

Table of Contents

S-NO	Topic	Page Number
1.	Introduction	3
2.	Scope	3
3.	Hardware / Software requirements	4
4.1	Tools	4
4.2	Libraries, IDE	4
4.	Module Description	4-8
5.	Screen Shots	8-18
6.	Task Sheet	19

1. Introduction:

A Process used to decide which program is going to take place in execution and which process on hold is called CPU Scheduling Algorithm. Based on designed algorithms, it is made sure that CPU always has some process to execute and it's not free. CPU scheduler is responsible for selecting processes.

There are 6 types of scheduling algorithms:

- I. First come First Serve (FCFS)
- II. Shortest Job First (SJF)
- III. Shortest Remaining Job First (SRTF)
- IV. Round Robin (RR)
- V. Priority Scheduling (PS)
- VI. Multilevel queue Scheduling (MQ)

The algorithms used in project are FCFS, SJF, RR, PS.

IDE used: *Net Beans (version 12.2)*

2. Scope:

The Application helps in arranging series of processes in accordance with the algorithms. It help in creating efficient environment for CPU, due to which processes are signed accordingly to and CPU does not remain idle. All algorithms implemented have their own unique efficient ways of dealing with the process execution.

User can select algorithm and processes will be assigned in that manner to CPU from ready queue.

3. Hardware / Software Requirements:

3.1 Tools:

Hardware Requirements: A Windows Machine.

Software Requirements: Windows OS.

3.2 Libraries, IDE:

IDE: net beans (version 12.2)

Libraries javax.swing.JOptionPane, javax.swing.table.DefaultTableModel, java.util.Collections, java.util.List, java.util.ArrayList, javax.swing.JPanel

4. Module Description:

CPU SCHEDULING SYSTEM

Process	Arrival	Burst	Priority	WTT	TAT
---------	---------	-------	----------	-----	-----

Process:

Arrival:

Burst:

Priority:

Add Delete

Edit Compute

Average Waiting Time: NULL

Average Turnaround Time: NULL

Select a Process

FCFS

Gant Chart

Following window is what appears after the execution of Program. The left side shows some labels and text boxes in which we enter Process name, Arrival time, Burst time and Priority (if required).

Towards the bottom of the form we see buttons that are used to add, delete, edit and execute process. Along with that is attached drop down menu which helps user select which algorithm user wants to execute.

The right side of the form contains a grid which displays data in tabular format after user enters the data. And once the user computes the values respected algorithm is applied to the processes.

Finally the portion with the heading “Gant Chart” shows graphical representation of the scheduling algorithm that is executed.

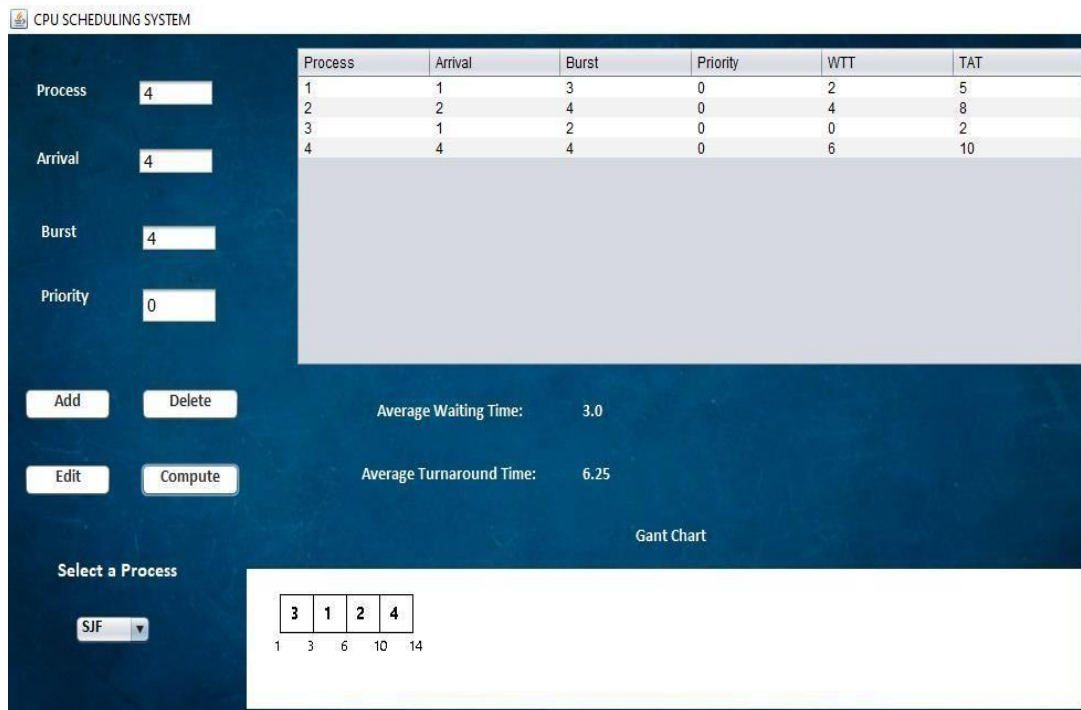
- **Execution of First Come First Serve Algorithm:**

This algorithm works on the basis of arrival time and is non preemptive scheduling algorithm. The output is shown below.



