CPU SCHEDULING SIMULATION

MINI PROJECT REPORT

Submitted by

AKSSHAYA B (202009003)
BHUVANA S (202009008)

in

19AD481 – OPERATING SYSTEM PRINCIPLES

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

MEPCO SCHLENK ENGINEERING COLLEGE

SIVAKASI

MAY 2022

MEPCO SCHLENK ENGINEERING COLLEGE, SIVAKASI

AUTONOMOUS

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

BONAFIDE CERTIFICATE

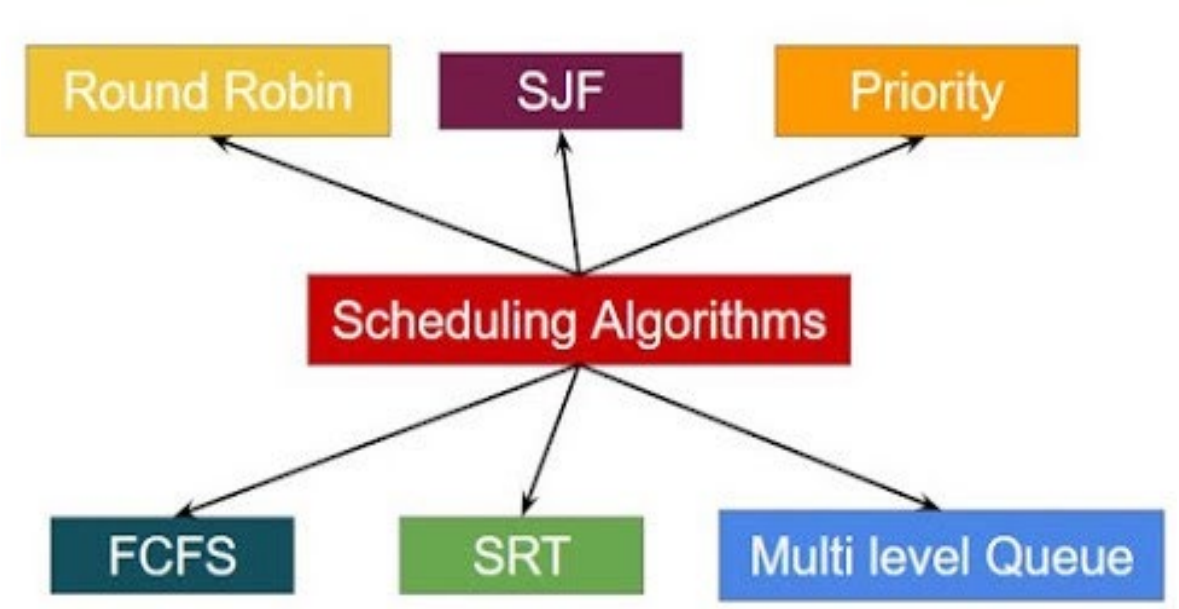
This is to certify that it is the bonafide work of "**Aksshaya.B (Reg. No.: 202009003)** and **Bhuvana.S (Reg.No.:202009008)"** for the mini project titled **"CPU SCHEDULING SIMULATION"** in 19AD481 – Operating System Principles during the fourth semester, January 2022 – May 2022 under my supervision.

SIGNATURE
Mrs.P.Swathika,
Asst. Professor (Senior Grade),
AI&DS Department,
Mepco Schlenk Engg. College, Sivakasi

SIGNATURE
Dr. J. Angela Jennifa Sujana,
Asso. Professor (Senior Grade)& Head,
AI&DS Department,
Mepco Schlenk Engg. College, Sivakasi

ABSTRACT

The Operating system schedules the processes on the CPU to have the maximum utilization of it and this procedure is called **CPU scheduling**. The Operating System uses various scheduling algorithm to schedule the processes. Some of the algorithms include FCFS, SJF and Round Robin. Based on the arrival time and burst time of the given processes, the corresponding waiting time(WT), turn around time(TAT) is calculated for each and every process. In the uniprogramming systems like MS DOS, when a process waits for any I/O operation to be done, the CPU remains idol. This is an overhead since it wastes the time and causes the problem of starvation. However, In Multiprogramming systems, the CPU doesn't remain idle during the waiting time of the Process and it starts executing other processes. Operating System has to define which process the CPU will be given. In Multiprogramming systems, the Operating system schedules the processes on the CPU to have the maximum utilization of it and this procedure is called CPU scheduling. The Operating System uses various scheduling algorithm to schedule the processes.



(1) Types of Scheduling Algorithms

The algorithms may be pre-emptive or non pre-emptive:

Pre-emptive include, First Come First Serve(FCFS) and Priority.
Non pre-emptive include, Shortest Job First(SJF) and Round Robin(RR).

TABLE OF CONTENTS

1. INTRODUCTION

1.1.Introduction

In Multiprogramming, if the long term scheduler picks more I/O bound processes then most of the time, the CPU remains idol. The task of Operating system is to optimize the utilization of resources. If most of the running processes change their state from running to waiting then there may always be a possibility of deadlock in the system. Hence to reduce this overhead, the OS needs to schedule the jobs to get the optimal utilization of CPU and to avoid the possibility to deadlock. CPU Scheduling is a process of determining which process will own CPU for execution while another process is on hold. The main task of CPU scheduling is to make sure that whenever the CPU remains idle, the OS at least select one of the processes available in the ready queue for execution. The selection process will be carried out by the CPU scheduler. It selects one of the processes in memory that are ready for execution.

1.2.Objectives

The Objective of a Scheduling algorithm is to:

- Maximum CPU utilization
- Fair allocation of CPU
- Maximum throughput
- Minimum turnaround time
- Minimum waiting time
- Minimum response time

For each process arrival time and burst time is given.

Arrival Time: Time at which the process arrives in the ready queue.

Completion Time: Time at which process completes its execution.

Burst Time: Time required by a process for CPU execution.

Turn Around Time: Time Difference between completion time and arrival time.

Turn Around Time = Completion Time – Arrival Time

Waiting Time(W.T): Time Difference between turn around time and burst time.

Waiting Time = Turn Around Time – Burst Time

Based on the formulas displayed above, the corresponding waiting time and turn around time is calculated respectively.

- FCFS:

First Come First Serve (FCFS) is an operating system scheduling algorithm that automatically executes queued requests and processes in order of their arrival. It is the easiest and simplest CPU scheduling algorithm. In this type of algorithm, processes which requests the CPU first get the CPU allocation first.

- SJF:

Shortest Job First (SJF) is an algorithm in which the process having the smallest execution time is chosen for the next execution. This scheduling method can be preemptive or non-preemptive. It significantly reduces the average waiting time for other processes awaiting execution.

- Pre-emptive SJF:

In **Preemptive SJF** Scheduling, jobs are put into the ready queue as they come. A process with shortest burst time begins execution. If a process with even a shorter burst time arrives, the current process is removed or preempted from execution, and the shorter job is allocated CPU cycle.

- Non pre-emptive SJF:

In **non-preemptive SJF** scheduling, once the CPU cycle is allocated to process, the process holds it till it reaches a waiting state or terminated.

- Round Robin:

Round Robin is a CPU scheduling algorithm where each process is assigned a fixed time slot in a cyclic way. It is basically the preemptive version of First come First Serve CPU Scheduling algorithm.

1.3.Algorithms

- **FCFS**

algorithm fcfsSchedule( )

{

get the number of processes.

Get process name, arrival time and burst time of all processes

Compute waiting time for all according to fcfs algorithm

Compute Total waiting time and Total turnaround time

Average Waiting time = total Waiting time / number of processes

Average Turn Around Time = total turnaround time / number of processes

}

- **SJF**

```
algorithm sjfSchedule( )
{
        Get process name, arrival time and burst time of all processes
        Compute waiting time for all according to sjf algorithm
        Compute Total waiting time and Total turnaround time
        Average Waiting time = Total Waiting time / number of processes
        Average Turn Around Time = Total turnaround time / number of processes
}
```

- **Round Robin**

```
algorithm roundRobinScheduling( )
{
        Get process name, arrival time and burst time of all processes
        Compute waiting time for all according to Round Robin algorithm
        Compute Total waiting time
        Average waiting time = total waiting time / no of process
}
```

2.IMPLEMENTATION

- All the CPU scheduling algorithms are implemented using Java for both frontend and backend.
- For frontend, we have used java.swing package.
- The Logic of the program is executed using Java Language.

