Samuel George
Comp 530H
November 1st, 2017

Practicality of Implementing a Weighted Round Robin Scheduler

Introduction:

In homework 5 I built a user weighted round robin scheduler (uswr) kernel module that allows real-time processes to alter the time slice available to them in a weighted manner. Real-time processes are generally run in a cyclic manner with each one receiving a system set slice of time. My system allows a process to call my module with an integer between 1 and 20 and the number of milliseconds of compute time that was allocated to that process was ten times that integer. I expect that if a process is given a higher user weight that it would be able to complete iterative tasks at a quantity proportionate to its weight. I would also expect that the amount of time required to complete a set task would also be proportionate to the user weight assigned.

Methods:

I tested my kernel module in two distinct manners. First, I wrote a program that waited on a queue until it was released. Upon release it would encrypt a password as many times as it could during a thirty second interval. It then prints the number of iterations that it achieved in the allotted time. Multiple processes performing this task were started with user weighted time weights of 1,4,8,12,16, and 20. All six tasks were released and at the end of thirty seconds each weighted task would print the number of iterations it achieved. Performance of each was evaluated based on the processes number of iterations. The second test was a program that would attempt to encrypt a password 2 million times and print how long it took to complete this task. Again, six tasks with this objective and same weights were queued, released, and each printed how long they took to complete the task. Additionally, the kernel module prints when a processes time slice starts and ends. These data were collected and a graph was generated to illustrate how much time each task was given based on their weight. The first program was used to perform this experiment.

Results:

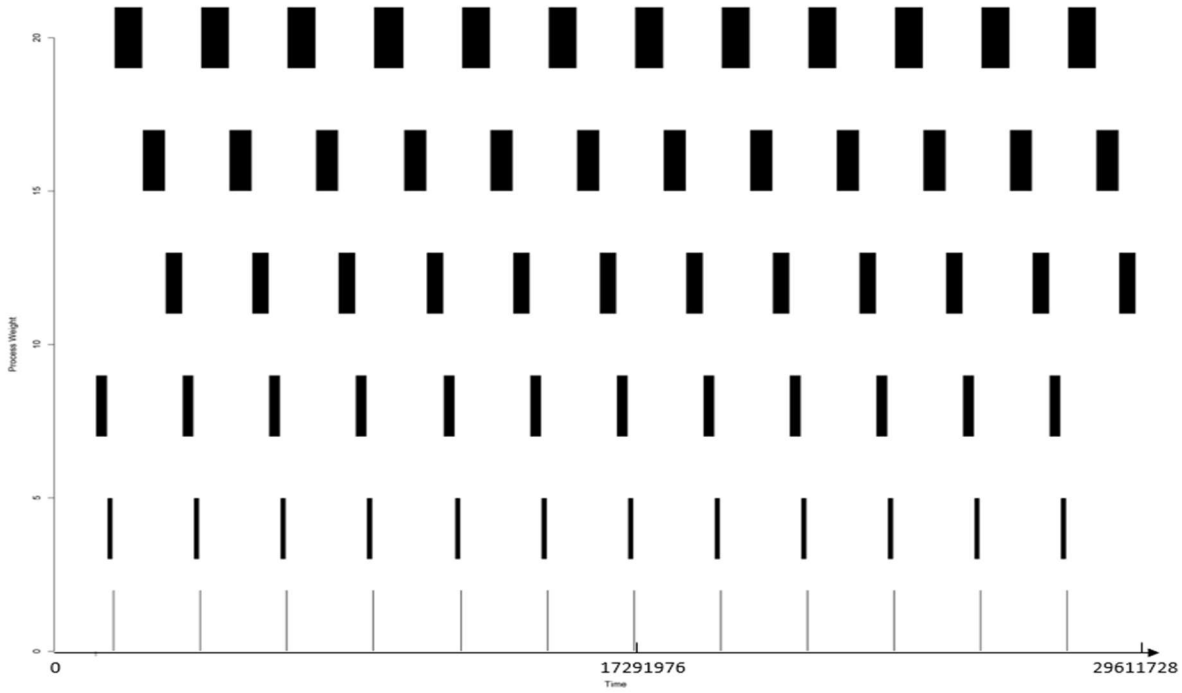
The Chart showing iterations achieved in test one is given below:

Iterations by Weight							
Run/Weight	1	4	8	12	16	20	Total
1	120155	517328	1014312	1299948	1846212	2502381	7300336
2	115510	506906	890586	1282486	1808874	2308055	6912417
3	138715	517703	962772	1392163	2017630	2410994	7439977
4	14022	520853	1177688	1383832	1999174	2526274	7621843
5	136944	554558	1147610	359023	2329453	2549890	7077478
6	118417	376459	774058	1390760	1910841	2125927	6696462
7	137924	431058	863322	1274092	1938857	2133187	6778440
8	137663	476740	741709	1386878	1861634	2311162	6915786
9	147690	478644	296623	1314219	1877404	2431041	6545621
10	120994	479778	926127	1524506	2129972	2325649	7507026
Average	118803.4	486002.7	879480.7	1260791	1972005	2362456	7079539
Standard Deviation	38410.12	51137.01	249068.5	325397.9	157744.2	150577.5	370094.8
Average Fold Iterations	1	4.090815	7.402824	10.61241	16.59889	19.88542	N/A