- **Execution of Shortest Job First Algorithm:**

This algorithm works on the basis of burst time. It is a non-preemptive scheduling algorithm. The output is shown below.



- **Execution of Round Robin Algorithm:**

This algorithm works on the basis of time quantum and is preemptive scheduling algorithm. The output is shown below.

CPU SCHEDULING SYSTEM

Process: 4
Arrival: 4
Burst: 1
Priority: 0

Add Delete
Edit Compute

Average Waiting Time: NULL
Average Turnaround Time: NULL

Select a Process
RR

Gant Chart

Input
Time Quantum: 2
OK Cancel

Process	Arrival	Burst	Priority	WTT	TAT
1	0	5	0		
2	1	4	0		
3	2	2	0		
4	4	1	0		

CPU SCHEDULING SYSTEM

Process: 4
Arrival: 4
Burst: 1
Priority: 0

Add Delete
Edit Compute

Average Waiting Time: 4.75
Average Turnaround Time: 7.75

Select a Process
RR

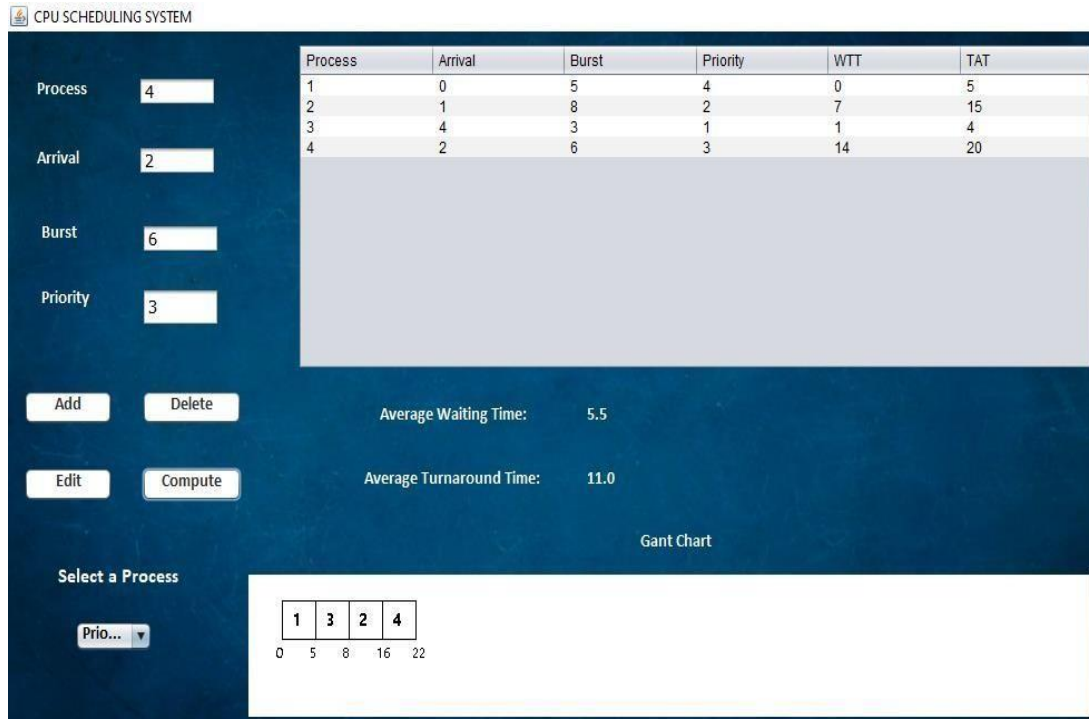
Gant Chart

Process	Arrival	Burst	Priority	WTT	TAT
1	0	5	0	7	12
2	1	4	0	6	10
3	2	2	0	2	4
4	4	1	0	4	5

1 2 3 1 4 2 1
0 2 4 6 8 9 11 12

- Execution of Priority Scheduling Algorithm:**

This algorithm works on the basis of priority and is non-preemptive scheduling algorithm. The output is shown below.



