CprE 308 Operating Systems: Principles and Practice **Spring 2016**

Midterm Exam 2

(Scheduling, Memory Management, and File System)

Student Name: Lab Section Number:

Note: **The total score on this exam is 50 points. For the bonus questions, if you have a working solution, you can get 5 additional bonus points. The maximum time allowed is 50 minutes. *This is a closed book exam. But you can bring a sheet of note to the exam.*** Explain your assumptions, if any, and your reasoning to get full credit on the problems. Doing so also helps to get **some partial points** (if meaningful), even though the answer is not exactly right.

1. (5 points) Mark each of the following statements as true or false.

a. The size of an inverted page table is determined by the size of virtual memory. FALSE

b. Each process needs its own page table. TRUE

c. Second Chance Algorithm replaces the page loaded longest time ago if R bit is 0, without giving it a 2nd chance. TRUE

d. A symptom of Belady's Anomaly is that the processor is spending a significant amount of time doing nothing—all of the processes are waiting on page-in requests (disk reads). FALSE

e. Copy-on-Write approach allows parent and child processes share read-only pages. When a shared page has to be modified by the child process, the page fault handler will make a copy of that page and update the page tables of both the parent and child processes. TRUE

f. On Unix, Case is important in the file names, e.g., "abc" is different "aBc". TRUE

g. Each process has its own *file descriptor table* for the set of open files. TRUE

h. The *read* system call **read(**int **fd,** void ***buffer,** size_t **n)** tells the system to read up to *n* bytes of data into the area of memory pointed to by *buffer* from the file referred to by *fd*. The return value can be smaller than *n.* TURE

i. A shell can handle I/O redirection (like running a command "who > file &") by creating a new process in which to run a command "who", where the new process closes the file descriptor 1 (standard output) and then opens "file" (the arguments indicate that "file" is only to be written to). When "who" runs, its output is then written to "file". TURE

j. On Linux/Unix, the command chmod (change file permission) will update the last modified time of inode data structure. FALSE

2. (4 points) Five jobs are waiting to be run. Their expected run times are 9, 6, 3, 5, and X. In what order should they be run to minimize average response time? (Your answer will depend on X).

Answer:

They should be run in increasing run time lengths. So 3, 5, 7, and 9 with X put in where it falls in the sequence. It can also be shown as follows:

Case 1: x<=3: X, 3, 5, 7, 9

Case 2: 3<X<=5: 3, X, 5, 7, 9
Case 3: 5<X<=6: 3, 5, X, 7, 9
Case 4: 6<X<=9: 3, 5,7, X, 9
Case 5: X>9: 3, 5, 7, 9, X

3. (7 points) Please use the Gantt chart to illustrate how the following three jobs are scheduled, and calculate the **average response times**, when (1) Round Robin (Quantum = 5) is used, and (2) FIFO is used, respectively.

Note: If you do not use Gantt chart, we reserve the right **to reduce up to 2 points,** though your answer may be right.

P1 (Arrival Time: 0): **3** (CPU Burst) 30 (IO Burst) **8** (CPU Burst) 12 (IO Burst) **1** (CPU Burst)

P2 (Arrival Time: 13): **23** (CPU Burst) 12 (IO Burst) **2** (CPU Burst)

P3 (Arrival Time: 16): **4** (CPU Burst) 6 (IO Burst) **8** (CPU Burst)

Answer: Please take a look at Lecture 21 slides.

Round Robin (Quantum = 5)

P1	IDLE	P2	P3	P2	P2	P3	P2	P1	P3	P2	P1	IDLE	P2	IDLE	P1	Finished
0	3	13	18	22	27	32	37	42	47	50	53	56	65	67	68	69

FIFO

P1	IDLE	P2	P3	P1	P3	P2	IDLE	P1	Finished							
0	3	13	36	40	48	56	58	60	61							

Average response time:

Round Robin = (69+(67-13)+(50-16))/3 =52.33

FIFO = (61+(58-13)+(56-16))/3 = 48.66

4. (4 points) For each of the following decimal virtual addresses, compute the virtual page number and offset for a 4-KB page and for an 8-KB page: 20016, 32729, and 60035.

Answer:

For 4-KB page size, the (page, offset) pairs are (4, 3632), (7, 4057) and (14, 2691)
For 8-KB page size, the (page, offset) pairs are (2, 3632), (3, 8153) and (7, 2691)

5. (4 points) A computer with a 64-bit address uses a three-level page table. Virtual addresses are split into a 12-bit top level page table field, 16 bit second level page table field, and 20-bit third level page table field, and an offset. How large are the pages and how many pages are there in the address space?

Answer:

48 bits are used for the virtual page numbers, leaving 16 over for the offset. This yields a 64-KB page. 48 bits for the virtual page implies 2^{48} pages.

6. (4 points) If an instruction takes 20 nsec and a page fault takes an additional 10000 nsec, then give a formula for the effective instruction time if page faults occur once every k instructions.

Answer:

Effective instruction time = 20*(1-1/k)+ (20+10000)*1/k=(20 + (10000/k)) nsec

7. (8 points) Please show the content of the buf1, buf2, buf4, buf5, buf6, buf7, and buf8, after the program ends.