The highest weighted process achieved the highest average iterations of 2362456. The number of average total iterations performed by all processes was 7079539. The fold increase in iterations complete by the weight 4, 8, 12, 16, and 20 processes were 4.09, 7.40, 10.61, 16.59, and 19.88-fold higher than the number of iterations completed by the weight 1 process respectively. Results for the time required to encrypt 2 million passwords given different weights is shown below:

2 Million Command Time						
Run/Weight	1	4	8	12	16	20
1	49	43	43	43	14	43
2	52	44	38	34	28	25
3	51	45	39	34	30	25
4	48	39	32	27	21	43
5	59	52	45	38	34	32
6	48	42	37	33	27	23
7	54	46	41	36	33	27
8	48	38	31	25	42	18
9	50	44	39	35	30	22
10	51	44	40	33	31	23
Average	51	43.7	38.5	33.8	29	28.1
Standard Deviation	3.431877	3.860052	4.377975	5.09466	7.527727	8.633912
Average Time over Longest Slice	1.814947	1.55516	1.370107	1.202847	1.032028	1

The highest weighted task finished in 28.1 seconds. Tasks given weights 16, 12, 8, 4, and 1 took 1.03, 1.20, 1.37, 1.55, and 1.814 times longer to complete the task respectively. In this case total time is simply given by the lowest weight task. Finally, a chart showing execution time over a set interval is given below:



Numbers on the X axis are time after 0. The Width of bars indicates runtime. For clarity simple divisions of real time are given. Raw data will be uploaded online. Here we see that processes with greater weight are indeed given greater runtime and for this experiment the process with weight 8 ran first.

Discussion/Conclusion:

The program that counted iterations based on weight showed a very nice linear correlation between the weight given to a process and the number of iterations it was able to complete. Below is a chart showing the relative iterations completed based on weight:

	1	4	8	12	16	20
Fold Iterations over 1 Achieved	1	4.090815	7.402824	10.61241	16.59889	19.88542

The chart shows that weight correlates almost perfectly with the fold number of iterations achieved. Therefore, if you have a task that is going to run continuously then using uswr is a great way to control which processes achieve the most calculations. The experiments showed relatively low standard deviation and all runs showed this same pattern. Overall, this section of the experiment showed excellent reproducibility and illustrated a clear pattern.

Interestingly when we look at the time required to finish a specific task, this does not correlate with weight in a linear way. Below is a chart showing weight vs fold time over the weight 20 process required to complete the task:

	1	4	8	12	16	20
Fold Time Taken to do 2 Million Encrypts	1.814947	1.55516	1.370107	1.202847	1.032028	1

Here we see that even giving a process a 20-fold increase in weight did not result in more than a 2-fold decrease in the amount of time required to complete the tasks when run simultaneously. Looking at the full experiment table, we see some strange outliers such as run 4. During run 4, the highest weight process took 43 seconds, while the processes with weights 4-16 all finished before the highest weight process. The standard deviations were also very high for the experimental values. Most of this is due to the heavy effect that order of execution has on the completion of a task. For example. If the weight 20 gets to run last, then it has to wait for all the other processes to run before it even starts. In addition, if the weight 20 process doesn't finish in its first run then it is possible for a lower weight process to actually get enough cpu time to complete before the one with weight 20. Imagine a scenario where it takes 22 time slices to complete a given task. If a process with weight 12, then 16, then 20 runs, the 12 and 16 will both finish before the weight 20 process because they will each get to go twice before the weight 20 process does. Overall, this section of the experiment showed that weight does not perfectly correlate with completion time of a finite task, instead order of tasks run has a much larger impact.

Lastly, looking at CPU execution time we can see that the overhead of switching between tasks in a real-time queue is miniscule and almost all of the execution time went to the real-time processes in question. We also see that the amount of time allocated is very consistent and processes have very little variation in the length of time they actually receive based on their time slice.

Overall, the experiments showed that weighted round robin is an excellent predictor of how much work will be complete in a “infinite” amount of work is to be done. A process like a prime number calculator for example running in parallel with a video decoder will make progress on these tasks in a way that correlates with their given weights and the amount of time they are given to execute. In contrasts, tasks that are finite such as encrypting 2 million passwords do not perform in direct correlation with their time slice. Other factors, such as scheduling and order, are far more important to the efficiency of their completion. The experiments also showed that the overhead from switching between real time tasks is minimal and the overhead can basically be factored out of the overall performance. In conclusion, the choice to implement a real time round robin scheduler with variable time slices as a method to improve performance should be made based on the type of task that’s performance and predictability you are trying to optimize.

Honor Pledge:

I have neither given nor received unauthorized aid on this assignment.

Samuel George