**PROJECT REPORT (CSF204)**

*A report submitted in partial fulfilment of the requirement for the course*

**OPERATING SYSTEM**

Part of the degree of

BACHELOR OF TECHNOLOGY

**In**

**CSE/IT/EE/CE/PE**

A picture containing drawing

Description automatically generated

**Submitted to *:***

**Mr Atul Kumar Srivastava**

**Assistant Professor**

**Submitted by:**

**Name- SATYAM RAJ**

**Roll No : 200102581**

**SAP ID: 1000015607**

**Section: I**

**SCHOOL OF COMPUTING**

**DIT UNIVERSITY, DEHRADUN**

(State Private University through State Legislature Act No. 10 of 2013 of Uttarakhand and approved by UGC)

**Mussoorie Diversion Road, Dehradun, Uttarakhand - 248009, India.**

**2021**

**CANDIDATES DECLARATION**

I hereby certify that the work, which is being presented in the Report, entitled Scheduling Algorithm Simulation, in partial fulfilment of the requirement as part of the course Data Structures of the Degree of **Bachelor of Technology** and submitted to the DIT University is an authentic record of my work carried out during the period ***15/04/2022***to ***17/04/2022***under the guidance of **Mr Atul Kumar Srivastava.**



**Date: 17/04/2022 Signature of the Candidate**

**TABLE OF CONTENT**

CHAPTER PAGE No.

1. What is Scheduling Algorithm?
2. Types of Scheduling Algorithm.

* 1. First Come First Serve (FCFS)
  2. Shortest Job First (Pre-emptive)
  3. Shortest Job First (Non- Pre-emptive)

1. Introduction to Project
2. Project Description
3. Methods and Implementation
4. Data Structure used.
5. Source Code
6. Snapshots
7. Result Analysis
8. Conclusion & Bibilography

**Chapter 1**

**What is Scheduling Algorithm?**

There are various algorithms which are used by the Operating System to schedule the processes on the processor in an efficient way.

The Purpose of a Scheduling algorithm

1. Maximum CPU utilization
2. Fare allocation of CPU
3. Maximum throughput
4. Minimum turnaround time
5. Minimum waiting time
6. Minimum response time

There are the following algorithms which can be used to schedule the jobs.

1. First Come First Serve

It is the simplest algorithm to implement. The process with the minimal arrival time will get the CPU first. The lesser the arrival time, the sooner will the process gets the CPU. It is the non-preemptive type of scheduling.

2. Round Robin

In the Round Robin scheduling algorithm, the OS defines a time quantum (slice). All the processes will get executed in the cyclic way. Each of the process will get the CPU for a small amount of time (called time quantum) and then get back to the ready queue to wait for its next turn. It is a preemptive type of scheduling.

3. Shortest Job First

The job with the shortest burst time will get the CPU first. The lesser the burst time, the sooner will the process get the CPU. It is the non-preemptive type of scheduling.

4. Shortest remaining time first

It is the preemptive form of SJF. In this algorithm, the OS schedules the Job according to the remaining time of the execution.

5. Priority based scheduling

In this algorithm, the priority will be assigned to each of the processes. The higher the priority, the sooner will the process get the CPU. If the priority of the two processes is same then they will be scheduled according to their arrival time.

6. Highest Response Ratio Next

In this scheduling Algorithm, the process with highest response ratio will be scheduled next. This reduces the starvation in the system.

**Chapter 2**

**Types of Scheduling Algorithm**

1. **First Come First Serve (FCFS)**

**First come first serve** (FCFS) scheduling algorithm simply schedules the jobs according to their arrival time. The job which comes first in the ready queue will get the CPU first. The lesser the arrival time of the job, the sooner will the job get the CPU. FCFS scheduling may cause the problem of starvation if the burst time of the first process is the longest among all the jobs.

Advantages of FCFS

* Simple
* Easy
* First come, First serve

Disadvantages of FCFS

1. The scheduling method is non preemptive, the process will run to the completion.
2. Due to the non-preemptive nature of the algorithm, the problem of starvation may occur.
3. Although it is easy to implement, but it is poor in performance since the average waiting time is higher as compare to other scheduling algorithms.
4. **Shortest Job First (Pre-emptive)**

This Algorithm is the **preemptive version** of **SJF scheduling**. In SRTF, the execution of the process can be stopped after certain amount of time. At the arrival of every process, the short term scheduler schedules the process with the least remaining burst time among the list of available processes and the running process.

Once all the processes are available in the **ready queue**, No preemption will be done and the algorithm will work as **SJF scheduling**. The context of the process is saved in the **Process Control Block** when the process is removed from the execution and the next process is scheduled. This PCB is accessed on the **next execution** of this process.

1. **Shortest Job First (Non Pre-emptive)**

Till now, we were scheduling the processes according to their arrival time (in FCFS scheduling). However, SJF scheduling algorithm, schedules the processes according to their burst time.

In SJF scheduling, the process with the lowest burst time, among the list of available processes in the ready queue, is going to be scheduled next.

However, it is very difficult to predict the burst time needed for a process hence this algorithm is very difficult to implement in the system.

## Advantages of SJF

1. Maximum throughput
2. Minimum average waiting and turnaround time

## Disadvantages of SJF

1. May suffer with the problem of starvation
2. It is not implementable because the exact Burst time for a process can't be known in advance.

There are different techniques available by which, the CPU burst time of the process can be determined. We will discuss them later in detail.

**Introduction**

The purpose of this project is to outline the Scheduling algorithms of operating system which helps in the scheduling the task and jobs inside the system that which must be executed first, and which jobs has to wait for other jobs to finish.

The name of this project is the OS Scheduling algorithm simulation because it simulates the three one of the best scheduling algorithms FCFS, SJF-P and SJF-NP.