5. Screen Shots:

```

1  import java.util.ArrayList;
2  import java.util.List;
3
4  public abstract class CPUScheduler
5  {
6      private final List<Row> rows;
7      private final List<Event> timeline;
8      private int timeQuantum;
9
10     public CPUScheduler()
11     {
12         rows = new ArrayList();
13         timeline = new ArrayList();
14         timeQuantum = 1;
15     }
16
17     public boolean add(Row row)
18     {
19         return rows.add(row);
20     }
21
22     public void setTimeQuantum(int timeQuantum)
23     {
24         this.timeQuantum = timeQuantum;
25     }
26
27     public int getTimeQuantum()
28     {
29         return timeQuantum;
30     }
31
32     public double round(double value, int places) {
33         if (places < 0) throw new IllegalArgumentException();
34     }
35

```



```

36     long factor = (long) Math.pow(10, places);
37     value = value * factor;
38     long tmp = Math.round(value);
39     return (double) tmp / factor;
40 }
41
42
43 public double getAverageWaitingTime()
44 {
45     double avg = 0.0;
46
47     for (Row row : rows)
48     {
49         avg += row.getWaitingTime();
50     }
51     // return avg / rows.size();
52     return round(avg / rows.size(), 2);
53 }
54
55 public double getAverageTurnAroundTime()
56 {
57     double avg = 0.0;
58
59     for (Row row : rows)
60     {
61         avg += row.getTurnaroundTime();
62     }
63
64     return round(avg / rows.size(), 2);
65     // return avg / rows.size();
66 }
67
68 public Event getEvent(Row row)
69 {
70     for (Event event : timeline)

```

```

70     for (Event event : timeline)
71     {
72         if (row.getProcessName().equals(event.getProcessName()))
73         {
74             return event;
75         }
76     }
77
78     return null;
79 }
80
81 public Row getRow(String process)
82 {
83     for (Row row : rows)
84     {
85         if (row.getProcessName().equals(process))
86         {
87             return row;
88         }
89     }
90
91     return null;
92 }
93
94 public List<Row> getRows()
95 {
96     return rows;
97 }
98
99 public List<Event> getTimeline()
100 {
101     return timeline;
102 }
103
104 //hr page me istamal kr skian yahan pr define nhi howawa
105
106 @
107 }

```

Start Page X Design.java X Event.java X Utility.java X hodi.pg X gent.java X CPUScheduler.java X

Source History

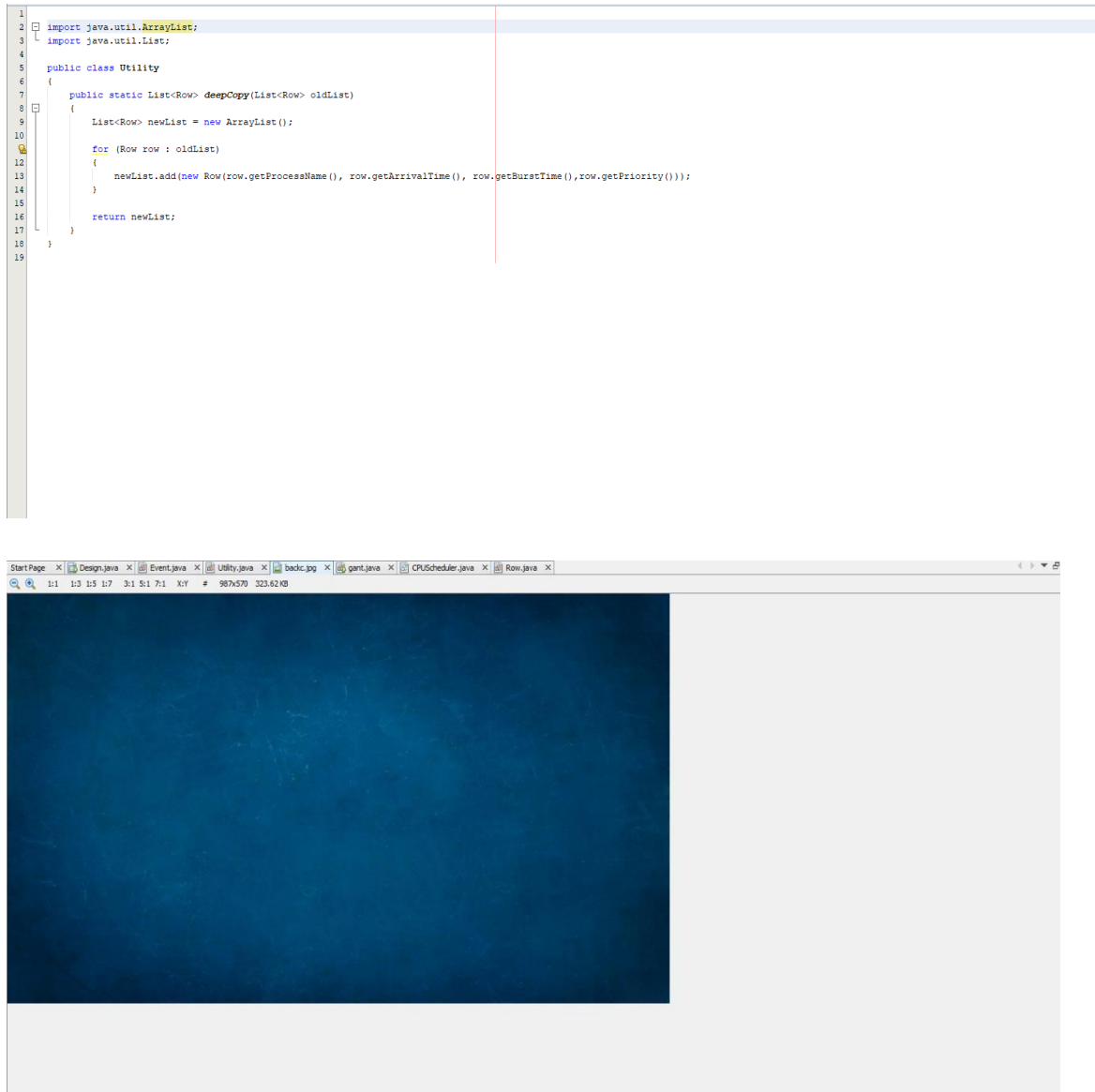
```

1 // rows event me save ho rhi hain (ik algo perform ho jaye tou wo rows event me aa k set ho jatin hain)
2 public class Event
3 {
4     private final String processName;
5     private final int startTime;
6     private int finishTime;
7
8     public Event(String processName, int startTime, int finishTime)
9     {
10         this.processName = processName;
11         this.startTime = startTime;
12         this.finishTime = finishTime;
13     }
14
15     public String getProcessName()
16     {
17         return processName;
18     }
19
20     public int getStartTime()
21     {
22         return startTime;
23     }
24
25     public int getFinishTime()
26     {
27         return finishTime;
28     }
29
30     public void setFinishTime(int finishTime)
31     {
32         this.finishTime = finishTime;
33     }
34 }
35

