CS2506 Operating Systems II Lab 1 1/02/2022
Processes and Scheduling

Laboratory Goals / Objectives

Students will investigate the multilevel feedback queues scheduling of processes and the power saving strategies by designing the algorithm in pseudo-code and implementing in Python/Java. By the end of the lab, an understanding of the algorithm should be demonstrated by providing a working simulation of the algorithm implementation and screenshots with results of the execution.

How the scheduling algorithm works

Multilevel feedback queues are an architectural solution for the concept of dynamic priorities of processes. Each queue has a priority level and a given size of the CPU time slice. If user processes don't complete the execution while controlling the CPU for the time slice allocated to them, they will be returned in a queue with a lower priority and a bigger time slice. If they reach the lowest allowed priority queue, they will be scheduled there round-robin until completion. However, if a user process is blocked for the execution of I/O operations, when it will be returned to the ready state, its priority will be increased; therefore, it will go into a different queue of higher priority. The occurrence of an interrupt will suspend the process for the time the interrupt is handled after which the process resumes execution.
The idle process waits in the lowest priority queue and if there is no process of higher priority ready, it will be given the control of the CPU. Its execution will correspond to the energy saving policy.

The kernel power manager strategy for the CPU

The kernel power manager works in two stages. During the first stage which corresponds to the CPU running processes, the configuration, {voltage, frequency}, is adapted to the load. If the load is high, the standard configuration in terms of voltage and frequency values is used. If the load is decreasing below a threshold, the configuration is changed to a lower one (lower frequency and voltage).
When there are no processes in the queues, the CPU is running the idle process that will switch the system to a sleep state. If the idle state extends in time, the CPU will switch into a deeper sleep state. The process continues down to the deepest sleep state.
When there are new processes arriving in the CPU queues, the CPU is switched back to the running state.

Lab Work

In this lab we will explore the multilevel feedback queues scheduling and power saving strategies. You may consider a set of assumptions like the following:

- The time slice of any queue is a multiple of the quantum (the time slice of the highest priority queue), according to the queue level – see lecture notes.
- A process that is created and is ready to run will be inserted on its arrival time (when its state becomes "ready") in a queue that corresponds to its priority.
- Each process needs a determined number of CPU time quanta to complete (a positive integer, 1, 2, 3,...).
- We'll also consider that some processes will execute I/O operations. When such an operation is started, the process is pre-empted and switched to the blocked state. The next ready process with highest priority takes control of the CPU. The event of I/O completion triggers the switch of the process from the blocked state to the ready state – the process is inserted into a queue of a higher priority than the one the process had when blocked.
- If the process completes its execution during a time slice, it will be terminated (exits the system), and the next process of the highest priority takes control of the CPU.
- When the system load (measured in terms of number of processes) decreases below a certain value (e.g., two processes on the lowest priority queue), the {frequency, voltage} configuration is switched from the standard one to a lower one.
- If there is no other process ready, the idle process will take control of the CPU.

Tasks to do

Task 1 First draw 8 multilevel feedback queues for user processes and allocate priorities to them starting with priority 0 – 0 is the highest priority.

- If q is the quantum for the queue with priority 0, give q a time value and determine the time slice for each user process queue.
- When there is less load – processes only in the lowest priority queue, the CPU frequency is lowered (new configuration) and the time slices change as well. Therefore, processes in the lowest priority queue will benefit of a longer execution time slice. As an example, you may consider the clock frequency switch from 1 GHz to 800 MHz and show the new time slice value.
- Explain what happens when all user processes terminate and there is no other user process ready to execute.

Task 2 Outline the multilevel feedback queues scheduling and power saving algorithm in pseudo-code. This will be your design for the algorithm. It should consider all aspects of a process lifecycle.

- Your design of the algorithm must take into account the possibility of the process having to block for I/O, in which case the process will forego the rest of its time slice.
- Define the rule by which the priority of a blocked process is increased when I/O is completed and the process becomes ready again.
- Describe the work (algorithm) carried out by the idle process.

Task 3 Provide an implementation of your algorithm in Python/Java. You should design this with good object oriented design in mind, encapsulating the different parts of your solution into appropriate classes. For example, you may have one class to represent a process and its attributes such as the execution duration, the presence of an I/O operation and its duration. You may use other classes to represent the ready and blocked queues and their properties. You may

also want to have a class that represents the scheduler, which may contain a method(s) implementing your scheduling algorithm.
Place your name and student number at the top of each class file.
Have your code commented. Place comments at the method level; use one-line comments inside method bodies to describe more complex statements if you feel they are not obvious.

Task 4 Test your implementation by simulating the execution of your algorithm: create a class which is running your scheduling algorithm. Provide some processes to run with varying properties. For example, you may consider six processes with different execution times (expressed in quanta), and two of them having I/O operations. Consider the idle process present. By executing this simulation, the output should be some text tracing the path your processes have taken through the algorithm until completion, i.e. the different time-slices they may have, the different states they switch into, etc. **Capture screenshots of the execution and include them in your report**.

Submission

Return your results to each of Tasks 1 - 4 above in a separate section, including the pseudo-code and the Python/Java code in a report (pdf file), by the deadline – 22/02.

Tasks 1 and 2 have 2 marks allocated each; tasks 3 and 4 are worth 3 marks each. The total for this lab is 10 marks.

For the completion of this lab, you will have three weeks:
- Task 1 and 2 should be completed by the end of the first week – 8/02.
- Task 3 should be completed by the end of the second week – 15/02.
- Task 4 should be completed by the end of the third week – 22/02. The full report documenting all tasks should be submitted by Canvas, by the deadline.