This project will take the processes their arrival and burst time and show their simulation of scheduling inside the operating system.

This project will help to understand the scheduling of processes in the best way.

This project is developed by using the Java Language in the IntelliJ idea code editor using JSwing for making interactive GUI’s in WINDOWS platform.

**Project Description**

**Purpose:** The purpose of this project is to outline the Scheduling algorithms of operating system which helps in the scheduling the task and jobs inside the system that which must be executed first, and which jobs has to wait for other jobs to finish.

The name of this project is the OS Scheduling algorithm simulation because it simulates the three one of the best scheduling algorithms FCFS, SJF-P and SJF-NP.

This project will take the processes their arrival and burst time and show their simulation of scheduling inside the operating system.

This project will help to understand the scheduling of processes in the best way.

This project is developed by using the Java Language in the IntelliJ idea code editor using JSwing for making interactive GUI’s in WINDOWS platform.

**Problem statement:** Write a GUI based application which does the following:

* 1. Receive arrival and burst time of different processes from the user.
  2. After pressing the solve button the Gantt chart and the complete table would be printed based on the selected CPU scheduling algorithm.
  3. You have to implement FCFS, SJF-Pre-emptive and SJF Non Pre-emptive scheduling algorithm.

**System Requirements:**

* **Minimum Intel i3 processor**
* **Code editor to make GUI**
* **JSwing packages**
* **JDK for making the project.**

**Methods and Implementation**

**Classes:**

1. **Main**

This class includes only main function which is used to just start the implementation of the whole project.

1. **InputFrame**

This class consist of a constructor which is used to make the frame for inputting the arrival and burst time of the different processes.

This will contains making:

1. Input heading using JLabel
2. Algorithm heading using JLabel
3. Combo box for algorithm using JComboBox
4. Arrival time heading using JLabel
5. Text Field to input the arrival times using JTextField
6. Burst Time heading using JLabel
7. Text Field to input the burst time using JTextField
8. **FinalFrame**

This class is used to check the index of the combo box button and call the different class constructor to give appropriate scheduling.

1. **FCFS**

This class is responsible to give output of the FCFS scheduling algorithm.

This class consist of:

1. Frame to show the gui and add component in it using JFrame.
2. Output heading using JLabel
3. FCFS heading at the top right to see which scheduling is going on using JLabel.
4. Gant Chart Heading using the JLabel.
5. Buttons to show the gantt chart using JButton
6. Labels to show the start and end of a particular process below the gantt char using JLabel.
7. Tables to show the desired output consisting of:
   1. JOB
   2. Arrival Time
   3. Burst Time
   4. Finish Time
   5. Waiting Time
   6. Turn Around Time
   7. Average waiting and turn around time.
8. **SJFP**

This class is responsible to give output of the pre–emptive SJF scheduling algorithm.

This class consist of:

1. Frame to show the gui and add component in it using JFrame.
2. Output heading using JLabel
3. FCFS heading at the top right to see which scheduling is going on using JLabel.
4. Gant Chart Heading using the JLabel.
5. Buttons to show the gantt chart using JButton
6. Labels to show the start and end of a particular process below the gantt char using JLabel.
7. Tables to show the desired output consisting of:

1. JOB

2. Arrival Time

3. Burst Time

4. Finish Time

5. Waiting Time

6. Turn Around Time

7. Average waiting and turn around time.

1. **SJFNP**

This class is responsible to give output of the non pre-emptive SJF scheduling algorithm.

This class consist of:

* 1. Frame to show the gui and add component in it using JFrame.
  2. Output heading using JLabel

1. FCFS heading at the top right to see which scheduling is going on using JLabel.
2. Gant Chart Heading using the JLabel.
3. Buttons to show the gantt chart using JButton
4. Labels to show the start and end of a particular process below the gantt char using JLabel.
5. Tables to show the desired output consisting of:

1. JOB

2. Arrival Time

3. Burst Time

4. Finish Time

5. Waiting Time

6. Turn Around Time

7. Average waiting and turnaround time.

**……… Source Code …..**

**Main Class**

**package OS\_Mini\_Project;  
  
public class Main {  
 public static void main(String[] args) {  
 InputFrame frame = new InputFrame();  
 }  
}**

InputFrame CLASS

**package OS\_Mini\_Project;  
  
  
import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.\*;  
  
public class InputFrame implements ActionListener {  
 JFrame frame;  
 JComboBox<String> comboBox;  
 JTextField arrival;  
 JTextField burst;  
 JButton solve;  
  
 InputFrame() {  
*// JFrame -> Satyam's Frame* frame = new JFrame();  
 frame.setTitle("OS Mini Project- Satyam");  
 frame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 frame.setSize(400, 450);  
  
 String[] algorithm = {"First Come First Serve (FCFS)", "Shortest Job First Preemptive(SJF-P)", "Shortest Job First Non Preemptive (SJF-NP)"};  
 comboBox = new JComboBox<>(algorithm);  
  