```

```
Start Page X Design.java X Event.java X Utility.java X back.jpg X gantt.java X CPUScheduler.java X Row.java X
Source History
1 public class Row
2 {
3     private String processName;
4     private int arrivalTime;
5     private int burstTime;
6     private int waitingTime;
7     private int turnaroundTime;
8     private int priority;
9
10
11     private Row(String processName, int arrivalTime, int burstTime, int priority, int waitingTime, int turnaroundTime)
12     {
13         this.processName = processName;
14         this.arrivalTime = arrivalTime;
15         this.burstTime = burstTime;
16         this.priority = priority;
17         this.waitingTime = waitingTime;
18         this.turnaroundTime = turnaroundTime;
19     }
20
21     public Row(String processName, int arrivalTime, int burstTime, int priority)
22     {
23         this(processName, arrivalTime, burstTime, priority, 0, 0);
24     }
25
26     public void setBurstTime(int burstTime)
27     {
28         this.burstTime = burstTime;
29     }
30
31     public void setWaitingTime(int waitingTime)
32     {
33         this.waitingTime = waitingTime;
34     }
35 }
```

```
Start Page X Design.java X Event.java X Utility.java X back.jpg X gantt.java X CPUScheduler.java X Row.java X
Source History
36 public void setTurnaroundTime(int turnaroundTime)
37 {
38     this.turnaroundTime = turnaroundTime;
39 }
40
41 public void setPriority(int priority)
42 {
43     this.priority = priority;
44 }
45
46 public String getProcessName()
47 {
48     return this.processName;
49 }
50
51 public int getArrivalTime()
52 {
53     return this.arrivalTime;
54 }
55
56 public int getBurstTime()
57 {
58     return this.burstTime;
59 }
60
61 public int getPriority()
62 {
63     return this.priority;
64 }
65
66 public int getWaitingTime()
67 {
68     return this.waitingTime;
69 }
70
71
72 public int getTurnaroundTime()
73 {
74     return this.turnaroundTime;
75 }
```



```

1
2 import java.awt.Color;
3 import java.awt.Font;
4 import java.awt.Graphics;
5 import java.util.List;
6 import javax.swing.JPanel;
7
8
9 public class gantt
10 {
11     CustomPanel chartPanel;
12
13
14     public gantt()
15     {
16
17         chartPanel = new CustomPanel();
18         chartPanel.setBackground(Color.WHITE);
19
20     }
21
22     public static void main(String[] args)
23     {
24
25     }
26
27
28     class CustomPanel extends JPanel
29     {
30         private List<Event> timeline;
31
32         @Override
33         protected void paintComponent(Graphics g)
34         {
35             super.paintComponent(g);
36
37             if (timeline != null)
38             {

