CS2506 Operating Systems II Lab 2 14, 16/02/2017
Processes and Scheduling

Laboratory Goals / Objectives

Students will investigate the multilevel feedback queues scheduling of processes by designing and implementing the algorithm in pseudo-code and Python respectively. 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 execution.

How the scheduling algorithm works

Multilevel feedback queues are an architectural solution for implementing the concept of dynamic priorities of processes. Each queue has a priority level and a given size of the 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 but 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. 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 algorithm will correspond to the energy saving policy.

Lab Work

In this lab we will explore the multilevel feedback queues scheduling. You may consider assumptions like the following:

- Each process that is created and is ready to run will be inserted on their arrival time (when their state becomes "ready") in a queue, according to its priority.
- Each process needs a determined number of CPU time quanta to complete (an integer, 1, 2, 3,...). Each 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.
- 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, and 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 in 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.

- If there is no process of higher priority ready, the idle process will be given the 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 4 (two queues, not considered here, with priorities 0 and 2 are used by kernel processes – 0 is the highest priority). If q is the quantum for the queue with priority 0, determine the time slices for each user process queue. 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 algorithm in pseudo-code. This will be your design for the algorithm. It should consider all aspects of processes and scheduling. Your design of the algorithm must also take into account the possibility of having to block for I/O, in which case the process will forego the rest of its time slice. Define the equation by which the priority of a blocked process is increased when I/O is completed and the process becomes ready again. Define the work (algorithm) of the idle process.

Task 3 Provide an implementation of your algorithm in Python. 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 duration expressed in terms of time slices, the presence of an I/O operation and its duration. You may use other classes to represent the ready and blocked data structures and their properties. These classes may also contain data structures to hold the processes in the respective states. You may also want to have a class that represents the scheduler, which may contain a method(s) implementing your scheduling algorithm.

Task 4 Test your implementation by simulating the execution of your algorithm: create a class which is capable of 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 move into, etc.

Submission

Return your results to Tasks 1-4 above, including the pseudo-code, the Python code and screenshots, in a pdf file on moodle, by the deadline.

Have comments in your code. Place your name and student number at the top of each class file. 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.

Tasks 1 and 2 have 1 mark allocated each; task 3 and 4 is worth 3 marks. The total for this lab is 5 marks.