 frame.getContentPane().setBackground(new Color(230, 230, 230));  
 frame.setLayout(null);  
 JLabel input = new JLabel();  
 input.setText("Input");  
 input.setForeground(Color.*BLACK*);  
 input.setBounds(30, 10, 100, 50);  
 input.setFont(new Font(null, Font.*BOLD*, 40));  
 JLabel algo = new JLabel("Algorithm");  
 algo.setBounds(25, 70, 90, 30);  
 algo.setFont(new Font(null, Font.*CENTER\_BASELINE*, 15));  
 comboBox.setBounds(20, 100, 350, 40);  
 comboBox.setFont(new Font("Sans Serif", Font.*PLAIN*, 15));  
*// Now adding the label Arrival Time* JLabel at = new JLabel("Arrival Time");  
 at.setBounds(25, 150, 90, 30);  
 at.setFont(new Font(null, Font.*CENTER\_BASELINE*, 15));  
 JLabel bt = new JLabel("Burst Time");  
 bt.setBounds(25, 230, 90, 30);  
 bt.setFont(new Font(null, Font.*CENTER\_BASELINE*, 15));  
 comboBox.setBackground(new Color(215, 245, 255));  
 comboBox.setForeground(Color.*BLACK*);  
 comboBox.addActionListener(this);  
*// TextFeild for arrival time* arrival = new JTextField();  
*// arrival.setFocusable(false);* arrival.setForeground(Color.*black*);  
 arrival.setBackground(new Color(215, 245, 255));  
 arrival.setFont(new Font("Sans Serif", Font.*PLAIN*, 15));  
 arrival.setBounds(20, 180, 350, 40);  
 burst = new JTextField();  
*// arrival.setFocusable(false);* burst.setForeground(Color.*black*);  
 burst.setBackground(new Color(215, 245, 255));  
 burst.setFont(new Font("Sans Serif", Font.*PLAIN*, 15));  
 burst.setBounds(20, 260, 350, 40);  
*// adding the submit button* solve = new JButton("Solve");  
 solve.setFont(new Font("Sans Serif", Font.*PLAIN*, 20));  
 solve.setBounds(20, 320, 90, 40);  
 solve.setBackground(new Color(100, 180, 255));  
 solve.setForeground(Color.*white*);  
 solve.addActionListener(this);  
 frame.add(solve);  
 frame.add(arrival);  
 frame.add(burst);  
 frame.add(algo);  
 frame.add(at);  
 frame.add(bt);  
 frame.add(input);  
 frame.add(comboBox);  
 frame.setVisible(true);  
 }  
  
 *// override* public void actionPerformed(ActionEvent e) {  
 int index = comboBox.getSelectedIndex();  
*// checking for the combo box* if (e.getSource() == solve) {  
 new FinalFrame(index, arrival.getText(), burst.getText());  
 }  
 }**}

**FINAL FRAME CLASS**

**package OS\_Mini\_Project;  
  
 public class FinalFrame {  
 FinalFrame(int index,String arrival,String burst)  
 {  
 if(index == 0)  
 {  
*// converting the arrival string to arrival integer array - SATYAM RAJ* String[] arrivalStringArray = arrival.split(" ");  
 int[] arrivalArray = new int[arrivalStringArray.length];  
 for(int i=0;i<arrivalArray.length;i++)  
 {  
 arrivalArray[i] = Integer.*parseInt*(arrivalStringArray[i]);  
 }  
*// converting the burst string to burst array - jatin bisht* String[] burstStringArray = burst.split(" ");  
 int[] burstArray = new int[burstStringArray.length];  
 for(int i=0;i<burstArray.length;i++)  
 {  
 burstArray[i] = Integer.*parseInt*(burstStringArray[i]);  
 }  
*// Calling fcfs class* new FCFS(arrivalArray,burstArray);  
 }  
 if(index!=0 && index!=1) {*// converting the arrival string to arrival integer array - SATYAM RAJ* String[] arrivalStringArray = arrival.split(" ");  
 int[] arrivalArray = new int[arrivalStringArray.length];  
 for (int i = 0; i < arrivalArray.length; i++) {  
 arrivalArray[i] = Integer.*parseInt*(arrivalStringArray[i]);  
 }  
*// converting the burst string to burst array -SATYAM RAJ* String[] burstStringArray = burst.split(" ");  
 int[] burstArray = new int[burstStringArray.length];  
 for (int i = 0; i < burstArray.length; i++) {  
 burstArray[i] = Integer.*parseInt*(burstStringArray[i]);  
 }  
*// calling SJF-P* new SJFP(arrivalArray,burstArray);  
 }  
 if(index==1)  
 {  
 *// converting the arrival string to arrival integer array - SATYAM RAJ* String[] arrivalStringArray = arrival.split(" ");  
 int[] arrivalArray = new int[arrivalStringArray.length];  
 for(int i=0;i<arrivalArray.length;i++)  
 {  
 arrivalArray[i] = Integer.*parseInt*(arrivalStringArray[i]);  
 }  
*// converting the burst string to burst array - SATYAM RAJ* String[] burstStringArray = burst.split(" ");  
 int[] burstArray = new int[burstStringArray.length];  
 for(int i=0;i<burstArray.length;i++)  
 {  
 burstArray[i] = Integer.*parseInt*(burstStringArray[i]);  
 }  
*// Calling SJFNP class* new SJFNP(arrivalArray,burstArray);  
 }  
 }  
}**

FCFS CLASS

**package OS\_Mini\_Project;  
  
import javax.swing.\*;  
*//import javax.swing.border.\*;*import javax.swing.table.DefaultTableModel;  
import java.awt.\*;  
public class FCFS {  
 JFrame frame;*// used to making the frame``* JLabel output;*//used for the heading of the frame* JLabel fcfs; *// used to tell that which algo is using at the top right side of the frame* JLabel ganttChart;*// used to write the gantChart above the gantt chart diagram  
 // JLabel Average;//used to find the average of tat and wt* FCFS(int[] arrival,int[] burst)  
 {  
*// specify the number of rows and columns in the table and the gantt chart  
// int rowInTable = arrival.length;//row will be same for gantt chart and table* int columnInGanttChart = arrival.length;  
*// int column = 6;  
  