Assume file "x" content is: 0123456789876543210.

buf3 = "abcde";

Program:

```
fdr = open("x", O_RDONLY);
read(fdr, buf1, 10);
fdrw = open("x", O_RDWR);
read(fdrw, buf2, 10);
write(fdrw, buf3, 5);
fdrw2 = dup(fdrw);
read(fdrw2, buf4, 2);
lseek(fdrw, (off_t)–12, SEEK_CUR);
read(fdrw2, buf5, 5);
lseek(fdrw, (off_t)8, SEEK_SET);
read(fdrw2, buf6, 5);
lseek(fdrw2, (off_t)8, SEEK_END);
read(fdrw, buf7, 5);
lseek(fdrw, (off_t)–15, SEEK_CUR);
read(fdrw, buf8, 5);
```

Answer: Please take a look at Lecture 35 slides.

buf1=0123456789

buf2=0123456789

buf4=32

buf5=56789

buf6=89abc

buf7=empty

buf8=45678

8. (4 points) Describe the disk operations, step by step, that are needed to fetch the inode for the following file.

/home/ course/cpre308/midterm2.pdf

Assume that the inode for the root directory is in memory, but none of the other required inodes or data blocks along the path are in memory. Also assume that all directories fit in one disk block.

Answer:

The following disk reads are needed:

directory for /
inode for /home
directory for /home
inode for /home/course
directory for /home/course
inode for /home/course/cpre308
directory for /home/course/cpre308
inode for /home/course/cpre308/midgterm2.pdf

Therefore, in all, 8 disk reads are required.

9. (10 points) Page Replacement Algorithms: Consider the following page-reference string

1,4,2,3,2,1,5,6,2,1,4,3,2

For each page reference, specify whether a page fault occurs or not with the following algorithms. Assume that the system has three page frames. Also assume that all the frames are initially empty, so that the first reference to a page will always cost a fault. Please answer the following questions:

(a). **(5 Points)** LRU (Least Recently Used)

Page Accessed	1	4	2	3	2	1	5	6	2	1	4	3	2
Mark if page fault occurs													
Contents of Page frame 1													
Contents of Page frame 2													
Contents of Page frame 3													

(b). **(5 points)** For the optimal algorithm (which has knowledge of future requests), show the page fault sequence as in the previous question.

Page Accessed	1	4	2	3	2	1	5	6	2	1	4	3	2
Mark if page fault occurs													
Contents of Page frame 1													
Contents of Page frame 2													
Contents of Page frame 3													

Answer:

LRU (Least Recently Used)

Page Accessed	1	4	2	3	2	1	5	6	2	1	4	3	2
Mark if page fault occurs	F	F	F	F		F	F	F	F	F	F	F	F
Contents of Page frame 1	1	1	1	3	3	3	5	5	5	1	1	1	2
Contents of Page frame 2		4	4	4	4	1	1	1	2	2	2	3	3
Contents of Page frame 3			2	2	2	2	2	6	6	6	4	4	4

For the optimal algorithm (which has knowledge of future requests), show the page fault sequence as in the previous question.

Page Accessed	1	4	2	3	2	1	5	6	2	1	4	3	2
Mark if page fault occurs	F	F	F	F			F	F			F	F	
Contents of Page frame 1	1	1	1	1	1	1	1	1	1	1	4	-	-
Contents of Page frame 2		4	4	3	3	3	5	6	6	6	-	3	-
Contents of Page frame 3			2	2	2	2	2	2	2	2	2	2	2

The end of the OPT algorithm may vary.
F – Page fault

10. (5 bonus points)

(a) **(2 bonus points)** Please answer the following yes/no or multiple-choice questions.

a.1. **(1 point)** When a process is first launched, the operating system does not know the size of these segments:

(a) Text
(b) Data
(c) Stack
(d) Heap

Answer: c and d

a.2 **(0.5 point)** Using mutual exclusion ensures that a system avoids deadlock.   True or False

Answer: False

a.3 **(0.5 point)** The value of a semaphore can never be negative.   True or False

Answer: True

(b) **(3 bonus points)** We have studied the Readers-Writers Problem in the earlier lecture. When there are multiple threads reading/writing a database, we allow that (1) many threads can read simultaneously, but (2) only one can be writing at any time (When a writer is executing, nobody else can read or write). The following is a solution to this problem.

Two semaphores: database and protector

Initial: protector=1; database =1; rc =0

READER:

```
While (1) {
        down(protector);
        rc++;
        if (rc == 1)  //first reader
                down(database);
        up(protector);

        read();

        down(protector);
        rc--;
        If (rc == 0) then // last one
                up(database);
        up(protector);
        ….
}
```

WRITER:

```
While (1) {
        generate_data();
        down(database);
        write();
        up(database);
}
```

However, there is a problem with this solution. Please clearly explain what this problem is. What are the possible solution(s) to this problem? Assumer there is only one writer in the system. **In case you need more space, you can write down your answer on the back of this page.**

Answer:

The solution above has the writer starvation problem. Readers might continuously enter, while a writer has to wait, though the writer may have arrived earlier than some of those readers. It is a reader-preference solution.

The possible solution can be like:

Four semaphores: database, rsem, xprotector, yprotector

Initial: xprotector=1;   yprotector=1; database =1;   rsem=1;   rc =0

READER:

```
    While (1) {
        down(yprotector);
          down(rsem);
            down(xprotector);
              rc++;
              if (rc == 1)  //first reader
                    down(database);
            up(xprotector);
          up(rsem);
        up(yprotector);

        read();

        down(xprotector);
          rc--;
          If (rc == 0) then // last one
                up(database);
        up(xprotector);
    }
```

WRITER:

```
    While (1) {
        generate_data();

        down(rsem);

        down(database);
          write();
        up(database);

        up(rsem);

    }
```

Cases:

Case 1: Only Readers

- database set
- no queue

Case 2: Only writer

- database and rsem set

Case 3: Both with Reader First

- database set by reader
- rsem set by writer
- 2nd reader wait on rsem
- other readers on yprotector

Case 4: Both with Writer First

- database set by writer
- rsem set by writer
- 1st reader wait on rsem
- other readers wait on yprotector