2.1.Program Coding

FCFS:

```
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;
import java.util.Queue;
import java.util.concurrent.TimeUnit;

public class FCFS
{
    int time = 0;
```

```
        Queue<ProcessObj> q = new LinkedList<>();
        List<ProcessObj> list = new ArrayList<>();
        List<ProcessObj> completed = new ArrayList<>();
        public List<ProcessObj> allocateResources( List<ProcessObj> l) throws
InterruptedException {
        System.out.println("Started the first come first serve algorithm--->");
        list=l;
//        addToQueue();
        while (list.size()>0 || q.size()>0) {
           addToQueue();
           while (q.size()==0) {
              System.out.println("------Waiting to recive a process---------");
              TimeUnit.SECONDS.sleep(1);
              time++;
              addToQueue();
           }
           ProcessObj processRun = q.remove();
           System.out.println("Process name - " + processRun.getName() + " and Id = " +
processRun.getId() + " is going run");
           for (int j = 0; j < processRun.getBrustTime(); j++) {
              System.out.println("------running a seccond--------");
              TimeUnit.SECONDS.sleep(1);
              time++;
              addToQueue();
           }
           processRun.setCompleteTime(time);
           processRun.setTurnaroundTime(time-processRun.getArrivalTime());
           processRun.setWaitingTime(processRun.getTurnaroundTime()-
processRun.getBrustTime());
           System.out.println("processcompleted time = "+processRun.getCompleteTime());

System.out.println("processturnaroundtime="+processRun.getTurnaroundTime());
           System.out.println(" process waiting time = "+processRun.getWaitingTime());

           this.completed.add(processRun);

        }
        return completed;
     }

    public void addToQueue()
    {
        if(!list.isEmpty())
        {
           for (int i = 0; i < list.size(); i++)
           {
```

```
            ProcessObj p = list.get(i);
            if (time >= p.getArrivalTime())
            {
               q.add(p);
               list.remove(i);
            }
            }
         }
      }
}
```

SJF (Non- Preemptive):

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.concurrent.TimeUnit;


public class SJF {
    int time = 0;
       List<ProcessObj> q = new ArrayList<>();
       List<ProcessObj> list = new ArrayList<>();
       List<ProcessObj> completed = new ArrayList<>();
   public List<ProcessObj> allocateResources( List<ProcessObj> l) throws
InterruptedException {
       System.out.println("Started the Shortest job first algorithm--->");
       list=l;
       while (list.size()>0 || q.size()>0) {
          addToQueue();
          while (q.isEmpty()) {
             System.out.println("------Waiting to recive a process---------");
             TimeUnit.SECONDS.sleep(1);
             time++;
             addToQueue();
          }
          ProcessObj processRun = selectTheBest();
          System.out.println("Process which name - " + processRun.getName() + " and Id =
" + processRun.getId() + " is going run");
          for (int j = 0; j < processRun.getBrustTime(); j++) {
             System.out.println("------running a seccond--------");
             TimeUnit.SECONDS.sleep(1);
             time++;
             addToQueue();
          }
```

```
            processRun.setCompleteTime(time);
            processRun.setTurnaroundTime(time-processRun.getArrivalTime());
            processRun.setWaitingTime(processRun.getTurnaroundTime()-
processRun.getBrustTime());

            System.out.println(" process waiting time = "+processRun.getWaitingTime());

            this.completed.add(processRun);

        }
        return completed;
    }

    public void addToQueue(){
        if(!list.isEmpty()){
            for (int i = 0; i < list.size(); i++) {
            ProcessObj p = list.get(i);
            if (time >= p.getArrivalTime()) {
                q.add(p);
                list.remove(i);
            }
        }

        }
    }

    private ProcessObj selectTheBest() {
        List<Integer> bTimelist = new ArrayList<>();
        if(!q.isEmpty()){
            for (int i = 0; i < q.size(); i++) {
                bTimelist.add(q.get(i).getBrustTime());
            }
            Collections.sort(bTimelist);

            for (int i = 0; i < q.size(); i++) {
                if (bTimelist.get(0)== q.get(i).getBrustTime()) {
                    return q.remove(i);
                }
            }
        }
        return null;
    }

}
```

SJF (Preemptive):

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.concurrent.TimeUnit;

public class SRTF {
   int time = 0;
   int timeQuantum =1;
      List<ProcessObj> q = new ArrayList<>();
      List<ProcessObj> cloneq = new ArrayList<>();
      List<ProcessObj> list = new ArrayList<>();
      List<ProcessObj> completed = new ArrayList<>();
   public List<ProcessObj> allocateResources( List<ProcessObj> l, List<ProcessObj>
completed) throws InterruptedException {
      System.out.println("Started the shortest remaining time first algorithm--->");
      list.addAll(l);
      this.completed.addAll(completed);
      while (list.size()>0 || q.size()>0) {
         addToQueue();
         while (q.isEmpty()) {
            System.out.println("------Waiting to recive a process---------");
            TimeUnit.SECONDS.sleep(1);
            time++;
            addToQueue();
         }
        ProcessObj processRun = selectTheBest();
        System.out.println("Process which name - " + processRun.getName() + " and Id =
" + processRun.getId() + " is going run");
         for (int j = 0; j < timeQuantum; j++) {
            System.out.println("------running a seccond--------");
            TimeUnit.SECONDS.sleep(1);
            time++;
            addToQueue();
         }

      }
      for (int j = 0; j < completed.size(); j++) {
         ProcessObj p = completed.get(j);
         p.setBrustTime(l.get(j).getBrustTime());
         p.setTurnaroundTime(p.getCompleteTime()- p.getArrivalTime());
         p.setWaitingTime(p.getTurnaroundTime()- p.getBrustTime());
         completed.set(j, p);
         }
```

```
        for (int i = 0; i < completed.size(); i++) {
            System.out.println(" process Arrival time = "+completed.get(i).getArrivalTime());

System.out.println("processcompletedtime="+completed.get(i).getCompleteTime());
            System.out.println(" process turnaround time =
"+completed.get(i).getTurnaroundTime());
            System.out.println(" process burst time = "+completed.get(i).getBrustTime());
            System.out.println(" process waiting time =
"+completed.get(i).getWaitingTime());
            System.out.println("-----------------------------------------------------");
        }

        return completed;
    }

    public void addToQueue(){
        if(!list.isEmpty()){
            for (int i = 0; i < list.size(); i++) {
            ProcessObj p = list.get(i);
            if (time >= p.getArrivalTime()) {
                q.add(p);
                list.remove(i);
            }
        }
        }
    }
```

Class for User Interface:

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import javax.swing.table.DefaultTableModel;

public class Starter extends javax.swing.JFrame
{
public Starter()
{
initComponents();
DefaultTableModel mdl = (DefaultTableModel) tblAddedProcess.getModel();
                                                                    mdl.setRowCount(0);
    }
    @SuppressWarnings("unchecked")
    private void initComponents() {
```

```
jScrollPane3 = new javax.swing.JScrollPane();
jLabel1 = new javax.swing.JLabel();
jPanel1 = new javax.swing.JPanel();
jLabel2 = new javax.swing.JLabel();
jLabel3 = new javax.swing.JLabel();
jLabel4 = new javax.swing.JLabel();
jLabel5 = new javax.swing.JLabel();
jLabel6 = new javax.swing.JLabel();
pIdTxt = new javax.swing.JTextField();
pNameTxt = new javax.swing.JTextField();
pATimeTxt = new javax.swing.JTextField();
pBTimeTxt = new javax.swing.JTextField();
createProcess = new javax.swing.JButton();
jPanel2 = new javax.swing.JPanel();
sJF = new javax.swing.JButton();
rRound = new javax.swing.JButton();
fCFS = new javax.swing.JButton();
sRT = new javax.swing.JButton();
jPanel3 = new javax.swing.JPanel();
jScrollPane1 = new javax.swing.JScrollPane();
tblAddedProcess = new javax.swing.JTable();
jLabel7 = new javax.swing.JLabel();
btnClearTbl = new javax.swing.JButton();
tQuantumtxt = new javax.swing.JTextField();
jLabel11 = new javax.swing.JLabel();
jPanel4 = new javax.swing.JPanel();
jScrollPane2 = new javax.swing.JScrollPane();
tblComplete = new javax.swing.JTable();
jPanel5 = new javax.swing.JPanel();
jLabel8 = new javax.swing.JLabel();
jLabel9 = new javax.swing.JLabel();
jLabel10 = new javax.swing.JLabel();
lblAvgTTime = new javax.swing.JLabel();
lblAvgWtime = new javax.swing.JLabel();
lblthroughput = new javax.swing.JLabel();
txtAvgTurn = new javax.swing.JTextField();
txtWait = new javax.swing.JTextField();
txtThroughput = new javax.swing.JTextField();

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
setTitle("CPU Scheduling Simulator");

jLabel1.setFont(new java.awt.Font("Palatino Linotype", 1, 14)); // NOI18N
jLabel1.setText("CPU Scheduling Algorithm Simulator");
```

```
jPanel1.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swing.border.Bev
elBorder.RAISED));

        jLabel2.setFont(new java.awt.Font("Tahoma", 1, 12)); // NOI18N
        jLabel2.setText("Create Process");

        jLabel3.setText("Process Id");

        jLabel4.setText("Process Name");

        jLabel5.setText("Arrival Time");

        jLabel6.setText("Burst Time");

        pIdTxt.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                pIdTxtActionPerformed(evt);
            }
        });

        pNameTxt.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                pNameTxtActionPerformed(evt);
            }
        });

        pBTimeTxt.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                pBTimeTxtActionPerformed(evt);
            }
        });

        createProcess.setText("Create Process");
        createProcess.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                createProcessActionPerformed(evt);
            }
        });

        javax.swing.GroupLayout jPanel1Layout = new javax.swing.GroupLayout(jPanel1);
        jPanel1.setLayout(jPanel1Layout);
        jPanel1Layout.setHorizontalGroup(

jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel1Layout.createSequentialGroup()
```

