Faculty of information technology

Computer Systems Engineering Department

Operating Systems

ENCS 339

CPU SCHEDULING

STUDENT NAME:
Lana Abu Hammad **ID:** (#1181219)
Tareq Shannak **ID:** (#1181404)

INSTRUCTOR: Dr. Adnan Yahya

SECTION: 1

DATE: 1/11/2020

Table of Contents

1. Introduction: -

The reason we need CPU Scheduling is that it's the ground on which the concept of multiprogramming is built on. Any process involves both CPU time and I/O time, however, while a process is involved in doing I/O events, the CPU remains idle which in turn decreases the overall CPU utilization. Hence, to increase the utilization of the CPU, CPU scheduling solves this problem by decreasing the time in which the CPU remains idle by using algorithms such as Round Robin, Shortest-remaining time first, Shortest job first, Priority scheduling without preemption and First come first served. Thus, this project aims to use a generated set of processes and implement the various CPU scheduling algorithms whilst providing the user with an interface that makes it as easy as possible to be able to visualize what each algorithm does.

2. Program Implementation: -

To implement this program, we have used Java language and to implement the $GUIs$, we have used $Scenebuilder$ for creating scenes. first, we have created a class named process which has the attributes needed for each scheduling algorithm, and for creating instances of this class. Those attributes include an id for each process, arrival time, CPU burst, finish time, wait time, turnaround time and its priority. In this class we have created methods for sorting processes by their ID, burst and priority. We have also created a Scheduler class that's constituted of methods of each scheduling algorithms. For the implementation of each scheduling algorithm, we have used a ready queue and a CPU queue and since the criteria for each scheduling algorithm differs, the way in which each algorithm chooses which process is to be on the CPU next also differs. For example, SJF sorts the processes on the ready queue on the basis of which has the smallest burst time. Priority sorts them in accordance of the one with the highest priority. $FCFS$ sorts them based on their arrival time and finally, round robin, assigns a process to a CPU for an amount of time called the time quantum which is provided by the user. For SJF scheduling algorithm we implemented priority with aging, we tested two aging algorithm and chose the one who gave us the best CPU utilization. We gave each process a priority in all of the scheduling algorithms implemented, **so if two process arrived at the same time and had the same attributes the process with the lowest priority will be assigned to the CPU, However if they also have the same priority the process with the lowest ID will be assigned to the CPU** ,hence we have solved the notion of tiebreaking by assigning the process with the lower ID the CPU. Also, we've included a method named $checkArrival$ which incessantly checks for the arrival of a new process. As mentioned earlier, we have used $scenebuilder$ to create the scenes for this project and for each scene we have assigned a $home\_page$ class that controls what each scene does and handled each button in such a way that fits the implementation of this project. The main class initialize the scenes that have been created.

