EECS3221 Project Report

Name: Ruojie Sun
Student number: 213107313

1. Summary

The system that we simulates in this project consists of three parts: ready queue, device queue and CPU. In general, one processing cycle of a process is

1. Waiting in ready until it reaches the head of the queue.
2. Being popped out from the queue to one CPU to process its current CPU burst.
3.After the CPU burst finishes, the process is put into the device queue to process the IO burst.
4. After its IO burst finishes, it is again put into the ready queue waiting to be process for its next CPU burst.

The main difference between Three-level Feedback Queue algorithm (fbq) and First Come First Serve (fcfs) algorithm as well as Round-Robin (rr), is that, the former has three ready queues each with a particular priority, but for fcfs and rr, there is only one ready queue and no concept of "priority".

2. Algorithm

(1). First come first serve:

For this particular scheduling algorithm, the implementation is relatively simple. There is only one ready queue to hold all processes which are waiting to be processed by a CPU. Moreover, the order of the process in that queue is simply based on the time order of arrival, which can be interpreted from the name of this algorithm.

(2). Round-Robin:

This scheduling algorithm is a bit harder than the previous one since a new attribute is involved to the system: time quantum. A CPU only keeps processing on a process for a specific period of time, which is known as "time quantum". If the current burst finished during the time quantum, the process would switch to the device queue, which is as same as fcfs algorithm. However, if the current burst did not finish in time, it would be mandatorily put into the tail of the ready queue and a context switch would occur at this time. That is how the rr algorithm is preemptive.

The result of the simulation under rr depends heavily on the value of time

quantum. In general, while the value of time quantum increases, the average waiting time, the average turnaround time, the finish time also increases but the times of context switches decrease. Since in this project we do not take the cost of context switches into account, it seems that lower the time quantum, better the performance.

(3) Feedback Queue:

The special part of this algorithm is that, instead of one ready queue, there are three ready queues which are scheduling in rr with time quantum q1, rr with time quantum q2 (q1 < q2) and fcfs respectively. In other words, if a process popped out from ready queue 1, the process would be processed at most q1 time units in a CPU. Similarly, for processes in ready queue 2, the upper limit is q2. However, for queue 3, the processes are scheduled by first-come-first-serve rule.

Another issue should be emphasized is the priority of processes. Obviously, processes in ready queue 1 have the highest priority, and processes in ready queue 2 have lower priority. Processes in ready queue 3 have the lowest priority. The first time when a process is queuing and waiting to be processed on a CPU burst, it is assigned to queue 1. If current burst did not finish during q1, the process would be put to the tail of queue 2. Same situation apply to process in queue 2, which mean the process would be put into queue 3. Moreover, if current burst finished during q1, the process would be put into device queue and the priority would stay the same. After a process finished its IO burst, the priority of it should be increased. However, there are several ways to achieve that and each could result in different simulation result. For example, we can change the priority to queue the highest no matter which priority it was, or just increase the priority by one (originally queue 2 -> now queue 1, originally queue 3 -> now queue 2).

3. Scheduler Configurations

1. For fcfs, this is only one way to do the simulation since there are no variants of the algorithm.

2. For rr, first I was trying to put newly arrived processes, processes which just finished IO burst, and processes which did not finish the cpu burst during the time quantum into a temp queue. After sort this queue by pid, put the processes into ready queue. However, the simulation result of this approach differs from the given result. So I decided to sort them by pid separately and by doing so I get the correct result.

3. For fbq, there are too many configurations of on this algorithm.

First, there are two methods to increase the priority of processes. The first one is directly promote every process that finished its IO burst to highest priority no matter what its original priority was; the second one is increase the priority by only one level, e.g. change from 3 to 2, or 2 to 1. These two methods could lead to totally different

results.

Second issue is about how to preempt the running process which has the lowest priority. The conditions to determine whether a process is with lowest priority are various. For example, which ready queue it came from and how many time unit left to stay at the CPU. In my final version of code, if two processes came from the same queue, the way to measure which one has lower priority is to compare the value of (time quantum – burst step) for queue 2 processes, and (burst length – burst step) for queue 3 processes (since if there are only queue 1 processes in CPUs, there is no need to preempt any running processes.) At the beginning I forgot to promote processes after they finished the CPU burst, so I got 60 times of context switch in the simulation result. After that I tried different combinations of such conditions, I found that the most efficient way is to compare the minimum of the previous values. The process with the bigger value has lower priority.

Optimal parameters

rr: time quantum = 9;

fbq: q1 = 5, q2 = 37.