OS Scheduler Project

Work Distribution

Name	ID	Coded
Ramez Yousri Adli	19190043	• Scheduler (RR , SRTF , HP , FCFS) • Project Integration • Parser
Zeyad Essam AbdelHakim	191900141	• GUI • Scheduler (FCFS) • Parser
Hassan Ahmed	191900178	• Scheduler (SRTF) • Process Generator

Table Of Contents

Keywords and Abbreviations

Abbreviation	Keyword
First Come First Serve	FCFS
Round Robin	RR
Non-preemptive Highest Priority First	Non-preemptive HPF
Shortest Remaining Time First	SRTF
Central Processing Unit	CPU

1.Project Idea

1.1.Idea description

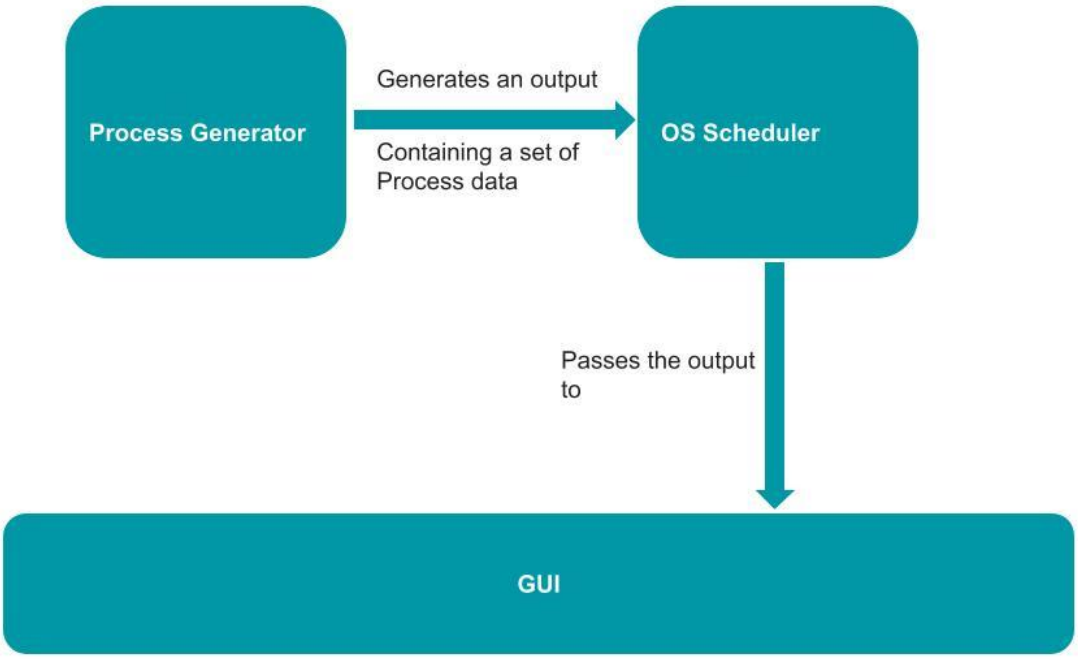
The OS scheduler is one of the most essential parts for an OS to operate. It simply schedules the processes entering the execution section.It selects the processes to execute based on certain criteria and by using a particular algorithm.Every process has certain traits (properties) associated with it in order for the OS scheduler to decide which ones to execute.These properties include but aren't limited to:

- Process ID
- Process Arrival Time : the time at which it was admitted to the ready queue
- Process Burst Time: the time which the process takes to finish its execution. This is also the amount of time the CPU is used by the process.
- Process Priority: the priority is a number given to a process to determine its execution turn (Highest priority first algorithm) , the higher the number the bigger the priority.

The idea is to code a full OS scheduler , which consists of 3 parts:

- Process Generator: takes a normal distribution as an input as well as the number of processes and a poisson distribution and generates a set of processes as output
- OS Scheduler: It has 4 algorithms
    - First Come First Serve (FCFS)
    - Non-preemptive Highest priority first (HPF)
    - Round Robin (RR)
    - Preemptive Shortest Remaining Time First (SRTF)
- GUI: The GUI is the front end of the whole project , it consists of
    - A chart
    - An OpenFileDialog
    - A text field
    - Radio buttons for the 4 algorithms for to make the user able to choose his/her preferred algorithm