Note That

3. How the Program runs: -

When pressing the run icon, the main scene appears which looks like this: -

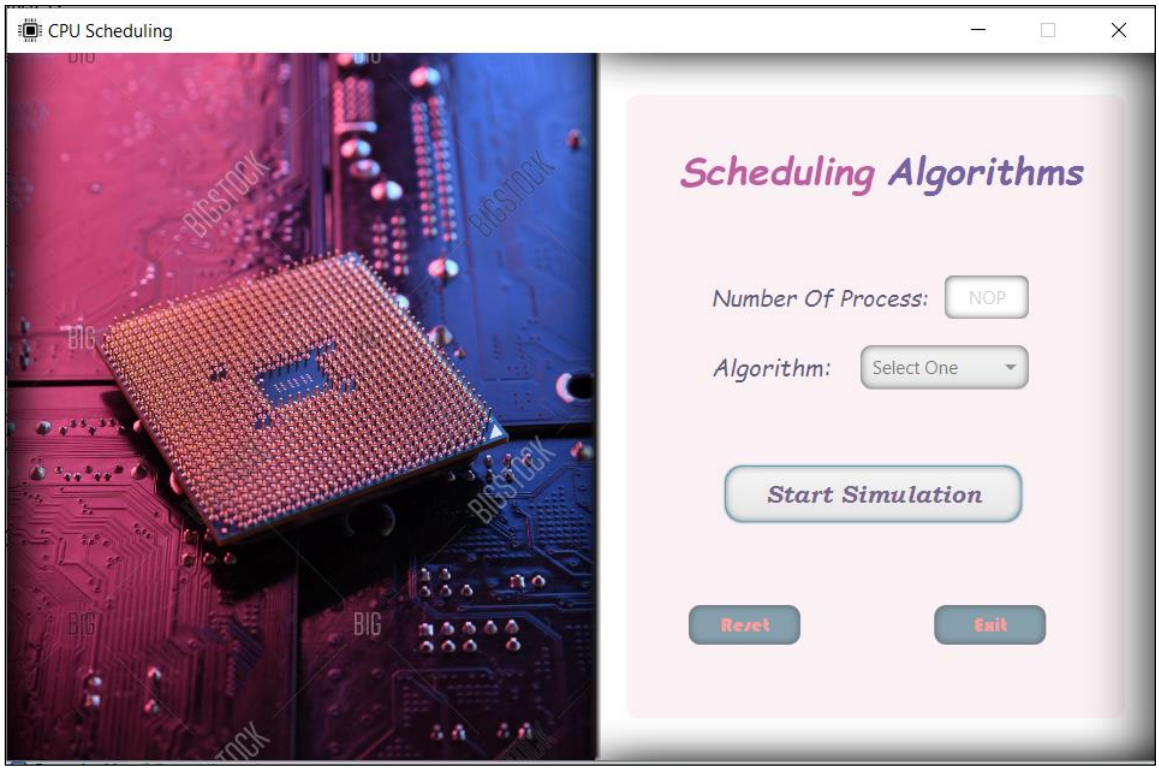

Fig.3.1: main scene

First the user enters the number of processes and chooses one of the scheduling algorithms listed. after pressing the simulation button. The input file will look like this:

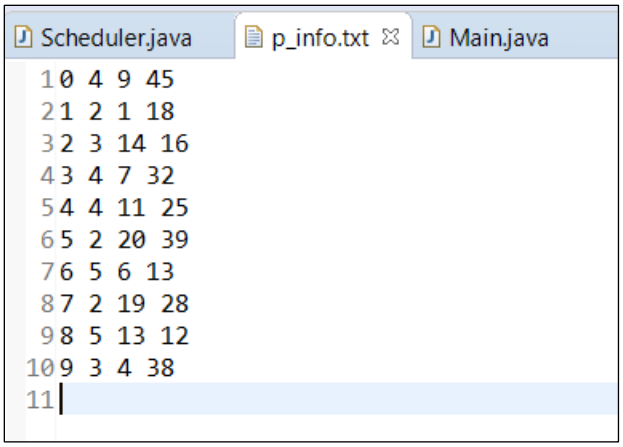

Fig.3.2: input File

After pressing simulation button, a table of the processes generated and their attributes will be shown and the $gantt\_chart$ which is a visual representation of the $CPU\ QUEUE$ will also be shown where a colored rectangle and the process ID will represent each process. Also, the average wait time and turn around will be shown. The scene will look like this after the simulation button is pressed:

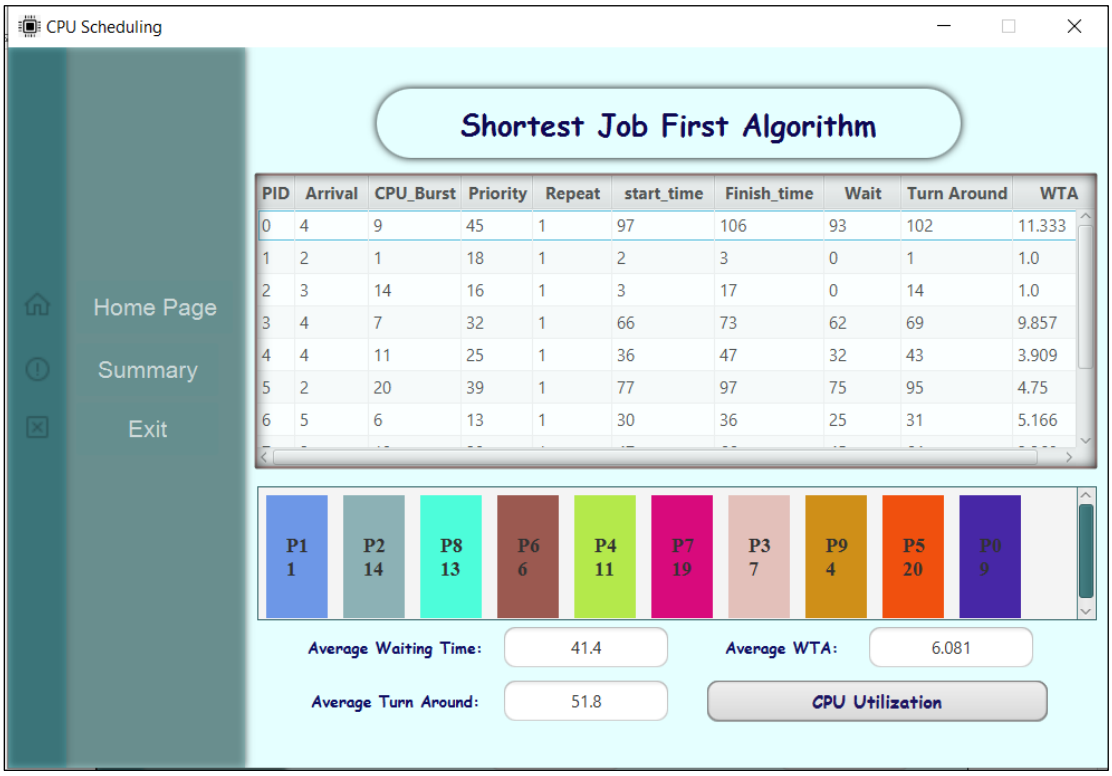

Fig.3.3: View2

In addition to a pie chart that represents the CPU utilization percentage after pressing the cpu utilization button: -

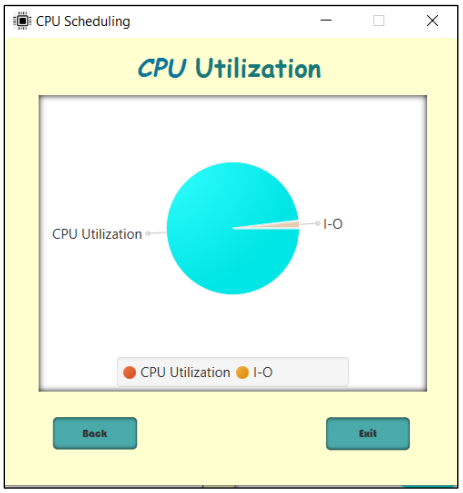

Fig.3.4: View3

4. Assumptions considering aging: -

We tested two aging methods twice and chose the one that gave us the highest CPU usage. Of course, because our system generates random processes the comparison between the three methods is not accurate but the results were as follows. **Priority with aging was implemented to SJF scheduling algorithm:**

