**Lab #10: Operating Systems**

Due: Tuesday, April 22, 2014, beginning of class

## Instructions

* You will need your WFU-issued ThinkPad for this lab.
* Submit your answers to the assignment in Sakai using this document and any other material you are asked to submit in the exercises.

## Part 0: Background

The purpose of this lab is to reinforce what you have learned about operating systems (memory management, process management, and other OS activities). Background comes from Chapter 10, from section 11.3, Disk Scheduling, in Chapter 11 (Computer Science Illuminated), and from lectures.

* Create a folder called ***Lab 10*** on your desktop.
* From the assignment on Sakai, right-click and *save link as* the ***OS\_Lab\_Applet\_Doc*** into your Lab 10 folder
* From the assignment on Sakai, right-click and *save link as* the ***Applets\_for\_OS\_Lab.zip***  into your Lab 10 folder. Unzip this file. You should get 3 folders.
* Download the lab manual and report.

## Part 1: Memory and Process (Job) Management

* Read the applet documentation then complete the exercises below.
* Note each exercise calls for you to start a particular applet. To do this, go into the appropriate applets sub-folder in your Lab 10 folder. In that sub-folder, double click on the ***applet\_frame.htm*** file. Once the applet frame starts, depending on your security settings, you may need to explicitly allow the applet to run. Screen captures at the end of this document illustrate how to do this. Note that if the applet does not work with Internet Explorer, right click on applet\_frame.htm and select ‘open with’ and select Firefox.

**Exercise 1: Memory Management**

* 1. Start the Memory\_Management applet and click on the *Example* button. Delete jobs 2 and 5. How many holes are there?

**3**

What is the total amount of free memory?

**275**

What is the largest job that could be entered without compacting the memory?

**100**

* 1. Try to add a job that is 125 Kb long. What happens?

**Says can’t find a slot.**

* 1. Compact the memory (click on the *Compact* button). Try to add a job that is 125 Kb long. What happens?

**Job 9 is added into combined empty space in memory that resulted after compaction.**

* 1. Try to add a job that is 125 Kb long. What happens?

**Job 10 is added to memory below job 9**.

**1.6)** Re-start the applet and click on the *Example* button again. (You can re-start the applet by closing

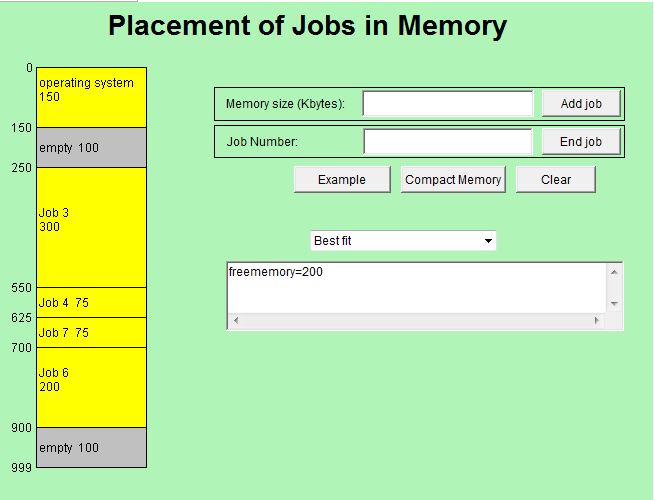
and reopening your browser.) Delete jobs 2 and 5 again.

* 1. Add a job that requires 75 Kb. Where do you expect it to go?

**Expect it to go into first empty slot with room.**

* 1. Redo Step **1.6.** elect the *Best fit* algorithm from the pull-down menu. Add a 75 Kb job. Where will it go and why? Take a screenshot.

**In empty 75 slot since it fits the new job best.**



* 1. Since the *Compact memory* button is there, why doesn’t the operating system always use it when a new job enters? Make a guess as to why frequent compaction is avoided.

