RICHMOND FRIMPONG

CSE23004

212114641

CSE3221 OPERATING SYSTEMS

CPU SCHEDULING ALGORITHMS

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

This report presents the design and implementation of various scheduling algorithms: first-come first-served scheduling, round robin and feed-back scheduling. First come first serve is the most basic and simplest scheduling algorithms out of the three. Round Robin is a scheduling algorithm very suitable for real time sharing systems. Lastly, multi-level feedback queue is a suitable algorithm for interactive systems for which modern operating such as BSD Unix derivatives, Solaris and subsequent Windows operating systems use as a form of their base schedulers.

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. Fcfs is non-pre-emptive scheduling algorithm which CPU executing a process must either terminate, or that process must go to IO before the CPU is assigned to another process. When a process is given to the CPU it finishes its job and terminates. Termination also happens when there is input and 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 finish 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.

For this implementation of first come first serve (fcfs) , a process that request the Central Processing Unit(CPU) first is allocated to it first. The way 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, the process with *ID 0* is the current process doing I/O or CPU burst work while waiting for the next process to be fetched into the CPU from the front of waiting process queue with process *ID 9*. Two process queues data structures are created, 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. The process with the least arrival time is fetched, put in the ready queue, and then passed to the CPU to be 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 known as the convey effect. For instance, imagine if all I/O processes have to wait 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 enable the system to switch between processes. Every virtual processor is appended at the front of the ready list. When it becomes ready, the front de-queues and gives to the physical processor. It is similar to fcfs but instead it receives a time quantum to run after whose end is signaled by a timer interrupt. Upon interrupt, the processor is recycled at the back of the ready queue and the next process is fetched. Round Robin gives preference to short tasks, 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's idle time drastically. It also allows more responsiveness which will benefit multiple users on a timed shared system. The average waiting time for Round Robin is often long. In addition, the average turnaround time increases with a smaller time quantum since it requires more context switches.

Lastly, we tackle multi-level feedback queues ***(MLFQ)*** which is a well-known scheduler for interactive systems. MLFQ uses past behavior to predict its future and assign job priorities due to the adaptive policy in its decision making process. This scheduler tends to favor jobs that have less CPU burst time. MLFQ has a number of distinct queues with each running on different priority levels. At any point in the system, a job or process is running a process from a single queue.

All incoming processes are assigned some priority depending on their burst time. A job with the highest priority is chosen to run. If there is more than one job in a given queue with the same priority, a round-robin scheduling algorithm is used to resolve this contention. Jobs start in the highest priority queue and the priority is dropped down one level *(first level)* if the first time slice expires. If the second time slice expires, the priority is dropped down another level *(second level)* and the job defaults to first come first serve scheduling algorithm with a maximum time slice if both time slices are not met. Processes in the second level will only run if the top level queues are empty and there is an available CPU. If a new job or process arrives at the top most level *(ready queue)*, it may preempt processes in the lower levels and cause a context switch. If process finish in first level, they move up to the second level or go directly to the ready queue which is the default for fcfs only if both queues are empty and they may be preempted at any time by processes in either first and second level queues causing a context switch. Figure 1.3 shows snapshot of fbq.c simulation.

Figure 1.3

```
STEPS: 1 . . . . . 10
First time slice is met
PID: 0      Lengths: 74
STEPS: 10 . . .  40
Second time slice is met
STEPS: 40 . . . . 74
PID: 0      Lengths: 74
DONE PROCESS ID: 0            && RT    : 0
Waiting process id: 0 && Current Process id: 0
STEPS: 1 . . . 4
Promotion 1 defaults to FCFS . . . . .
```

Observe how a CPU burst of 74 which was greater than both time slices went through all the different levels before it finished it work. Promotion happens when the CPU burst length is less than both time slices and the current process are assigned the highest priority. This is solely based on observation of the nature of the job and a prior knowledge of the job. In this way, it manages and achieves best of both worlds by delivering excellent overall performance and makes progress for long-running CPU–intensive burst.

