Name: CENSORED

Due Date: 6/19/19

Class: COMP-3500

Assignment: Lab 3

**Abstract:**

For our third and final lab assignment, we were given three files: processesgenerator.o, common2.h, and processesmanagement2.c. We were not supposed to modify the first two files, but the last file had a lot to add to it. We needed to modify it so that we could observe the average waiting time in the job queue and allow our output to be modified by different memory policies. The program was finished in jGrasp and tested using a Tux Machine, which was accessed remotely using a Unix terminal and a sftp client (the latter being used to move files to the Tux machine). Our program (after implementing the aforementioned changes) was designed to run multiple processes with the new memory policies and return very specific times. With these times, we were able to determine which of the CPU Scheduling Policies (combined with specific memory policies) is the most efficient and which was the most inefficient. The times that we received from running the program are as follows: the average turnaround times, the average response times, the CPU busy times, the throughput times, the average waiting times, and the average waiting times in the job queue. We will explain in detail later as to what these times are and how they relate to the CPU Scheduling Policies.

**Introduction:**

The turnaround time is the time it takes for a process to complete and terminate, the response time is the time a process gets in the CPU the first time, the CPU busy time is the fraction of time a CPU is used by user processes, the throughput time is the number of processes completed divided by the observation time, the waiting time is the total time spent by a process in the ready queue, and the waiting time in the job queue is just the total time spent by a process in the job queue. The three CPU Scheduling Policies we implemented and observed times for were First Come First Serve (FCFS), Shortest Remaining Time First (SRTF), and Round Robin (RR). We implemented the RR policy with several different quantum times (a quantum is a specified amount of milliseconds that must elapse before the CPU is preempted and the previously running process is added to the end of the ready queue), namely 10ms, 20ms, 50ms, 250ms, and 500ms. This was done to see how the difference in the quantum time would affect the measurable times for this specific policy. We evaluated our code through careful analysis of our data output in comparison with expected estimations of outputs for each policy. For example, we would not expect a low turnaround time from the initial FCFS policy (which is the unmodified FCFS policy that came with the code template we were given), and checking the data yielded from the program against this expectation proved correct. The specifics on how each policy was implemented will be covered in detail in the next few paragraphs.

