

# **End Term Project**

## **On**

### **Design Principles of Operating System (CSE 3249)**

**Submitted by**

**Name : E. Jagadeeswar Patro**  
**Reg. No. : 2241016309**  
**Branch : Computer Science & Engineering**  
**Semester : 5<sup>th</sup>**  
**Section : 2241044**  
**Session : 2024-2025**  
**Admission Batch : 2022**



**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**  
**FACULTY OF ENGINEERING & TECHNOLOGY (ITER)**  
**SIKSHA 'O' ANUSANDHAN DEEMED TO BE UNIVERSITY**  
**BHUBANESWAR, ODISHA – 751030**

**Objective of this Assignment:**

- To design a CPU scheduler for simulating a few CPU scheduling policies.
- Implementation of Banker's algorithm to avoid deadlock.

**Overview of the Project:**

1. One of the main tasks of an operating system is scheduling processes to run on the CPU. The goal of this programming project is to build a program (*use C or Java programming language*) to implement a simulator with different scheduling algorithms discussed in theory. The simulator should select a process to run from the ready queue based on the scheduling algorithm chosen at runtime. Since the assignment intends to simulate a CPU scheduler, it does not require any actual process creation or execution.

2. The goal of this programming project is to build a program (*use C or Java programming language*) to implement banker's algorithms discussed in theory. Create 5 process that request and release resources from the bank. The banker will grant the request only if it leaves the system in a safe state. It is important that shared data be safe from concurrent access. To ensure safe access to shared data, you can use mutex locks.

**Objective 1:** The C program provides an interface to the user to implement the following scheduling policies as per the choice provided:

1. First Come First Served (FCFS)
2. Round Robin (RR)

Appropriate option needs to be chosen from a switch case based menu driven program with an option of “Exit from program” in case 5 and accordingly a scheduling policy will print the Gantt chart and the average waiting time, average turnaround time and average response time. The program will take Process ids, its arrival time, and its CPU burst time as input. For implementing RR scheduling, user also needs to specify the time quantum. Assume that the process ids should be unique for all processes. Each process consists of a single CPU burst (no I/O bursts), and processes are listed in order of their arrival time. Further assume that an interrupted process gets placed at the back of the Ready queue, and a newly arrived process gets placed at the back of the Ready queue as well. The output should be displayed in a formatted way for clarity of understanding and visual.

#### Test Cases:

The program should be able to produce correct answer or appropriate error message corresponding to the following test cases:

1. Consider the set of processes with arrival time (in milliseconds), CPU burst time (in milliseconds), and time quantum = 4ms as shown below.

Process	Arrival time	Burst Time
P1	0	10
P2	1	1
P3	2	2
P4	3	1
P5	6	5

- Input choice 1, and print the Gantt charts that illustrate the execution of these processes using the FCFS scheduling algorithm and then print the average turnaround time, average waiting time and average response time.
- Input choice 2, and print the Gantt charts that illustrate the execution of these processes using the RR scheduling algorithm and then print the average turnaround time, average waiting time and average response time.
- Analyze the results and determine which of the algorithms results in the minimum average waiting time over all processes?

