***CSE3221 OPERATING SYSTEMS***

***CPU SCHEDULING ALGORITHMS***

***FIRST COME FIRST SERVE (FCFS), ROUND ROBIN (RR) AND MULTI-LEVEL FEEDBACK QUEUE (FBQ)***

This report presents the design and implementation of various scheduling algorithms in some operating systems. First come first serve is the most basic and simplest scheduling algorithms found in some operating. Round Robin is a scheduling algorithm very suitable for real time sharing systems and multi-level feedback queue is a very good algorithm for interactive systems.

Firstly, first come first serve (fcfs) is a non-preemptive scheduling algorithm meaning an operating system never initiates from a running process to another context switch. When a process is given to the CPU it finishes its job and terminates. Termination also happens when there is input output request but during that time the CPU sits idle and no other process is running. Processes that receive system resources during its execution are never shared and have to finishes before handing over the resources back to the operating system. It does not use any priorities and the set of ready virtual processors *(NUMBER\_OF\_PROCESSORS) is implemented as one queue.*

Implementation of the fcfs algorithm was done in C programming language which is more suitable for systems and provides an efficient mapping to machine instructions. Fcfs It is also very efficient for batch systems.

For this implementation of first come first serve (fcfs) , a process that request the Central Processing Unit(CPU) first is allocated to it first. How the program actually measures progress is through a variable step which is part of the process data structure. Each iteration increases the step of a working CPU until the CPU burst length or I/0 burst length is reached. If that condition is satisfied the program fetches the next burst and repeat the whole process until there is no more processes to be fed into the CPU. The following figure 1.1 shows a snapshot of the simulation of previously discussed steps.

***figure 1.1***

***Waiting process id: 9 && Current Process id: 0***

***CPU BURST: 4 STEPS: 1***

***CPU BURST: 4 STEPS: 2***

***CPU BURST: 4 STEPS: 3***

***CPU BURST: 4 STEPS: 4***

***Waiting process id: 9 && Current Process id: 7***

***CPU BURST: 3 STEPS: 1***

***CPU BURST: 3 STEPS: 2***

***CPU BURST: 3 STEPS: 3***

***Waiting process id: 0 && Current Process id: 9***

***CPU BURST: 1 STEP: 1***

***Waiting process id: 7 && Current Process id: 0***

As you can observe from the figure above, process with *PID 0* is the current process doing some I/O or CPU burst work and the waiting or next process to be fetch into the CPU from the front of a process queue is process with *PID 9*. Instead of fetching from the tail of a process queue as to how the standard implementation will be, a there are two process queues, the ready and waiting queue. A new process enters the front of the waiting queue and a schedule process is selected from the front of the ready queue. Before the processes are fed into the CPU, the processes are given to a temporary data structure to be sorted by their arrival times and the process with the least arrival time is fetched, put in the ready queue and executed.

The main drawback of fcfs algorithm is that the virtual processors running short jobs may have to wait in line behind processors running very long jobs also known as the convey effect. For instance, imagine if all I/O processes have to wait up in the ready queue for a long CPU burst to finish up before they get the chance to do some work or the CPU sits idle until an I/O burst finishes it work. This leads to a waste of system resources.

Secondly, Round Robin is a timed algorithm which yields better performance for time sharing systems. It does not use priorities but preemption is added to it to enable the system to switch between processes. Every virtual processor is appended at the front of the ready list in this implementation when it becomes ready the front is de-queue and giving to the physical processor. It’s similar to fcfs but instead it receives a time quantum to run after whose end is signaled by a timer interrupt, which is part of the process data structure. Upon interrupt, the processor is recycled at the back of the ready queue and the next process is fetch. Round Robin gives preference to short task and the cost of overhead is high due to context switching which requires saving all process information in some data structure. The smaller the time quantum the higher the number of context switches. Figure 1.2 shows a snapshot of rr.c simulation.

***Figure 1.2***

***STEP: 1 . . . . . .***

***STEPS: 12***

***\*\* Context Switching is in process \*\****

***PID: 0 is Pre-empted***

***Remaining Length of Process: 0 is: 62***

***\*\*\* Pre-emption is done \*\****

Round Robin as pointed out earlier is good for time sharing platforms because it allows more process to contend for resources which may reduce the CPU idle time drastically. It also allows more responsiveness which will beneficial to multiple users on a timed shared system. The average waiting time for Round Robin is often long. And also the average turnaround time increases with a smaller time quantum since it requires more context switches.