Name: Gustavo Hammerschmidt.

CS433 Written Homework 5

(70 points) All question numbers refer to exercises in the textbook (10th edition). Make sure you use the right textbook and answer the right questions. Students must finish written questions individually. Type your answers and necessary steps clearly.

1. (10 points) 9.13 Given six memory partitions of 100 MB, 170 MB, 40 MB, 205 MB, 300 MB, and 185 MB (in order), how would the first-fit, best-fit, and worst-fit algorithms place processes of size 200 MB, 15 MB, 185 MB, 75 MB, 175 MB, and 80 MB (in order)? Indicate which—if any—requests cannot be satisfied. Comment on how efficiently each of the algorithms manages memory.

Memory partition of size MB:

```
Index:  0    1     2   3     4   5                                   FIRST-FIT

        (100, 170, 40, 205, 300, 185)    ->  Initially

        (100, 170, 40, 5, 300, 185)    -> 200 MB placed in partition 3.

        (85, 170, 40, 5, 300, 185)    ->  15 MB placed in partition 0.

        (85, 170, 40, 5, 115, 185)    -> 185 MB placed in partition 4.

        (10, 170, 40, 5, 115, 185)    -> 75 MB placed in partition 0.

        (10, 170, 40, 5, 115, 10)    -> 175 MB placed in partition 5.

        (10, 90, 40, 5, 115, 10)    -> 80 MB placed in partition 1.
```

Memory partition of size MB:

```
Index:  0    1     2   3     4   5                                   BEST-FIT

        (100, 170, 40, 205, 300, 185)    ->  Initially
```

(100, 170, 40, 5, 300, 185) -> 200 MB placed in partition 3.

(100, 170, 25, 5, 300, 185) -> 15 MB placed in partition 2.

(100, 170, 25, 5, 300, 0) -> 185 MB placed in partition 5.

(25, 170, 25, 5, 300, 0) -> 75 MB placed in partition 0.

(25, 170, 25, 5, 125, 0) -> 175 MB placed in partition 4.

(25, 170, 25, 5, 45, 0) -> 80 MB placed in partition 4.

Memory partition of size MB:

Index: 0 1 2 3 4 5 WORST-FIT

(100, 170, 40, 205, 300, 185) -> Initially

(100, 170, 40, 205, 100, 185) -> 200 MB placed in partition 4.

(100, 170, 40, 190, 100, 185) -> 15 MB placed in partition 3.

(100, 170, 40, 5, 100, 185) -> 185 MB placed in partition 3.

(100, 170, 40, 5, 100, 110) -> 75 MB placed in partition 5.

(100, 170, 40, 5, 100, 110) -> 175 MB -- REQUEST COULD NOT BE SATISFIED

(100, 90, 40, 5, 100, 110) -> 80 MB placed in partition 1.

The first-fit generates free blocks of a larger size towards the end of the address space; the best-fit uses the address space better, while it searches for the best fittable lacune, which leads to small lacunes of memory throught the address space; and the worst-fit tends to use all the larger memory blocks, fragmenting the address space into medium-size lacunes of memory, which increase the number of requests of large memory that are not satisfied.

2. (5 points) 9.15 Compare the memory organization schemes of contiguous memory allocation and paging with respect to the following issue0s:

a. External fragmentation

Both of them can have external fragmentation

b. Internal fragmentation

Both of them will suffer internal fragmentation if their memory space isn’t fulfilled.

c. Ability to share code across processes

Only Paging allows to share code across processes.

3. (10 points) 9.21 Assuming a 1-KB page size, what are the page numbers and offsets for the following address references (provided as decimal numbers)?

a.	21205	=>	Page Number: 20;	Offset:	725.
b.	164250	=>	Page Number: 160;	Offset:	410.
c.	121357	=>	Page Number: 118;	Offset:	525.
d.	16479315	=>	Page Number: 16093;	Offset:	83.
e.	27253187	=>	Page Number: 26614;	Offset:	451.

4. (5 points) 9.22 The MPV operating system is designed for embedded systems and has a 24-bit virtual address, a 20-bit physical address, and a 4-KB page size. How many entries are there in each of the following?

a. A conventional, single-level page table

2 ^ (24 – 12) = 2 ^ (12) = 4096 entries.

b. An inverted page table

2 ^ (20 – 12) = 2 ^ (8) = 256 entries.