## Program:

### < FCFS.c >

```
FCFS.c
~/DOS_2241016309/Pr... Save

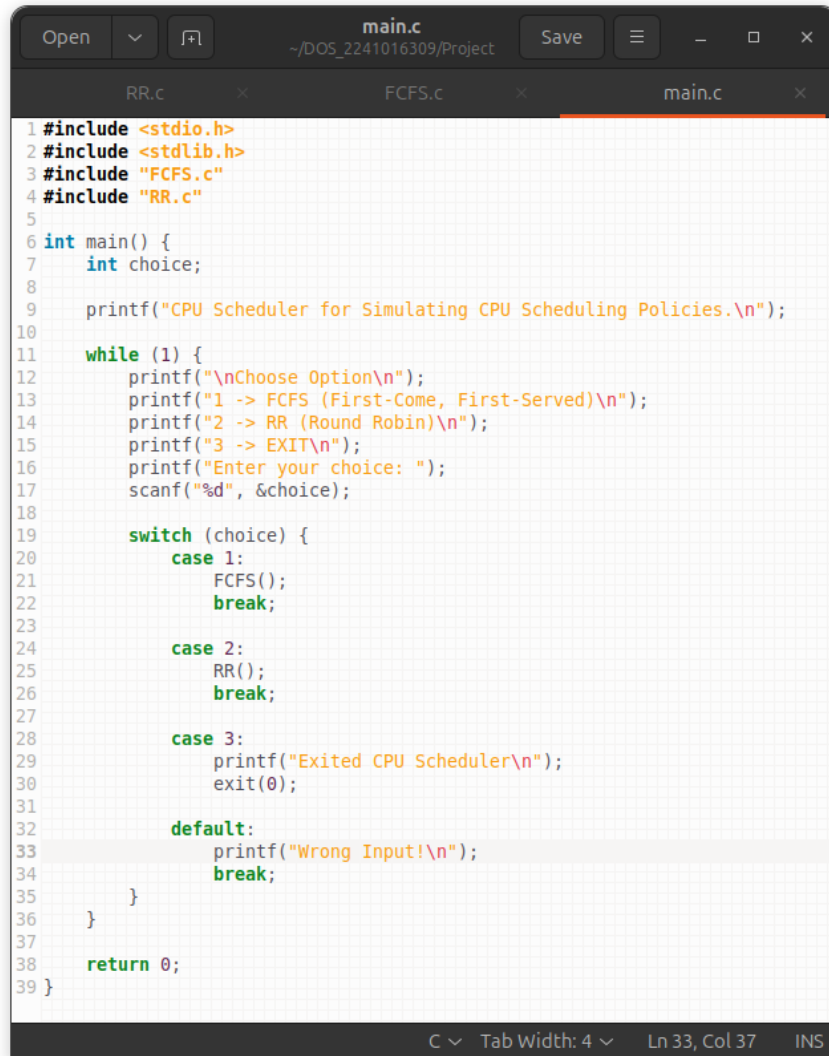
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 typedef struct node
5 {
6     int no;
7     float at, bt;
8     float pc, tat;
9     float wt, rt, rd;
10    struct node *next;
11 } NODE;
12
13 void create_insert(NODE **p, int no, float at, float bt, float *fr)
14 {
15     NODE *q, *r = *p;
16     q = (NODE *)malloc(sizeof(NODE));
17     q->no = no;
18     q->at = at;
19     q->bt = bt;
20     q->rt = *fr - at;
21     q->pc = *fr + bt;
22     q->tat = q->pc - at;
23     q->wt = q->tat - bt;
24     q->rd = q->tat / bt;
25     *fr = *fr + bt;
26     q->next = NULL;
27
28     if (*p == NULL)
29         *p = q;
30     else
31     {
32         while (r->next != NULL)
33             r = r->next;
34         r->next = q;
35     }
36 }
37
38 void gantt_chart(NODE *p, int process)
39 {
40     int i;
41     NODE *r = p;
42
43     printf("\nGantt Chart:\n");
44
45     for (i = 0; i < process; i++)
46         printf("|\\tP%d\\t", p->no), p = p->next;
47     printf("\\n");
48
49     printf("%.1f\\t", r->at);
50     for (i = 0; i < process; i++)
51     {
52         printf("%.1f\\t", r->pc);
53         r = r->next;
54     }
55     printf("\\n");
56 }
57
58 void display(NODE *p, int process)
59 {
60     float ttat = 0, twt = 0, trt = 0, tbt = 0;
61
62     printf("\\nProcess Details:\\n");
63     printf("*****\\n");
64     printf("Pro\\tArTi\\tBuTi\\tTaTi\\tWtTi\\tRTi\\n");
65     printf("*****\\n");
66
67     while (p != NULL)
68     {
69         printf("%d\\t", p->no);
70         printf("%.2f\\t", p->at);
71         printf("%.2f\\t", p->bt);
72         printf("%.2f\\t", p->tat);
73         printf("%.2f\\t", p->wt);
74         printf("%.2f\\t", p->rt);
75
76         ttat += p->tat;
77         twt += p->wt;
78         trt += p->rt;
79         tbt += p->bt;
80
81         p = p->next;
82     }
83
84     printf("\\nOverall Details:\\n");
85     printf("Average Turn Around Time: %.2f\\n", ttat / process);
86     printf("Average Waiting Time: %.2f\\n", twt / process);
87     printf("Average Response Time: %.2f\\n", trt / process);
88 }
89
90 void FCFS()
91 {
92     NODE *head = NULL;
93     int process, i;
94     float arrival_time, burst_time, first_response = 0;
95
96     printf("First-Come, First-Served (FCFS) Scheduling\\n");
97     printf("Enter Number of Processes: ");
98     scanf("%d", &process);
99
100    for (i = 1; i <= process; i++)
101    {
102        printf("\\nEnter the Details for Process %d:\\n", i);
103        printf("Arrival Time: ");
104        scanf("%f", &arrival_time);
105        printf("Burst Time: ");
106        scanf("%f", &burst_time);
107
108        if (i == 1)
109            first_response = arrival_time;
110
111        create_insert(&head, i, arrival_time, burst_time,
112                    &first_response);
113    }
114
115    gantt_chart(head, process);
116    display(head, process);
117 }
```