```

```

32         @Override
33         protected void paintComponent(Graphics g)
34         {
35             super.paintComponent(g);
36
37             if (timeline != null)
38             {
39                 //
40                 int width = 30;
41
42                 for (int i = 0; i < timeline.size(); i++)
43                 {
44                     Event event = timeline.get(i);
45                     int x = 30 * (i + 1);
46                     int y = 20;
47
48                     g.drawRect(x, y, 30, 30);
49                     g.setFont(new Font("Segoe UI", Font.BOLD, 13));
50                     g.drawString(event.getProcessName(), x + 10, y + 20);
51                     g.setFont(new Font("Segoe UI", Font.PLAIN, 11));
52                     g.drawString(Integer.toString(event.getStartTime()), x - 5, y + 45);
53
54                     if (i == timeline.size() - 1)
55                     {
56                         g.drawString(Integer.toString(event.getFinishTime()), x + 27, y + 45);
57                     }
58                 }
59             }
60
61         }
62     }
63
64     public void setTimeline(List<Event> timeline)
65     {
66         this.timeline = timeline;
67         repaint();
68     }
69

```

```

65     public void setTimeline(List<Event> timeline)
66     {
67         this.timeline = timeline;
68         repaint();
69     }
70
71
72

```

```

StartPage X Design.java X Event.java X Utility.java X back.jpg X gantt.java X CPUScheduler.java X Row.java X FirstComeFirstServe.java X Priority.java X RoundRobin.java X ShortestJobFirst.java X
Source History
1
2 import java.util.Collections;
3 import java.util.List;
4
5 public class FirstComeFirstServe extends CPUScheduler
6 {
7     @Override
8     public void process()
9     {
10         Collections.sort(this.getRows(), (Object o1, Object o2) -> {
11             if (((Row) o1).getArrivalTime() == ((Row) o2).getArrivalTime())
12             {
13                 return 0;
14             }
15             else if (((Row) o1).getArrivalTime() < ((Row) o2).getArrivalTime())
16             {
17                 return -1;
18             }
19             else
20             {
21                 return 1;
22             }
23         });
24
25         List<Event> timeline = this.getTimeline();
26
27         for (Row row : this.getRows())
28         {
29             if (timeline.isEmpty())
30             {
31                 timeline.add(new Event(row.getProcessName(), row.getArrivalTime(), row.getArrivalTime() + row.getBurstTime()));
32             }
33             else
34             {
35                 Event event = timeline.get(timeline.size() - 1);
36                 timeline.add(new Event(row.getProcessName(), event.getFinishTime(), event.getFinishTime() + row.getBurstTime()));
37             }
38         }

```

```

StartPage X Design.java X Event.java X Utility.java X back.jpg X gantt.java X CPUScheduler.java X Row.java X FirstComeFirstServe.java X Priority.java X RoundRobin.java X ShortestJobFirst.java X
Source History
23
24
25         List<Event> timeline = this.getTimeline();
26
27         for (Row row : this.getRows())
28         {
29             if (timeline.isEmpty())
30             {
31                 timeline.add(new Event(row.getProcessName(), row.getArrivalTime(), row.getArrivalTime() + row.getBurstTime()));
32             }
33             else
34             {
35                 Event event = timeline.get(timeline.size() - 1);
36                 timeline.add(new Event(row.getProcessName(), event.getFinishTime(), event.getFinishTime() + row.getBurstTime()));
37             }
38         }
39
40         for (Row row : this.getRows())
41         {
42             row.setWaitingTime(this.getEvent(row).getStartTime() - row.getArrivalTime());
43             row.setTurnaroundTime(row.getWaitingTime() + row.getBurstTime());
44         }
45     }
46 }
47

```

```

1  import java.util.ArrayList;
2  import java.util.Collections;
3  import java.util.List;
4
5  public class ShortestJobFirst extends CPUScheduler
6  {
7
8      @Override
9      // to sort process according to arrival time
10     public void process()
11     {
12         // Current row jo aye osko sort karain
13         Collections.sort(this.getRows(), (Object o1, Object o2) -> {
14             if (((Row) o1).getArrivalTime() == ((Row) o2).getArrivalTime())
15             {
16                 return 0;
17             }
18             else if (((Row) o1).getArrivalTime() < ((Row) o2).getArrivalTime())
19             {
20                 return -1;
21             }
22             else
23             {
24                 return 1;
25             }
26         });
27
28         List<Row> rows = Utility.deepCopy(this.getRows());
29         int time = rows.get(0).getArrivalTime();
30
31         while (!rows.isEmpty())
32         {
33             List<Row> availableRows = new ArrayList();
34
35             for (Row row : rows)
36             {
37                 if (row.getArrivalTime() <= time)
38