What is the maximum amount of physical memory in the MPV operating system?

Physical Memory amount = 2 ^ (20) = 1,048,576 bytes.

5. (5 points) 9.23 Consider a logical address space of 2,048 pages with a 4-KB page size, mapped onto a physical memory of 512 frames.

a. How many bits are required in the logical address?

log(2 ^ (page size + nº pages) = 2 ^ (12 + 11))/log(2) = 23
23 bits are required.

b. How many bits are required in the physical address?

log(2 ^ (page size + nº frames) = 2 ^ (12 + 9))/log(2) = 21
21 bits are required.

6. (5 points) 9.25 Consider a paging system with the page table stored in memory.

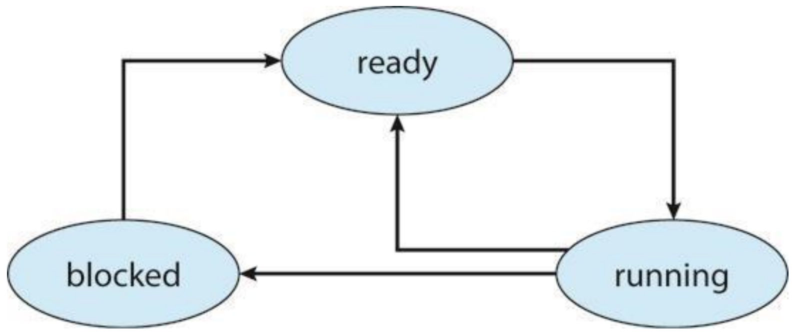
a. If a memory reference takes 50 nanoseconds, how long does a paged memory reference take?

It takes 100 nanoseconds.

b. If we add TLBs, and 75 percent of all page-table references are found in the TLBs, what is the effective memory reference time? (Assume that finding a page-table entry in the TLBs takes 2 nanoseconds, if the entry is present.)

Effective Access Time = 0.75*(50ns) + 0.25*(100ns) + 2 = 64.5 nanoseconds.

7. (5 points) 10.16 A simplified view of thread states is ready, running, and blocked, where a thread is either ready and waiting to be scheduled, is running on the processor, or is blocked (for example, waiting for I/O).



Assuming a thread is in the running state, answer the following questions, and explain your answers:

a. Will the thread change state if it incurs a page fault? If so, to what state will it change?

If a page fault occurs, it will change from running to blocked.

b. Will the thread change state if it generates a TLB miss that is resolved in the page table? If so, to what state will it change?

    It will change the state if the page is not in the main memory to blocked.

c. Will the thread change state if an address reference is resolved in the page table? If so, to what state will it change?

    It won't change its state.

8. (5 points) 10.21 Assume we have a demand-paged memory. The page table is held in registers. It takes 8 milliseconds to service a page fault if an empty page is available or the replaced page is not modified, and 20 milliseconds if the replaced page is modified. Memory access time is 100 nanoseconds.

Assume that the page to be replaced is modified 70 percent of the time. What is the maximum acceptable page-fault rate for an effective access time of no more than 200 nanoseconds?

Effective Access Time = 100 + 7,999,900 * p

200 ns + 0.7 * 200 ns > 100 + 7999900 * p

200 ns + 140 ns > 100 + 7999900 *p

240 ns / 7,999,900 > p

100 ns > 7999900 * p

p < 0.00003

We can allow fewer than one memory access out of 33,332 to page-fault.

9. (10 points) 10.22 Consider the page table for a system with 16-bit virtual and physical addresses and 4,096-byte pages.

Page	Page Frame	Reference Bit
0	9	1
1	–	0

2	10	0
3	15	0
4	6	0
5	13	1
6	8	1
7	12	0
8	7	0
9	–	0
10	5	0
11	4	1
12	1	0
13	0	0
14	–	0
15	2	1

The reference bit for a page is set to 1 when the page has been referenced. Periodically, a thread zeroes out all values of the reference bit. A dash for a page frame indicates that the page is not in memory. The page-replacement algorithm is localized LRU, and all numbers are provided in decimal.

a. Convert the following virtual addresses (in hexadecimal) to the equivalent physical addresses. You may provide answers in either hexadecimal or decimal. Also set the reference bit for the appropriate entry in the page table.