```
.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LE
ADING)
                .addGroup(jPanel1Layout.createSequentialGroup()
                    .addContainerGap()

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LE
ADING)
                        .addComponent(jLabel3,
javax.swing.GroupLayout.PREFERRED_SIZE,                                    68,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addComponent(jLabel4)
                        .addComponent(jLabel5,
javax.swing.GroupLayout.PREFERRED_SIZE,                                    67,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addComponent(jLabel6,
javax.swing.GroupLayout.PREFERRED_SIZE,                                    67,
javax.swing.GroupLayout.PREFERRED_SIZE))
                    .addGap(44, 44, 44)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LE
ADING, false)
                        .addComponent(pIdTxt)
                        .addComponent(pNameTxt)
                        .addComponent(pATimeTxt)
                        .addComponent(pBTimeTxt,
javax.swing.GroupLayout.DEFAULT_SIZE, 124, Short.MAX_VALUE)))
                .addGroup(jPanel1Layout.createSequentialGroup()
                    .addGap(79, 79, 79)
                    .addComponent(jLabel2))
                .addGroup(jPanel1Layout.createSequentialGroup()
                    .addGap(71, 71, 71)
                    .addComponent(createProcess)))
            .addContainerGap(20, Short.MAX_VALUE))
        );
        jPanel1Layout.setVerticalGroup(

jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jLabel2)
                .addGap(18, 18, 18)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BA
SELINE)
                    .addComponent(jLabel3)
```

```
                .addComponent(pIdTxt,       javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BA
SELINE)
                .addComponent(jLabel4)
                .addComponent(pNameTxt,  javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BA
SELINE)
                .addComponent(jLabel5)
                .addComponent(pATimeTxt,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BA
SELINE)
                .addComponent(jLabel6)
                .addComponent(pBTimeTxt,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
            .addGap(18, 18, 18)
            .addComponent(createProcess)
            .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
      );


jPanel2.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swing.border.Bev
elBorder.RAISED));

      sJF.setText("SJF (Non- Preemptive)");
      sJF.addActionListener(new java.awt.event.ActionListener() {
         public void actionPerformed(java.awt.event.ActionEvent evt) {
            sJFActionPerformed(evt);
         }
```

```
        });

        rRound.setText("Round Robin");
        rRound.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                rRoundActionPerformed(evt);
            }
        });

        fCFS.setText("FCFS");
        fCFS.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                fCFSActionPerformed(evt);
            }
        });

        sRT.setText("SJF (Preemptive)");
        sRT.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                sRTActionPerformed(evt);
            }
        });

        javax.swing.GroupLayout jPanel2Layout = new javax.swing.GroupLayout(jPanel2);
        jPanel2.setLayout(jPanel2Layout);
        jPanel2Layout.setHorizontalGroup(

    jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel2Layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(sJF,     javax.swing.GroupLayout.PREFERRED_SIZE,     150,
    javax.swing.GroupLayout.PREFERRED_SIZE)

    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                .addComponent(rRound,  javax.swing.GroupLayout.PREFERRED_SIZE,  150,
    javax.swing.GroupLayout.PREFERRED_SIZE)

    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                .addComponent(fCFS,    javax.swing.GroupLayout.PREFERRED_SIZE,    165,
    javax.swing.GroupLayout.PREFERRED_SIZE)

    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                .addComponent(sRT,     javax.swing.GroupLayout.PREFERRED_SIZE,     163,
    javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
    Short.MAX_VALUE))
```

```
        );

        jPanel2Layout.linkSize(javax.swing.SwingConstants.HORIZONTAL, new
java.awt.Component[] {fCFS, rRound, sJF, sRT});

        jPanel2Layout.setVerticalGroup(

jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel2Layout.createSequentialGroup()
                .addContainerGap()

.addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LE
ADING)
                    .addComponent(sJF, javax.swing.GroupLayout.PREFERRED_SIZE, 41,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                        .addComponent(rRound)
                        .addComponent(fCFS, javax.swing.GroupLayout.PREFERRED_SIZE,
41, javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addComponent(sRT, javax.swing.GroupLayout.PREFERRED_SIZE, 41,
javax.swing.GroupLayout.PREFERRED_SIZE)))
                .addContainerGap())
        );

        jPanel2Layout.linkSize(javax.swing.SwingConstants.VERTICAL, new
java.awt.Component[] {fCFS, rRound, sJF, sRT});


jPanel3.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swing.border.Bev
elBorder.RAISED));

        tblAddedProcess.setModel(new javax.swing.table.DefaultTableModel(
            new Object [][] {
                {null, null, null, null},
                {null, null, null, null},
                {null, null, null, null},
                {null, null, null, null},
                {null, null, null, null},
                {null, null, null, null},
                {null, null, null, null}
            },
            new String [] {
                "Process Id", "Name", "Arrival time", "Burst time"
            }
        ));
```

```
        jScrollPane1.setViewportView(tblAddedProcess);

        jLabel7.setFont(new java.awt.Font("Tahoma", 1, 12)); // NOI18N
        jLabel7.setText("processes");

        btnClearTbl.setText("Clear ");
        btnClearTbl.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                btnClearTblActionPerformed(evt);
            }
        });

        tQuantumtxt.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                tQuantumtxtActionPerformed(evt);
            }
        });

        jLabel11.setText("Time Quantum :");

        javax.swing.GroupLayout jPanel3Layout = new javax.swing.GroupLayout(jPanel3);
        jPanel3.setLayout(jPanel3Layout);
        jPanel3Layout.setHorizontalGroup(

jPanel3Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
jPanel3Layout.createSequentialGroup()
                .addGap(0, 0, Short.MAX_VALUE)
                .addComponent(jScrollPane1,   javax.swing.GroupLayout.PREFERRED_SIZE,
440, javax.swing.GroupLayout.PREFERRED_SIZE))
            .addGroup(jPanel3Layout.createSequentialGroup()
                .addGap(170, 170, 170)
                .addComponent(jLabel7)
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
jPanel3Layout.createSequentialGroup()
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
                .addComponent(jLabel11)
                .addGap(18, 18, 18)
                .addComponent(tQuantumtxt,   javax.swing.GroupLayout.PREFERRED_SIZE,
38, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGap(83, 83, 83)
                .addComponent(btnClearTbl,    javax.swing.GroupLayout.PREFERRED_SIZE,
120, javax.swing.GroupLayout.PREFERRED_SIZE)
```

```
            .addContainerGap())
        );
        jPanel3Layout.setVerticalGroup(

jPanel3Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
jPanel3Layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jLabel7)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(jScrollPane1,   javax.swing.GroupLayout.PREFERRED_SIZE,
138, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)

.addGroup(jPanel3Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LE
ADING, false)
                    .addComponent(btnClearTbl)
                    .addGroup(jPanel3Layout.createSequentialGroup()
                       .addGap(1, 1, 1)

.addGroup(jPanel3Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BA
SELINE)
                            .addComponent(tQuantumtxt)
                            .addComponent(jLabel11,
javax.swing.GroupLayout.PREFERRED_SIZE,                                    21,
javax.swing.GroupLayout.PREFERRED_SIZE))))
                .addContainerGap())
        );


jPanel4.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swing.border.Bev
elBorder.RAISED));

        tblComplete.setModel(new javax.swing.table.DefaultTableModel(
            new Object [][] {
                {null, null, null, null, null, null, null},
                {null, null, null, null, null, null, null},
                {null, null, null, null, null, null, null},
                {null, null, null, null, null, null, null}
            },
            new String [] {
                "Process Id", "Name", "Arrival Time", "Brust Time", "Completed Time",
"Waiting Time", "Turnaround Time"
            }
        ));
```

```
        jScrollPane2.setViewportView(tblComplete);

        javax.swing.GroupLayout jPanel4Layout = new javax.swing.GroupLayout(jPanel4);
        jPanel4.setLayout(jPanel4Layout);
        jPanel4Layout.setHorizontalGroup(

jPanel4Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
jPanel4Layout.createSequentialGroup()
                .addContainerGap(16, Short.MAX_VALUE)
                .addComponent(jScrollPane2,   javax.swing.GroupLayout.PREFERRED_SIZE,
700, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap())
        );
        jPanel4Layout.setVerticalGroup(

jPanel4Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel4Layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jScrollPane2,   javax.swing.GroupLayout.PREFERRED_SIZE,
104, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
        );


jPanel5.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swing.border.Bev
elBorder.RAISED));

        jLabel8.setFont(new java.awt.Font("Tahoma", 0, 12)); // NOI18N
        jLabel8.setText("Average Turnaround Time    :");

        jLabel9.setFont(new java.awt.Font("Tahoma", 0, 12)); // NOI18N
        jLabel9.setText("Average Waiting Time         :");

        jLabel10.setFont(new java.awt.Font("Tahoma", 0, 12)); // NOI18N
        jLabel10.setText("Throughput                      :");

        lblAvgTTime.setFont(new java.awt.Font("Tahoma", 0, 12)); // NOI18N

        lblAvgWtime.setFont(new java.awt.Font("Tahoma", 0, 12)); // NOI18N

        lblthroughput.setFont(new java.awt.Font("Tahoma", 0, 12)); // NOI18N

        txtAvgTurn.setEditable(false);
        txtAvgTurn.addActionListener(new java.awt.event.ActionListener() {
```