```

```

39             availableRows.add(row);
40         }
41     }
42
43     Collections.sort(availableRows, (Object o1, Object o2) -> {
44         if (((Row) o1).getBurstTime() == ((Row) o2).getBurstTime())
45         {
46             return 0;
47         }
48         else if (((Row) o1).getBurstTime() < ((Row) o2).getBurstTime())
49         {
50             return -1;
51         }
52         else
53         {
54             return 1;
55         }
56     });
57     // demag chl giya ha
58
59     Row row = availableRows.get(0);
60     this.getTimeline().add(new Event(row.getProcessName(), time, time + row.getBurstTime()));
61     time += row.getBurstTime();
62
63     for (int i = 0; i < rows.size(); i++)
64     {
65         if (rows.get(i).getProcessName().equals(row.getProcessName()))
66         {
67             rows.remove(i);
68             break;
69         }
70     }
71
72     for (Row row : this.getRows())
73     {
74         row.setWaitingTime(this.getEvent(row).getStartTime() - row.getArrivalTime());
75         row.setTurnaroundTime(row.getWaitingTime() + row.getBurstTime());
76

```

```

Start Page | Design.java | Event.java | Utility.java | back.jpg | gantt.java | CPUScheduler.java | Row.java | FirstComeFirstServe.java | Priority.java | RoundRobin.java | ShortestJobFirst.java
Source | History |
1
2 import java.util.Collections;
3 import java.util.HashMap;
4 import java.util.List;
5 import java.util.Map;
6
7 public class RoundRobin extends CPUScheduler
8 {
9     @Override
10    public void process()
11    {
12        Collections.sort(this.getRows(), (Object o1, Object o2) -> {
13            if (((Row) o1).getArrivalTime() == ((Row) o2).getArrivalTime())
14            {
15                return 0;
16            }
17            else if (((Row) o1).getArrivalTime() < ((Row) o2).getArrivalTime())
18            {
19                return -1;
20            }
21            else
22            {
23                return 1;
24            }
25        });
26
27        List<Row> rows = Utility.deepCopy(this.getRows());
28        int time = rows.get(0).getArrivalTime();
29        int timeQuantum = this.getTimeQuantum();
30
31        while (!rows.isEmpty())
32        {
33            Row row = rows.get(0);
34            int bt = (row.getBurstTime() < timeQuantum ? row.getBurstTime() : timeQuantum);
35            this.getTimeline().add(new Event(row.getProcessName(), time, time + bt));
36            time += bt;
37            rows.remove(0);
38        }
39    }
40 }

```

```

Start Page | Design.java | Event.java | Utility.java | back.jpg | gantt.java | CPUScheduler.java | Row.java | FirstComeFirstServe.java | Priority.java | RoundRobin.java | ShortestJobFirst.java
Source | History |
39
40 if (row.getBurstTime() > timeQuantum)
41 {
42     row.setBurstTime(row.getBurstTime() - timeQuantum);
43
44     for (int i = 0; i < rows.size(); i++)
45     {
46         if (rows.get(i).getArrivalTime() > time)
47         {
48             rows.add(i, row);
49             break;
50         }
51         else if (i == rows.size() - 1)
52         {
53             rows.add(row);
54             break;
55         }
56     }
57 }
58
59 Map map = new HashMap();
60
61 for (Row row : this.getRows())
62 {
63     map.clear();
64
65     for (Event event : this.getTimeline())
66     {
67         if (event.getProcessName().equals(row.getProcessName()))
68         {
69             if (map.containsKey(event.getProcessName()))
70             {
71                 int w = event.getStartTime() - (int) map.get(event.getProcessName());
72                 row.setWaitingTime(row.getWaitingTime() + w);
73             }
74             else
75             {
76                 row.setWaitingTime(event.getStartTime() - row.getArrivalTime());
77             }
78             map.put(event.getProcessName(), event.getFinishTime());
79         }
80     }
81
82     row.setTurnaroundTime(row.getWaitingTime() + row.getBurstTime());
83 }
84
85 }
86
87 }

