Laboratory 10

Operating Systems

Objective

- Watch how the operating system places jobs in memory and schedules jobs.

References

Software needed:

1) A web browser (Internet Explorer or Netscape)
2) Applets from the CD-ROM:
   a) Placement of jobs in memory
   b) Scheduling of jobs

Textbook reference: Chapter 10, pp. 331–337, 347–352

Background

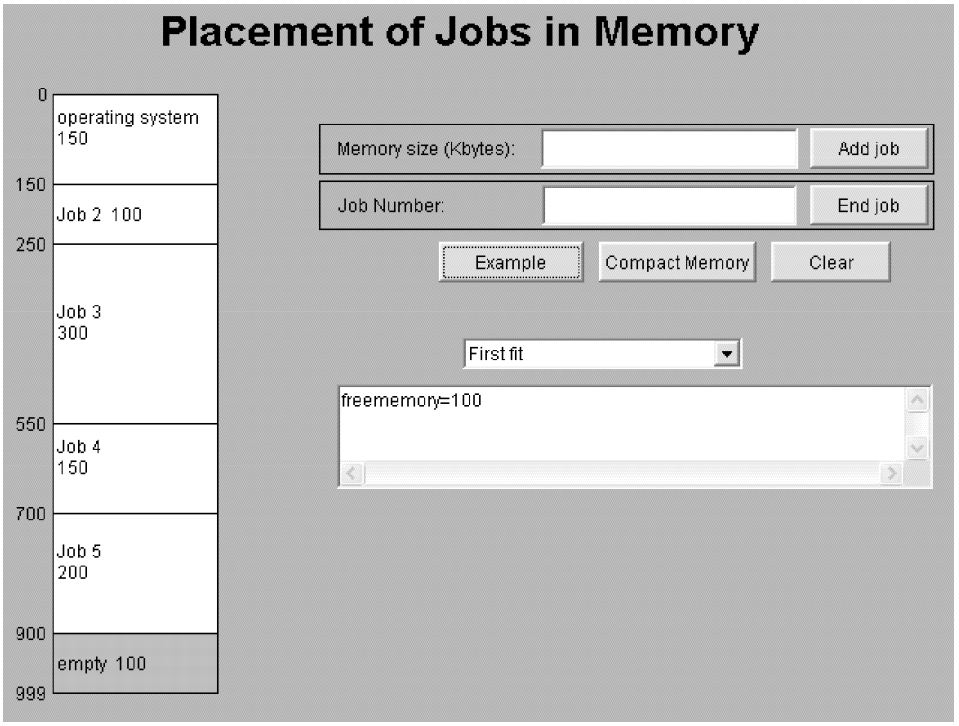
Everything you need to learn is explained in Chapter 10, "Operating Systems."

Activity

Part 1

Operating systems do an amazing variety of tasks, far too many for us to study in detail. The textbook explains several of these in detail, and this lab will show you how to study these and draw your own conclusions about which methods are best.

The first simulation shows how *jobs* that are (programs or sequences of programs submitted by a user) placed in the main memory of a computer by the operating system using single contiguous memory management (textbook p. 339). Start the "Placement of jobs in memory" applet. There is an *Example* button, which places several jobs in memory.



Notice that the operating system always occupies the first 150 kb (kilobytes) of memory, from address 0 up to 149 inclusive.

You can add a new job by typing in a number of kilobytes in the text area near the top and clicking the *Add job* button. Try adding a 200 Kb job. What happens and why?

You can delete a job by typing its job number in the text area next to the *End job* button. Delete Job 2. Notice that the area turns gray to symbolize that this is unused memory. (Of course, the memory itself doesn't change in any way—the power doesn't go off; the bits don't freeze. Jobs are merely an idea in the "mind" of the operating system.)

The power of the simulation comes when several jobs are deleted, leaving "holes" (gray areas) between the jobs. When a new job arrives, it must be placed in memory somewhere. The operating system uses one of three algorithms to find a hole big enough:

1) First fit—the simplest, finds the first hole
2) Best fit—finds the smallest hole
3) Worst fit—finds the largest hole

Sometimes, the total amount of space in all the holes may be enough for the new job, but there is no one hole big enough. In this case, the *Compact Memory* button saves the day—it moves all the jobs to one end and squishes all the free space into one big hole.

Sadly, sometimes there is just not enough memory even after compacting. The operating system then informs the unhappy user to either quit some running programs or give up! Up until recently, home PCs didn't have much memory and messages like this were frequently seen.