// Making the frame know : SATYAM RAJ* frame = new JFrame();  
 frame.setTitle("First Come First Serve (FCFS)");  
 frame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
*// first grid layout for gantt chart* frame.setLayout(null);  
*// Making panel for the gant chat* JPanel gantChartPanel = new JPanel();  
 gantChartPanel.setBounds(100,100,300,50);  
 gantChartPanel.setLocation(200,100);  
 gantChartPanel.setLayout(new GridLayout(1,columnInGanttChart,0,0));  
 gantChartPanel.setBackground(new Color(215, 245, 255));  
 gantChartPanel.setForeground(Color.*BLACK*);  
*// making buttons in the gantt chart* for(int i=0;i<columnInGanttChart;i++)  
 {  
 char ch = (char)(65+i);  
 JButton button = new JButton(String.*valueOf*(ch));  
 button.setBackground(new Color(215, 245, 255));  
 button.setFont(new Font(null,Font.*BOLD*,20));  
 button.setFocusable(false);  
 gantChartPanel.add(button);  
 }  
*// making the label for the heading output* output = new JLabel("Output");  
 output.setForeground(Color.*BLACK*);  
 output.setBounds(35,10,200,50);  
 output.setFont(new Font(null,Font.*BOLD*,35));  
*// writing gantt chart above the grid layout* ganttChart = new JLabel("Gantt Chart");  
 ganttChart.setBounds(270,65,200,30);  
 ganttChart.setFont(new Font("Courier",Font.*PLAIN*,30));  
*// adding panel for the algorithm* fcfs = new JLabel("FCFS");  
 fcfs.setBounds(635,12,50,30);  
 fcfs.setFont(new Font(null,Font.*BOLD*,15));  
 fcfs.setForeground(new Color(40, 220, 255));  
  
*// ------------------------------------------------------------------------  
// Now making the values for the finish time in the gant chart* int[] finish = finishTime(arrival,burst);  
 for(int i=0;i<finish.length;i++)  
 {  
 String str = Integer.*toString*(finish[i]);  
*// str+=" ";* JLabel ft = new JLabel(str);  
 int start = 190 + (i\*60);  
 ft.setBounds(start,145,300,30);  
 ft.setFont(new Font(null,Font.*PLAIN*,13));  
 frame.add(ft);  
 }  
*// ------------------------------------------------------------------------  
// Making the table format for final table* JTable table = new JTable();  
*// table.setBounds();* table.setFont(new Font(null,Font.*BOLD*,15));  
 table.setBackground(new Color(215, 245, 255));  
 table.setBounds(100,100,700,480);  
 table.setRowHeight(30);  
 table.setFocusable(false);  
  
*// table.setSize(600,500);* JPanel tablePanel = new JPanel();  
 int height = (arrival.length)\*35;  
 tablePanel.setBounds(35,250,600,height);  
 tablePanel.setLocation(50,200);  
*// tablePanel.setBackground(Color.BLACK);* JScrollPane scrollPane = new JScrollPane(table);  
 scrollPane.setPreferredSize(new Dimension(600,600));  
*// scrollPane.setViewportView(table);* DefaultTableModel tableModel = (DefaultTableModel)table.getModel();  
*// adding the column name* tableModel.addColumn("Job");  
 tableModel.addColumn("Arrival Time");  
 tableModel.addColumn("Burst Time");  
 tableModel.addColumn("Finish Time");  
 tableModel.addColumn("Turnaround Time");  
 tableModel.addColumn("Waiting Time");  
*// Getting the values of turnaround array* int[] turnAround = turnaroundTime(finish,arrival);  
*// getting the values of waiting time* int[] waiting = waitingTime(turnAround,burst);  
*// Adding values to the column* for(int i=0;i<arrival.length;i++)  
 {  
*// selecting the job* char ch = (char)(65+i);  
 String str1 = Character.*toString*(ch);  
*// value of arrival time* String str2 = Integer.*toString*(arrival[i]);  
*// value of burst time* String str3 = Integer.*toString*(burst[i]);  
*// value of finish time* String str4 = Integer.*toString*(finish[i+1]);  
*// value of Turnaround Time* String str5 = Integer.*toString*(turnAround[i]);  
*// value of Waiting time* String str6 = Integer.*toString*(waiting[i]);  
 tableModel.addRow(new Object[0]);  
 tableModel.setValueAt(str1,i,0);  
 tableModel.setValueAt(str2,i,1);  
 tableModel.setValueAt(str3,i,2);  
 tableModel.setValueAt(str4,i,3);  
 tableModel.setValueAt(str5,i,4);  
 tableModel.setValueAt(str6,i,5);  
 }  
*// set the column width for each column* table.getColumnModel().getColumn(0).setPreferredWidth(5);  
 table.getColumnModel().getColumn(1).setPreferredWidth(20);  
 table.getColumnModel().getColumn(2).setPreferredWidth(15);  
 table.getColumnModel().getColumn(3).setPreferredWidth(17);  
*// Adding the average of waiting and turnaround time* JLabel avg = new JLabel("Average");  
 avg.setFont(new Font(null,Font.*PLAIN*,14));  
 avg.setBounds(300,370,100,30);  
*// avg.setForeground(new Color(215, 245, 255));  
// getting the average waitning time* int waitSum = 0;  
 for(int i=0;i<waiting.length;i++)  
 {  
 waitSum+=waiting[i];  
 }  
 int waitAvg = waitSum/waiting.length;  
*// getting the turnaround average* int tatSum = 0;  
 for(int i=0;i<turnAround.length;i++)  
 {  
 tatSum+=turnAround[i];  
 }  
 int tatAvg = tatSum/turnAround.length;  
*// Making 2 labels to add the waitAvg and tatAvg* JLabel tA = new JLabel(tatSum+"/"+ turnAround.length+" = "+tatAvg);  
 tA.setFont(new Font(null,Font.*PLAIN*,14));  
 tA.setBounds(375,370,100,30);  
 JLabel wA = new JLabel(waitSum+"/"+ waiting.length+" = "+waitAvg);  
 wA.setFont(new Font(null,Font.*PLAIN*,14));  
 wA.setBounds(510,370,100,30);  
 frame.add(tA);  
 frame.add(wA);  
 frame.add(avg);  
 frame.setSize(700,500);  
 tablePanel.add(scrollPane);  
 frame.add(tablePanel);  
 frame.add(fcfs);  
 frame.add(ganttChart);  
 frame.add(output);  
 frame.add(gantChartPanel);  
 frame.setVisible(true);  
 }  
 *// FIND FINISH TIME* public int[] finishTime(int[] arrival,int[] burst)  
 {  
 int[] finish = new int[(arrival.length+1)];  
 for(int i=0;i<finish.length;i++)  
 {  
 if(i==0)  
 {  
 finish[i] = arrival[i];  
 }  
 else{  
 finish[i] = finish[i-1] + burst[i-1];  
 }  
 }  
 return finish;  
 }  
 *// FIND WAITING TIME* public int[] waitingTime(int[]turnAround,int[] burst) {  
 int[] wait = new int[turnAround.length];  
 for (int i = 0; i < wait.length;i++)  
 {  
 wait[i] = turnAround[i] - burst[i];  
 }  
 return wait;  
 }  
 *// FIND TURNAROUND TIME* public int[] turnaroundTime(int[] finish,int[] arrival )  
 {  
 int[] tat = new int[arrival.length];  
 for(int i=0;i<tat.length;i++)  
 {  
 tat[i] = finish[i+1] - arrival[i];  
 }  
 return tat;  
 }  
}**

