Operating Systems Lab (CS 470):

Lab 5: Given a binary file containing PCB (Process Control Block) information about multiple processes –the processes being written in sequential order (one after another), simulate a CPU scheduling/execution using different scheduling algorithms and different processors on an imaginary computer. Each processor is running completely independent governed by a certain scheduling algorithm (Priority Scheduling = 1, Shortest Job First = 2, Round Robin = 3, FCFS = 4). Make sure to provide load balancing when such operation is necessary.

Overview

PCB is a data structure holding information about processes, such as id, name, activity status, CPU burst time, priority, files open per process, etc. This data descriptor is necessary for the CPU to know which process is running, what is the current state of the process, where the program is in its execution, etc.

Instructions

The input binary file contains multiple PCB descriptors in sequential order. In our case it will be not a real PCB descriptor (see sched.h for such a descriptor), but each process has the following fields:

Offset	**Type**	**Value**	**Description**
0000	1 byte	??	priority
0001	24 bytes	??	process name
0025	4 bytes	??	process_id
0029	1 byte	??	activity status
0030	4 bytes	??	CPU burst time
0034	4 bytes	??	base register
0038	8 bytes	??	limit register
0046	4 byte	??	number of files

Notes

- Read the binary file and print out the number of processes available in the file, the total number of memory allocated by the processes and the overall number of open files – considering all the processes. The binary file with the PCBs will be provided as command line argument with the structure described above. [The test file will be same in structure but not exactly same in size and content as the example provided with the assignment.

Make sure to create a generic solution to read all files following the structure mentioned earlier.]

- The different processors and their governing scheduling algorithm will be provided as command line arguments (variable number), along with their initial load.
  Example: ***./scheduling PCB.bin 1 0.4 2 0.5 4 0.1***
  meaning that there will be 3 processors running the simulation. The first one running Priority Scheduling (see **1**) and 40% (see **0.4**) of the original processes from the process file (see **PCB.bin**) should be assigned to this processor. The second one will run Shortest Job First (see **2**) and 50% (see **0.5**) of the *PCB.bin* file will be assigned to this processor. Finally, the third processor will run based on First Come First Served (see **4**) and the remaining 10% (see **0.1**) of the processes from the *PCB.bin* (also provided as command line argument) will run on this processors. It is recommended (not enforced though) to populate originally the ready queues of each processor in a sequential order from the original file, meaning that the first 40% of the PCB records can go to the ready queue of processor 1, the next 50% of the PCB records can be assigned to the second processor's ready queue, etc. Random assignment is also allowed, but the data percentage should be strictly considered. Each processor should run in a separate thread. The number of processors and their corresponding load is not limited. It can be one processor, it can be two processors and in general it can be *n* processors. However, their cumulative load should be always 100%. Everything depends only on how you call the program from the command line.
- Each process is "executed" when it is its "turn" (considering the given scheduling algorithm) by decrementing the CPU burst with a certain value. Let that quantum be 2. To mimic the "execution" introduce a sleep() for a certain number of milliseconds. Ex. 200 or 400 milliseconds. (Make sure that you are not waiting forever to finish the simulation!) You can store the processes in a ready queue (each processor will have its own).
- For the priority scheduling (if any involved) an aging mechanism should kick in at each time quantum of 10 seconds. In that case the priority of each process from the priority based scheduling queue should be increased by one. Do not forget the priority is inversely proportional with the priority number!
- When one ready queue is empty (all the processes originally have been executed from that particular processor using the selected scheduling algorithm) the system should perform a load balancing (moving processes from the other ready queues into the empty one in such a way to balance out (have equal or close to equal number of processes) the system.
- For the load balancing you can select the processor which has the most processes yet to run and do a balancing between that processor and the one which has no processes to run (see Line 4 in the Rubric). You can also distribute the load from all existing processors

which is an optimal solution. Once the load balancing has happened, each CPU is running its own scheduling algorithm as initially provided in the command line. If a processor is not involved in the load balancing that processor will continue executing its processes without stopping.

- Each step in the simulation should be documented (printed) on the screen to see which processor is running which process, using what algorithm, what CPU burst has left, what is their current priority for those where priority does matter, when the load balancing is happening, which ready queues are involved in the load balancing, how many processes were transferred, when a processor completed the processes allocated to it, etc.
- The simulation stops when all the processes from the original file have been "executed".

Rubric

Task	**Points**
Error handling	2
Printing results	2
"Executing" the processes	5
Load balancing (only two CPUs involved – the new load will come from only one processor for the empty processor)	1 **or**
Load balancing (all the CPUs involved – the load will come from all existing processors for the empty processor)	**1+3 (3 extra points added to the final grade)**