```
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            txtAvgTurnActionPerformed(evt);
        }
    });

    txtWait.setEditable(false);
    txtWait.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            txtWaitActionPerformed(evt);
        }
    });

    txtThroughput.setEditable(false);

    javax.swing.GroupLayout jPanel5Layout = new javax.swing.GroupLayout(jPanel5);
    jPanel5.setLayout(jPanel5Layout);
    jPanel5Layout.setHorizontalGroup(

jPanel5Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(jPanel5Layout.createSequentialGroup()
            .addGap(73, 73, 73)

.addGroup(jPanel5Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LE
ADING)
                .addComponent(jLabel8)
                .addComponent(jLabel9)
                .addComponent(jLabel10))
            .addGap(18, 18, 18)

.addGroup(jPanel5Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LE
ADING, false)
                .addComponent(txtWait,  javax.swing.GroupLayout.DEFAULT_SIZE,  59,
Short.MAX_VALUE)
                .addComponent(txtThroughput)
                .addComponent(txtAvgTurn))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(jPanel5Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LE
ADING)
                .addComponent(lblAvgTTime)
                .addComponent(lblAvgWtime)
                .addComponent(lblthroughput))
            .addGap(406, 406, 406))
    );
    jPanel5Layout.setVerticalGroup(
```

```
jPanel5Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel5Layout.createSequentialGroup()
               .addGap(27, 27, 27)

.addGroup(jPanel5Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TR
AILING)
                  .addGroup(jPanel5Layout.createSequentialGroup()
                     .addComponent(lblAvgTTime,
javax.swing.GroupLayout.PREFERRED_SIZE,                                    15,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                     .addComponent(lblAvgWtime)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                     .addComponent(lblthroughput))
                  .addGroup(jPanel5Layout.createSequentialGroup()

.addGroup(jPanel5Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BA
SELINE)
                        .addComponent(jLabel8,
javax.swing.GroupLayout.PREFERRED_SIZE,                                    15,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addComponent(txtAvgTurn,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(jPanel5Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BA
SELINE)
                        .addComponent(jLabel9)
                        .addComponent(txtWait,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(jPanel5Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BA
SELINE)
                        .addComponent(jLabel10)
                        .addComponent(txtThroughput,
javax.swing.GroupLayout.PREFERRED_SIZE,
```

```
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))))
            .addContainerGap(143, Short.MAX_VALUE))
      );

      javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(getContentPane());
      getContentPane().setLayout(layout);
      layout.setHorizontalGroup(
         layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
         .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING,
false)
               .addComponent(jPanel2, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
               .addGroup(javax.swing.GroupLayout.Alignment.LEADING,
layout.createSequentialGroup()
                  .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                  .addComponent(jPanel3, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
               .addGroup(javax.swing.GroupLayout.Alignment.LEADING,
layout.createSequentialGroup()
                  .addGap(217, 217, 217)
                  .addComponent(jLabel1, javax.swing.GroupLayout.PREFERRED_SIZE,
268, javax.swing.GroupLayout.PREFERRED_SIZE))
               .addComponent(jPanel4, javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
               .addComponent(jPanel5, javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
            .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
      );
      layout.setVerticalGroup(
         layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
         .addGroup(layout.createSequentialGroup()
            .addContainerGap()
            .addComponent(jLabel1, javax.swing.GroupLayout.PREFERRED_SIZE, 39,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
```

```
.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING,
false)
                .addComponent(jPanel3,          javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                .addComponent(jPanel1,          javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(jPanel2,          javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(jPanel4,          javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
            .addComponent(jPanel5,          javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addContainerGap())
        );

        pack();
    }// </editor-fold>//GEN-END:initComponents

    private void pIdTxtActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_pIdTxtActionPerformed
        // TODO add your handling code here:
    }//GEN-LAST:event_pIdTxtActionPerformed

    private void pNameTxtActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_pNameTxtActionPerformed
        // TODO add your handling code here:
    }//GEN-LAST:event_pNameTxtActionPerformed

    private void pBTimeTxtActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_pBTimeTxtActionPerformed
        // TODO add your handling code here:
    }//GEN-LAST:event_pBTimeTxtActionPerformed

    private void sJFActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_sJFActionPerformed
        List<ProcessObj> list = new ArrayList<>();
        List<ProcessObj> listCompleted = new ArrayList<>();
//        int timeQ = Integer.parseInt(this.tQuantumtxt.getText());
```

```
        DefaultTableModel mdl = (DefaultTableModel) tblAddedProcess.getModel();
        for (int i = 0; i < mdl.getRowCount(); i++) {
            int pId = (int) mdl.getValueAt(i, 0);
            String pName = (String) mdl.getValueAt(i, 1);
            int aTime = (int) mdl.getValueAt(i, 2);
            int bTime = (int) mdl.getValueAt(i, 3);
            ProcessObj p1 = new ProcessObj(pName,pId,aTime,bTime);
            list.add(p1);
//                System.out.println(p1.getId()+" "+p1.getName()+" "+p1.getArrivalTime()+"
"+p1.getBrustTime());
        }
//       mdl.setRowCount(0);
         SJF sjf = new SJF();
        try {
            listCompleted=sjf.allocateResources(list);

        } catch (Exception e) {
            System.out.println("Fail to complete the task=>");
        }
        DefaultTableModel mdlComplete = (DefaultTableModel) tblComplete.getModel();
        mdlComplete.setRowCount(0);
        int totWaitingTime=0;
        int totTurnTime = 0;
        float avgWait,avgTurn;

        for (int i = 0; i < listCompleted.size(); i++) {
            ProcessObj p1 = listCompleted.get(i);
            mdlComplete.addRow(new
Object[]{p1.getId(),p1.getName(),p1.getArrivalTime(),p1.getBrustTime(),p1.getComplet
eTime(),p1.getWaitingTime(),p1.getTurnaroundTime()});
            totWaitingTime = totWaitingTime + p1.getWaitingTime();
            totTurnTime = totTurnTime + p1.getTurnaroundTime();
        }
        avgWait=(float)totWaitingTime/listCompleted.size();
        float avgWaitRound = (float) (Math.round(avgWait * 100.0) / 100.0);

        avgTurn=(float)totTurnTime/listCompleted.size();
        float avgTurnRound = (float) (Math.round(avgTurn * 100.0) / 100.0);

        txtAvgTurn.setText(String.valueOf(avgTurnRound));
        txtWait.setText(String.valueOf(avgWaitRound));
        txtThroughput.setText(calculateThroughput(listCompleted));
    }//GEN-LAST:event_sJFActionPerformed

    private void fCFSActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_fCFSActionPerformed
```

```
        List<ProcessObj> list = new ArrayList<>();
        List<ProcessObj> listCompleted = new ArrayList<>();
        DefaultTableModel mdl = (DefaultTableModel) tblAddedProcess.getModel();
        for (int i = 0; i < mdl.getRowCount(); i++) {
            int pId = (int) mdl.getValueAt(i, 0);
            String pName = (String) mdl.getValueAt(i, 1);
            int aTime = (int) mdl.getValueAt(i, 2);
            int bTime = (int) mdl.getValueAt(i, 3);
            ProcessObj p1 = new ProcessObj(pName,pId,aTime,bTime);
            list.add(p1);
        }
        FCFS f1 = new FCFS();
        try {
            listCompleted=f1.allocateResources(list);

        } catch (Exception e) {
            System.out.println("Fail to complete the task=>");
        }
        DefaultTableModel mdlComplete = (DefaultTableModel) tblComplete.getModel();
        mdlComplete.setRowCount(0);
        int totWaitingTime=0;
        int totTurnTime = 0;
        float avgWait,avgTurn;

        for (int i = 0; i < listCompleted.size(); i++) {
            ProcessObj p1 = listCompleted.get(i);
            mdlComplete.addRow(new
Object[]{p1.getId(),p1.getName(),p1.getArrivalTime(),p1.getBrustTime(),p1.getComplet
eTime(),p1.getWaitingTime(),p1.getTurnaroundTime()});
            totWaitingTime = totWaitingTime + p1.getWaitingTime();
            totTurnTime = totTurnTime + p1.getTurnaroundTime();
        }
        avgWait=(float)totWaitingTime/listCompleted.size();
        float avgWaitRound = (float) (Math.round(avgWait * 100.0) / 100.0);

        avgTurn=(float)totTurnTime/listCompleted.size();
        float avgTurnRound = (float) (Math.round(avgTurn * 100.0) / 100.0);

        txtAvgTurn.setText(String.valueOf(avgTurnRound));
        txtWait.setText(String.valueOf(avgWaitRound));
        txtThroughput.setText(calculateThroughput(listCompleted));
//        avgWait=(float)totWaitingTime/listCompleted.size();
//        avgTurn=(float)totTurnTime/listCompleted.size();
//
//        txtAvgTurn.setText(String.valueOf(avgTurn));
//        txtWait.setText(String.valueOf(avgWait));
```

