



CS-2003L-CS-201-BS-CS-4B

Operating Systems Lab Fall
2020

CPU Scheduling System

GROUP MEMBERS

1. **Aaisha Motan Cs-182009**
2. **Muhammad Danish Cs-182019**
3. **Iqra Anwar Cs-182015**
4. **Winona Fernandes Cs-182032**

Report

Lab Instructor: Miss Ansha Zahid

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).

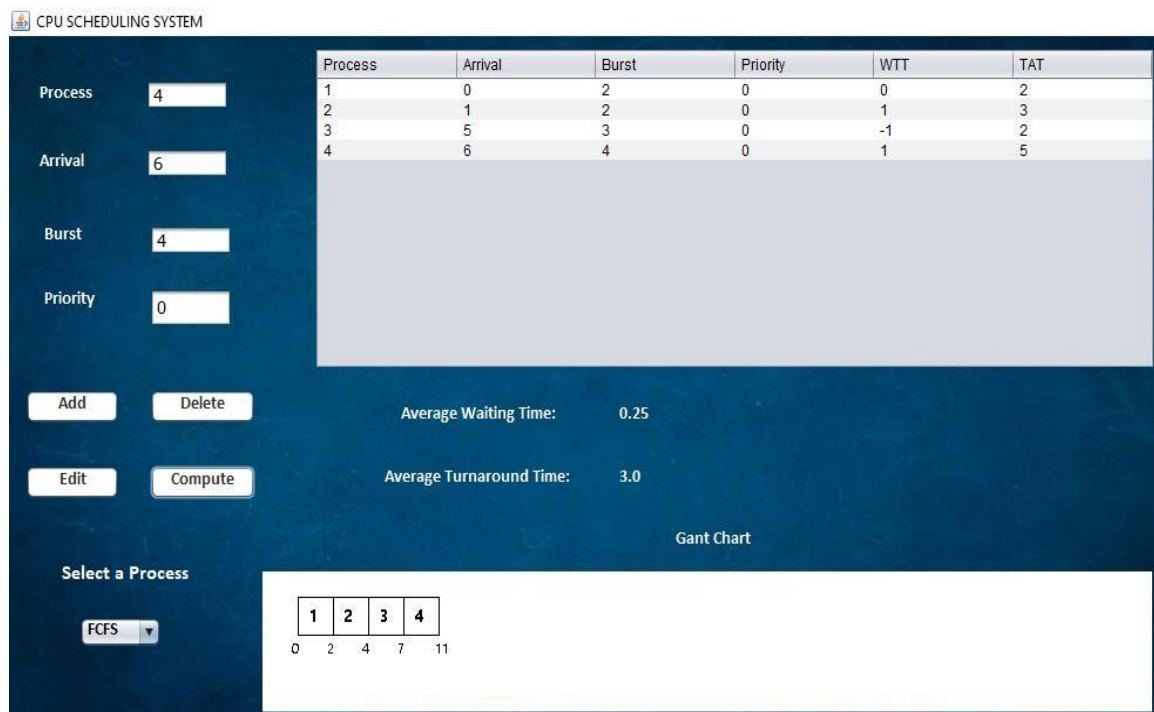
Towards the bottom of the form we see buttons that are used to add, delete, edit and execute process. Along with that is attached drop down menu which helps user select which algorithm user wants to execute.

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.

