CMPE 322: Project2- To Schedule or Not To Schedule?

Ahmet Fırat Gamsız 2020400180

In this project I implemented a preemptive priority scheduler that breaks ties with round robin. My implementation uses enumerators to hold process types and names. Also, I used structs for instructions (which have string name and integer duration) and processes (which have enum name, int priority, arrival time, absolute arrival time, enum type, instruction, number of instructions, last executed instruction, quantum threshold, quantum count, cpu time, waiting time and turnaround time).

Firstly, I read the input files, create the appropriate structs and put the variables in their corresponding spaces. This is lengthy but standard input handling however there is a point needs to be mentioned: last executed instructions in the processes are initialized to -1, because this number incremented just before the next instruction fetched. Also, quantum count holds how many quantum process executed (for promotions) and cpu time holds how much time process spend in the cpu without getting preempted (to detect whether process passed quantum threshold).

Then the most important part is executed, the bubble sort function. This function constitutes the priority queue. It is simple bubble sort with custom comparator run on the processes array. Custom comparator have many criteria (given in decreasing priority): If a process finished it is pushed directly to the end. If process hasn't arrived yet it is pushed to back (but not back of finished processes because that criterion has higher priority). A platinum process always has the priority. If both or none are platinum, then priority field is compared. The one with the higher number is selected. Ties are broken with arrival time; smaller arrival time has the priority. Also, arrival time used for round robin scheduling. If a process re-enters the queue its arrival time is updated with that time (original arrival time kept at abs arrival time field). This enables scheduler to round robin processes with same priority. If this is a tie too, then name is the absolute tie breaker. Process with smaller index (P1<P2) has the higher priority. So, using these criteria the process list is sorted.

After sorting, the first element in the list is selected. Since a process will always come at time 0, there is no need to check for idling. I select the first process from the list, do context switch (just increase current time) and enter the while loop. This while loop terminates when all the processes complete. This check is done by comparing last executed instructions and number of instructions. In the loop, process is executed. Its last executed instruction incremented, instruction duration is added to its cpu time and duration is returned. Current time is incremented with this return value. If process completes after this execution, turnaround time is calculated with standard formula. For waiting time, burst time is calculated by summing all instruction durations. Then this is subtracted from turnaround time to get waiting time. Also process type is assigned to a non-existing type to prevent any bugs.

If the process is platinum then the process is never preempted. There is no special treatment for this case and just loops continues after process completes. For other types, scheduler checks whether the quantum threshold exceeded or not. If exceeded then it resets cpu time and increases the quantum count. It updates the arrival time to current time for round robin scheduling. If any promotion is available it is done in here. Then this process is saved as current processes (because bubble sort does in-place sorting and I lost the reference to this process) and do bubble sort again. If first process in the sorted hasn't arrived yet, then no other process arrived (property of first criterion) so I idle the cpu. There is no need to redo sorting since other criteria did their job by tie breaking for first criterion. If the new first process is different than previous one then it does context switch. Either same or different, the head of list is assigned as new current process and while loop restarts.

If threshold is not exceeded than the current process is saved, and it does bubble sort first. Here I get the reference of saved process from sorted list because I need to mutate it from the list. Check for idling is done and saved process' name compared with head process' name. If new head is different then, quantum number is increased, cpu time reset and arrival time updated. Promotions and context switch are done. Whether or no names are same, head process is assigned as new current process and while loop restarts.

Loop finishes when all instructions complete, and then average waiting time and average turnaround time calculated. In the description it is said to print with only one decimal place, however in the example integers are not printed with .0. So, following this example I print floats with one decimal and integers with none. After this program execution completes.

One final note is that I commented out all the print statements that I used, instead of deleting them because I pretty much like how they log everything and don't want to lose those lines.

The biggest difficulty I faced in this project was understanding how round robin should work. I first though that we were supposed to round robin all the time and priorities only used for ordering in round robin. I implemented a real queue with relevant structures for one or two hours. Then, I thought some things (that were asked in the forum) should work differently (and I still believe some of them make more sense) and worked on fixing/debugging those points.