**Lab 3 - Process Schedulers**

Christian Poynter and

John Frederick

[poyntecd@uwec.edu](mailto:poyntecd@uwec.edu)

[frederja@uwec.edu](mailto:frederja@uwec.edu)

**Abstract**

*In this lab, we set forward to implement three different process schedulers in C++. While we were not forging any new ground in the field of Computer Science, it was our first interaction with both schedulers and the language of C++. We successfully accomplished our objective, and implemented a Real-Time Scheduler, a Multi-Level Feedback Queue Scheduler, and a Hybrid Windows Scheduler.*

**1. Introduction**

The purpose of this document is to evaluate the results of the three schedulers. We will look at the average waiting time, average turnaround time, and total time taken for each scheduler and compare them. We will then draw our conclusions on which scheduler is most efficient and which is most fair.

**2. Details/Process**

**2.1. Real Time Scheduler (RTS)**

The RTS uses the deadlines of the processes in queue to determine which process to run at any given time. The process with the nearest deadline is run until it either finishes normally or is interrupted by another process with a sooner deadline.

We tested the RTS with three different input files. The first was *testfile1*, the test file provided with the assignment. This is a very basic set of 10 processes, only 7 of which are valid processes (3 have negative values in them). These processes do not overlap, so it provided a good test of the scheduler being able to simply run the processes in order.

The second test file was *testfile2*, which is based off of the slides for the class, with 2 extra processes added. These processes demonstrate the capabilities of the scheduler to dynamically change what process

is running based on the process’ deadline, and also the

ability to terminate processes that run out of time to complete.

The third and final test file was *1m\_processes*, a list of 1,000,000 processes provided by Dr. Jack Tan. These processes were randomly generated, and they cover every case. They also place a large load on the program, demonstrating its efficiency and exposing any memory leaks (of which we found none).

**2.2. Multi-Level Feedback Queue Scheduler (MFQS)**

**2.3. Hybrid Windows Scheduler (HWS)**

The HWS puts processes into separate queues based on priority, with one queue for each priority level. A process can have a priority between 1 and 99, inclusive. Throughout the scheduler, each process’ priority is dynamically changed so as to make sure all processes run in a fair way.

The scheduler runs the highest priority queue using Round Robin. A process either goes into IO one clock tick before the time quantum, or, if it doesn’t do IO, breaks at the time quantum. Its priority is decremented by the amount of time spent running in its current time quantum. If it does not do IO, it simply gets added to a new priority queue. If it does do IO, it goes into the IO queue and stays there the duration of its IO. Its priority is then increased by the amount of time spent in IO, and it comes back into the appropriate priority queue. At no time does a process’ priority go below its original priority, and a process that starts with a priority less than 50 can never go above 50.

Every 100 clock ticks, the scheduler checks for any processes that haven’t run in the last 100 ticks, and increments their priority by 10 in the hopes that they will be able to run.

We ran this scheduler using the same 3 test files as the previous schedulers. We also ran the test file *HWS starving* to test the scheduler’s ability to handle starving processes. All results are shown in section 3.

**3. Results**

**3.1. Testfile1**

**RTS:**

850 <- Process 1 -> 851 | 3980 <- Process 2 -> 3995 | 5148 <- Process 7 -> 5157 | 6529 <- Process 5 -> 6629 | 7262 <- Process 4 -> 7358 | 8095 <- Process 3 -> 8178 | 9068 <- Process 6 -> 9126

AWT: 0

ATT: 51.7143

Processes scheduled: 7

Processes finished: 7

**HWS (tq = 4):**

850 <- Process 1 -> 851 | 3980 <- Process 2 -> 4003 | 5148 <- Process 7 -> 5165 | 6529 <- Process 5 -> 6629 | 7262 <- Process 4 -> 7358 | 8095 <- Process 3 -> 8178 | 9068 <- Process 6 -> 9126

AWT: 0

ATT: 53

Processes scheduled: 7

**HWS (tq = 8):**

850 <- Process 1 -> 851 | 3980 <- Process 2 -> 3999 | 5148 <- Process 7 -> 5161 | 6529 <- Process 5 -> 6629 | 7262 <- Process 4 -> 7358 | 8095 <- Process 3 -> 8178 | 9068 <- Process 6 -> 9126 |

AWT: 0

ATT: 51.8571

Processes scheduled: 7

**3.2. Testfile2**

**RTS:**

0 <- Process 1 -> 1 | 1 <- Process 6 -> 2 | 2 <- Process 2 -> 3 | 3 <- Process 3 -> 4 | 4 <- Process 2 -> 8 | 8 <- Process 4 -> 12 | 12 <- Process 6 -> 13 | 13 <- Process 5 -> 20 |