```

```

StartPage X Design.java X Event.java X Utility.java X badc.jpg X gantt.java X CPUScheduler.java X Row.java X FirstComeFirstServe.java X Priority.java X RoundRobin.java X ShortestJobFirst.java X
Source History
1 import java.util.ArrayList;
2 import java.util.Collections;
3 import java.util.List;
4
5 public class Priority extends CPUScheduler {
6
7     @Override
8     public void process()
9     {
10         Collections.sort(this.getRows(), (Object o1, Object o2) -> {
11             if (((Row) o1).getArrivalTime() == ((Row) o2).getArrivalTime())
12             {
13                 if (((Row) o1).getPriority() == ((Row) o2).getPriority())
14                 {
15                     return 0;
16                 }
17             }
18             else if (((Row) o1).getPriority() < ((Row) o2).getPriority())
19             {
20                 return -1;
21             }
22             else
23             {
24                 return 1;
25             }
26         });
27
28         else if (((Row) o1).getArrivalTime() < ((Row) o2).getArrivalTime())
29         {
30             return -1;
31         }
32     }
33 }

```

```

StartPage X Design.java X Event.java X Utility.java X badc.jpg X gantt.java X CPUScheduler.java X Row.java X FirstComeFirstServe.java X Priority.java X RoundRobin.java X ShortestJobFirst.java X
Source History
39 }
40 else
41 {
42     return 1;
43 }
44 });
45
46 List<Row> rows = Utility.deepCopy(this.getRows());
47 int time = rows.get(0).getArrivalTime();
48
49 while(!rows.isEmpty())
50 {
51     List<Row> availableRows = new ArrayList();
52
53     for (Row row : rows)
54     {
55         if (row.getArrivalTime() <= time)
56         {
57             availableRows.add(row);
58         }
59     }
60
61     Collections.sort(availableRows, (Object o1, Object o2) -> {
62         if (((Row) o1).getPriority() == ((Row) o2).getPriority())
63         {
64             return 0;
65         }
66         else if (((Row) o1).getPriority() < ((Row) o2).getPriority())
67         {
68             return -1;
69         }
70         else
71         {
72             return 1;
73         }
74     });
75 }
76

```

```

74 }
75
76 });
77
78 Row row = availableRows.get(0);
79 this.getTimeline().add(new Event(row.getProcessName(), time, time + row.getBurstTime()));
80 time += row.getBurstTime();
81
82 for(int i = 0; i < rows.size(); i++)
83 {
84     if (rows.get(i).getProcessName().equals(row.getProcessName()))
85     {
86         rows.remove(i);
87         break;
88     }
89 }
90
91 for (Row row: this.getRows())
92 {
93     row.setWaitingTime(this.getEvent(row).getStartTime() - row.getArrivalTime());
94     row.setTurnaroundTime(row.getWaitingTime() + row.getBurstTime());
95 }
96 }
97
98 }

```



```

1  import javax.swing.JOptionPane;
2  import javax.swing.table.DefaultTableModel;
3
4
5  /**
6   * To change this license header, choose License Headers in Project Properties.
7   * To change this template file, choose Tools | Templates
8   * and open the template in the editor.
9   */
10
11  /**
12   *
13   *
14   */
15  public class Design extends javax.swing.JFrame {
16
17      /**
18       *
19       */
20      DefaultTableModel model;
21      public Design() {
22          initComponents();
23          model=(DefaultTableModel)tbl.getModel();
24          setExtendedState(java.awt.Frame.MAXIMIZED_BOTH);
25      }
26
27      /**
28       * This method is called from within the constructor to initialize the form.
29       * WARNING: Do NOT modify this code. The content of this method is always
30       * regenerated by the Form Editor.
31       */
32      @SuppressWarnings("unchecked")
33      // Generated Code
273
34      private void tfprocessActionPerformed(java.awt.event.ActionEvent evt) {
275          // TODO add your handling code here:
276      }
277

```

```

278
279      private void haddActionPerformed(java.awt.event.ActionEvent evt) {
280          // TODO add your handling code here:
281          model.insertRow(model.getRowCount(),new Object[] {tfprocess.getText(),tfairival.getText(),tfburst.getText(),tfpriority.getText()});
282      }
283
284      private void hdeleteActionPerformed(java.awt.event.ActionEvent evt) {
285          // TODO add your handling code here:
286          model.removeRow(tbl.getSelectedRow());
287      }
288      //grid work
289      private void heditActionPerformed(java.awt.event.ActionEvent evt) {
290          // TODO add your handling code here:
291          model.setValueAt(tfprocess.getText(),tbl.getSelectedRow(),0);
292          model.setValueAt(tfairival.getText(),tbl.getSelectedRow(),1);
293          model.setValueAt(tfburst.getText(),tbl.getSelectedRow(),2);
294          model.setValueAt(tfpriority.getText(),tbl.getSelectedRow(),3);
295      }
296
297      private void hcomputeActionPerformed(java.awt.event.ActionEvent evt) {
298          // TODO add your handling code here:
299          String selected = (String) opt.getSelectedItem();
300
301          CPUScheduler scheduler;
302
303          switch (selected) {
304              case "FCFS":
305                  scheduler = new FirstComeFirstServe();
306                  break;
307              case "SJFF":
308                  scheduler = new ShortestJobFirst();
309                  break;
310              case "Priority":
311                  scheduler = new Priority();
312                  break;
313              case "RR":
314                  scheduler = new RoundRobin();
315                  break;
316          }
317          SchedulerDialog dialog = new SchedulerDialog(scheduler, this);
318          dialog.setVisible(true);