```
//       txtThroughput.setText(calculateThroughput(listCompleted));

    }//GEN-LAST:event_fCFSActionPerformed

    private void sRTActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_sRTActionPerformed
        List<ProcessObj> list = new ArrayList<>();
        List<ProcessObj> listCompleted = new ArrayList<>();
        DefaultTableModel mdl = (DefaultTableModel) tblAddedProcess.getModel();
        for (int i = 0; i < mdl.getRowCount(); i++) {
            int pId = (int) mdl.getValueAt(i, 0);
            String pName = (String) mdl.getValueAt(i, 1);
            int aTime = (int) mdl.getValueAt(i, 2);
            int bTime = (int) mdl.getValueAt(i, 3);
            ProcessObj p1 = new ProcessObj(pName,pId,aTime,bTime);
            ProcessObj p2 = new ProcessObj(pName,pId,aTime,bTime);
            list.add(p1);
            listCompleted.add(p2);
//                System.out.println(p1.getId()+" "+p1.getName()+" "+p1.getArrivalTime()+"
"+p1.getBrustTime());
        }
//        mdl.setRowCount(0);
        SRTF srt = new SRTF();
        try {

            listCompleted = resetBurst(srt.allocateResources(list, listCompleted));

        } catch (Exception e) {
            System.out.println("Fail to complete the task=>");
        }
        DefaultTableModel mdlComplete = (DefaultTableModel) tblComplete.getModel();
        mdlComplete.setRowCount(0);
        int totWaitingTime=0;
        int totTurnTime = 0;
        float avgWait,avgTurn;

        for (int i = 0; i < listCompleted.size(); i++) {
            ProcessObj p1 = listCompleted.get(i);
            mdlComplete.addRow(new
Object[]{p1.getId(),p1.getName(),p1.getArrivalTime(),p1.getBrustTime(),p1.getComplet
eTime(),p1.getWaitingTime(),p1.getTurnaroundTime()});
            totWaitingTime = totWaitingTime + p1.getWaitingTime();
            totTurnTime = totTurnTime + p1.getTurnaroundTime();
        }
        avgWait=(float)totWaitingTime/listCompleted.size();
        float avgWaitRound = (float) (Math.round(avgWait * 100.0) / 100.0);
```

```
        avgTurn=(float)totTurnTime/listCompleted.size();
        float avgTurnRound = (float) (Math.round(avgTurn * 100.0) / 100.0);

        txtAvgTurn.setText(String.valueOf(avgTurnRound));
        txtWait.setText(String.valueOf(avgWaitRound));
        txtThroughput.setText(calculateThroughput(listCompleted));

//        avgWait=(float)totWaitingTime/listCompleted.size();
//        avgTurn=(float)totTurnTime/listCompleted.size();
//
//        txtAvgTurn.setText(String.valueOf(avgTurn));
//        txtWait.setText(String.valueOf(avgWait));
//        txtThroughput.setText(calculateThroughput(listCompleted));
    }//GEN-LAST:event_sRTActionPerformed

    private void createProcessActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_createProcessActionPerformed
        int pId = Integer.parseInt(this.pIdTxt.getText());
        String pName = this.pNameTxt.getText();
        int aTime = Integer.parseInt(this.pATimeTxt.getText());
        int bTime = Integer.parseInt(this.pBTimeTxt.getText());
        ProcessObj p1 = new ProcessObj(pName,pId,aTime,bTime);

        DefaultTableModel mdl = (DefaultTableModel) tblAddedProcess.getModel();
        mdl.addRow(new
Object[]{p1.getId(),p1.getName(),p1.getArrivalTime(),p1.getBrustTime()});
    }//GEN-LAST:event_createProcessActionPerformed

    private void btnClearTblActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_btnClearTblActionPerformed
        DefaultTableModel mdl = (DefaultTableModel) tblAddedProcess.getModel();
        mdl.setRowCount(0);
    }//GEN-LAST:event_btnClearTblActionPerformed

    private void tQuantumtxtActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_tQuantumtxtActionPerformed
        // TODO add your handling code here:
    }//GEN-LAST:event_tQuantumtxtActionPerformed

    private void rRoundActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_rRoundActionPerformed
        List<ProcessObj> list = new ArrayList<>();
        List<ProcessObj> listCompleted = new ArrayList<>();
        int timeQ = Integer.parseInt(this.tQuantumtxt.getText());
        DefaultTableModel mdl = (DefaultTableModel) tblAddedProcess.getModel();
```

```
        for (int i = 0; i < mdl.getRowCount(); i++) {
            int pId = (int) mdl.getValueAt(i, 0);
            String pName = (String) mdl.getValueAt(i, 1);
            int aTime = (int) mdl.getValueAt(i, 2);
            int bTime = (int) mdl.getValueAt(i, 3);
            ProcessObj p1 = new ProcessObj(pName,pId,aTime,bTime);
            ProcessObj p2 = new ProcessObj(pName,pId,aTime,bTime);
            list.add(p1);
            listCompleted.add(p2);
//                System.out.println(p1.getId()+" "+p1.getName()+" "+p1.getArrivalTime()+"
"+p1.getBrustTime());
        }
//       mdl.setRowCount(0);
         RoundRobin r = new RoundRobin();
        try {
//           listCompleted=r.allocateResources(list, timeQ);
            listCompleted = resetBurst(r.allocateResources(list, listCompleted,timeQ));

        } catch (Exception e) {
            System.out.println("Fail to complete the task=>");
        }
        DefaultTableModel mdlComplete = (DefaultTableModel) tblComplete.getModel();
        mdlComplete.setRowCount(0);
        int totWaitingTime=0;
        int totTurnTime = 0;
        float avgWait,avgTurn;

        for (int i = 0; i < listCompleted.size(); i++) {
            ProcessObj p1 = listCompleted.get(i);
            mdlComplete.addRow(new
Object[]{p1.getId(),p1.getName(),p1.getArrivalTime(),p1.getBrustTime(),p1.getComplet
eTime(),p1.getWaitingTime(),p1.getTurnaroundTime()});
            totWaitingTime = totWaitingTime + p1.getWaitingTime();
            totTurnTime = totTurnTime + p1.getTurnaroundTime();
        }
        avgWait=(float)totWaitingTime/listCompleted.size();
        float avgWaitRound = (float) (Math.round(avgWait * 100.0) / 100.0);

        avgTurn=(float)totTurnTime/listCompleted.size();
        float avgTurnRound = (float) (Math.round(avgTurn * 100.0) / 100.0);

        txtAvgTurn.setText(String.valueOf(avgTurnRound));
        txtWait.setText(String.valueOf(avgWaitRound));
        txtThroughput.setText(calculateThroughput(listCompleted));

//
```

```
//        avgWait=(float)totWaitingTime/listCompleted.size();
//        avgTurn=(float)totTurnTime/listCompleted.size();
//
//        txtAvgTurn.setText(String.valueOf(avgTurn));
//        txtWait.setText(String.valueOf(avgWait));
//        txtThroughput.setText(calculateThroughput(listCompleted));
    }//GEN-LAST:event_rRoundActionPerformed

    private void txtWaitActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_txtWaitActionPerformed
        // TODO add your handling code here:
    }//GEN-LAST:event_txtWaitActionPerformed

    private void txtAvgTurnActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_txtAvgTurnActionPerformed
        // TODO add your handling code here:
    }

    public static void main(String args[]) {
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(Starter.class.getName()).log(java.util.logging.Level.S
EVERE, null, ex);
        } catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(Starter.class.getName()).log(java.util.logging.Level.S
EVERE, null, ex);
        } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(Starter.class.getName()).log(java.util.logging.Level.S
EVERE, null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(Starter.class.getName()).log(java.util.logging.Level.S
EVERE, null, ex);
        }
        //</editor-fold>
```

```
    /* Create and display the form */
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new Starter().setVisible(true);
        }
    });
}

// Variables declaration - do not modify//GEN-BEGIN:variables
private javax.swing.JButton btnClearTbl;
private javax.swing.JButton createProcess;
private javax.swing.JButton fCFS;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel10;
private javax.swing.JLabel jLabel11;
private javax.swing.JLabel jLabel2;
private javax.swing.JLabel jLabel3;
private javax.swing.JLabel jLabel4;
private javax.swing.JLabel jLabel5;
private javax.swing.JLabel jLabel6;
private javax.swing.JLabel jLabel7;
private javax.swing.JLabel jLabel8;
private javax.swing.JLabel jLabel9;
private javax.swing.JPanel jPanel1;
private javax.swing.JPanel jPanel2;
private javax.swing.JPanel jPanel3;
private javax.swing.JPanel jPanel4;
private javax.swing.JPanel jPanel5;
private javax.swing.JScrollPane jScrollPane1;
private javax.swing.JScrollPane jScrollPane2;
private javax.swing.JScrollPane jScrollPane3;
private javax.swing.JLabel lblAvgTTime;
private javax.swing.JLabel lblAvgWtime;
private javax.swing.JLabel lblthroughput;
private javax.swing.JTextField pATimeTxt;
private javax.swing.JTextField pBTimeTxt;
private javax.swing.JTextField pIdTxt;
private javax.swing.JTextField pNameTxt;
private javax.swing.JButton rRound;
private javax.swing.JButton sJF;
private javax.swing.JButton sRT;
private javax.swing.JTextField tQuantumtxt;
private javax.swing.JTable tblAddedProcess;
private javax.swing.JTable tblComplete;
private javax.swing.JTextField txtAvgTurn;
private javax.swing.JTextField txtThroughput;
```