There are a few problems with associated with MLFQ. Starvation occurs when there are too many interactive jobs in the system and they consume all CPU resources and thus long-burst process never receives any CPU time. CPU bound jobs experiences a drop in priority and I/O bound jobs or processes stay at high priority.

As the operating system knows what is best for each and every process of the system, it is often ideal to provide some optimal parameters through an interface to allow users and administrators to provide some hints to the OS in order to make the best decisions.

SUMMARY OF ALL EXPERIMENTS DONE WITH FCFS, RR && MLFQ

```
FCFS
************************************************
Average Waiting Time              : 23.20
Average Turn Around Time          : 40492.60
Time all for all CPU processes    : 68571
CPU Utilization Time              : 78.91%
Total Number of Context Switches  : 0
Last Process to finish PID        : 28
************************************************
```

```
RR WITH QUANTUM 12
***************************************************
Average Waiting Time              : 22.70
Average Turn Around Time          : 40492.10
Time all for all CPU processes    : 68563
CPU Utilization Time              : 78.91%
Total Number of Context Switches  : 2408
Last Process to finish PID        : 28
*************************************************
```

```
RR WITH QUANTUM 80
**********************************************
Average Waiting Time              : 24.57
Average Turn Around Time          : 40493.97
Time all for all CPU processes    : 68581
CPU Utilization Time              : 78.89%
Total Number of Context Switches  : 131
Last Process to finish PID        : 28
************************************************
```

```
RR WITH QUANTUM 100
****************************************************
Average Waiting Time              : 26.23
Average Turn Around Time          : 40495.63
Time all for all CPU processes    : 68574
CPU Utilization Time              : 78.90%
Total Number of Context Switches  : 70
Last Process to finish PID        : 28
****************************************************
```

```
*************OPTIMAL PARAMETERS FOR ROUND ROBIN RANGES FROM 10 TO 30 TIME SLICES************
```

```
FBQ WITH QUANTUMS 10 100
************************************************
Average Waiting Time              : 12.87
Average Turn Around Time          : 40487.17
Time all for all CPU processes    : 68543
CPU Utilization Time              : 78.94%
Total Number of Context Switches  : 1083
Last Process to finish PID        : 28
***********************************************
```

```
FBQ WITH QUANTUMS 10 50
**********************************************
Average Waiting Time              : 16.40
Average Turn Around Time          : 40494.30
Time all for all CPU processes    : 68558
CPU Utilization Time              : 78.92%
Total Number of Context Switches  : 1364
Last Process to finish PID        : 28
************************************************
```

```
FBQ WITH QUANTUMS 100 50
***********************************************
Average Waiting Time              : 19.23
Average Turn Around Time          : 40490.70
Time all for all CPU processes    : 68555
CPU Utilization Time              : 78.92%
```

```
FBQ WITH QUANTUMS 16 96
************************************************
Average Waiting Time              : 15.53
Average Turn Around Time          : 40488.13
Time all for all CPU processes    : 68546
CPU Utilization Time              : 78.93%
```

```
Total Number of Context Switches    : 73                Total Number of Context Switches    : 781

Last Process to finish PID          : 28                Last Process to finish PID          : 28

******************************************              *************************************************

*************OPTIMAL PARAMETERS FOR MLFQ RANGES FROM 10 && 100, 16 && 96 TIME SLICES************
```

The above optimal times for round robin were chosen because shorter quantum allows many processors to circulate processes quickly, giving each job a chance to run. This way, highly interactive jobs with round robin do not have to use their quantum and will not have to wait long before they get the CPU again. Optimal times for multilevel feedback queue were chosen because it offered less waiting time and less context switches making the system more responsive to user process with fewer overheads. Increase in quantum for lower level queues favors CPU bound processes by giving them larger chunk of CPU time when they are allowed to run.