**Body:**

Below is a table comparing all the values we received from our different CPU Scheduling Policies combined with the different Memory Policies. It should be noted that we had 3 CPU Scheduling Policies and 6 different Memory Policies (including the initial results that we received from running the code “as-is”). There are no times given for the average waiting time in the job queue for the initial run due to it not being implemented initially, and we were told not to modify the code before collecting the data. In total, we had 42 different policy combinations to observe times for, and all the times are documented in the table below. We will delve more into the details with individual charts after this table.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | ATAT | ART | CBT | T | AWT | AWTJQ |
| Initial FCFS | 17.645918 | 3.047055 | 0.971225 | 0.987630 | 12.438119 | N/A |
| Initial SRTF | 4.167688 | 1.422049 | 0.970897 | 1.047933 | 2.098073 | N/A |
| Initial RR (q = 10) | 10.548371 | 0.103391 | 0.967029 | 1.013795 | 9.258256 | N/A |
| Initial RR (q = 20) | 10.289114 | 0.191964 | 0.970411 | 1.013815 | 8.889810 | N/A |
| Initial RR (q = 50) | 10.984953 | 0.462164 | 0.976775 | 1.013680 | 9.227262 | N/A |
| Initial RR (q = 250) | 14.581814 | 1.857907 | 0.980148 | 0.983577 | 10.916514 | N/A |
| Initial RR (q = 500) | 16.393368 | 2.620341 | 0.980404 | 0.982863 | 11.682897 | N/A |
| OMAP FCFS | 17.166385 | 3.408817 | 0.971220 | 0.987624 | 11.559703 | 0.539989 |
| OMAP SRTF | 4.182759 | 1.430318 | 0.971025 | 1.048071 | 2.107486 | 0.001027 |
| OMAP RR (q = 10) | 10.354262 | 0.101761 | 0.968668 | 1.013903 | 9.068508 | 0.000999 |
| OMAP RR (q = 20) | 10.239099 | 0.192256 | 0.971770 | 1.013838 | 8.841898 | 0.001064 |
| OMAP RR (q = 50) | 10.924752 | 0.463360 | 0.977139 | 1.013833 | 9.171097 | 0.001021 |
| OMAP RR (q = 250) | 14.507688 | 1.974098 | 0.980323 | 0.987156 | 10.695091 | 0.161974 |
| OMAP RR (q = 500) | 16.147570 | 2.809357 | 0.981086 | 0.983423 | 11.213781 | 0.299555 |
| Paging (256) FCFS | 17.156220 | 3.435298 | 0.972493 | 0.987720 | 11.563717 | 0.571876 |
| Paging (256) SRTF | 4.175654 | 1.412969 | 0.970956 | 1.047997 | 2.101737 | 0.001104 |
| Paging (256) RR (q = 10) | 10.269462 | 0.100470 | 0.969557 | 1.013868 | 8.986466 | 0.001047 |
| Paging (256) RR (q = 20) | 10.205009 | 0.190126 | 0.971673 | 1.013827 | 8.807750 | 0.001079 |
| Paging (256) RR (q = 50) | 10.900322 | 0.462882 | 0.977274 | 1.013749 | 9.149781 | 0.001043 |
| Paging (256) RR (q = 250) | 14.508097 | 1.970307 | 0.980255 | 0.987088 | 10.691832 | 0.162051 |
| Paging (256) RR (q = 500) | 16.188337 | 2.854083 | 0.981085 | 0.983422 | 11.227980 | 0.326618 |
| Paging (8192) FCFS | 16.644926 | 3.549465 | 0.971227 | 0.987631 | 10.905334 | 0.829181 |
| Paging (8192) SRTF | 4.180278 | 1.427658 | 0.970980 | 1.048023 | 2.104718 | 0.001140 |
| Paging (8192) RR (q = 10) | 10.321589 | 0.101383 | 0.969122 | 1.013860 | 9.037519 | 0.001023 |
| Paging (8192) RR (q = 20) | 10.247727 | 0.191277 | 0.971537 | 1.013865 | 8.850397 | 0.001038 |
| Paging (8192) RR (q = 50) | 10.890679 | 0.476053 | 0.977146 | 1.013841 | 9.133591 | 0.010345 |
| Paging (8192) RR (q = 250) | 14.451484 | 2.208626 | 0.980302 | 0.987135 | 10.393626 | 0.432436 |
| Paging (8192) RR (q = 500) | 15.870141 | 2.985508 | 0.981042 | 0.983379 | 10.763719 | 0.574606 |
| Best Fit FCFS | 17.182111 | 3.411628 | 0.971295 | 0.987701 | 11.604862 | 0.539935 |
| Best Fit SRTF | 4.167661 | 1.421526 | 0.970966 | 1.048008 | 2.097515 | 0.001073 |
| Best Fit RR (q = 10) | 10.311057 | 0.100771 | 0.969213 | 1.013865 | 9.027333 | 0.001023 |
| Best Fit RR (q = 20) | 10.237342 | 0.192276 | 0.971587 | 1.013827 | 8.840354 | 0.001096 |
| Best Fit RR (q = 50) | 10.871857 | 0.462918 | 0.977411 | 1.013826 | 9.120966 | 0.001038 |
| Best Fit RR (q = 250) | 14.504738 | 1.969637 | 0.980381 | 0.987215 | 10.692372 | 0.161966 |
| Best Fit RR (q = 500) | 16.137925 | 2.824628 | 0.981059 | 0.983396 | 11.211418 | 0.299507 |
| Worst Fit FCFS | 17.188443 | 3.416023 | 0.971278 | 0.987684 | 11.608174 | 0.539990 |
| Worst Fit SRTF | 4.171851 | 1.410313 | 0.971089 | 1.048141 | 2.096041 | 0.001057 |
| Worst Fit RR (q = 10) | 10.241940 | 0.100141 | 0.969342 | 1.013869 | 8.960015 | 0.001020 |
| Worst Fit RR (q = 20) | 10.239799 | 0.191750 | 0.971476 | 1.013801 | 8.841634 | 0.001034 |
| Worst Fit RR (q = 50) | 10.881706 | 0.463509 | 0.977396 | 1.013653 | 9.133898 | 0.001069 |
| Worst Fit RR (q = 250) | 14.517231 | 1.975869 | 0.980400 | 0.987234 | 10.701029 | 0.161941 |
| Worst Fit RR (q = 500) | 16.131121 | 2.823788 | 0.981097 | 0.983435 | 11.212059 | 0.299564 |

Below are comparisons for all the FCFS times measured in the above table. The rough times are relatively close to each other; but using these charts can show more easily which policy combinations had the highest times and which ones had the lowest times. It should be noted that the initial FCFS run had the highest ATAT time and AWT time, the 8192 bit paging FCFS policy had the highest ART time and AWTJQ time, and the 256 bit paging FCFS policy had the highest CBT time and the highest T time. The 8192 bit paging FCFS policy also had the lowest ATAT time and AWT time, the initial FCFS policy had the lowest ART time, the OMAP FCFS policy had the lowest CBT time and T time, and the Best Fit FCFS policy had the lowest AWTJQ time.

Below are comparisons for all the SRTF times measured in the above table. The rough times are relatively close to each other; but using these charts can show more easily which policy combinations had the highest times and which ones had the lowest times. It should be noted that the OMAP SRTF policy had the highest ATAT time and AWT time, the 8192 bit paging SRTF policy had the highest AWTJQ time, the Best Fit SRTF policy had the highest ART time, and the Worst Fit SRTF policy had the highest CBT time and T time. The Worst Fit SRTF policy also had the lowest ART time and AWT time, the initial SRTF policy had the lowest ATAT time, T time, and CBT time, and the OMAP SRTF policy had the lowest AWTJQ time.