**SJFP CLASS**

**package OS\_Mini\_Project;  
  
import javax.swing.\*;  
import javax.swing.table.DefaultTableModel;  
import java.awt.\*;  
public class SJFP {  
 JFrame frame;*// used to making the frame``* JLabel output;*//used for the heading of the frame* JLabel sjfp; *// used to tell that which algo is using at the top right side of the frame* JLabel ganttChart;*// used to write the gantChart above the gantt chart diagram  
 // JLabel Average;//used to find the average of tat and wt* SJFP(int[] arrival,int[] burst)  
 {  
*// specify the number of rows and columns in the table and the gantt chart  
// int rowInTable = arrival.length;//row will be same for gantt chart and table* int columnInGanttChart = arrival.length;  
*// int column = 6;  
  
// Making the frame know* frame = new JFrame();  
 frame.setTitle("Shortest Job First Preemptive (SJF-P)");  
 frame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
*// first grid layout for gantt chart* frame.setLayout(null);  
*// Making panel for the gant chat* JPanel gantChartPanel = new JPanel();  
 gantChartPanel.setBounds(100,100,300,50);  
 gantChartPanel.setLocation(200,100);  
 gantChartPanel.setLayout(new GridLayout(1,columnInGanttChart,0,0));  
 gantChartPanel.setBackground(new Color(215, 245, 255));  
 gantChartPanel.setForeground(Color.*BLACK*);  
*// making buttons in the gantt chart* for(int i=0;i<columnInGanttChart;i++)  
 {  
 char ch = (char)(65+i);  
 JButton button = new JButton(String.*valueOf*(ch));  
 button.setBackground(new Color(215, 245, 255));  
 button.setFont(new Font(null,Font.*BOLD*,20));  
 button.setFocusable(false);  
 gantChartPanel.add(button);  
 }  
*// making the label for the heading output* output = new JLabel("Output");  
 output.setForeground(Color.*BLACK*);  
 output.setBounds(35,10,200,50);  
 output.setFont(new Font(null,Font.*BOLD*,35));  
*// writing gantt chart above the grid layout* ganttChart = new JLabel("Gantt Chart");  
 ganttChart.setBounds(270,65,200,30);  
 ganttChart.setFont(new Font("Courier",Font.*PLAIN*,30));  
*// adding panel for the algorithm* sjfp = new JLabel("SJF-P");  
 sjfp.setBounds(635,12,50,30);  
 sjfp.setFont(new Font(null,Font.*BOLD*,15));  
 sjfp.setForeground(new Color(40, 220, 255));  
  
*// ------------------------------------------------------------------------  
// Now making the values for the finish time in the gant chart* int[] finish = finishTime(arrival,burst);  
 for(int i=0;i<=finish.length;i++)  
 {  
 if(i==0)  
 {  
 String str = Integer.*toString*(1);  
*// str+=" ";* JLabel ft = new JLabel(str);  
 int start = 200;  
 ft.setBounds(start,145,300,30);  
 ft.setFont(new Font(null,Font.*PLAIN*,13));  
 frame.add(ft);  
 }  
 else{  
 int j = i-1;  
 String str = Integer.*toString*(finish[j]);  
*// str+=" ";* JLabel ft = new JLabel(str);  
 int start = 200 + (i\*70);  
 ft.setBounds(start,145,300,30);  
 ft.setFont(new Font(null,Font.*PLAIN*,13));  
 frame.add(ft);  
 }  
 }  
*// ------------------------------------------------------------------------  
// Making the table format for final table* JTable table = new JTable();  
*// table.setBounds();* table.setFont(new Font(null,Font.*BOLD*,15));  
 table.setBackground(new Color(215, 245, 255));  
 table.setBounds(100,100,700,480);  
 table.setRowHeight(30);  
 table.setFocusable(false);  
  
