translation. Suppose, like in Chapter 16 of OSTEP, that we have segments “Code”, “Heap”, and “Stack”, where the base and bounds of each segment are as follows (all values are decimal):

Segment	Base	Bound
Code	1000	4096
Heap	16000	2048
Stack	8000	2048

For each of the following virtual addresses, give the physical address (assume addresses “grow positive” in each segment), or that a segmentation fault has occurred. Provide the physical address as a decimal number.

g. (Code, 153)

h. (Heap, 3327)

i. (Code, 6211)

j. (Stack, 32)

3. **AWK.** Problem 3 of last week’s AWK lab asks you to create 5 small awk programs. Write these awk programs and submit them as described below. Note: your programs will be tested on simulator output, but not exactly the same output as shown in the lab.

Here’s some example simulator output:

```
ptr[2] = Alloc(5)  returned 1001 (searched 3 elements)
Free List [ Size 3 ]:  [ addr:1000 sz:1 ] [ addr:1006 sz:2 ] [ addr:1008 sz:92 ]

ptr[3] = Alloc(8)  returned 1008 (searched 3 elements)
Free List [ Size 3 ]:  [ addr:1000 sz:1 ] [ addr:1006 sz:2 ] [ addr:1016 sz:84 ]

Free(ptr[3]) returned 0
Free List [ Size 4 ]:  [ addr:1000 sz:1 ] [ addr:1006 sz:2 ] [ addr:1008 sz:8 ] [ addr:1016
sz:84 ]
```

The idea is that the operating system keeps track of memory that it can provide to processes. The processes make Alloc() calls to allocate memory, and Free() calls to release memory. For example, a process calls Alloc(5) to get 5 bytes of memory.

The operating system keeps track of free memory in a list, called the ‘free list’. The free list contains “chunks” of free memory. Each line above that starts with ‘Free List’ is showing the number of elements in the free list, followed by details about each element. Each element represents a chunk of available memory, with values giving the beginning address of the chunk, and the number of bytes in the chunk.

In the first ‘Free List’ line above, the free list has 3 elements. The first element of the list shows 1 byte available at address 1000; the second elements shows 2 bytes available at address 1006, etc.

Testing your code: On mlc104, the directory `/home/CLASSES/brunsglenn/cst334/hw/hw6` contains a tar file hw6.tar. Copy this to your own directory and untar it. You will see a folder 'awk_problem'. Underneath this directory is one sub-directory for each of the five small awk programs you need to write. In each of these subdirectories is the file for the awk program you need to write, a Makefile, and a test script test1.sh. You know how to use the test script to test your code. Read the Makefile in each subdirectory and use it if you like.

4. **Peerwise.** Spend an hour on Peerwise. Consider creating a question related to chapters 15 or 16 of OSTEP , or something from lecture. Also, answer, rate, and comment meaningfully on other student's questions.

Submitting. Submit your homework on iLearn as 2 files:

- **parts 1 and 2:** an edited version of the **hw6.txt** file included in the homework assignment on iLearn

- **part 3:** your files **count_allocs.awk**, **freesize.awk**, **list_sizes.awk**, **num_bytes.awk**, and **succ_reqs.awk**. Please use these filenames when you submit. When I download your files, iLearn will automatically add your names to the filenames. Do not submit a tar file or other archive file!

Grading: Part 1 of the homework is worth 21 points (3 pts/question), part 2 is worth 30 points (3 pts/question), part 3 is worth 50 points (10 points for each awk program). The maximum score for the homework is 100.

Some of the tests I use to grade your awk scripts will differ from the scripts I have provided you. I will deduct 3 points per script if your code is untidy. Peerwise work is graded separately.