- 0x621C => (0110 0010 0001 1010) => (1000 0010 0001 1100) => 0x821C
- 0xF0A3 => (1111 0000 1010 0011) => (0010 0000 1010 0011) => 0x20A3
- 0xBC1A => (1011 1100 0001 1010) => (0100 1100 0001 1010) => 0x4C1A
- 0x5BAA => (0101 1011 1010 1010) => (1101 1011 1010 1010) => 0xDBAA
- 0x0BA1 => (0000 1011 1010 0001) => (1001 1011 1010 0001) => 0x9BA1

b. Using the above addresses as a guide, provide an example of a logical address (in hexadecimal) that results in a page fault.

A logical address that would result in a page fault would be, in hexadecimal, 0xEC1A for example, because the reference bit is not set in the page table.

c. From what set of page frames will the LRU page-replacement algorithm choose in resolving a page fault?

For the following page frame set: { 10, 15, 6, 12, 7, 5, 1, 0 }.

10 (10 points) 10.24 Apply the (1) FIFO, (2) LRU, and (3) optimal (OPT) replacement algorithms for the following page-reference strings:

- 2,6,9,2,4,2,1,7,3,0,5,2,1,2,9,5,7,3,8,5
- 3,1,4,2,5,4,1,3,5,2,0,1,1,0,2,3,4,5,0,1

Indicate the number of page faults for each algorithm assuming demand paging with three frames. (You only need to do the two above sequences)

```
FIFO


•         2, 6, 9, 2, 4, 2, 1, 7, 3, 0, 5, 2, 1, 2, 9, 5, 7, 3, 8, 5
          |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
Array =   *  *  9  9  9  9  1  1  1  0  0  0  1  1  1  1  7  7  7  5
          *  6  6  6  6  2  2  2  3  3  3  2  2  2  2  5  5  5  8  8
          2  2  2  2  4  4  4  7  7  7  5  5  5  5  9  9  9  3  3  3

          f  f  f  h  f  f  f  f  f  f  f  f  f  h  f  f  f  f  f  f


        18 faults.


•       3, 1, 4, 2, 5, 4, 1, 3, 5, 2, 0, 1, 1, 0, 2, 3, 4, 5, 0, 1
        |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
Array = *  *  4  4  4  4  1  1  1  1  0  0  0  0  0  0  4  4  4  1
        *  1  1  1  5  5  5  5  5  2  2  2  2  2  2  3  3  3  0  0
        3  3  3  2  2  2  2  3  3  3  3  1  1  1  1  1  1  5  5  5

        f  f  f  f  f  h  f  f  h  f  f  f  h  h  h  f  f  f  f  f

        15 faults
```

```
   LRU


   •        2, 6, 9, 2, 4, 2, 1, 7, 3, 0, 5, 2, 1, 2, 9, 5, 7, 3, 8, 5
            |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
   Array =  *  *  9  9  9  9  1  1  1  0  0  0  1  1  1  1  7  7  7  5
            *  6  6  6  4  4  4  4  3  3  3  2  2  2  2  5  5  5  8  8
            2  2  2  2  2  2  2  7  7  7  5  5  5  5  9  9  9  3  3  3

            f  f  f  h  f  h  f  f  f  f  f  f  f  h  f  f  f  f  f  f


            17 faults.
```

OPT

```
•         2, 6, 9, 2, 4, 2, 1, 7, 3, 0, 5, 2, 1, 2, 9, 5, 7, 3, 8, 5
          |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
Array =   *  *  9  9  9  9  1  1  1  1  1  1  1  1  9  9  9  3  8  8
          *  6  6  6  4  4  4  7  3  0  5  5  5  5  5  5  5  5  5  5
          2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  7  7  7  7

          f  f  f  h  f  h  f  f  f  f  f  h  h  h  f  h  f  f  f  h

          13 faults.
```

```
•        3, 1, 4, 2, 5, 4, 1, 3, 5, 2, 0, 1, 1, 0, 2, 3, 4, 5, 0, 1
         |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
Array =  *  *  4  4  4  4  4  4  4  2  2  2  2  2  2  3  4  5  5  5
         *  1  1  1  1  1  1  3  3  3  3  1  1  1  1  1  1  1  1  1
         3  3  3  2  5  5  5  5  5  5  0  0  0  0  0  0  0  0  0  0

         f  f  f  f  f  h  h  f  h  f  f  f  h  h  h  f  f  f  h  h

         12 faults.
```