Aayush Patel - CS20BTECH11001
CS3510 - Operating System 2
13 February 2022

OS Programming Assignment 3

Explanation of Algorithm Implementation

1. Two entities are used for the implementation(Event and Process). Each process will be represented by a Process object. Since that process can execute repeat number of times, each such repeat of the process will be represented by an Event object.
2. Line 17 lets you set the context switch time.
3. Line 19 defines a global variable is_rms which can be true or false. It is set to true if command line passes rms flag otherwise false for edf flag.
4. Execute the instructions as given in the readme file.
5. From line 75-86 file reading happens that initializes the Process and Event objects.
6. Line 88 converts the events vector into a events heap. The first two args passes to make_heap() are the iterators of the vector and the third parameter is a functor which indicates that a *min-heap* will be created and sorting will be on the arrival times attribute.
7. Line 89 defines a prev_executing Event pointer which keeps track of the Event that was in the CPU at 1 unit before the current time.
8. Line 90 defines a ready vector that keeps track of processes in the ready queue(processes that have arrived but are not executing).
9. At line 93, the simulation of time begins.
10. At line 95, the while loop will push all the newly arrived events in the ready heap. The ready heap is a max heap and uses the priority attribute for arranging. If is_rms is false, priority will be 1/deadline, if true, priority be 1/period.
11. Then the loop iteraters over all the possible cases: new process, preemption, deadline, etc. as mentioned in the comments.
12. Whenever a new process is added, preempted or a process leaves the system, the variable-i is incremented by the CONTEXT_SWITCH_TIME macro defined in line17.
13. Next all the statistics are calculated and waiting times are averaged and the stats are written to the file.

14. Run the file in the Simulation/simulator.py to get the plots that will be shown below.

Sample Input should look like:

3

1,20,80,50

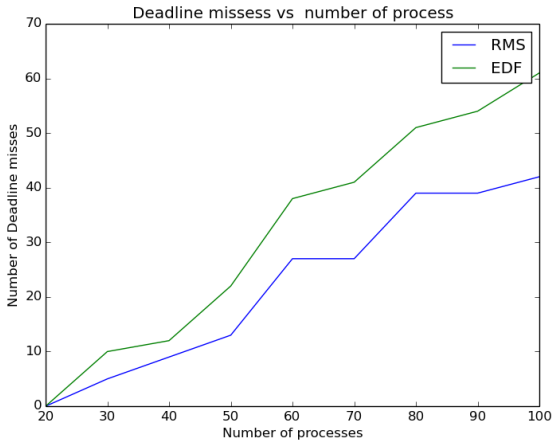
2,30,50,40

3,25,60,50

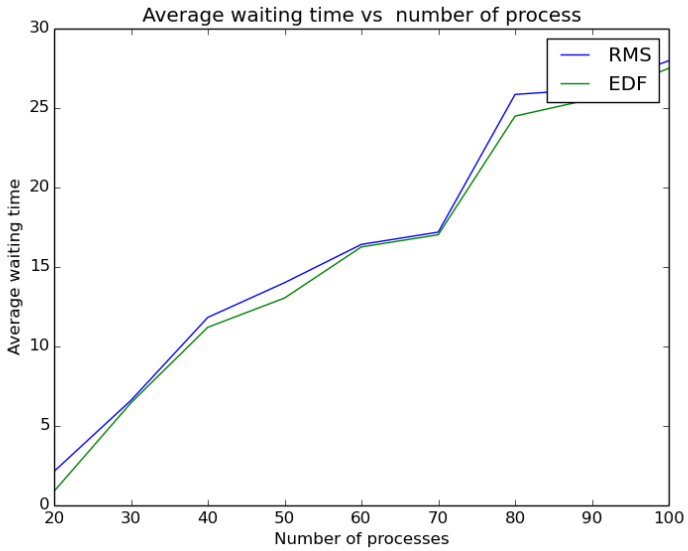
Complications During Implementation

1. The hardest part was the implementation of the for loop for time. To check for deadline miss or completion, the cpu_time parameter had to be compared. During the implementation in some cases, there were cases when the cpu_time was off by a factor of 1. This cascaded to the following events and there were unexpected outcomes.
2. It also took time to come up with all the exhaustive list of cases and modularly arranging them in the if-else-if condition and in the right order. Once that was complete, the execution happened smoothly.
3. Also, Segementation fault occured in many cases and to resolve that, condtions such as prev_executing != nullptr, events.size() != 0 and ready.size() != 0 had to checked at the appropriate places.

Analysis



The given figure shows that as the number of processes increases and system becomes more loaded with processes, number of deadline missess increases in both algorithms which is fairly obvious. It also shows that for the given set of process taken as sample by me, EDF has more deadline missess than RMS. Since the CPU utilisation > 1, EDF and RMS had high chances of missing deadlines. EDF missing more deadlines states that is cases where CPU utilisation is > 1, RMS would be a better algorithm.



The waiting times for both algorithms are approximately same. RMS has a slighly higher waiting time than EDF.

Since the two plots are from the same sample data, we can provide a conclusion that when cpu utilisation is > 1, then there is a tradeoff between waiting time and deadline missess in RMS and EDF. The former has less deadline missess and vice-versa.

*****