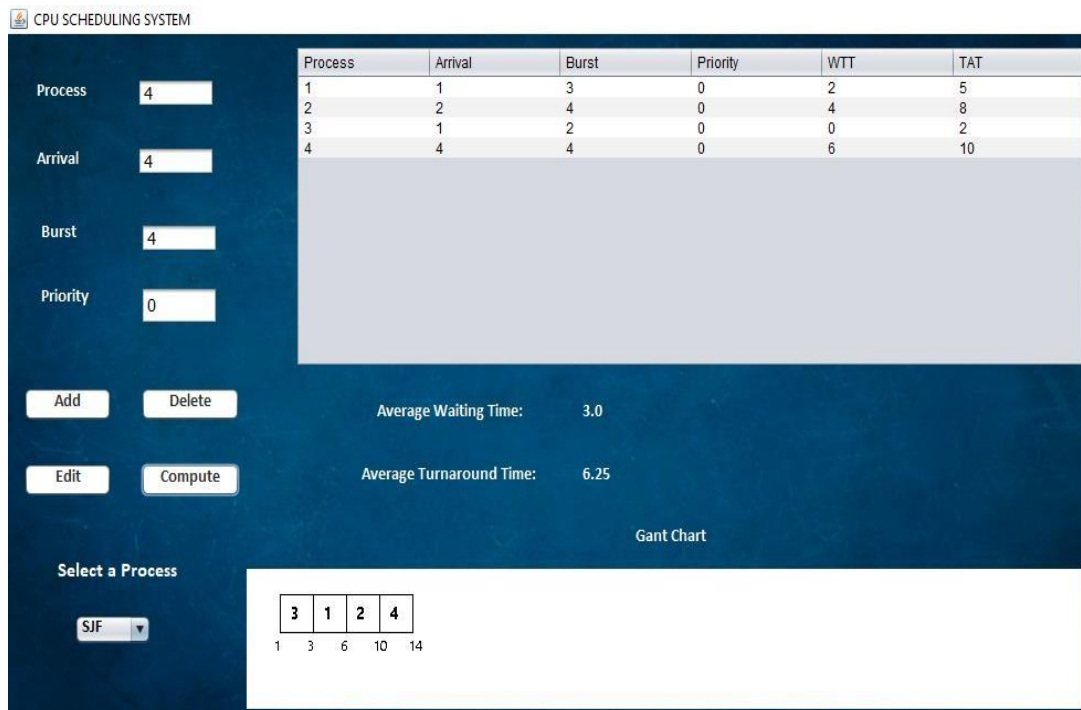
- **Execution of First Come First Serve Algorithm:**