*// table.setSize(600,500);* JPanel tablePanel = new JPanel();  
 int height = (arrival.length)\*35;  
 tablePanel.setBounds(35,250,600,height);  
 tablePanel.setLocation(50,200);  
*// tablePanel.setBackground(Color.BLACK);* JScrollPane scrollPane = new JScrollPane(table);  
 scrollPane.setPreferredSize(new Dimension(600,600));  
*// scrollPane.setViewportView(table);* DefaultTableModel tableModel = (DefaultTableModel)table.getModel();  
*// adding the column name* tableModel.addColumn("Job");  
 tableModel.addColumn("Arrival Time");  
 tableModel.addColumn("Burst Time");  
 tableModel.addColumn("Finish Time");  
 tableModel.addColumn("Turnaround Time");  
 tableModel.addColumn("Waiting Time");  
*// Getting the values of turnaround array* int[] turnAround = turnaroundTime(arrival,burst);  
*// getting the values of waiting time* int[] waiting = waitingTime(arrival,burst);  
*// Adding values to the column* for(int i=0;i<arrival.length;i++)  
 {  
*// selecting the job* char ch = (char)(65+i);  
 String str1 = Character.*toString*(ch);  
*// value of arrival time* String str2 = Integer.*toString*(arrival[i]);  
*// value of burst time* String str3 = Integer.*toString*(burst[i]);  
*// value of finish time* String str4 = Integer.*toString*(finish[i]);  
*// value of Turnaround Time* String str5 = Integer.*toString*(turnAround[i]);  
*// value of Waiting time* String str6 = Integer.*toString*(waiting[i]);  
 tableModel.addRow(new Object[0]);  
 tableModel.setValueAt(str1,i,0);  
 tableModel.setValueAt(str2,i,1);  
 tableModel.setValueAt(str3,i,2);  
 tableModel.setValueAt(str4,i,3);  
 tableModel.setValueAt(str5,i,4);  
 tableModel.setValueAt(str6,i,5);  
 }  
*// set the column width for each column* table.getColumnModel().getColumn(0).setPreferredWidth(5);  
 table.getColumnModel().getColumn(1).setPreferredWidth(20);  
 table.getColumnModel().getColumn(2).setPreferredWidth(15);  
 table.getColumnModel().getColumn(3).setPreferredWidth(17);  
 frame.setSize(700,500);  
 tablePanel.add(scrollPane);  
 frame.add(tablePanel);  
 frame.add(sjfp);  
 frame.add(ganttChart);  
 frame.add(output);  
 frame.add(gantChartPanel);  
 frame.setVisible(true);  
 }  
 public int[] finishTime(int[] at,int[] bt)  
 {  
 int n = at.length;  
 int[] pid = new int[n];  
 int ct[] = new int[n]; *// ct means complete time* int ta[] = new int[n];*// ta means turn around time* int wt[] = new int[n]; *// wt means waiting time* int f[] = new int[n]; *// f means it is flag it checks process is completed or not* int k[]= new int[n]; *// it is also stores brust time* int i, st=0, tot=0;  
 float avgwt=0, avgta=0;  
  
 for (i=0;i<n;i++)  
 {  
 pid[i] = i+1;  
 k[i]= bt[i];  
 f[i]= 0;  
 }  
  
 while(true){  
 int min=99,c=n;  
 if (tot==n)  
 break;  
  
 for ( i=0;i<n;i++)  
 {  
 if ((at[i]<=st) && (f[i]==0) && (bt[i]<min))  
 {  
 min=bt[i];  
 c=i;  
 }  
 }  
  
 if (c==n)  
 st++;  
 else  
 {  
 bt[c]--;  
 st++;  
 if (bt[c]==0)  
 {  
 ct[c]= st;  
 f[c]=1;  
 tot++;  
 }  
 }  
 }  
  
 for(i=0;i<n;i++)  
 {  
 ta[i] = ct[i] - at[i];  
 wt[i] = ta[i] - k[i];  
 avgwt+= wt[i];  
 avgta+= ta[i];  
 }  
  
 return ct;  
 }  
 public int[] waitingTime(int[] at,int[] bt)  
 {  
 int n = at.length;  
 int[] pid = new int[n];  
 int ct[] = new int[n]; *// ct means complete time* int ta[] = new int[n];*// ta means turn around time* int wt[] = new int[n]; *// wt means waiting time* int f[] = new int[n]; *// f means it is flag it checks process is completed or not* int k[]= new int[n]; *// it is also stores brust time* int i, st=0, tot=0;  
 float avgwt=0, avgta=0;  
  
 for (i=0;i<n;i++)  
 {  
 pid[i] = i+1;  
 k[i]= bt[i];  
 f[i]= 0;  
 }  
  
 while(true){  
 int min=99,c=n;  
 if (tot==n)  
 break;  
  
 for ( i=0;i<n;i++)  
 {  
 if ((at[i]<=st) && (f[i]==0) && (bt[i]<min))  
 {  
 min=bt[i];  
 c=i;  
 }  
 }  
  
 if (c==n)  
 st++;  
 else  
 {  
 bt[c]--;  
 st++;  
 if (bt[c]==0)  
 {  
 ct[c]= st;  
 f[c]=1;  
 tot++;  
 }  
 }  
 }  
  
 for(i=0;i<n;i++)  
 {  
 ta[i] = ct[i] - at[i];  
 wt[i] = ta[i] - k[i];  
 avgwt+= wt[i];  
 avgta+= ta[i];  
 }  
  
 return wt;  
 }  
 public int[] turnaroundTime(int[] at,int[] bt)  
 {  
 int n = at.length;  
 int[] pid = new int[n];  
 int ct[] = new int[n]; *// ct means complete time* int ta[] = new int[n];*// ta means turn around time* int wt[] = new int[n]; *// wt means waiting time* int f[] = new int[n]; *// f means it is flag it checks process is completed or not* int k[]= new int[n]; *// it is also stores brust time* int i, st=0, tot=0;  
 float avgwt=0, avgta=0;  
  
 for (i=0;i<n;i++)  
 {  
 pid[i] = i+1;  
 k[i]= bt[i];  
 f[i]= 0;  
 }  
  
 while(true){  
 int min=99,c=n;  
 if (tot==n)  
 break;  
  