Below are comparisons for all the RR times with a quantum of 10 measured in the above table. The rough times are relatively close to each other; but using these charts can show more easily which policy combinations had the highest times and which ones had the lowest times. It should be noted that the initial RR (q = 10) policy had the highest ATAT time, AWT time, and ART time, the 256 bit paging RR (q = 10) policy had the highest AWTJQ time and CBT time, and the OMAP RR (q = 10) policy had the highest T time. The Worst Fit RR (q = 10) policy also had the lowest ATAT time, ART time, and AWT time, the initial RR (q = 10) policy had the lowest CBT time and T time, and the OMAP RR (q = 10) policy had the lowest AWTJQ time.

Below are comparisons for all the RR (q = 20) times measured in the above table. The rough times are relatively close to each other; but using these charts can show more easily which policy combinations had the highest times and which ones had the lowest times. It should be noted that the initial RR (q = 20) policy had the highest ATAT time and AWT time, the 8192 bit paging RR (q = 20) policy had the highest T time, the Best Fit RR (q = 20) policy had the highest ART time and AWTJQ time, and the OMAP RR (q = 20) policy had the highest CBT time. The 256 bit paging RR (q = 20) policy also had the lowest ATAT time, ART time, and AWT time, the initial RR (q = 20) policy had the lowest CBT time, and the Worst Fit RR (q = 20) policy had the lowest AWTJQ time and T time.

Below are comparisons for all the RR (q = 50) times measured in the above table. The rough times are relatively close to each other; but using these charts can show more easily which policy combinations had the highest times and which ones had the lowest times. It should be noted that the initial RR (q = 50) policy had the highest ATAT time and AWT time, the 8192 bit paging RR (q = 50) policy had the highest ART time, AWTJQ time, and T time, and the Best Fit RR (q = 50) policy had the highest CBT time. The OMAP RR (q = 50) policy also had the lowest AWTJQ time, the initial RR (q = 50) policy had the lowest ART time and CBT time, the Worst Fit RR (q = 50) policy had the lowest T time, and the Best Fit RR (q = 50) policy had the lowest ATAT time and AWT time.

Below are comparisons for all the RR (q = 250) times measured in the above table. The rough times are relatively close to each other; but using these charts can show more easily which policy combinations had the highest times and which ones had the lowest times. It should be noted that the initial RR (q = 250) policy had the highest ATAT time and AWT time, the 8192 bit paging RR (q = 250) policy had the highest ART time and AWTJQ time, and the Worst Fit RR (q = 250) policy had the highest T time and CBT time. The initial RR (q = 250) policy had the lowest ART time, T time, and CBT time, the Worst Fit RR (q = 250) policy had the lowest AWTJQ time, and the 8192 bit paging RR (q = 250) policy had the lowest ATAT time and AWT time.

Below are comparisons for all the RR (q = 500) times measured in the above table. The rough times are relatively close to each other; but using these charts can show more easily which policy combinations had the highest times and which ones had the lowest times. It should be noted that the initial RR (q = 500) policy had the highest ATAT time and AWT time, the 8192 bit paging RR (q = 500) policy had the highest ART time and AWTJQ time, and the Worst Fit RR (q = 500) policy had the highest T time and CBT time. The initial RR (q = 500) policy had the lowest ART time, T time, and CBT time, the Best Fit RR (q = 500) policy had the lowest AWTJQ time, and the 8192 bit paging RR (q = 500) policy had the lowest ATAT time and AWT time.

**Conclusion:**

Based on the data we received from running the program, we have decided that the most efficient FCFS policy is either the 8192 bit paging implementation or the OMAP implementation, and the most inefficient FCFS policy is a tie between the initial implementation, the 8192 bit paging implementation, and the 256 bit paging implementation. The most efficient SRTF policy is the initial implementation, and the most inefficient SRTF policy is the 8192 bit paging implementation. The most efficient RR (q = 10) policy is the Worst Fit implementation, and the most inefficient RR (q = 10) policy is the initial implementation. The most efficient RR (q = 20) policy is the 256 bit paging implementation, and the most inefficient RR (q = 20) policy is either the initial implementation or the Best Fit implementation. The most efficient RR (q = 50) policy is either the initial implementation or the Best Fit implementation, and the most inefficient RR (q = 50) is the 8192 bit paging implementation. The most efficient RR (q = 250) policy is the initial implementation, and the most inefficient RR (q = 250) is either the 8192 bit paging implementation or the Worst Fit implementation. The most efficient RR (q = 500) policy is the initial implementation, and the most inefficient RR (q = 500) is either the 8192 bit paging implementation or the Worst Fit implementation. Based on this data, it seems that the initial implementation is the most efficient, seeing as it is the best for SRTF, RR (q = 50), RR (q = 250), and RR (q = 500). It also seems that the most inefficient implementation is the 8192 bit paging implementation, seeing as it is the worst for FCFS, SRTF, and RR (q = 50, 250, and 500).

**References:**

* Class Powerpoints on Canvas
* Class Textbook - Operating System Concepts Essentials