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

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

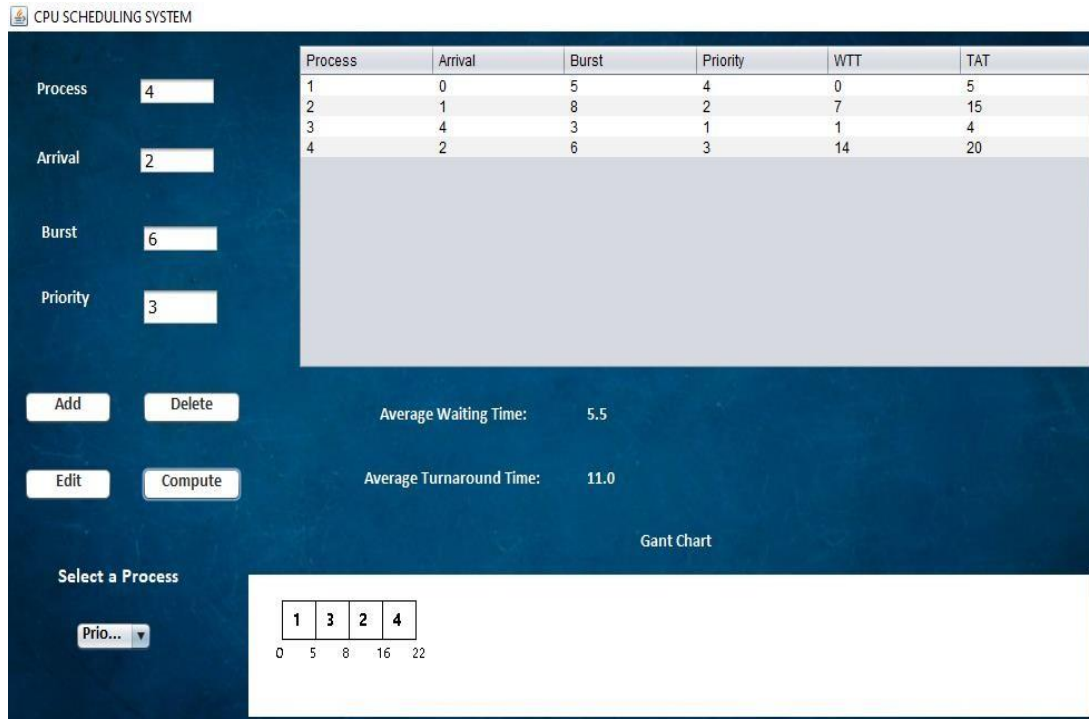
Gant Chart

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
11      public CPUScheduler()
12      {
13          rows = new ArrayList();
14          timeline = new ArrayList();
15          timeQuantum = 1;
16      }
17
18      public boolean add(Row row)
19      {
20          return rows.add(row);
21      }
22
23      public void setTimeQuantum(int timeQuantum)
24      {
25          this.timeQuantum = timeQuantum;
26      }
27
28      public int getTimeQuantum()
29      {
30          return timeQuantum;
31      }
32
33      public double round(double value, int places) {
34          if (places < 0) throw new IllegalArgumentException();
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
44 public double getAverageWaitingTime()
45 {
46     double avg = 0.0;
47
48     for (Row row : rows)
49     {
50         avg += row.getWaitingTime();
51     }
52     // return avg / rows.size();
53     return round(avg / rows.size(), 2);
54 }
55
56 public double getAverageTurnAroundTime()
57 {
58     double avg = 0.0;
59
60     for (Row row : rows)
61     {
62         avg += row.getTurnaroundTime();
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     return null;
78 }
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     return null;
91 }
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 public abstract void process();

```

StartPage X Design.java X Event.java X Utility.java X badcpg X gant.java X CPUScheduler.java X

Source History

```

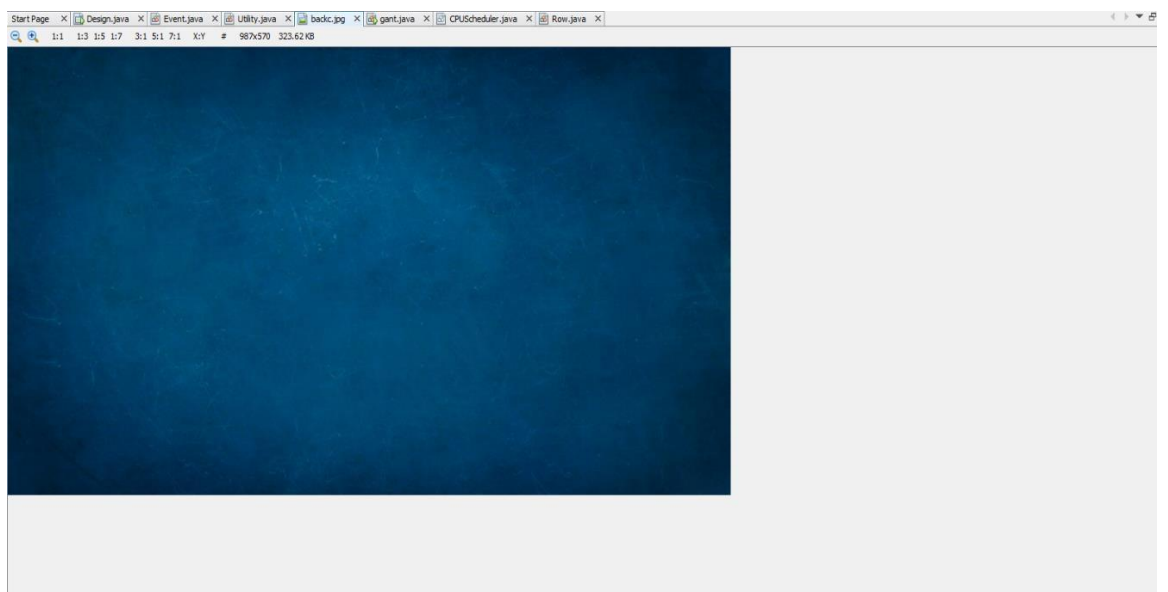
1 // rows event me save ho rhi hain (ik algo perform ho jaye tou wo rows event me aa k set ho jatin hain)
2 public class Event
3 {
4     private final String processName;
5     private final int startTime;
6     private int finishTime;
7
8     public Event(String processName, int startTime, int finishTime)
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 gent.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    private Row(String processName, int arrivalTime, int burstTime, int priority, int waitingTime, int turnaroundTime)
11    {
12        this.processName = processName;
13        this.arrivalTime = arrivalTime;
14        this.burstTime = burstTime;
15        this.priority = priority;
16        this.waitingTime = waitingTime;
17        this.turnaroundTime = turnaroundTime;
18    }
19
20    public Row(String processName, int arrivalTime, int burstTime, int priority)
21    {
22        this(processName, arrivalTime, burstTime, priority, 0, 0);
23    }
24
25    public void setBurstTime(int burstTime)
26    {
27        this.burstTime = burstTime;
28    }
29
30    public void setWaitingTime(int waitingTime)
31    {
32        this.waitingTime = waitingTime;
33    }
34
35
```