```
    private javax.swing.JTextField txtWait;
    // End of variables declaration//GEN-END:variables

    private String calculateThroughput(List<ProcessObj> listCompleted) {
        List<Integer> completedTimes = new ArrayList<>();
        for (int i = 0; i < listCompleted.size(); i++) {
            completedTimes.add(listCompleted.get(i).getCompleteTime());
        }
        Collections.sort(completedTimes);
        float tPut = (float) completedTimes.size()/completedTimes.get(completedTimes.size()-1);
        float roudTput = (float) (Math.round(tPut * 100.0) / 100.0);
        return Float.toString(roudTput);
    }
    private List<ProcessObj> resetBurst(List<ProcessObj> recive) {
        DefaultTableModel mdl = (DefaultTableModel) tblAddedProcess.getModel();
        List<ProcessObj> sendList = new ArrayList<>();
        for (int i = 0; i < mdl.getRowCount(); i++) {
            String pName = (String) mdl.getValueAt(i, 1);
            int bTime = (int) mdl.getValueAt(i, 3);

            for (ProcessObj recive1 : recive) {
                if (pName.equals(recive1.getName())) {
                    recive1.setBrustTime(bTime);
                    recive1.setWaitingTime(recive1.getTurnaroundTime()- bTime);
                    sendList.add(recive1);
                }

            }
        }
        return sendList;
    }
}



    private ProcessObj selectTheBest() {
        List<Integer> bTimelist = new ArrayList<>();
        if(!q.isEmpty()){
            for (int i = 0; i < q.size(); i++)
            {
                bTimelist.add(q.get(i).getBrustTime());
            }
            Collections.sort(bTimelist);

            for (int i = 0; i < q.size(); i++) {
```

```
            if (bTimelist.get(0)== q.get(i).getBrustTime()) {
               if (bTimelist.get(0)==1) {
                  for (int j = 0; j < completed.size(); j++) {
                     if (q.get(i).getId()==completed.get(j).getId()) {
                        ProcessObj p = completed.get(j);
                        p.setCompleteTime(time+1);
                        completed.set(j, p);
                     }
                  }
                  return q.remove(i);
               } else {
                  ProcessObj p = q.get(i);
                  ProcessObj pNext = p;
                  pNext.setBrustTime(p.getBrustTime()-1);
                  q.set(i, pNext);
                  return p;

               }
            }
         }
      }
      return null;
   }

}
```

Round Robin:

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.concurrent.TimeUnit;

public class RoundRobin {

   int time = 0;
   int timeQuantum;
   List<ProcessObj> q = new ArrayList<>();
   List<ProcessObj> cloneq = new ArrayList<>();
   List<ProcessObj> list = new ArrayList<>();
   List<ProcessObj> completed = new ArrayList<>();

   public List<ProcessObj> allocateResources(List<ProcessObj> l, List<ProcessObj> c,
int timeQ) throws InterruptedException {
      System.out.println("Started the round robin algorithm--->");
```

```
        list.addAll(l);
        completed.addAll(c);
        timeQuantum=timeQ;
        while (list.size() > 0 || q.size() > 0) {
            addToQueue();
            while (q.isEmpty()) {
                System.out.println("------Waiting to recive a process---------");
                TimeUnit.SECONDS.sleep(1);
                time++;
                addToQueue();
            }
             selectTheBest();
        }
        for (int j = 0; j < completed.size(); j++) {
            ProcessObj p = completed.get(j);
            p.setBrustTime(l.get(j).getBrustTime());
            p.setTurnaroundTime(p.getCompleteTime()- p.getArrivalTime());
            p.setWaitingTime(p.getTurnaroundTime()- p.getBrustTime());
            completed.set(j, p);
            }

        for (int i = 0; i < completed.size(); i++) {
            System.out.println(" process Arrival time = "+completed.get(i).getArrivalTime());
            System.out.println(" process completed time =
"+completed.get(i).getCompleteTime());
            System.out.println(" process turnaround time =
"+completed.get(i).getTurnaroundTime());
            System.out.println(" process burst time = "+completed.get(i).getBrustTime());
            System.out.println(" process waiting time =
"+completed.get(i).getWaitingTime());
            System.out.println("-------------------------------------------------------");
        }
        return completed;
    }

    public void addToQueue() {
        if (!list.isEmpty()) {
            for (int i = 0; i < list.size(); i++) {
                ProcessObj p = list.get(i);
                if (time >= p.getArrivalTime()) {
                    q.add(p);
                    list.remove(i);
                }
            }
        }
    }
```

```
    private void selectTheBest() throws InterruptedException {
        List<Integer> bTimelist = new ArrayList<>();
        if (!q.isEmpty()) {
            for (int i = 0; i < q.size(); i++) {
                bTimelist.add(q.get(i).getBrustTime());
            }
            Collections.sort(bTimelist);

            for (int i = 0; i < q.size(); i++) {
                if (bTimelist.get(0) == q.get(i).getBrustTime()) {
                    if (bTimelist.get(0) > timeQuantum) {
                        ProcessObj p = q.get(i);
                        ProcessObj pNext = p;
                        pNext.setBrustTime(p.getBrustTime() - timeQuantum);
                        q.set(i, pNext);
                         run(p,timeQuantum);

                    } else if (bTimelist.get(0) == timeQuantum) {
                        for (int j = 0; j < completed.size(); j++) {
                            if (q.get(i).getId() == completed.get(j).getId()) {
                                ProcessObj p = completed.get(j);
                                p.setCompleteTime(time + timeQuantum);
                                completed.set(j, p);
                            }
                        }
                        run(q.remove(i),timeQuantum);

                    } else if (bTimelist.get(0) < timeQuantum) {
                        for (int j = 0; j < completed.size(); j++) {
                            if (q.get(i).getId() == completed.get(j).getId()) {
                                ProcessObj p = completed.get(j);
                                p.setCompleteTime(time + bTimelist.get(0));
                                completed.set(j, p);
                            }
                        }
                        run(q.remove(i),bTimelist.get(0));
                    }
                }
            }
        }
    }

    private void run(ProcessObj processRun, Integer t) throws InterruptedException {
        System.out.println("Process which name - " + processRun.getName() + " and Id = "
+ processRun.getId() + " is going run");
```

```
        for (int j = 0; j < t; j++) {
                System.out.println("------running a seccond--------");
                TimeUnit.SECONDS.sleep(1);
                time++;
                addToQueue();
            }
    }

}

public class ProcessObj
{
    private String name;
    private int id;
    private int arrivalTime;
    private int brustTime;
    private int completeTime;
    private int turnaroundTime;
    private int waitingTime;

    public ProcessObj(String name, int id, int arrivalTime, int brustTime){
        this.id = id;
        this.name = name;
        this.arrivalTime= arrivalTime;
        this.brustTime = brustTime;
    }

    public ProcessObj(ProcessObj obj) {
        throw new UnsupportedOperationException("Not supported yet."); //To change
body of generated methods, choose Tools | Templates.
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }


    public int getArrivalTime() {
        return arrivalTime;
    }

    public void setArrivalTime(int arrivalTime) {
        this.arrivalTime = arrivalTime;
    }
```

```
    public int getBrustTime() {
        return brustTime;
    }


    public void setBrustTime(int brustTime) {
        this.brustTime = brustTime;
    }
    public int getCompleteTime() {
        return completeTime;
    }
    public void setCompleteTime(int completeTime) {
        this.completeTime = completeTime;
    }
    public int getTurnaroundTime() {
        return turnaroundTime;
    }
    public void setTurnaroundTime(int turnaroundTime) {
        this.turnaroundTime = turnaroundTime;
    }
    public int getWaitingTime() {
        return waitingTime;
    }
    public void setWaitingTime(int waitingTime) {
        this.waitingTime = waitingTime;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    @Override
    public String toString() {
        return super.toString(); //To change body of generated methods, choose Tools | Templates.
    }
}
```

Main Class:

