CSS 430 Operating Systems

Professor Stephen Dame

Program 2

Christopher Tran

**Part 1 Performance Evaluation**

Base Scheduler.java

|  |  |  |  |
| --- | --- | --- | --- |
| Thread | Response Time | Turn Around Time | Execution Time |
| A | 1996 | 7000 | 5004 |
| B | 2996 | 3997 | 1001 |
| C | 3997 | 6999 | 3002 |
| D | 4996 | 11000 | 6004 |
| E | 5996 | 6496 | 500 |

RoundRobin Scheduler.java

|  |  |  |  |
| --- | --- | --- | --- |
| Thread | Response Time | Turn Around Time | Execution Time |
| A | 1987 | 28989 | 27002 |
| B | 2986 | 9987 | 7001 |
| C | 3986 | 20988 | 17002 |
| D | 4986 | 32989 | 28003 |
| E | 5987 | 6487 | 500 |

We enforced the Round Robin Scheduling and this caused an increase in the execution time of almost all the threads. This is due to wasted CPU cycles of the Round Robin scheduler that keeps sleeping even if a process has completed. The base scheduler sets priority levels and allows other processes to use the remaining CPU time and why we get mixed thread output.

By enforcing the round robin system, we block threads from using that valuable CPU time and increase execution time overall. The drastic increase is due to the two processes we do not see output from. One is the Thread for the Prompt and the other is the Thread for Test2/Test2b. These are still processed by the Scheduler and can be noticed when the output from Thread A,B,C,D,E pauses for moments at a time.

**Part 2 Implementation**

Initial Diagram



**Part 2 Implementation and Design Choices**

Problem

Check every timeSplice / 2 if there are Threads in lower Queues to be processed

Solution

Integer variable equal to timeSplice / 2 (500ms) that is used as a metric for how often while loop is repeated. At each loop, Queue is checked from queue[0] for any Threads. If they exist, then they are processed. If no new threads have been added, then loop will get to the previously executed position and continue. No calls to check if the other queues have Threads to process as the check is built in by design and cycle time is easily changed.

Problem

Limit CPU time even when interrupted by Thread in lower Queue

Solution

Two integer arrays of the same length as Queue levels. One stores number of cycles the head of each Queue has received. The other stores the maximum cycles allowed to execute. When the limit is reached, Thread is moved to the next Queue. This allows a Thread to be interrupted and when continued, will only execute remaining cycles until maximum.

**Part 2 Performance Evaluation**

RoundRobin Scheduler.java

|  |  |  |  |
| --- | --- | --- | --- |
| Thread | Response Time | Turn Around Time | Execution Time |
| A | 1987 | 28989 | 27002 |
| B | 2986 | 9987 | 7001 |
| C | 3986 | 20988 | 17002 |
| D | 4986 | 32989 | 28003 |
| E | 5987 | 6487 | 500 |

MLQS Scheduler.java

|  |  |  |  |
| --- | --- | --- | --- |
| Thread | Response Time | Turn Around Time | Execution Time |
| A | 500 | 22506 | 22006 |
| B | 1000 | 5502 | 4502 |
| C | 1501 | 14504 | 13003 |
| D | 2000 | 29505 | 27505 |
| E | 2500 | 8001 | 5501 |

Performance of the enforced round robin and the MLQS is similar in Execution time and Turn Around Time. The only change is response time but that is expected due to the 500ms cycle of the implementation of MLQS versus the 1000ms of the Round Robin.

Similar to the Round Robin, the MLQS is affected by wasted CPU time. When a Thread completes, there are CPU cycles that are idle and wasted until the scheduler wakes up again. In addition, because all the CPU bursts are divisible by 500ms, they hit a situation where they would complete with just a little more cycle time but they don’t and get pushed into another cycle. This was tested by setting the cycle time to 501 and the reduction of execution time was 2-2.5 seconds. Setting cycle time to 515 had a greater effect with a reduction in 5-6 seconds. This is likely due to the introduced overhead with the MLQS where CPU cycles are used to check and move Threads.

Thread “e” will rarely complete in Queue 0 due to the CPU burst being the same as the cycle time. There are additional CPU cycles needed to mark the thread as complete. It gets moved into Queue 1 and wastes almost a full second of execution time.

Similar to the Round Robin, there are two additional threads that are causing an increase in Execution time. The thread for the prompt and the thread for Test2/Test2b. These make their way into Queue[2] and eventually increase execution time for all threads that almost make it into Queue[2].

Considering that if we implemented a FCFS in Queue2, the Scheduler would likely hang due to the Thread for the prompt and the Thread for Test2\Test2b. If we didn’t have to consider that situation, the execution time would be similar since once all threads are in Queue2, the performance of a RoundRobin vs a FCFS wouldn’t change the overall execution time but would have an effect on the Turn Around Time based on which thread was first. This is likely the reason why MLQS simply implement a FCFS in the last queue since it doesn’t make sense to rotate threads.

Performance Metrics

Base Scheduler.java

|  |  |  |  |
| --- | --- | --- | --- |
| Thread | Response Time | Turn Around Time | Execution Time |
| A | 1996 | 7000 | 5004 |
| B | 2996 | 3997 | 1001 |
| C | 3997 | 6999 | 3002 |
| D | 4996 | 11000 | 6004 |
| E | 5996 | 6496 | 500 |

RoundRobin Scheduler.java

|  |  |  |  |
| --- | --- | --- | --- |
| Thread | Response Time | Turn Around Time | Execution Time |
| A | 1987 | 28989 | 27002 |
| B | 2986 | 9987 | 7001 |
| C | 3986 | 20988 | 17002 |
| D | 4986 | 32989 | 28003 |
| E | 5987 | 6487 | 500 |

MLQS Scheduler.java

|  |  |  |  |
| --- | --- | --- | --- |
| Thread | Response Time | Turn Around Time | Execution Time |
| A | 500 | 22506 | 22006 |
| B | 1000 | 5502 | 4502 |
| C | 1501 | 14504 | 13003 |
| D | 2000 | 29505 | 27505 |
| E | 2500 | 8001 | 5501 |