## <RR.c>

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 struct times {
5     int p, art, but, wtt, tat, rnt;
6 };
7
8 void sortart(struct times a[], int pro) {
9     int i, j;
10    struct times temp;
11    for (i = 0; i < pro; i++) {
12        for (j = i + 1; j < pro; j++) {
13            if (a[i].art > a[j].art) {
14                temp = a[i];
15                a[i] = a[j];
16                a[j] = temp;
17            }
18        }
19    }
20    return;
21 }
22
23 void RR() {
24     int i, j, pro, time, remain, flag = 0, ts;
25     struct times a[100];
26     float avgwt = 0, avgtt = 0;
27
28     printf("Round Robin (RR) Scheduling\n");
29     printf("Enter Number of Processes: ");
30     scanf("%d", &pro);
31     remain = pro;
32
33     for (i = 0; i < pro; i++) {
34         printf("Enter arrival time and burst time for Process P%d: ", i);
35         scanf("%d%d", &a[i].art, &a[i].but);
36         a[i].p = i;
37         a[i].rnt = a[i].but;
38     }
39
40     sortart(a, pro);
41
42     printf("Enter Time Quantum: ");
43     scanf("%d", &ts);
44
45     printf("\n< START >\n");
46     printf("\n*****\n");
47     printf("Gantt Chart\n");
48     printf("0");
49
50     for (time = 0, i = 0; remain != 0;) {
51         if (a[i].rnt <= ts && a[i].rnt > 0) {
52             time += a[i].rnt;
53             printf(" -> [P%d] <- %d", a[i].p, time);
54             a[i].rnt = 0;
55             flag = 1;
56         } else if (a[i].rnt > 0) {
57             a[i].rnt -= ts;
58             time += ts;
59             printf(" -> [P%d] <- %d", a[i].p, time);
60         }
61
62         if (a[i].rnt == 0 && flag == 1) {
63             remain--;
64             a[i].tat = time - a[i].art;
```

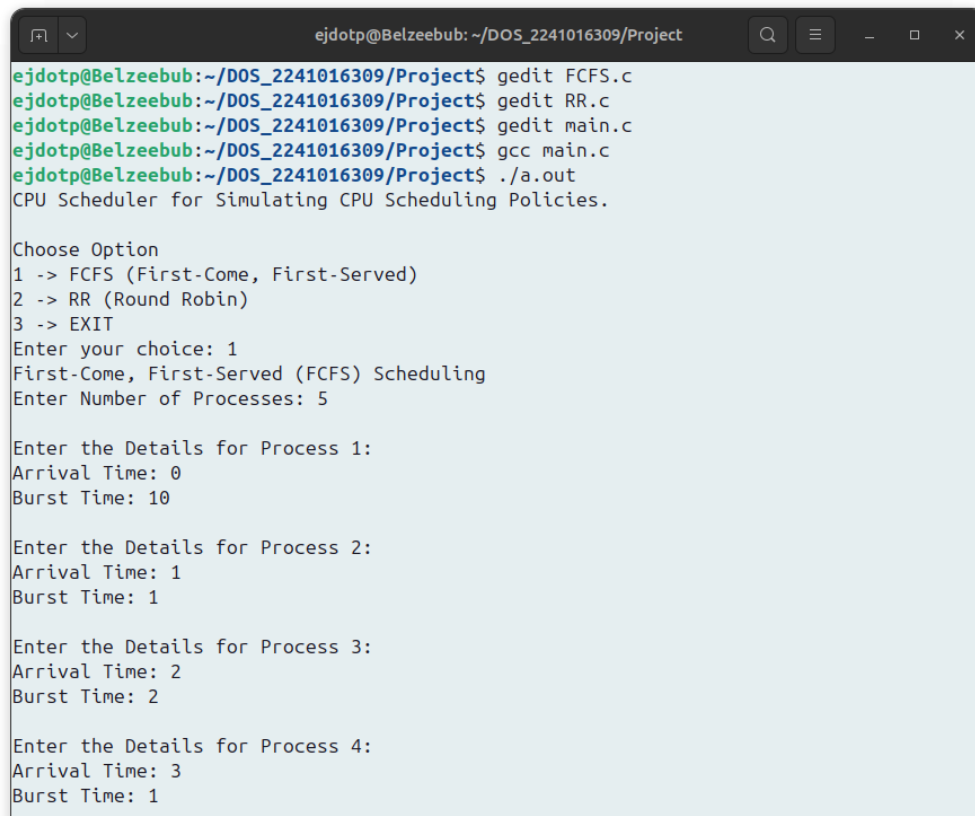
```
33     for (i = 0; i < pro; i++) {
34         printf("Enter arrival time and burst time for Process P%d: ", i);
35         scanf("%d%d", &a[i].art, &a[i].but);
36         a[i].p = i;
37         a[i].rnt = a[i].but;
38     }
39
40     sortart(a, pro);
41
42     printf("Enter Time Quantum: ");
43     scanf("%d", &ts);
44
45     printf("\n< START >\n");
46     printf("\n*****\n");
47     printf("Gantt Chart\n");
48     printf("0");
49
50     for (time = 0, i = 0; remain != 0;) {