AWT: 5.8

ATT: 7.4

Processes scheduled: 6

Processes finished: 5

**HWS (tq = 4):**

0 <- Process 1 -> 1 | 1 <- Process 6 -> 2 | 2 <- Process 2 -> 5 | 5 <- Process 4 -> 7 | 7 <- Process 2 -> 9 | 9 <- Process 4 -> 10 | 10 <- Process 6 -> 11 | 11 <- Process 3 -> 12 | 12 <- Process 1 -> 14 | 14 <- Process 5 -> 15 | 15 <- Process 4 -> 16 | 16 <- Process 5 -> 17 | 17 <- Process 1 -> 19 | 19 <- Process 5 -> 36

AWT: 6.83333

ATT: 13.5

Processes scheduled: 6

**HWS (tq = 8):**

0 <- Process 1 -> 1 | 1 <- Process 6 -> 2 | 2 <- Process 2 -> 7 | 7 <- Process 4 -> 11 | 11 <- Process 6 -> 12 | 12 <- Process 3 -> 13 | 13 <- Process 1 -> 17 | 17 <- Process 5 -> 24

AWT: 7.33333

ATT: 10.3333

Processes scheduled: 6

**3.3. 1m\_processes**

**RTS:**

AWT: 1027.18

ATT: 5.63186

Processes scheduled: 997000

Processes finished: 5536

**MFQS (5 queues, tq = 4, aging time = 100):**

AWT: 3.34451e+07

ATT: 3.34452e+07

3.33448e+13   3.33448e+13

Processes scheduled: 997000

Time elapsed: 205

**HWS (tq = 100):**

**3.4. HWS starving**

**HWS (tq = 4):**

0 <- Process 1 -> 1 | 1 <- Process 6 -> 2 | 2 <- Process 2 -> 5 | 5 <- Process 4 -> 7 | 7 <- Process 2 -> 9 | 9 <- Process 4 -> 10 | 10 <- Process 6 -> 12 | 12 <- Process 3 -> 15 | 15 <- Process 4 -> 16 | 16 <- Process 6 -> 19 | 19 <- Process 3 -> 23 | 23 <- Process 6 -> 26 | 26 <- Process 3 -> 30 | 30 <- Process 6 -> 33 | 33 <- Process 3 -> 37 | 37 <- Process 6 -> 40 | 40 <- Process 3 -> 44 | 44 <- Process 6 -> 47 | 47 <- Process 3 -> 51 | 51 <- Process 6 -> 53 | 53 <- Process 3 -> 100 | 100 <- Process 5 -> 103 | 103 <- Process 3 -> 109 | 109 <- Process 5 -> 112 | 112 <- Process 3 -> 118 | 118 <- Process 5 -> 119 | 119 <- Process 3 -> 200 | 200 <- Process 1 -> 202 | 202 <- Process 3 -> 205 | 205 <- Process 1 -> 207 | 207 <- Process 3 -> 241

AWT: 57

ATT: 103.833

Processes scheduled: 6

**HWS (tq = 8):**

0 <- Process 1 -> 1 | 1 <- Process 6 -> 2 | 2 <- Process 2 -> 7 | 7 | 7 <- Process 4 -> 11 | 11 | 11 <- Process 6 -> 17 | 17 <- Process 3 -> 21 | 21 <- Process 6 -> 28 | 28 <- Process 3 -> 32 | 32 <- Process 6 -> 38 | 38 | 38 <- Process 3 -> 100 | 100 <- Process 5 -> 107 | 107 | 107 <- Process 3 -> 200 | 200 <- Process 1 -> 204 | 204 | 204 <- Process 3 -> 241 |

AWT: 57.1667

ATT: 97.6667

Processes scheduled: 6

**4. Comparison**

**5. Conclusion**

**Test Files**

**testfile1:**

Pid Bst Arr Pri Dline I/O

1 1 850 61 852 3

2 15 3980 82 3998 2

3 83 8095 51 8179 0

4 96 7262 66 7365 0

5 100 6529 1 6630 0

6 58 9068 24 9132 0

7 9 5148 35 5166 4

8 -8 5924 3 5925 0

9 100 -1 34 101 0

10 72 1918 43 -5 0

**testfile2:**

Pid Bst Arr Pri Dline I/O

1 5 0 10 15 3

2 5 2 56 8 2

3 1 3 24 6 10

4 4 5 33 12 5

5 7 5 7 26 6

6 2 1 24 14 4

**HWS starving:**

Pid Bst Arr Pri Dline I/O

1 5 0 10 15 3

2 5 2 56 8 2

3 200 3 24 6 0

4 4 5 33 12 5

5 7 5 20 26 6

6 20 1 24 14 4