**Compaction takes time. Doing it too frequently would slow down the running of processes.**

**Exercise 2: Process (Job) Scheduling**

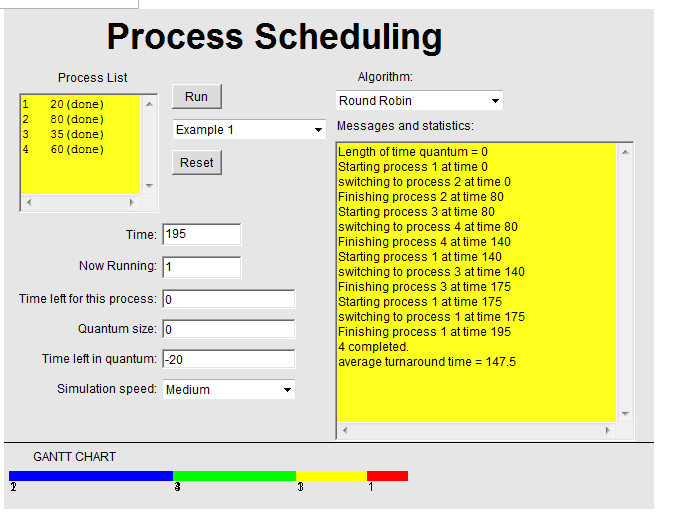
**2.1)** Start the “Process\_scheduling” applet and click on the *Example 1* button. Four jobs appear in the process list. Run the program using the *First-Come, First-Served* algorithm. Write below the total time and average turnaround time.

**Total time = 195, average turnaround time = 112.5.**

**2.2)** Select *Shortest Job first* from the drop-down menu and click on the *Run* button. Does this method have the same total time requirement as the *First-Come, First-Served* algorithm? Write below the total time and average turnaround time. Is it better than *First-Come, First-Served*?

**Total time = 195, average turnaround time = 96.25.**

**2.3)** Select *Round Robin* from the drop-down menu and click on the *Run* button. Write below the total time and average turnaround time. How does it compare to the other two methods? Take a screenshot.

**Total time = 195, average turnaround time = 147.5.** 

**2.4)** Looking at the log of activity, what is fundamentally different about *Round Robin* from the other two job scheduling algorithms? What kind of computing system would likely prefer to use *Round Robin*?

Note to grader: This question is somewhat open ended. So, just look for some kind of reasonable response and a reason. There really isn’t any absolute that I can see.

**Different? I guess the obvious, that with round robin the CPU is shared among the processes. What kind of computing system? One with more than one user?**

**2.5)** Based on your observations of the three algorithms on the example request list, state what kind of request list would make First-Come First-Served (FCFS) have a long average turnaround time?

. In other words, what pattern would the numbers in the request list have to show so that FCFS consumes a lot of time.

**Longest to shortest?**

## Part 2: Disk Scheduling

**Exercise** 3

Start the “Disk Scheduling” applet. Click on the *Example* button, which fills the list with some requests.

3.1) Select *First-Come, First-Served* from the pull-down menu and click the *Run* button. Time the applet with your watch, or the computer’s clock, and write down how many seconds it takes.

**37**

3.2) Select *Shortest Seek Time First* and run the applet again. Measure the time it takes and write it down. Also write down the order of the tracks that are visited. (This is different from the original request list.)

**12, sequence 40 21 19 10 4 3 2 62 86 97**

3.3) Repeat Step 3.2 after selecting *SCAN Disk (elevator)*. Which algorithm took the least time?

**12, sequence 40 21 19 10 4 3 2 62 86 97, first come first served**

3.4) Based on your observations of the three algorithms on the example request list, state what kind of request list would make First-Come First-Served (FCFS) take up a lot of time. In other words, what pattern would the numbers in the request list have to follow so that FCFS consumes a lot of time?

**I would guess where the requests alternate from one end to the other of the disk. So low #, then high, then low, then high**

**Exercise 4**

4.1) Some request lists might cause the disk scheduler to act the same when the three different algorithms are run. Create a request list of five track numbers that will cause all three algorithms to visit the same tracks in the same order. What track numbers did you use?

**40, 30, 20, 10, 5**

4.2) If shortest seek time first starts with the disk head positioned at either 0 or 99, instead of at 50 (in the middle), which algorithm would it resemble: FCFS or SCAN? Why?

Note to grader: I don’t think there is a right answer. Just look to see that they give a reason that makes sense regardless of which one they choose.

**SCAN, because the shortest scan time will be in one direction which is what SCAN does (I’m assuming we are also starting SCAN at one end also?)**

**Exercise 5**

Start the “Disk Scheduling” applet and type the following numbers into the *Requests* text area:

8, 20, 35, 80, 10, 90, 5, 87, 26, 94

These numbers have been chosen so that there are two clusters, one at the lower end of the scale and the other at the upper end.

