Roll No____________________ Fall 2012 Section________________

Operating Systems- (A, B, C)
FINAL-FALL 2012

Total Marks: 70

Total Time: 3 hours

Q# 1: [5+5+5+5 Marks]
Give Short answers to the following questions:

a) What kind of fragmentation is introduced in

I) Paged Architecture
II) Segmented Architecture

How are they removed in "Hierarchical page" ("Paged Segment" or Multilevel Paged) architecture. Explain the architecture for a 32 bit system.

b) A number of web browsers open each tab in a separate process, even though each tab essentially runs the same code. This adds to the bulk of the application; increases memory utilization and may slow down the machine. Can browsers use threads instead of processes? Explain?

c) Mr Herry and David both write code to perform the same functionality. Mr Hennry had taken the OS course with you. Now they both run their codes on different hardware systems with different OSes and do a benchmarking analysis of Page faults. Mr. David's program is producing 50% more page faults in comparison to you. Based on your knowledge of OS you can tell him why is that so?

Solution

```
1) Too small physical memory.
2) Page replace policy of particular OS is not good.
3) Code is haphazard.
```

Q# 2: [10 Marks]

Solution:

```
e.a. = 1 us + (0.20 * 1 us) + (0.02 * 20,000 us)
     = 401.2 us
```

Or, if you prefer

```
e.a. = (0.80 * 1 us) + (0.18 * 2 us) + (0.02 * 20,002 us)
     = .8 us + .36 us + 400.04 us
     = 401.2 us
```

Q# 3: [12 Marks]
Five batch jobs A through E, arrive at a computer center at almost the same time. They have estimated running times of A:10, B:6, C:2, D:4, and E:8 minutes. Their (externally determined) priorities are A:3, B:5, C:2, D:1, and E:4, respectively, with 5 being the highest priority. For each of the following scheduling algorithms, draw the

gantt chart and determine the average waiting time. Ignore context switching overhead. For partial credit, you should list the finishing times:

i) Round-Robin (so assume a time slice of 1 unit time)
ii) Preemptive Priority scheduling.
iii) First-come, First-served
iv) Shortest Job First.

```
(a)Round Robin. Preemptive multiprogramming. Each job gets its fair share
of CPU.

Initially all five processes are running, so they each get 1/5 of the CPU,
which means they take 5 times longer to run. This means that it will take
10 mins for C (the shortest job) to complete.

After 10 mins each job will have used 2 mins of CPU time each, so A will
have 8 mins left, B 4, D 2 and E will have 6. Since there are now 4 jobs
they will each get 1/4 the CPU time, which means they will take four times
as long to run. So, it will take 8 mins for D to complete. (So, D took 10
+ 8 = 18 mins).

Now, A will have 6 mins left, B 2, and E 4.
Each process gets 1/3rd of the CPU. Hence B will complete execution in 6
minutes. (B took 10 + 8+ 6= 24 mins)

Now A will have 4 mins and E 2mins left. Each process gets 1/2 of the CPU,
so E will finish executing in 4 mins. (E took 24 + 4= 28 mins)

Since A has only 2 mins left to run and it is the only job on the CPU, it
will finish in to minutes with a total time of (28 + 2 )=30 mins.

Averaging we get (10+18+24+28+30)/5 = 22 mins average turnaround.

(b)
Priority. Each job runs to completion without being preempted.

Since B has the highest priority it will run first, followed by E,
followed by A, followed by C and finally D. There is no preemption, so B
completes after 6 mins, E after 6 mins of waiting for B and 8 mins of
processing (14 mins total), A after 14 mins of waiting for B and E, and 10
mins processing (24 mins total), C after 24 mins of waiting for B,E and A,
and 2 mins processing (26 mins total)and D after 26 mins of waiting for
B,E,A & C and 4 mins of processing (30 mins total). The average is
(6+14+24+26+30)/5 = 100/5 = 20.

(c)

First Come First Served. Processes run to completion in the order of
arrival.

A completes after 10 mins, B completes after (10 +6)=16 mins, C completes
after (16 + 2)=18 mins, D completes after (18+4)=22 mins, and E completes
after (22+8)=30 mins.
```

```
The average is : (10+16+18+22+30)/5 = 96/5 = 19.2

(d)

Shortest job first. Processes will run to completion in the order : C, D,
B, E, A

C completes after 2 mins, D completes after (2+4)=6 mins, B completes
after (6+6)=12 mins, E completes after (12+8)=20 mins, and A completes
after (20+10)=30 mins.
The average is : (2+6+12+20+30)/5 = 70/5 = 14 mins
```

Q# 4: [8 Marks]

Five jobs are waiting to be run. Their expected running times are 9, 6, 3, 5, and X. In what order should they be run to minimize average response time? State the scheduling algorithm that should be used AND the order in which the jobs should be run. (Your answer will depend on X).

