Name: - Elias Werede
Cpt_S: - 370
Program 3: Operating System Scheduler
Due: - March 22, 2024

Project 3 Report

The implementation of a multilevel feedback queue (MFQS) scheduler program is for a simple operating system. The scheduler is responsible for managing the execution of threads in the system. The threads are divided into three priority levels. The MFQS uses three queues to store the threads belonging to each priority level. The queues are implemented as vectors. Each thread is represented by a Thread Control Block (TCB).

The algorithm used by the scheduler is a round-robin algorithm with a varying time quantum. The scheduler begins by checking the highest priority queue (Queue0). If this queue is not empty, the first thread in the queue is removed and its TCB is retrieved. The thread is then started, and it is allowed to execute for the time quantum assigned to its priority level. After the time quantum has elapsed, the thread is suspended, and its TCB is placed back into the queue. The scheduler then moves on to the next priority level (Queue1) and repeats the same process. If all queues are empty, the scheduler goes to sleep for a fixed time interval before starting again.
The scheduler also includes several additional features like the ability for a thread to delete itself by calling the deleteThread() method. This method sets the state of the thread's TCB to "terminated", which causes the scheduler to remove it from the queue. In addition to that feature it has the ability for a thread to sleep for a specified amount of time by calling the sleepThread() method. This method suspends the thread for the specified amount of time before resuming its execution.

Comparing the test results

Part 1 **Round-robin Scheduler**

Name	Response T.	Turnaround T.	Execution T.
Thread[a]	1999	23904	21905
Thread[b]	2999	3900	901
Thread[c]	4000	15903	11903
Thread[d]	5000	27904	22904
Thread[e]	6001	6401	400
Total	19999	78012	58013

Part 2 - **Multilevel Feed Back-Queue (MFQS) Scheduler**

Name	Response T.	Turnaround T.	Execution T.
Thread[a]	499	23409	22910
Thread[b]	1000	5401	4401
Thread[c]	1499	15407	13908
Thread[d]	1999	30307	28308
Thread[e]	2500	2900	400
Total	7497	77424	69927

From the test results, we can see that in the Round-robin Scheduler, Thread [a] had the longest execution time of 21905 whereas in the MFQS Scheduler, Thread [a] had the second longest execution time of 22910. Interestingly, Thread [e] execution time remains consistent across both scheduling systems 400. it was likely a short task that could complete within a single quantum in

the Round-robin or quickly execute in the high-priority queue before any major preemptions in the MFQS.

We can see that in the Round-robin Scheduler, Thread [a] had the longest response time of 1999, whereas in the MFQS Scheduler, Thread [a] had the shortest response time of 499. This shows that the MFQS Scheduler is better at providing quicker response time to the threads. Similarly, in the Round-robin Scheduler, Thread [a] had the longest turnaround time of 23904, whereas in the MFQS Scheduler, Thread [a] had 23409. Which indicates that the turnaround time is slightly better in the MFQS.

Overall, the MFQS scheduler demonstrates a better response time across all threads when compared to the round-robin scheduler. This improvement is due to the prioritization that allows shorter or more critical processes to begin execution sooner. However, when examining the total execution time, the MFQS scheduler shows an increased. This may occur because threads in the MFQS, especially if they are longer-running, can be preempted by newly arriving higher-priority threads. In contrast, the round-robin scheduler, with its fixed time quantum for all processes, is simpler and may offer more predictable performance for each thread but at the cost of potentially slower initial response, especially for long-running processes. The choice between round-robin and MFQS scheduling would typically depend on the specific requirements of the system. If the goal is to minimize the response time, especially for high-priority processes, the MFQS approach is beneficial. However, if the system demands predictability for CPU time allocation, a round-robin scheduler might be preferred despite the potentially slower response.

If we were to implement part 2

If we were to implement part 2 using FCFS instead of Round Robin, it would significantly affect the performance of the MFQS scheduler. In FCFS, threads are executed in the order they arrive, which is the scheduler will not preempt any running threads until they complete their execution.

If we run Test2.java with the FCFS-based, we would notice that the response time for each thread will be significantly higher than what we got with Round Robin. This is because threads

that arrive earlier will have to wait for the running threads to complete their execution, which can take a long time depending on the burst time of the running threads.