```

```

319
320      String tq = JOptionPane.showInputDialog("Time Quantum");
321      if (tq == null) {
322          return;
323      }
324      scheduler = new RoundRobin();
325      scheduler.setTimeQuantum(Integer.parseInt(tq));
326      break;
327      default:
328          return;
329      }
330
331      for (int i = 0; i < model.getRowCount(); i++)
332      {
333          String process = (String) model.getValueAt(i, 0);
334          int at = Integer.parseInt((String) model.getValueAt(i, 1));
335          int bt = Integer.parseInt((String) model.getValueAt(i, 2));
336          int priority = Integer.parseInt((String) model.getValueAt(i, 3));
337          // in a form of int
338          scheduler.add(new Row(process, at, bt,priority));
339      }
340
341      scheduler.process();
342
343      for (int i = 0; i < model.getRowCount(); i++)
344      {
345          String process = (String) model.getValueAt(i, 0);
346          Row row = scheduler.getRow(process);
347          model.setValueAt(row.getWaitingTime(), i, 4);
348          model.setValueAt(row.getTurnaroundTime(), i, 5);
349      }
350
351      wresult.setText(Double.toString(scheduler.getAverageWaitingTime()));
352      tatresult.setText(Double.toString(scheduler.getAverageTurnaroundTime()));
353
354      gantt.chartPanel.setTimeline(scheduler.getTimeline());
355

```

```

412 private javax.swing.JLabel jLabel11;
413 private javax.swing.JLabel jLabel12;
414 private javax.swing.JLabel jLabel13;
415 private javax.swing.JLabel jLabel14;
416 private javax.swing.JLabel jLabel15;
417 private javax.swing.JLabel jLabel16;
418 private javax.swing.JLabel jLabel17;
419 private javax.swing.JLabel jLabel18;
420 private javax.swing.JLabel jLabel19;
421 private javax.swing.JScrollPane jScrollPane1;
422 private javax.swing.JTextField jTextField1;
423 private javax.swing.JComboBox<String> opt;
424 private javax.swing.JScrollPane scroll;
425 private javax.swing.JLabel tresult;
426 private javax.swing.JTable tbl;
427 private javax.swing.JTextField tfarrival;
428 private javax.swing.JTextField tfburst;
429 private javax.swing.JTextField tfpriority;
430 private javax.swing.JLabel wresult;
431 // End of variables declaration
432 }
433
434
435
436

```

```

355 private void tblMouseClicked(java.awt.event.MouseEvent evt) {
356 // TODO add your handling code here:
357 tfprocess.setText(String.valueOf(model.getValueAt(tbl.getSelectedRow(), 0)));
358 tfarrival.setText(String.valueOf(model.getValueAt(tbl.getSelectedRow(), 1)));
359 tfburst.setText(String.valueOf(model.getValueAt(tbl.getSelectedRow(), 2)));
360 tfpriority.setText(String.valueOf(model.getValueAt(tbl.getSelectedRow(), 3)));
361 }
362
363 private void optActionPerformed(java.awt.event.ActionEvent evt) {
364 // TODO add your handling code here:
365 }
366
367 /**
368 * @param args the command line arguments
369 */
370 public static void main(String args[]) {
371 // Set the Nimbus look and feel
372 LookAndFeel.setLookAndFeel(SwingUtilities.getSystemLookAndFeelClassName());
373
374 // Create and display the form
375 java.awt.EventQueue.invokeLater(new Runnable() {
376     @Override
377     public void run() {
378         new Design().setVisible(true);
379     }
380 });
381 }
382
383 // Variables declaration - do not modify
384 private javax.swing.JButton hadd;
385 private javax.swing.JButton hcompute;
386 private javax.swing.JButton hdelete;
387 private javax.swing.JButton hedit;
388 private javax.swing.JList<String> gant;
389 private javax.swing.JComboBox<String> jComboBox1;
390 private javax.swing.JLabel jLabel1;
391 private javax.swing.JLabel jLabel10;

```