 for ( i=0;i<n;i++)  
 {  
 if ((at[i]<=st) && (f[i]==0) && (bt[i]<min))  
 {  
 min=bt[i];  
 c=i;  
 }  
 }  
  
 if (c==n)  
 st++;  
 else  
 {  
 bt[c]--;  
 st++;  
 if (bt[c]==0)  
 {  
 ct[c]= st;  
 f[c]=1;  
 tot++;  
 }  
 }  
 }  
  
 for(i=0;i<n;i++)  
 {  
 ta[i] = ct[i] - at[i];  
 wt[i] = ta[i] - k[i];  
 avgwt+= wt[i];  
 avgta+= ta[i];  
 }  
  
  
 return ta;  
 }  
}**

**SJFNP CLASS**

**package OS\_Mini\_Project;  
  
import javax.swing.\*;  
*//import javax.swing.border.\*;*import javax.swing.table.DefaultTableModel;  
import java.awt.\*;  
public class SJFNP {  
 static int[][] *mat* = new int[10][6];  
 JFrame frame;*// used to making the frame``* JLabel output;*//used for the heading of the frame* JLabel sjfnp; *// used to tell that which algo is using at the top right side of the frame* JLabel ganttChart;*// used to write the gantChart above the gantt chart diagram  
 // JLabel Average;//used to find the average of tat and wt* SJFNP(int[] arrival,int[] burst)  
 {  
*// specify the number of rows and columns in the table and the gantt chart  
// int rowInTable = arrival.length;//row will be same for gantt chart and table* int columnInGanttChart = arrival.length;  
*// int column = 6;  
  
// Making the frame know* frame = new JFrame();  
 frame.setTitle("Shortest Job First Non Preemptive (SJF-NP)");  
 frame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
*// first grid layout for gantt chart* frame.setLayout(null);  
*// Making panel for the gant chat* JPanel gantChartPanel = new JPanel();  
 gantChartPanel.setBounds(100,100,300,50);  
 gantChartPanel.setLocation(200,100);  
 gantChartPanel.setLayout(new GridLayout(1,columnInGanttChart,0,0));  
 gantChartPanel.setBackground(new Color(215, 245, 255));  
 gantChartPanel.setForeground(Color.*BLACK*);  
*// making buttons in the gantt chart* for(int i=0;i<columnInGanttChart;i++)  
 {  
 char ch = (char)(65+i);  
 JButton button = new JButton(String.*valueOf*(ch));  
 button.setBackground(new Color(215, 245, 255));  
 button.setFont(new Font(null,Font.*BOLD*,20));  
 button.setFocusable(false);  
 gantChartPanel.add(button);  
 }  
*// making the label for the heading output* output = new JLabel("Output");  
 output.setForeground(Color.*BLACK*);  
 output.setBounds(35,10,200,50);  
 output.setFont(new Font(null,Font.*BOLD*,35));  
*// writing gantt chart above the grid layout* ganttChart = new JLabel("Gantt Chart");  
 ganttChart.setBounds(270,65,200,30);  
 ganttChart.setFont(new Font("Courier",Font.*PLAIN*,30));  
*// adding panel for the algorithm* sjfnp = new JLabel("SJF-P");  
 sjfnp.setBounds(635,12,50,30);  
 sjfnp.setFont(new Font(null,Font.*BOLD*,15));  
 sjfnp.setForeground(new Color(40, 220, 255));  
*// calling all IN one function* allInOne(arrival,burst);  
*// ------------------------------------------------------------------------  
// Now making the values for the finish time in the gant chart* for(int i=0;i<=arrival.length;i++)  
 {  
 if(i==0)  
 {  
 String str = Integer.*toString*(0);  
*// str+=" ";* JLabel ft = new JLabel(str);  
 int start = 200;  
 ft.setBounds(start,145,300,30);  
 ft.setFont(new Font(null,Font.*PLAIN*,13));  
 frame.add(ft);  
 }  
 else{  
 String str = Integer.*toString*(*mat*[i-1][3]);  
*// str+=" ";* JLabel ft = new JLabel(str);  
 int start = 200 + (i\*60);  
 ft.setBounds(start,145,300,30);  
 ft.setFont(new Font(null,Font.*PLAIN*,13));  
 frame.add(ft);  
 }  
 }  
*// ------------------------------------------------------------------------  
// Making the table format for final table* JTable table = new JTable();  
*// table.setBounds();* table.setFont(new Font(null,Font.*BOLD*,15));  
 table.setBackground(new Color(215, 245, 255));  
 table.setBounds(100,100,700,480);  
 table.setRowHeight(30);  
 table.setFocusable(false);  
  
*// table.setSize(600,500);* JPanel tablePanel = new JPanel();  
 int height = (arrival.length)\*35;  
 tablePanel.setBounds(35,250,600,height);  
 tablePanel.setLocation(50,200);  
*// tablePanel.setBackground(Color.BLACK);* JScrollPane scrollPane = new JScrollPane(table);  
 scrollPane.setPreferredSize(new Dimension(600,600));  
*// scrollPane.setViewportView(table);* DefaultTableModel tableModel = (DefaultTableModel)table.getModel();  
*// adding the column name* tableModel.addColumn("Job");  
 tableModel.addColumn("Arrival Time");  
 tableModel.addColumn("Burst Time");  
 tableModel.addColumn("Finish Time");  
 tableModel.addColumn("Turnaround Time");  
 tableModel.addColumn("Waiting Time");  
  