1.2.Project Structure and Architecture



Figure(1.2.1) Project Architecture

- Process Generator: Is a stand alone program that generates its output in a text file (.txt) format. This file contains a set of processes with its corresponding data in the following order
    - pID
- OS Scheduler: The generated file is then passed to the first part of the OS Scheduler which is the Parser
    - Parser: it takes the input as a text file and then returns a List of processes
    - OS Scheduler: this set of processes generated by the parser is considered the ready queue which is fed into the scheduler in order to start the CPU scheduling process based on one of the following 4 algorithms
        - FCFS
        - RR
        - SRTF
        - Non-preemptive HPF

2.Project Components

2.1.Process Generator

The process generator is a program which for the scheduler's purpose, is used to turn an input text file that includes: number of processes, mean and standard deviation for the arrival and burst times, and a lambda for priority. The main parts of the process generator are the loader, saver, and the linked list within the main for the output.

In order for the input and output to be in text file format, a saver and a loader classes were used, where the loader is a tuple class as tuple is a data structure that can contain any number of different types of data as long as they don't exceed the coded amount. Moreover, it allows the combination of the various types of data into a single object without the need to create a custom class, which increases the performance and the reliability of the code. Both the loader and saver made use of the "stream" class in order to read and write text files, as they resemble an additional layer between the program and a text file, and executes actions between them smoothly.

With the loader and saver smoothly running, usage of linked lists seemed quite efficient for the output of the program as it links every piece of data to the next using pointers, which is assisted with the ".split" function in the loader which split the string of code into multiple sub-strings (strings are separated by delimiter function), creating a process array, hence ceasing the need of using a parser.

2.2.GUI

GUI created by windows forms. GUI was supported by a side menu to switch among app pages in the current case there is only one page. GUI supports an open file dialog button to display the content of text file and a list of radio buttons to choose among the available algorithms. In the case of RR algo there is a text box to enter the quanta. This text box has no meaning in other algos so it will be disabled. GUI has a chart to display processes and its burst time.

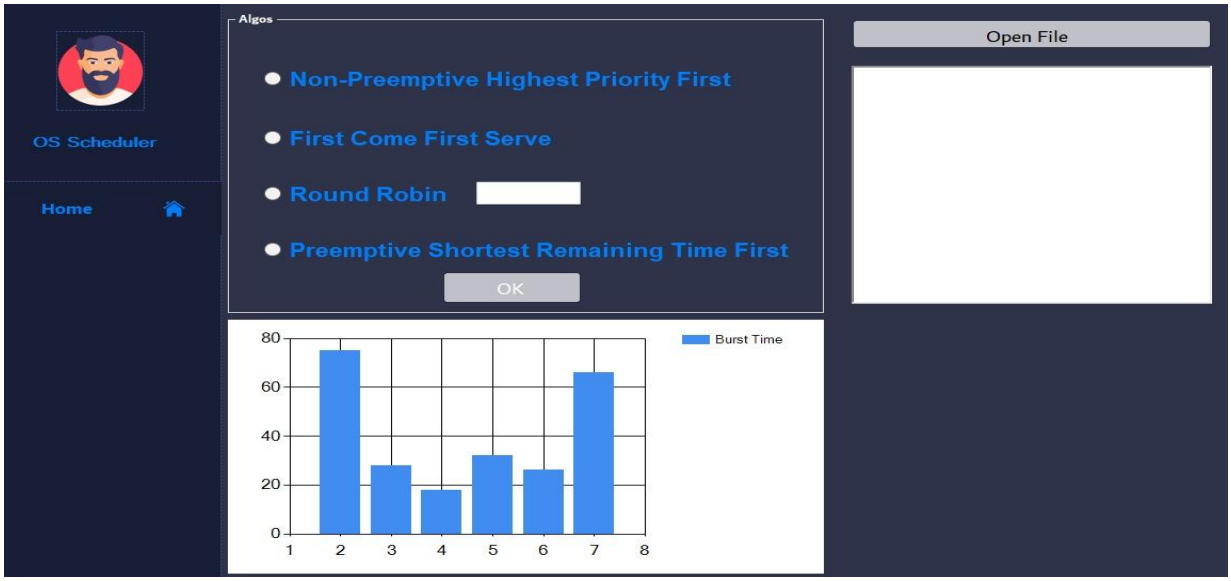


Figure (2.1.1) GUI

3.OS Scheduler

The process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy.
The OS maintains all PCBs in Process Scheduling Queues. The OS maintains a separate queue for each of the process states and PCBs of all processes in the same execution state are placed in the same queue. When the state of a process is changed, its PCB is unlinked from its current queue and moved to its new state queue.
The list of queues:

- Ready Queue:
  - It contains all the processes ready to be executed
- Waiting Queue:
  - It contains all the processes waiting for a certain event to occur
- Finished Queue:
  - A queue that contains all the data about a process that has finished and it is used to make the program able to display the outputs.