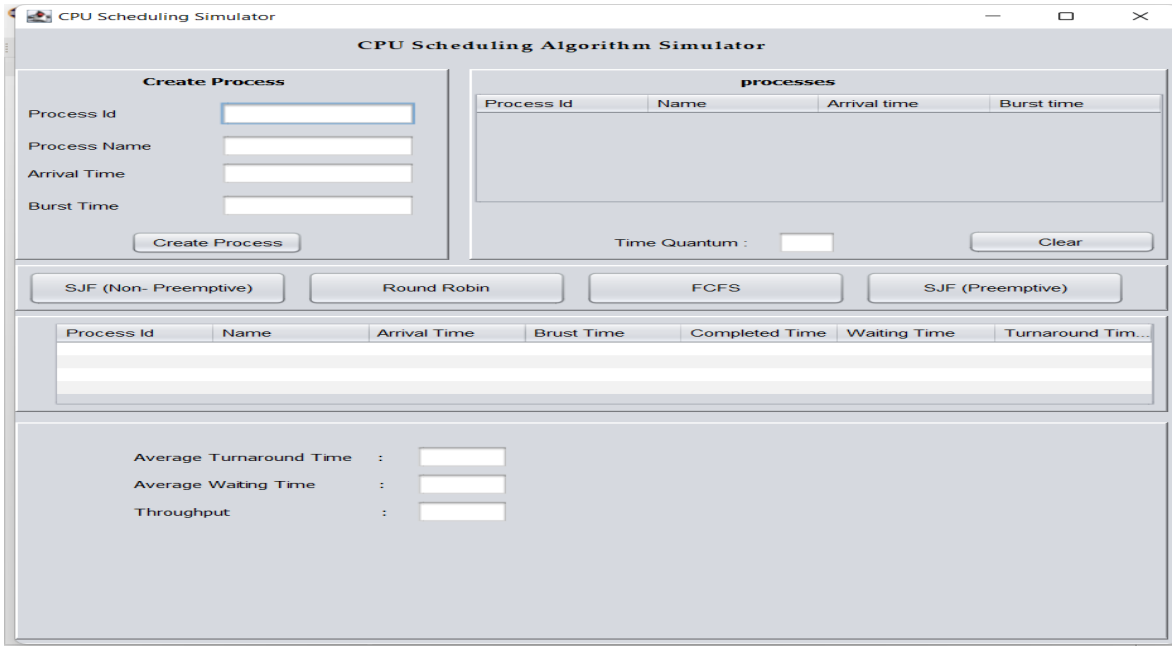
```
import java.util.ArrayList;
import java.util.List;

public class ProcessSchedulingSimulator {
```

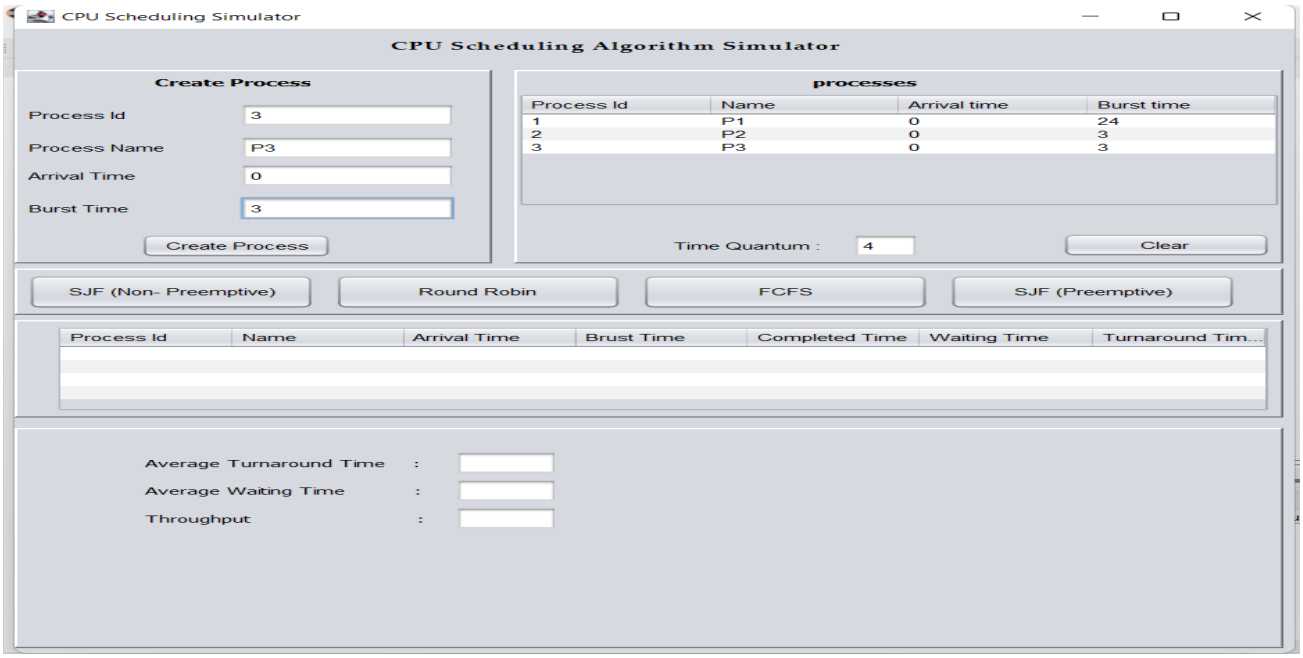
```
    public static void main(String[] args) throws InterruptedException
{
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new Starter().setVisible(true);
            }
        });
    }

}
```