51         if (a[i].rnt <= ts && a[i].rnt > 0) {
52             time += a[i].rnt;
53             printf(" -> [P%d] <- %d", a[i].p, time);
54             a[i].rnt = 0;
55             flag = 1;
56         } else if (a[i].rnt > 0) {
57             a[i].rnt -= ts;
58             time += ts;
59             printf(" -> [P%d] <- %d", a[i].p, time);
60         }
61
62         if (a[i].rnt == 0 && flag == 1) {
63             remain--;
64             a[i].tat = time - a[i].art;
65             a[i].wtt = a[i].tat - a[i].but;
66             avgwt += a[i].wtt;
67             avgtt += a[i].tat;
68             flag = 0;
69         }
70
71         if (i == pro - 1)
72             i = 0;
73         else if (a[i + 1].art <= time)
74             i++;
75         else
76             i = 0;
77     }
78
79     printf("\n\nProcess Details:\n");
80     printf("*****\n");
81     printf("Pro\tArTi\tBuTi\tTaTi\tWtTi\n");
82     printf("*****\n");
83
84     for (i = 0; i < pro; i++) {
85         printf("P%d\t%d\t%d\t%d\t%d\t%d\n", a[i].p, a[i].art, a[i].but,
86             a[i].tat, a[i].wtt);
87     }
88
89     avgwt /= pro;
90     avgtt /= pro;
91
92     printf("\nOverall Details:\n");
93     printf("Average Waiting Time: %.2f\n", avgwt);
94     printf("Average Turnaround Time: %.2f\n", avgtt);
95     printf("< END >\n");
96 }
```

## <main.c>



```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "FCFS.c"
4 #include "RR.c"
5
6 int main() {
7     int choice;
8
9     printf("CPU Scheduler for Simulating CPU Scheduling Policies.\n");
10
11     while (1) {
12         printf("\nChoose Option\n");
13         printf("1 -> FCFS (First-Come, First-Served)\n");
14         printf("2 -> RR (Round Robin)\n");
15         printf("3 -> EXIT\n");
16         printf("Enter your choice: ");
17         scanf("%d", &choice);
18
19         switch (choice) {
20             case 1:
21                 FCFS();
22                 break;
23
24             case 2:
25                 RR();
26                 break;
27
28             case 3:
29                 printf("Exited CPU Scheduler\n");
30                 exit(0);
31
32             default:
33                 printf("Wrong Input!\n");
34                 break;
35         }
36     }
37
38     return 0;
39 }
```

## Output:



```
ejdotp@Belzeebub: ~/DOS_2241016309/Project
ejdotp@Belzeebub:~/DOS_2241016309/Project$ gedit FCFS.c
ejdotp@Belzeebub:~/DOS_2241016309/Project$ gedit RR.c
ejdotp@Belzeebub:~/DOS_2241016309/Project$ gedit main.c
ejdotp@Belzeebub:~/DOS_2241016309/Project$ gcc main.c
ejdotp@Belzeebub:~/DOS_2241016309/Project$ ./a.out
CPU Scheduler for Simulating CPU Scheduling Policies.

Choose Option
1 -> FCFS (First-Come, First-Served)
2 -> RR (Round Robin)
3 -> EXIT
Enter your choice: 1
First-Come, First-Served (FCFS) Scheduling
Enter Number of Processes: 5

Enter the Details for Process 1:
Arrival Time: 0
Burst Time: 10

Enter the Details for Process 2:
Arrival Time: 1
Burst Time: 1

Enter the Details for Process 3:
Arrival Time: 2
Burst Time: 2

Enter the Details for Process 4:
Arrival Time: 3
Burst Time: 1
```

Enter the Details for Process 5:

Arrival Time: 6

Burst Time: 5

Gantt Chart:

```
|      P1      |      P2      |      P3      |      P4      |      P5      |
0.0      10.0    11.0    13.0    14.0    19.0
```

Process Details:

\*\*\*\*\*

Pro	ArTi	BuTi	TaTi	WtTi	RTi
1	0.00	10.00	10.00	0.00	0.00

```
ejdotp@Belzeebub: ~/DOS_2241016309/Project
2      1.00    1.00    10.00    9.00    9.00
3      2.00    2.00    11.00    9.00    9.00
4      3.00    1.00    11.00    10.00   10.00
5      6.00    5.00    13.00    8.00    8.00

Overall Details:
Average Turn Around Time: 11.00
Average Waiting Time: 7.20
Average Response Time: 7.20

Choose Option
1 -> FCFS (First-Come, First-Served)
2 -> RR (Round Robin)
3 -> EXIT
Enter your choice: 2
Round Robin (RR) Scheduling
Enter Number of Processes: 5
Enter arrival time and burst time for Process P0: 0 10
Enter arrival time and burst time for Process P1: 1 1
Enter arrival time and burst time for Process P2: 2 2
Enter arrival time and burst time for Process P3: 3 1
Enter arrival time and burst time for Process P4: 6 5
Enter Time Quantum: 4

< START >

*****
Gantt Chart
0 -> [P0] <- 4 -> [P1] <- 5 -> [P2] <- 7 -> [P3] <- 8 -> [P4] <- 12 -> [P0] <- 16 ->
[P4] <- 17 -> [P0] <- 19

Process Details:
*****
Pro    ArTi    BuTi    TaTi    WtTi
*****
P0      0      10      19      9
P1      1       1       4       3
P2      2       2       5       3
P3      3       1       5       4
P4      6       5      11       6

Overall Details:
Average Waiting Time: 5.00
Average Turnaround Time: 8.80
< END >