*// Adding values to the column* for(int i=0;i<arrival.length;i++)  
 {  
*// selecting the job(from the mat array)* int process = *mat*[i][0];  
 char ch = (char)(64+process);  
 String str1 = Character.*toString*(ch);  
*// value of arrival time* String str2 = Integer.*toString*(*mat*[i][1]);  
*// value of burst time* String str3 = Integer.*toString*(*mat*[i][2]);  
*// value of finish time* String str4 = Integer.*toString*(*mat*[i][3]);  
*// value of Turnaround Time* String str5 = Integer.*toString*(*mat*[i][5]);  
*// value of Waiting time* String str6 = Integer.*toString*(*mat*[i][4]);  
 tableModel.addRow(new Object[0]);  
 tableModel.setValueAt(str1,i,0);  
 tableModel.setValueAt(str2,i,1);  
 tableModel.setValueAt(str3,i,2);  
 tableModel.setValueAt(str4,i,3);  
 tableModel.setValueAt(str5,i,4);  
 tableModel.setValueAt(str6,i,5);  
 }  
*// set the column width for each column* table.getColumnModel().getColumn(0).setPreferredWidth(5);  
 table.getColumnModel().getColumn(1).setPreferredWidth(20);  
 table.getColumnModel().getColumn(2).setPreferredWidth(15);  
 table.getColumnModel().getColumn(3).setPreferredWidth(17);  
 frame.setSize(700,500);  
 *// Adding the average of waiting and turnaround time* JLabel avg = new JLabel("Average");  
 avg.setFont(new Font(null,Font.*PLAIN*,14));  
 avg.setBounds(300,370,100,30);  
  
 *// getting the average waitning time* int waitSum = 0;  
 for(int i=0;i<arrival.length;i++)  
 {  
 waitSum+=*mat*[i][4];  
 }  
 int waitAvg = waitSum/arrival.length;  
*// getting the turnaround average* int tatSum = 0;  
 for(int i=0;i<arrival.length;i++)  
 {  
 tatSum+=*mat*[i][5];  
 }  
 int tatAvg = tatSum/arrival.length;  
*// Making 2 labels to add the waitAvg and tatAvg* JLabel tA = new JLabel(tatSum+"/"+ arrival.length+" = "+tatAvg);  
 tA.setFont(new Font(null,Font.*PLAIN*,14));  
 tA.setBounds(375,370,100,30);  
 JLabel wA = new JLabel(waitSum+"/"+ arrival.length+" = "+waitAvg);  
 wA.setFont(new Font(null,Font.*PLAIN*,14));  
 wA.setBounds(510,370,100,30);  
 frame.add(tA);  
 frame.add(wA);  
 frame.add(avg);  
 tablePanel.add(scrollPane);  
 frame.add(tablePanel);  
 frame.add(sjfnp);  
 frame.add(ganttChart);  
 frame.add(output);  
 frame.add(gantChartPanel);  
 frame.setVisible(true);  
 }  
 *// all in one method for setting the answer for all the times* public void allInOne(int[] arrival,int[]burst) {  
 for (int i = 0; i < arrival.length; i++) {  
 *mat*[i][0] = i + 1;  
 *mat*[i][1] = arrival[i];  
 *mat*[i][2] = burst[i];  
 }  
 *arrangeArrival*(arrival.length,*mat*);  
 *completionTime*(arrival.length,*mat*);  
 }  
 *//arrange arrival ime method* static void arrangeArrival(int num, int[][] mat)  
 {  
 for (int i = 0; i < num; i++) {  
 for (int j = 0; j < num - i - 1; j++) {  
 if (mat[j][1] > mat[j + 1][1]) {  
 for (int k = 0; k < 5; k++) {  
 int temp = mat[j][k];  
 mat[j][k] = mat[j + 1][k];  
 mat[j + 1][k] = temp;  
 }  
 }  
 }  
 }  
 }  
 *// completion time method* static void completionTime(int num, int[][] mat)  
 {  
 int temp, val = -1;  
 mat[0][3] = mat[0][1] + mat[0][2];  
 mat[0][5] = mat[0][3] - mat[0][1];  
 mat[0][4] = mat[0][5] - mat[0][2];  
  
 for (int i = 1; i < num; i++) {  
 temp = mat[i - 1][3];  
 int low = mat[i][2];  
 for (int j = i; j < num; j++) {  
 if (temp >= mat[j][1] && low >= mat[j][2]) {  
 low = mat[j][2];  
 val = j;  
 }  
 }  
 mat[val][3] = temp + mat[val][2];  
 mat[val][5] = mat[val][3] - mat[val][1];  
 mat[val][4] = mat[val][5] - mat[val][2];  
 for (int k = 0; k < 6; k++) {  
 int tem = mat[val][k];  
 mat[val][k] = mat[i][k];  
 mat[i][k] = tem;  
 }  
 }  
 }  
}**

**Snapshots**

**Graphical user interface, table

Description automatically generated**

* 1. **FCFS**

**Table

Description automatically generated**

**SJF (Non-Pre-emptive)**

**Graphical user interface, table

Description automatically generated with medium confidence**

**Table

Description automatically generated**

**Graphical user interface

Description automatically generated SJF Pre-emptive**

**Table

Description automatically generated**

**Conclusion**

OS scheduling algorithm simulation can be used by educational institution to teach and demonstrate in a better way so that it is quite easy to the students to grasp the correct logic and concepts behind it. Teaching them manually in board will take lots of efforts and time so this simulation will saves there time and effort in an efficient way. Also it can be used by the operating systems architect to better analyse there product based on one of these scheduling algorithms.

**Bibliography**

1. [**https://www.youtube.com/watch?v=xk4\_1vDrzzo**](https://www.youtube.com/watch?v=xk4_1vDrzzo)
2. [**https://www.javatpoint.com/java-swing**](https://www.javatpoint.com/java-swing)
3. [**https://www.jetbrains.com/idea/**](https://www.jetbrains.com/idea/)