```
Start Page X Design.java X Event.java X Utility.java X back.jpg X gent.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.util.ArrayList;
3 import java.util.List;
4
5 public class Utility
6 {
7     public static List<Row> deepCopy(List<Row> oldList)
8     {
9         List<Row> newList = new ArrayList();
10
11         for (Row row : oldList)
12         {
13             newList.add(new Row(row.getProcessName(), row.getArrivalTime(), row.getBurstTime(), row.getPriority()));
14         }
15
16         return newList;
17     }
18 }
19
```



```

1  import java.awt.Color;
2  import java.awt.Font;
3  import java.awt.Graphics;
4  import java.util.List;
5  import javax.swing.JPanel;
6
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                 int width = 30;
40
41                 for (int i = 0; i < timeline.size(); i++)
42                 {
43                     Event event = timeline.get(i);
44                     int x = 30 * (i + 1);
45                     int y = 20;
46
47                     g.drawRect(x, y, 30, 30);
48                     g.setFont(new Font("Segoe UI", Font.BOLD, 13));
49                     g.drawString(event.getProcessName(), x + 10, y + 20);
50                     g.setFont(new Font("Segoe UI", Font.PLAIN, 11));
51                     g.drawString(Integer.toString(event.getStartTime()), x - 5, y + 45);
52
53                     if (i == timeline.size() - 1)
54                     {
55                         g.drawString(Integer.toString(event.getFinishTime()), x + 27, y + 45);
56                     }
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

```

```

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         }
39     }
40 }

```

```

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      @Override
8      // to sort process according to arrival time
9      public void process()
10     {
11         // Current row jo aye osko sort karain
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
30         while (!rows.isEmpty())
31         {
32             List<Row> availableRows = new ArrayList();
33
34             for (Row row : rows)
35             {
36                 if (row.getArrivalTime() <= time)
37

```

```

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

```

```

1  import java.util.Collections;
2  import java.util.HashMap;
3  import java.util.List;
4  import java.util.Map;
5  import java.util.Map;
6
7  public class RoundRobin extends CPUScheduler
8  {
9
10     @Override
11     public void process()
12     {
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         int timeQuantum = this.getTimeQuantum();
31
32         while (!rows.isEmpty())
33         {
34             Row row = rows.get(0);
35             int bt = (row.getBurstTime() < timeQuantum ? row.getBurstTime() : timeQuantum);
36             this.getTimeline().add(new Event(row.getProcessName(), time, time + bt));
37             time += bt;
38             rows.remove(0);
39         }
40     }
41 }

```

```

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

```

```

76     row.setWaitingTime(event.getStartTime() - row.getArrivalTime());
77 }
78
79     map.put(event.getProcessName(), event.getFinishTime());
80 }
81
82     row.setTurnaroundTime(row.getWaitingTime() + row.getBurstTime());
83 }
84 }
85 }
86 }
87 }

```