Choose Option
1 -> FCFS (First-Come, First-Served)
2 -> RR (Round Robin)
3 -> EXIT
Enter your choice: 3
Exited CPU Scheduler
ejdotp@Belzeebub:~/DOS_2241016309/Project$
```

## Objective 2:

The banker's algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation for predetermined maximum possible amounts of all resources, then makes an state" check to test for possible activities, before deciding whether allocation should be allowed to continue.

### Example: Snapshot at the initial stage:

1. Consider the following resource allocation state with 5 processes and 4 resources: There are total existing resources of 6 instances of type R1, 7 instances of type R2, 12 instance of type R3 and 12 instances of type R4.

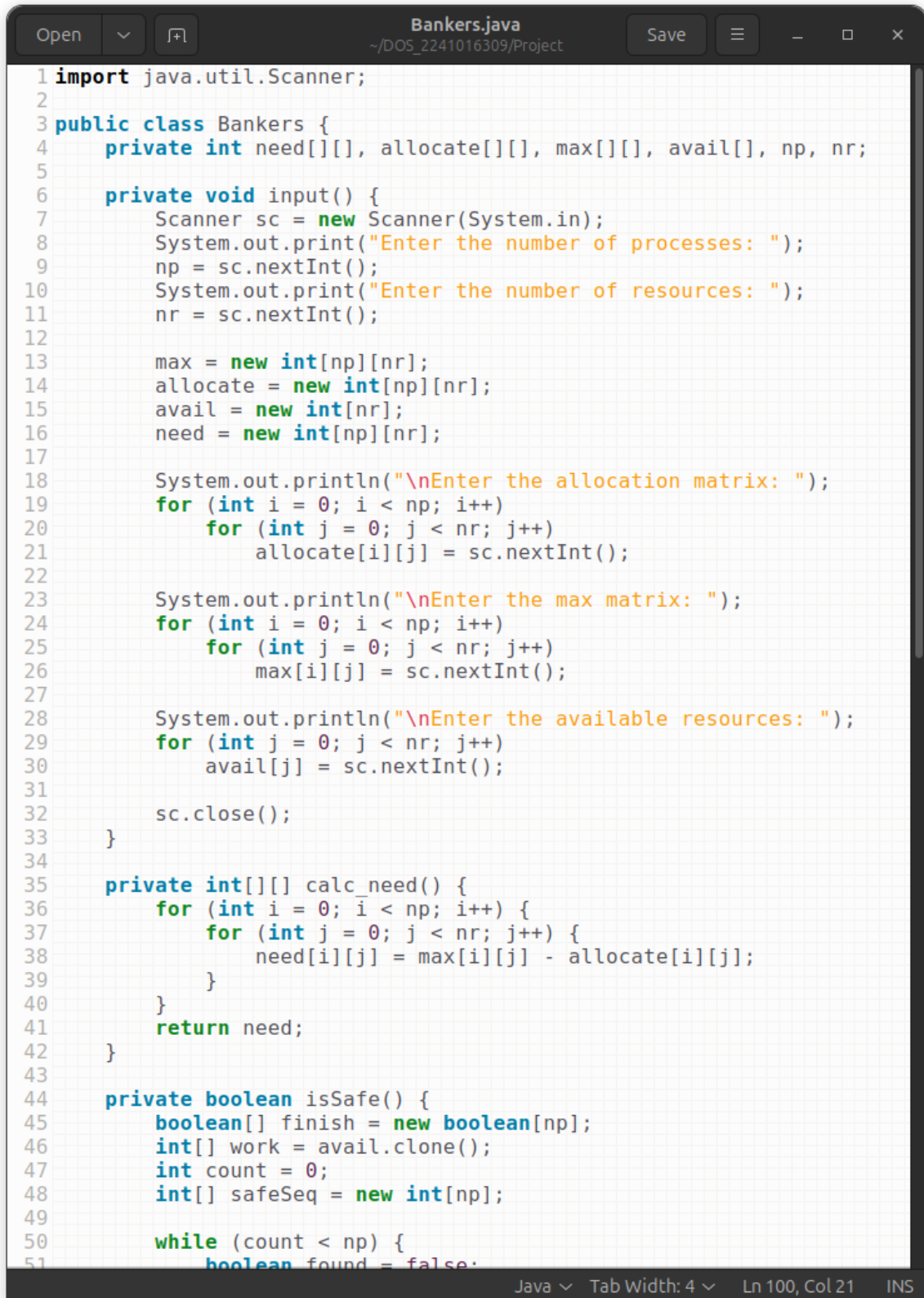
Process	Allocation				Max			
	R1	R2	R3	R4	R1	R2	R3	R4
P <sub>1</sub>	0	0	1	2	0	0	1	2
P <sub>2</sub>	2	0	0	0	2	7	5	0
P <sub>3</sub>	0	0	3	4	6	6	5	6
P <sub>4</sub>	2	3	5	4	4	3	5	6
P <sub>5</sub>	0	3	3	2	0	6	5	2

- a) Find the content of the need matrix.
- b) Is the system in a safe state? If so, give a safe sequence of the process.
- c) If P<sub>3</sub> will request for 1 more instances of type R2, Can the request be granted immediately or not?



## Program:

<Bankers.java>