4.1 Increase the priority of the process if it has been waiting more than twice time of its remaining burst time: -

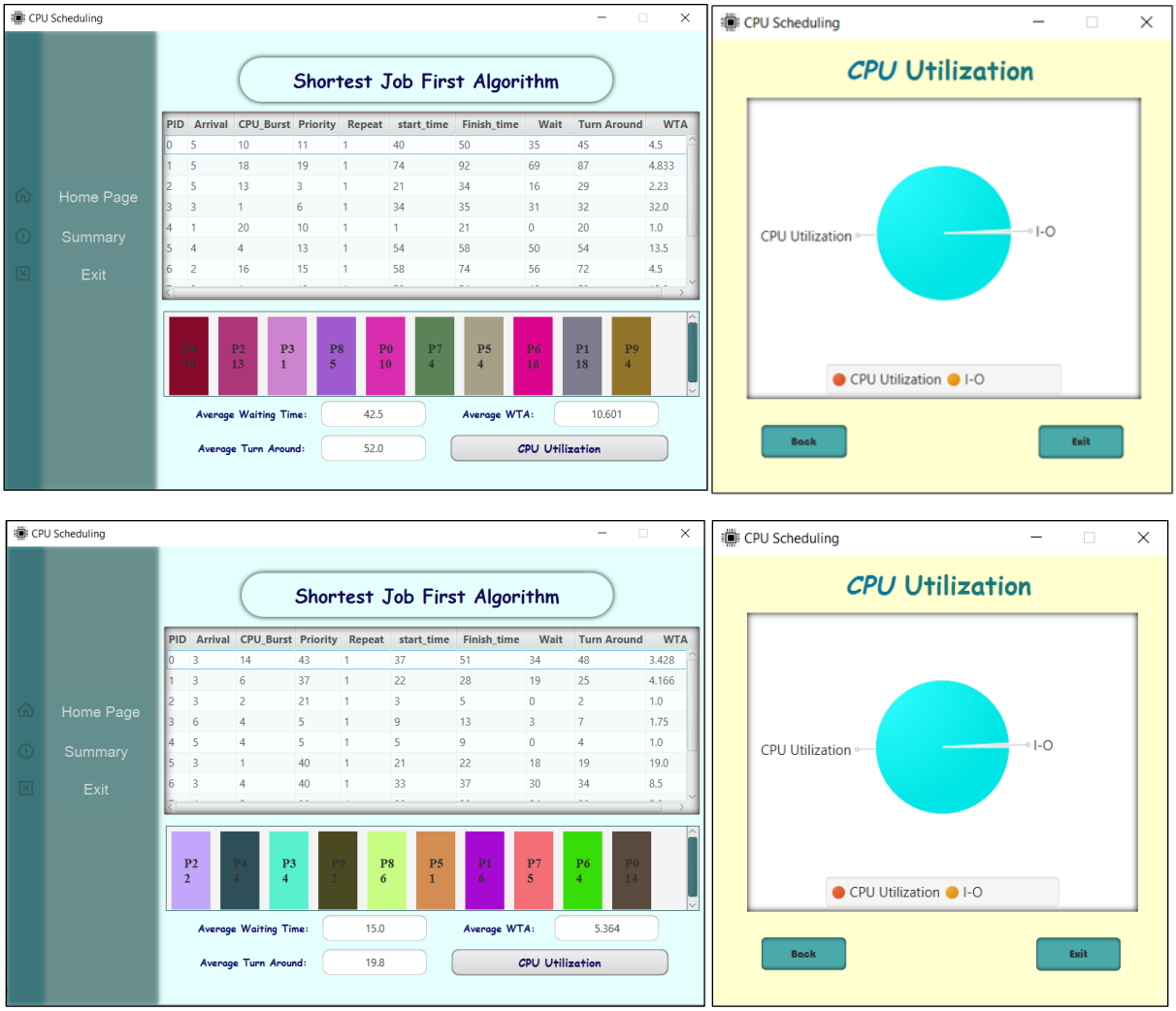

Fig.4.1: CPU Utilization

4.2 Increase the priority of the process if it has been waiting more than difference between the maximum and the minimum waiting time of all processes in the ready queue.