This scheduler deals with 4 scheduling algorithms , every algorithm can be preemptive or non-preemptive

3.1.FCFS

The first algorithm is called First Come First Serve.In the design of this algorithm,jobs are executed on first come first serve basis , meaning the ready queue is sorted ascendingly based on their arrival times.The process that has the earliest arrival time (least arrival time) is executed first and so on.Its implementation is based on the FIFO principle.It is easy to understand and easy to implement but its average waiting time is high.

Algorithm

```
1-  Input the processes along with their burst time (bt).
2-  Find waiting time (wt) for all processes.
3-  As first process that comes need not to wait so
    waiting time for process 1 will be 0 i.e. wt[0] = 0.
4-  Find waiting time for all other processes i.e. for
     process i ->
       wt[i] = bt[i-1] + wt[i-1] .
5-  Find turnaround time = waiting_time + burst_time
    for all processes.
6-  Find average waiting time =
```

```
                total_waiting_time / no_of_processes.
7-  Similarly, find average turnaround time =
                total_turn_around_time / no_of_processes.
```

3.2.Non-Preemptive Highest priority first

The second algorithm is called Highest priority first. Its design in this project is non-preemptive , meaning that the process once it starts executing no interrupts can stop its execution for any reason.Jobs are executed based on priority. The priority is a number given to each process in order to determine its importance (the higher the number the higher the priority).The ready queue is sorted descendingly according to every process' priority.If 2 processes have the same priority, they are executed based on FCFS basis.

Algorithm

1. First input the processes with their arrival time, burst time and priority.
2. First process will schedule, which have the lowest arrival time, if two or more processes will have lowest arrival time, then whoever has higher priority will schedule first.
3. Now further processes will be schedule according to the arrival time and priority of the process. (Here we are assuming that lower the priority number having higher priority). If two process priority are same then sort according to process number.

   Note: In the question, They will clearly mention, which number will have higher priority and which number will have lower priority.
4. Once all the processes have been arrived, we can schedule them based on their priority.

3.3.Round Robin

Round Robin is a CPU scheduling algorithm where each process is assigned a fixed time quantum and when that quantum is over the process stops executing till the following finishes.It's easy to implement , simple and starvation free as every process gets an equal share of the CPU.Its main disadvantage is the excess of context switches which causes a system overhead.

Algorithm (From GFG)

```
1- Create an array rem_bt[] to keep track of remaining
   burst time of processes. This array is initially a
   copy of bt[] (burst times array)
2- Create another array wt[] to store waiting times
   of processes. Initialize this array as 0.
3- Initialize time : t = 0
4- Keep traversing the all processes while all processes
   are not done. Do following for i'th process if it is
   not done yet.
    a- If rem_bt[i] > quantum
       (i)  t = t + quantum
       (ii) rem_bt[i] -= quantum;
    c- Else // Last cycle for this process
       (i)  t = t + rem_bt[i];
       (ii) wt[i] = t - bt[i]
       (ii) rem_bt[i] = 0; // This process is over
```

