Operating Systems Lab (CS 470):

Lab 5: Given a binary file containing PCB (Process Control Block) information about multiple processes –written in sequential order, simulate a CPU scheduling using different scheduling algorithms and different processors. Each processor is running completely independent using a certain scheduling algorithm (Priority Scheduling = 1, Shortest Job First = 2, Round Robin = 3). 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, 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 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	16 char	??	process name
0016	32 bit int	??	process_id
0020	1 char	??	activity status
0021	32 bit int	??	CPU burst time
0025	32 bit int	??	base register
0029	64 bit long	??	limit register
0037	1 char	??	priority

Notes

- Read the binary file and print out the number of processes available in the file, and the total number of memory allocated by the processes – considering all the processes. The binary file will be provided as command line argument with the structure described above.
- The different processors and their governing scheduling algorithm will be provided in the command line argument, along with their initial load.
  Example: ***./scheduling 1 0.4 2 0.5 1 0.1 PCB.bin***
  Which means that there will be 3 processors running the simulation. The first one running Priority Scheduling and 40% of the original processes from the process file are assigned to this processor, the second one will run Shortest Job First and 50% of the *PCB.bin* file will be assigned to this processor and finally the third processor will run based on Priority

Scheduling and the remaining 10% of the processes from the *PCB.bin* (also provided as command line argument) will run on this processors. It is recommended to populate the ready queues of each processor in a sequential order. Each processor should run in a separate thread. The number of processors and their corresponding load is not limited. It can be one, it can be two and in general it can be *n*. However, their cumulative load should be always 100%.

- 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 3. To mimic the “execution” introduce a sleep() for a certain number of milliseconds. Ex. 300 or 500 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), however, when a process is executed completely (the CPU burst ==0) that record should be updated in the file as its CPU burst will be 0 and the activity status (see activity status in the binary file) will be changed. Make sure to avoid incidences with other processes, as each processor is running in parallel in a separate thread and they can access the file in the same time.
- For the priority scheduling an aging mechanism should kick in at each time quantum of 20 seconds. In that case the priority of each process 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) 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 processes in the system. If it is easier, you can select that 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. Once the load balancing has happened, each CPU is running its own scheduling algorithm.
- Each step in the simulation should be documented (printed) on the screen to see which processor is running which process, using what algorithm, parameters, etc.
- The simulation stops when all the processes from the original file have been “executed”.

Rubric

Task	**Points**
Error handling	2
Printing results	1
“Executing” the processes	6

Load balancing (only two CPUs involved)	2 or
Load balancing (all the CPUs involved)	2 + 1