Solution:

```
Shortest job first is the way to minimize average response time (we also accepted
Shorted Remaining Time to Completion First).
0 < X <= 3: X, 3, 5, 6, 9.
3 < X <= 5: 3, X, 5, 6, 9.
5 < X <= 6: 3, 5, X, 6, 9.
6 < X <= 9: 3, 5, 6, X, 9.
X> 9: X, 3, 5, 6, 9, X.
```

.

Q# 5: [3+3+4 Marks]

A computer system has a 36-bit virtual address space with a page size of 8K, and 4 bytes per page table entry.
a) How many pages are in virtual address space?
b) What is the maximum size of addressable physical memory in this system?
c) If the average process size is 8GB, would you use a one level, two level or three level page table? Why?

Solution:

a. A 36 bit address can address 2^36 bytes in a byte addressable machine. Since the size of a page 8K bytes (2^13), the number of addressable pages is 2^36 / >2^13 = 2^23
b. With 4 byte entries in the page table we can reference 2^32 pages. Since each page is 2^13 B long, the maximum addressable physical memory size is 2^32 * 2^13 = 2^45 B (assuming no protection bits are used).
c. 8 GB = 2^33 B We need to analyze memory and time requirements of paging schemes in order to make a decision. Average process size is considered in the calculations below.

   1 Level Paging

   Since we have 2^23 pages in each virtual address space, and we use 4 bytes per page table entry, the size of the page table will be 2^23 * 2^2 = 2^25. This is 1/256 of the process' own memory space, so it is quite costly. (32 MB)

   2 Level Paging

   The address would be divided up as 12 | 11 | 13 since we want page table pages to fit into one page and we also want to divide the bits roughly equally.

   Since the process' size is 8GB = 2^33 B, I assume what this means is that the total size of all the distinct pages that the process accesses is 2^33 B.

Hence, this process accesses 2^33 / 2^13 = 2^20 pages. The bottom level of the page table then holds 2^20 references. We know the size of each bottom level chunk of the page table is 2^11 entries. So we need 2^20 / 2^11 = 2^9 of those bottom level chunks.

The total size of the page table is then:

//size of the outer page table		//total size of the inner pages	
1 * 2^12 * 4	+	2^9 * 2^11 * 4	= 2^20 * (2^-6 + 4) ~4MB

3 Level Paging

For 3 level paging we can divide up the address as follows:
8 | 8 | 7 | 13

Again using the same reasoning as above we need 2^20/2^7 = 2^13 level 3 page table chunks. Each level 2 page table chunk references 2^8 level 3 page table chunks. So we need 2^13/2^8 = 2^5 level-2 tables. And, of course, one level-1 table.

The total size of the page table is then:

//size of the outer page table	//total size of the level 2 tables	//total size of innermost tables	
1 * 2^8 * 4	2^5 * 2^8 *4	2^13 * 2^7 * 4	~4MB

As easily seen, 2-level and 3-level paging require much less space then level 1 paging scheme. And since our address space is not large enough, 3-level paging does not perform any better than 2 level paging. Due to the cost of memory accesses, choosing a 2 level paging scheme for this process is much more logical.

Q# 6: [10 Marks]

Suppose we are building printed circuit boards (PCBs) in a factory. This involves two steps: the first is to mount electrical components on a board and the second is to solder them. There are a number of Mount and Solder processes in our system. These processes can arrive in any order and their execution may be arbitrarily interleaved. The role of the Mount process is to carry out the mounting on a single board and it is followed by a Solder process. The Mount and Solder processes that work on the same board should only exit after the soldering of that board is complete. You are required to give the pseudo code for the Mount and Solder processes using semaphores. Make sure that you clearly represent the initialization values of your semaphores.

***Variable initialization*:**

```
Semaphore mount = 0;

Semaphore board_complete = 0;
```

Mount Process

```
    // start mounting

    signal(mount);

    wait(board_complete);
```

Solder Process

```
    wait(mount);

    //do soldering

    signal(board_complete);
```