Operating Systems Project – Part 3 – Process Scheduling

In this assignment, you will make your OS Simulator handle the Process Scheduling methods discussed in class. For simplicity, we will not worry about "User Input" while our Processes are running. (*You are welcome*)

Step 1: Remove the following User Commands from your program:

CreatePCB

DeletePCB

Block

Unblock

If you do not wish to delete them, you may comment them out, but they should not be present in your help menu and they should not be valid commands for the user.

Step 2: Create functionality to read process information from a file

Your schedulers will get their process information from a file. You should create a functionality that prompts the user for a filename and reads process information from the file. The scheduling methods (described below) will determine how you are to handle the information gained from the file.

Note: I put this step here so that if you need to remember how to read from files, you can review and test it before continuing. You should read the scheduling methods steps before creating any production C++ functions to read from the files.

The file will consist of several lines of process information. Your program should read until the end of the file in the following format:

Process Name Class Priority Memory Time Remaining Time of Arrival Percentage of CPU

Process Name is a String, Class is a Character, all other fields are integers.

Note: Each element is separated by a space. Notice that I do not ask for the Process State. All Processes are assumed to be entering the Ready Queue when they are scheduled. You can ignore the Percentage of CPU field until you get to the Lottery scheduler.

Process classes will be represented as follows:

A – Application

S – System

Example Process line:

TestProcess A 24 128 500 15 25

Note: You can assume that the file will be sorted by Time of Arrival, from earliest to latest. This attribute will be used in several of your scheduling methods. Assume that your schedulers are starting at time 0. (You're welcome once again. I almost made you write Quicksort to do it yourself)

You will be responsible for creating seven scheduling methods. These are in two categories: Full Knowledge Schedulers and Incomplete Knowledge Schedulers. Full Knowledge Schedulers can read the entire file beforehand and can ignore the time of arrival attribute.

Note: In real-world scheduling, this is extremely rare. I chose to do this to make it easier for you to gain understanding of the project. That's why you will write two versions of Shortest-Job First. It is possible to write FIFO as a full-knowledge scheduler, but it is better to do it as an Incomplete Knowledge Scheduler.

Incomplete Knowledge Schedulers will need to read each line of the file and deal with that Process' information before moving on to the next Process. They also need to take the time of arrival into consideration.

Step 3: Create and Implement Full Knowledge Shortest Job First

You should create a new command: **SJF** that prompts the user for a filename (***as usual with your OS, you should assume the user is an idiot and perform some error checking to see if the file is open***) and uses the information in the file to create a Shortest Job First scheduler. Your scheduler should create PCBs for each Process in the file and insert them into the Ready queue in the proper order. Proper order is determined using the Time Remaining field.

Once all processes are in the Ready Queue, you should show the Ready Queue in an easily readable format (***if you did a good job with your ShowReady command, this should work***). You should, at minimum show the name of the Process and its time remaining. This is used to demonstrate that your scheduler works properly.

After that, your scheduler should Dispatch the processes from Ready to Running in the proper order. At the end of the command, the Ready Queue should be empty and there should be nothing running (***If you constructed your program to have an always on process like "System" or "OS", then it is okay for it to remain***) You should then print out a list of the Process Names in the order that they were completed, the Total Time to Completion (***for the whole Ready Queue, not each job individually***) and the Average Turnaround time. (***See the textbook for how to calculate this if you're not sure***)

Step 4: Implement the Incomplete Knowledge Schedulers

The biggest difference here is that you should implement the schedulers as if you don't know the full schedule before you start. (***At least in theory, in reality you'll have an input file that lists all jobs.)*** This is mainly done with the Time of Arrival field. Also, some scheduling methods are Pre-Emptive or only allow the Process to run for a specified amount of time.

For this section of the assignment, you will implement the following schedulers. The command names you should use are listed in bold.

First In First Out - **FIFO**

Shortest Time to Completion First (Pre-Emptive Shortest Job First) - **STCF**

Fixed Priority Pre-Emptive Scheduling – **FPPS (This is similar to STCF but you use Priorities instead of Time Remaining for your pre-emption condition)**

Round-Robin Scheduling – **RR (You should prompt the user for the time quantum after you prompt for the filename)**

Multilevel Feedback Queue – **MLFQ (Hint: Use the Priority field in your object instead of making multiple queues. You should prompt the user for the number of queues in the system, the time allotment in each queue and how many cycles should pass until all processes are moved to the top queue. See the rules in Section 8.6)**

Lottery Scheduling – **LS (Prompt the user for the total number of tickets in the system, use the percentage of CPU field to decide on the ticket distribution)**

For each scheduler, you should prompt the user for a filename as you did in the SFJ portion of the assignment. You should also output the same information as you did for the SJF scheduler. However, since these schedulers are more complicated and involve you moving processes from the Ready Queue to the Running state and then perhaps to the Blocked Queue, we need to see more information. For every action that you perform in each scheduler, you should write it to an output file named after the scheduler. You should also be able to demonstrate that your processes are being correctly moved between queues and following the State Transition rules. It is your choice on how you would like to do this, but you should indicate the following things:

When a process changes state

When a process enters the system

When a process completes

It may also be helpful to show the contents of each Queue as well as what is currently running, but you should be careful to do this in a readable format (you don't need to list full contents of each PCB for instance).

Once you can demonstrate that each of your schedulers is working correctly, you are finished with this assignment.