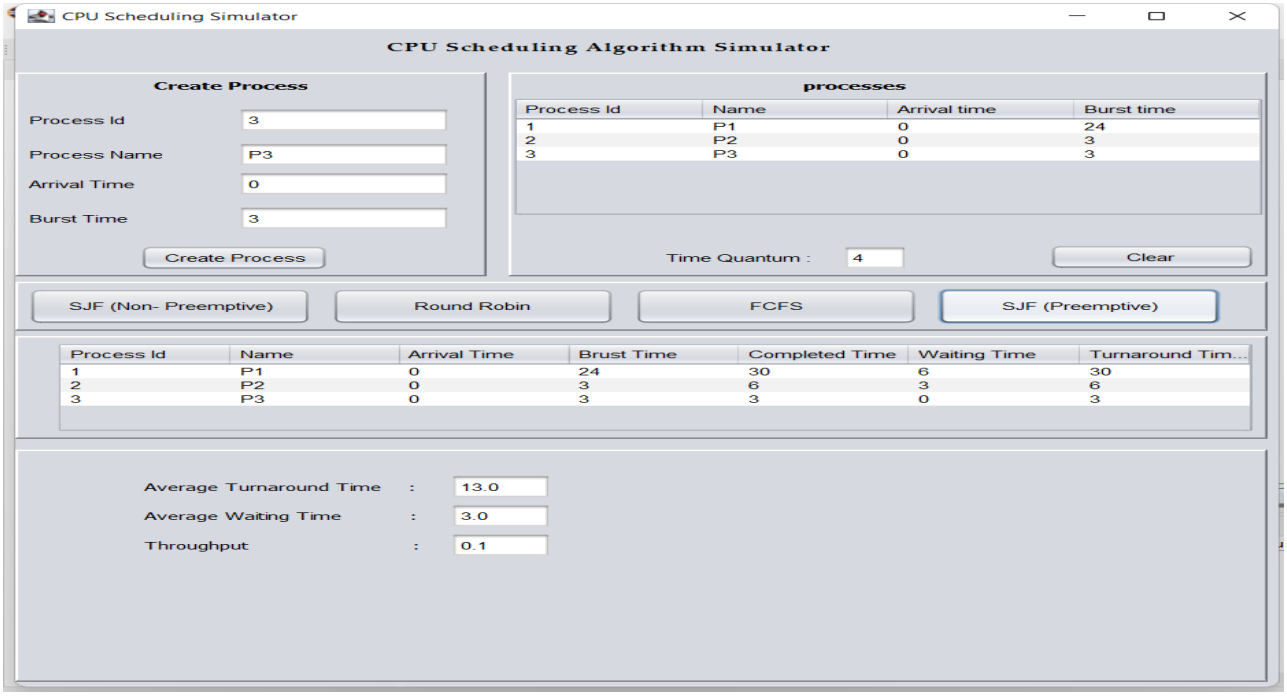
2.2.Output



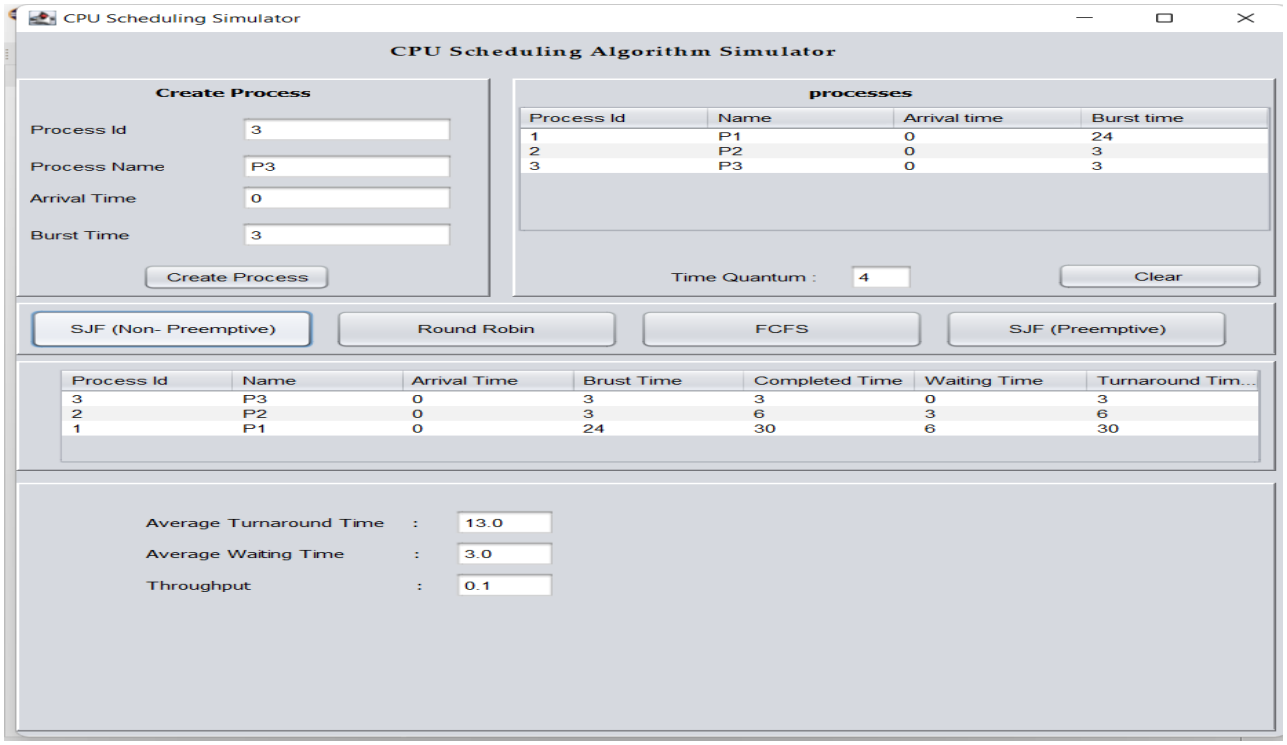
(2) GUI Pop up Window



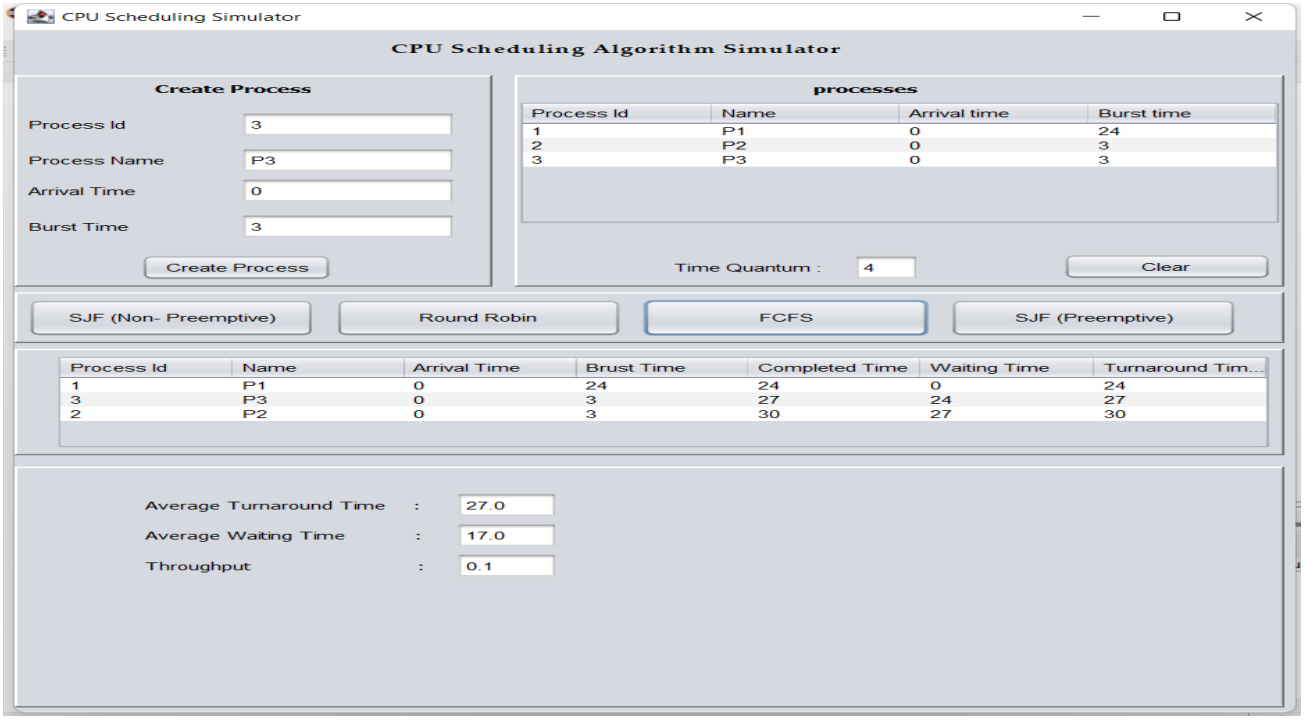
(3) Creating Processes



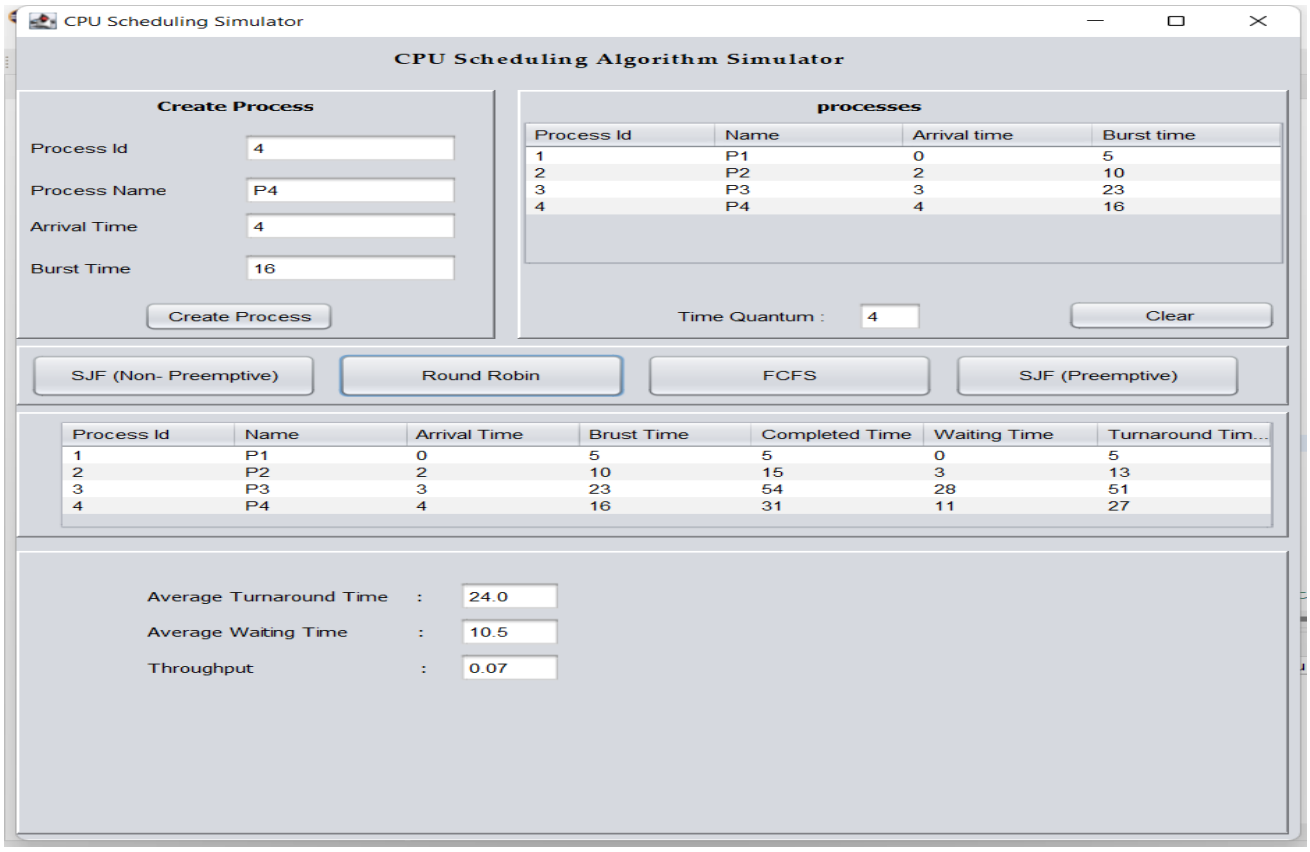
(4) SJF (Preemptive)



(5) SJF (Non-Preemptive)



(6) FCFS



(7) Round Robin

3.CONCLUSION:

Hence, CPU Scheduling Algorithms are implemented using Java language and the simulation is also shown successfully. The snapshot of processes queued according to scheduling algorithms are scheduled and verified.

References:

- https://www.javatpoint.com/os-cpu-scheduling#
- https://www.geeksforgeeks.org/cpu-scheduling-in-operating-systems/
- https://www.tutorialspoint.com/operating_system/os_process_scheduling_algorithms.htm
- https://www.guru99.com/cpu-scheduling-algorithms.html

****