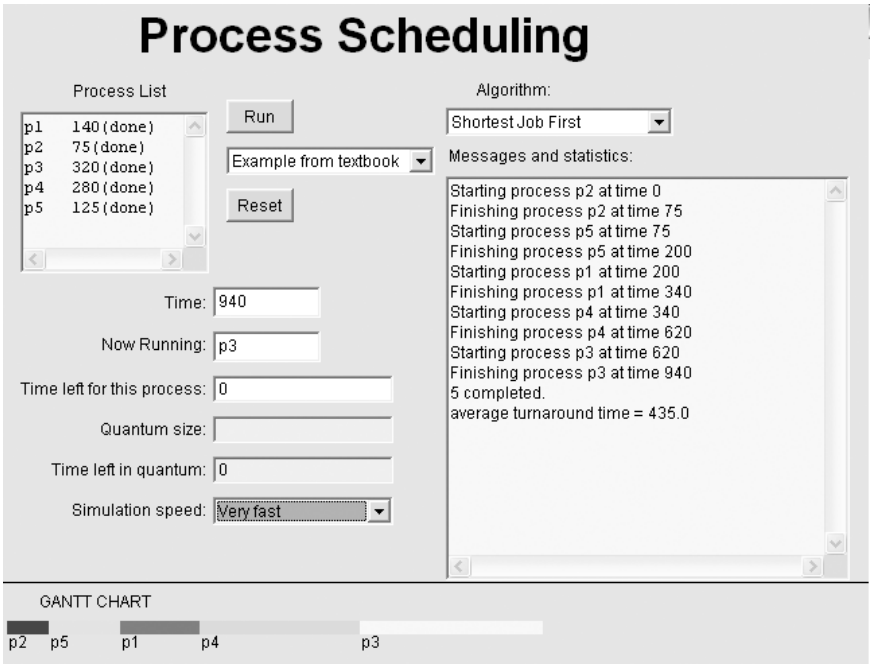
This applet allows you to experiment with the three fitting algorithms. Their benefits and drawbacks are explained in the textbook on pp. 341–343.

Part 2

The "Job Scheduling" applet simulates the decisions that an operating system makes when it determines which process gets to use the CPU next. Three algorithms that are discussed in your textbook on pp. 348-350 (First-Come, First Served; Shortest Job Next; Round Robin) are implemented in the applet.

Pull down on the Examples choice menu and select "Example from textbook." A list of jobs is placed in the process list. The first column is the process identifier and the second column of numbers is the list of corresponding service times. These are the time requirements for the process to finish its work. See p. 348 of your textbook.

Select an algorithm and click on Run. If you are impatient, pull down the choice next to "Simulation Speed" and select a faster rate. A log of messages is shown in the big text area on the right, and a Gantt chart is drawn across the bottom of the applet.



The applet calculates the turnaround time of a process, as described on p. 348 of your textbook. Basically, turnaround time is the service time required by the process plus all the time it spent waiting to obtain the CPU. If there are two processes with service times of 20 and 80, the turnaround time of the first will be 20 for FCFS and SJN, while the turnaround time for the second one will be 100, which is 20 + 80. Thus, the average turnaround time is 120 ÷ 2 = 60.

There are two short example sets of processes and a longer one that appears in the textbook on p. 348. Try all three algorithms on the book example. Notice that when using Round Robin, you can set the quantum size. Its initial value is 50, so that the example on p. 351 will match the applet's result.

Exercise 1

Name _____ Date _____

Section _____

1) Start the "Placement of jobs in memory" applet and click on the *Example* button. Delete jobs 2 and 5. How many holes are there? What is the total amount of free memory? What is the largest job that could be entered without compacting the memory?

2) Try to add a job that is 125 Kb long. What happens?

3) Compact the memory (click on the *Compact* button) and answer Questions 1 and 2 again.

4) 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.

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

6) Redo Step 4. Select the *Best fit* algorithm from the pull-down menu. Add a 75 Kb job. Where will it go and why? Take a screenshot.

7) 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 show so that FCFS consumes a lot of time.

8) 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.

Exercise 2

Name _____ Date _____

Section _____

1) Start the "Job Scheduling" applet and click on the *Example* button. Four jobs appear in the job list.
2) Run the program using the *First-Come, First-Served* algorithm. You should get the same answers as shown in the Activity section.
3) 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? Note the average time to completion. Is it better than *First-Come, First-Served*? Take a screenshot.

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

5) 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*?

Deliverables

Turn in your hand-written answers. You should also turn in one screenshot from the "Placement of jobs in memory" applet and two screenshots from the "Job Scheduling" applet.

Deeper Investigation

Suppose you have just invented a new job scheduling algorithm and wish to compare its performance to the old standard algorithms. The "Scheduling of jobs" applet can be used as a starting point, but it hardly gives a realistic or easy way to simulate a stream of jobs. Discuss what new features you would put into the applet to enable it to help you in your research. What new buttons and what new functions would you include?