5.1) Select the *First-Come, First-Served* algorithm and start. When the disk head has reached 35, type 30 into the “Add” area and press RETURN. This will add a request to seek to track 30 to the list. What happens? Does the disk drive respond to this new request or not?

**It does the 30 request last.**

5.2) Stop the applet and remove 30 from the end of the list. Choose “Shortest Seek-time first” and rerun the applet. When it consumes 35, type 30 into the add area and press RETURN. Write down what happens. (Warning! You have to be fast because the applet might move into the upper cluster quickly. In that case, just retry.)

**It services the 30 request after doing 35**

5.3) Stop the applet, remove 30 from the end of the list and choose “SCAN.” Run it and when it consumes 35, type 30 again. Write down what happens. Again you must be fast!

**Same as with SSTF, it does 35 then 30**

5.4) Which algorithm is least responsive to new requests?

**Duh, FCFS**

5.5) Stop the applet, remove 30 from the end of the list, and choose “Shortest seek-time first” and start it. Now try to “trap” the disk head into the lower cluster by typing in disk tracks that are in the lower half of the disk drive, pressing return after each one. You have to be quick, and you may have to try it several times. Were you successful?

**I couldn’t get this to work. I typed in a track very close to the ‘head’ and it kept going down**

5.6) Now choose “SCAN” instead. Once again, try to “trap” the disk head into the lower cluster by typing in disk tracks that are in the lower half of the disk drive, pressing return after each one. Were you successful?

**Not going to play with it. SCAN shouldn’t get caught.**

5.7) In real life, disk drives may see a cluster of track requests that could trap it in one section of the disk drive. What implications does this have for programs that requested tracks outside the busy area? (Hint: Computer Scientists have a gruesome term for this phenomenon. It is called starvation. Why do you think they chose this term?)

**I assume that this conversation is with respect to SSTF as both SCAN and FCFS shouldn’t get trapped.**

## Part 3: CPU Scheduling During Process I/O and Cache Memory

**Exercise 6: CPU scheduling versus disk access.**

Suppose Process A needs to read a single sector from a disk. You are given the following information: The disk rotates at 7,200 rpm (rpm is revolutions per minute). Average seek time for the disk is 12 milliseconds (ms). The average latency is one-half the time for a full rotation of the disk. A sector contains 1024 bytes (1024 bytes is 1 kilobyte), and there are 16 sectors per track. Transfer rate from the disk is 1024 megabits per second (Mbits/s). Transfer rate is the rate that the disk copies data from the disk into memory.

6.1) How long does it take to get to the right track, find the sector, and transfer the sector? That is, you must add the average seek time, the average rotational latency, and compute the transfer time.

Seek time: 12 ms (given)

Latency: one-half a revolution. At 7,200 rpm gives 0.120 revolutions per millisecond or 1/0.120 = 8.333 ms per revolution. Therefore, ½ revolution is 4.16 ms, which is the latency.

Transfer: Transferring one sector, which is 1/16 of a track. Therefore, transfer time is 8.333/16 or 0.52 ms

Total disk time: 12 + 4.16 + 0.52 = 16.68 ms

There is an alternative. The highlighted information in the question should not be there. However, if the students use this information, the will get a different transfer time. 1024 Mbits/s gives 128 Mbytes/s. Therefore, to transfer 1024 bytes only takes 0.007 ms. So the answer using this information is:

Total disk time: 12 + 4.16 + 0.007 = 16.167 ms

6.2) If a CPU executes 100 million instructions per second. How many instructions could be executed by Process B (which would be in the ready state) while Process A is waiting for its disk data (Process A would move to the wait state)?

100 million instructions per second is 100,000 instructions per ms. Therefore, Process B could execute

(100,000 instr/ms) (16.68 ms) = 1,668,000 instructions

**Exercise 7: Cache memory.**

7.1) Cache memory. Read the Cache Memory document in Sakai -> Resources. Work Practice Problems 3 and 4 in the document. Show all work and explain as needed. Pay attention to the Principle of Locality.

3) (0.80 x 10) + (0.20 x (10 + 25)) = 15 ns (same as in the reading)

4) Let R be the cache hit rate. Substitute R in the equation above and solve as follows

(R x 10) + ((1 – R) x (10 + 25)) = 12 ns

Solving for R gives: R = 0.92