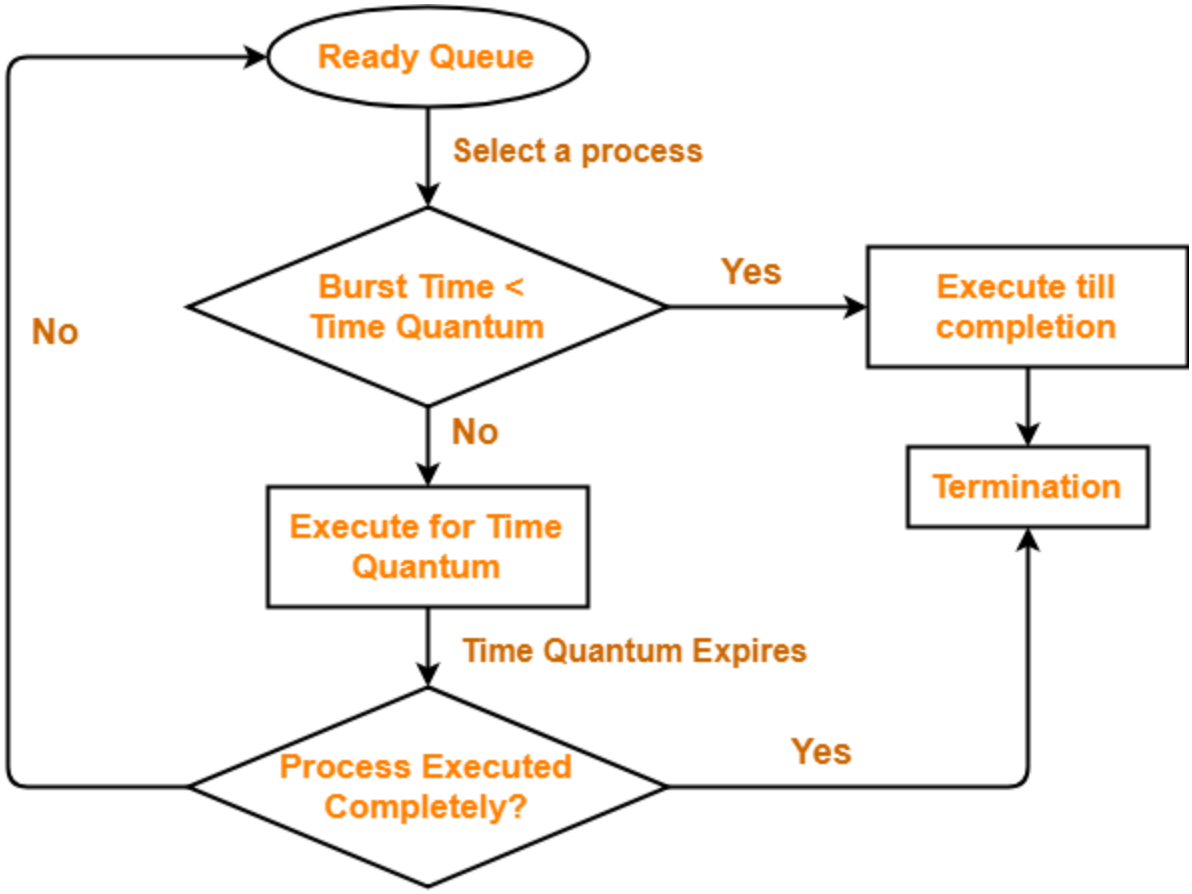


Figure (3.3.1) Round Robin FlowChart

3.4.Shortest Remaining Time First

In the Shortest Remaining Time First (SRTF) scheduling algorithm, the process with the smallest amount of time remaining until completion is selected to execute. if the currently executing process is the one with the shortest amount of time remaining ,that time should only reduce as execution progresses, this process will get executed.Processes will always get executed until they complete or a new process is added that requires a smaller amount of time.

Algorithm(From GFG)

```
1- Traverse until all process gets completely
   executed.
   a) Find process with minimum remaining time at
     every single time lap.
   b) Reduce its time by 1.
   c) Check if its remaining time becomes 0
```

d) Increment the counter of process completion.
e) Completion time of current process =
current_time +1;
e) Calculate waiting time for each completed
process.
wt[i]= Completion time - arrival_time-burst_time
f)Increment time lap by one.
2- Find turnaround time (waiting_time+burst_time).

4.Problems Faced

4.1.Process Generator

The only problem was that it was hard to integrate the process generator and the os scheduler into one program. One executable file

4.2.OS Scheduler

FCFS , RR , HPF

The algorithm for each scheduling technique was kind of hard to implement but with the help of various sources , it was implemented successfully.

SRTF

The SRTF scheduler part was quite troublesome to deal with as a lot of conditions were to be stated in order to get an accurate output, hence I had to work on manually-set data, and took question 5.5 from the past assignment as my example to work on in order to also make sure the SRTF works well on corner cases as well.
The main problems I met with the SRTF were adjusting the operations' conditions in order to get an accurate end-result, and integrating it within the scheduler, as I worked on it as its own separate program, On that note, the SRTF had to be remade in order to fit within the scheduler so a pseudo code was written by me explaining the SRTF from scratch an remade by Ramez, but the new SRTF had issues as It only operated on the initial process only deeming it unusable.

4.3.GUI (By Zeyad)

I faced one problem which is the integration between the open file dialogue button and the rich box and I solved it by seeing videos on YouTube

5.Conclusion

The OS scheduler project is complete but with some problems that faced me and my team.The current project contains the process generator , an OS Scheduler and a GUI.The OS Scheduler contains 4 algorithms , FCFS , RR , HPF , SRTF. It also contains a parser to parse the output generated by the process generator.

The following link contains videos showcasing the programs:
OS proj videos - Google Drive