```

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 }

```

```

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

```



```

Start Page X Design.java X Event.java X Utility.java X back.jpg X genti.java X CPUScheduler.java X Row.java X FirstComeFirstServe.java X Priority.java X RoundRobin.java X ShortestJobFirst.java X Design.java X
Source Design History
1
2 import javax.swing.JOptionPane;
3 import javax.swing.table.DefaultTableModel;
4
5
6 /*
7  * To change this license header, choose License Headers in Project Properties.
8  * To change this template file, choose Tools | Templates
9  * and open the template in the editor.
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
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

```

278
279
280 private void haddActionPerformed(java.awt.event.ActionEvent evt) {
281     // TODO add your handling code here:
282     model.insertRow(model.getRowCount(),new Object[] {tfprocess.getText(),tfairival.getText(),tfburst.getText(),tfpriority.getText()});
283 }
284
285 private void hdeleteActionPerformed(java.awt.event.ActionEvent evt) {
286     // TODO add your handling code here:
287     model.removeRow(tbl.getSelectedRow());
288 }
289
290 //grid work
291 private void heditActionPerformed(java.awt.event.ActionEvent evt) {
292     // TODO add your handling code here:
293     model.setValueAt(tfprocess.getText(), tbl.getSelectedRow(),0);
294     model.setValueAt(tfairival.getText(), tbl.getSelectedRow(),1);
295     model.setValueAt(tfburst.getText(), tbl.getSelectedRow(),2);
296     model.setValueAt(tfpriority.getText(), tbl.getSelectedRow(),3);
297 }
298
299 private void hcomputeActionPerformed(java.awt.event.ActionEvent evt) {
300     // TODO add your handling code here:
301     String selected = (String) opt.getSelectedItem();
302
303     CPUScheduler scheduler;
304
305     switch (selected) {
306         case "FCFS":
307             scheduler = new FirstComeFirstServe();
308             break;
309         case "SJFF":
310             scheduler = new ShortestJobFirst();
311             break;
312         case "Priority":
313             scheduler = new Priority();
314             break;
315         case "RR":
316             scheduler = new RoundRobin();
317             break;
318     }
319
320     String tq = JOptionPane.showInputDialog("Time Quantum");
321     if (tq == null) {
322         return;
323     }
324     scheduler.setTimeQuantum(Integer.parseInt(tq));
325     break;
326     default:
327         return;
328     }
329
330 for (int i = 0; i < model.getRowCount(); i++)
331 {
332     String process = (String) model.getValueAt(i, 0);
333     int at = Integer.parseInt((String) model.getValueAt(i, 1));
334     int bt = Integer.parseInt((String) model.getValueAt(i, 2));
335     int priority = Integer.parseInt((String) model.getValueAt(i, 3));
336     // in a form of int
337     scheduler.add(new Row(process, at, bt,priority));
338 }
339
340 scheduler.process();
341
342 for (int i = 0; i < model.getRowCount(); i++)
343 {
344     String process = (String) model.getValueAt(i, 0);
345     Row row = scheduler.getRow(process);
346     model.setValueAt(row.getWaitingTime(), i, 4);
347     model.setValueAt(row.getTurnaroundTime(), i, 5);
348 }
349
350 wresult.setText(Double.toString(scheduler.getAverageWaitingTime()));
351 tresult.setText(Double.toString(scheduler.getAverageTurnaroundTime()));
352
353 genti.chartPanel.setTimeline(scheduler.getTimeline());
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

```

318
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 tresult.setText(Double.toString(scheduler.getAverageTurnaroundTime()));
353
354 genti.chartPanel.setTimeline(scheduler.getTimeline());
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

```

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     Look and feel setting code (optional)
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 badd;
385 private javax.swing.JButton bcompute;
386 private javax.swing.JButton bdelete;
387 private javax.swing.JButton bedit;
388 private javax.swing.JList gantl;
389 private javax.swing.JComboBox<String> jComboBox1;
390 private javax.swing.JLabel jLabel1;
391 private javax.swing.JLabel jLabel10;
392
393 private javax.swing.JLabel jLabel11;
394 private javax.swing.JLabel jLabel12;
395 private javax.swing.JLabel jLabel13;
396 private javax.swing.JLabel jLabel14;
397 private javax.swing.JLabel jLabel15;
398 private javax.swing.JLabel jLabel16;
399 private javax.swing.JLabel jLabel17;
400 private javax.swing.JLabel jLabel18;
401 private javax.swing.JLabel jLabel19;
402 private javax.swing.JScrollPane jScrollPane1;
403 private javax.swing.JTextField jTextField1;
404 private javax.swing.JComboBox<String> opt;
405 private javax.swing.JScrollPane scroll;
406 private javax.swing.JLabel tresult;
407 private javax.swing.JTable tbl;
408 private javax.swing.JTextField tfarrival;
409 private javax.swing.JTextField tfburst;
410 private javax.swing.JTextField tfpriority;
411 private javax.swing.JTextField tfprocess;
412 private javax.swing.JLabel wresult;
413
414 // End of variables declaration
415 }
416

```

6. Task Sheet:

Names of students	Tasks
1. Aaisha Motan	Shortest Job First, Rows.java. Gant.java
2. Iqra Anwar	First Come First Serve, Design. Java,Gant.java
3. Muhammad Danish	Priority Scheduling Algorithm,Event.java,back.java
4. Winona Fernandes	Round Robin Algorithm, Cpu Scheduler.java, utility.java