```
1 import java.util.Scanner;
2
3 public class Bankers {
4     private int need[][], allocate[][], max[][], avail[], np, nr;
5
6     private void input() {
7         Scanner sc = new Scanner(System.in);
8         System.out.print("Enter the number of processes: ");
9         np = sc.nextInt();
10        System.out.print("Enter the number of resources: ");
11        nr = sc.nextInt();
12
13        max = new int[np][nr];
14        allocate = new int[np][nr];
15        avail = new int[nr];
16        need = new int[np][nr];
17
18        System.out.println("\nEnter the allocation matrix: ");
19        for (int i = 0; i < np; i++)
20            for (int j = 0; j < nr; j++)
21                allocate[i][j] = sc.nextInt();
22
23        System.out.println("\nEnter the max matrix: ");
24        for (int i = 0; i < np; i++)
25            for (int j = 0; j < nr; j++)
26                max[i][j] = sc.nextInt();
27
28        System.out.println("\nEnter the available resources: ");
29        for (int j = 0; j < nr; j++)
30            avail[j] = sc.nextInt();
31
32        sc.close();
33    }
34
35    private int[][] calc_need() {
36        for (int i = 0; i < np; i++) {
37            for (int j = 0; j < nr; j++) {
38                need[i][j] = max[i][j] - allocate[i][j];
39            }
40        }
41        return need;
42    }
43
44    private boolean isSafe() {
45        boolean[] finish = new boolean[np];
46        int[] work = avail.clone();
47        int count = 0;
48        int[] safeSeq = new int[np];
49
50        while (count < np) {
51            boolean found = false;
```

```
Bankers.java
~/DOS_2241016309/Project

49
50     while (count < np) {
51         boolean found = false;
52         for (int i = 0; i < np; i++) {
53             if (!finish[i]) {
54                 int j;
55                 for (j = 0; j < nr; j++)
56                     if (need[i][j] > work[j])
57                         break;
58                 if (j == nr) {
59                     for (int k = 0; k < nr; k++) {
60                         work[k] += allocate[i][k];
61                     }
62                     finish[i] = true;
63                     found = true;
64                     safeSeq[count++] = i;
65                 }
66             }
67         }
68         if (!found) {
69             break;
70         }
71     }
72
73     if(count < np){
74         System.out.println("System is not in a safe state.");
75         return false;
76     } else {
77         System.out.println("System is in a safe state.");
78         System.out.print("Safe Sequence: ");
79         for(int i = 0; i < np; i++)
80             System.out.print("p" + (safeSeq[i] + 1) + " ");
81         System.out.println();
82         return true;
83     }
84 }
85
86 public static void main(String[] args) {
87     Bankers ba = new Bankers();
88     ba.input();
89     int[][] needMat = ba.calc_need();
90     System.out.println("\nNeed Matrix: ");
91     for( int i = 0; i < ba.np; i++){
92         for(int j = 0; j < ba.nr; j++){
93             System.out.print(needMat[i][j] + " ");
94         }
95         System.out.println();
96     }
97     ba.isSafe();
98 }
99 }
```

Java ▾ Tab Width: 4 ▾ Ln 96, Col 10 INS

**Output:**

**(a, b):**

```
ejdotp@Belzeebub: ~/DOS_2241016309/Project
ejdotp@Belzeebub:~/DOS_2241016309/Project$ gedit Bankers.java
ejdotp@Belzeebub:~/DOS_2241016309/Project$ javac Bankers.java
ejdotp@Belzeebub:~/DOS_2241016309/Project$ java Bankers
Enter the number of processes: 5
Enter the number of resources: 4

Enter the allocation matrix:
0 0 1 2
2 0 0 0
0 0 3 4
2 3 5 4
0 3 3 2

Enter the max matrix:
0 0 1 2
2 7 5 4
6 6 5 6
4 3 5 6
0 6 5 2

Enter the available resources:
6 7 12 12

Need Matrix:
0 0 0 0
0 7 5 4
6 6 2 2
2 0 0 2
0 3 2 0

System is in a safe state.
Safe Sequence: p1 p2 p3 p4 p5
ejdotp@Belzeebub:~/DOS_2241016309/Project$
```

**(c):** If P3 will request for 1 more instances of type R2, there will be no problem in executing the processes in the same order. The request can be granted immediately.