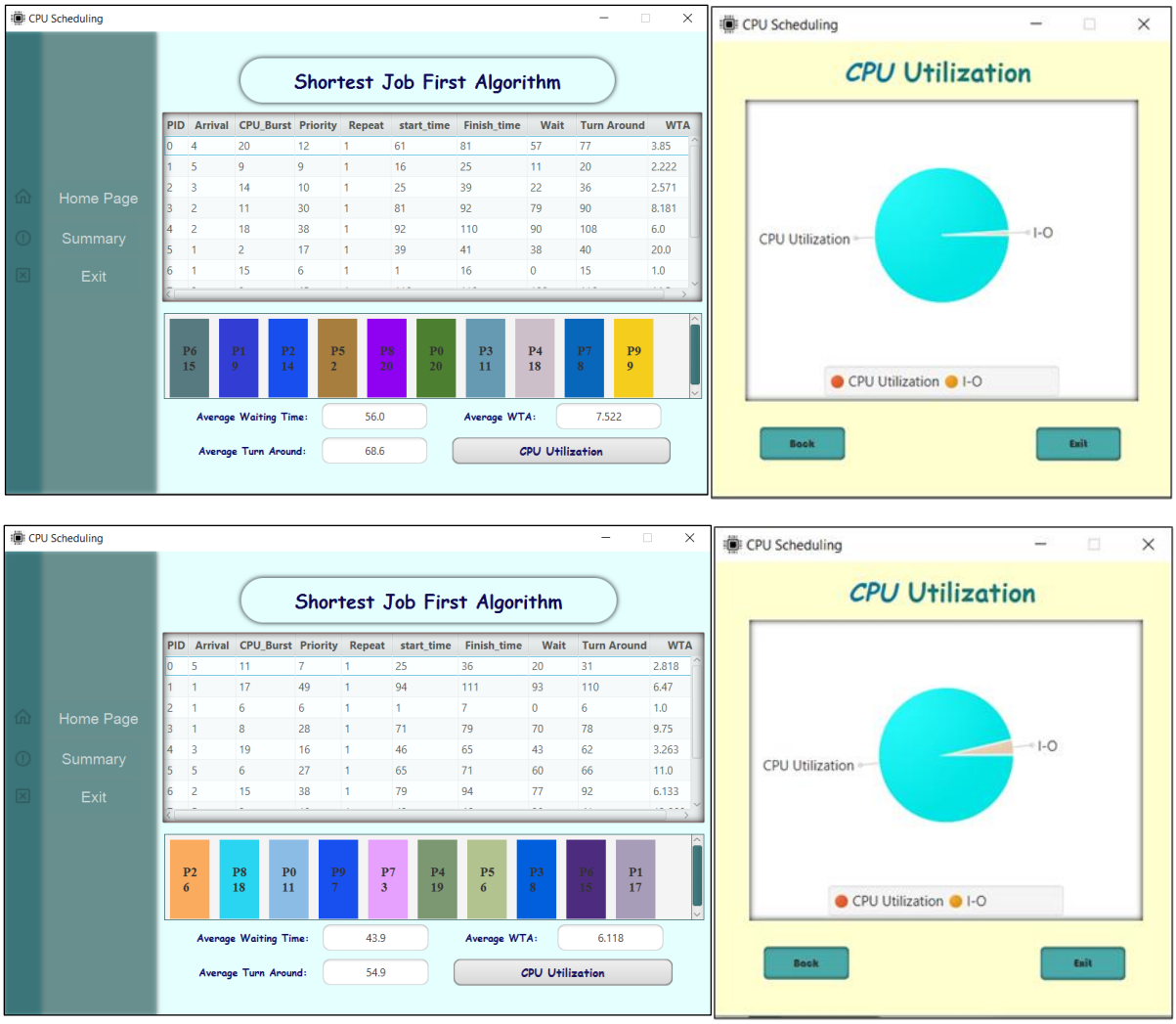

Fig.4.2: CPU Utilization

5. Problems we've faced: -

As it is more than normal when it comes to programming, we have encountered multiple problems but successfully got over them. Few of the problems include:

- Since it was our first time working with $Scenebuilder$, we weren't at first as flexible as we are now in working with it.

- We wanted to find the best method for aging, and we did some research online. It wasn't very helpful so we tested several methods as previously shown and chose the one that gave us the best CPU usage for most test cases.

- We've faced a problem with incorporating threads in our project, but upon research, we eliminated this problem.

6. Conclusion: -

In conclusion, CPU scheduling is a process which allows one process to use the CPU while the execution of another process is on hold (in waiting state) due to unavailability of any resource like I/O, thereby making full use of CPU. The aim of CPU scheduling is to make the system efficient, fast and fair. one can see how each scheduling algorithm produces a different waiting time and different turnaround time and therefore the criterion for choosing which algorithms suits best requires studying the situation and see which one approximates what we want. Through working on this project, we've learned from the challenges we faced and the problems we've encountered.