End Term Project Report
On
Design of Operating System (CSE 4049)

Submitted by

Name	**: SHAILENDRA SUMAN**
Reg. No.	**: 2141019394**
Semester	**: 5th**
Section	**: K**
Session	**: 2023-2024**
Admission Batch	**2021**

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

FACULTY OF ENGINEERING & TECHNOLOGY (ITER)

SIKSHA 'O' ANUSANDHAN DEEMED TO BE UNIVERSITY

BHUBANESWAR, ODISHA – 751030

Project Description 1: The Java 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 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.

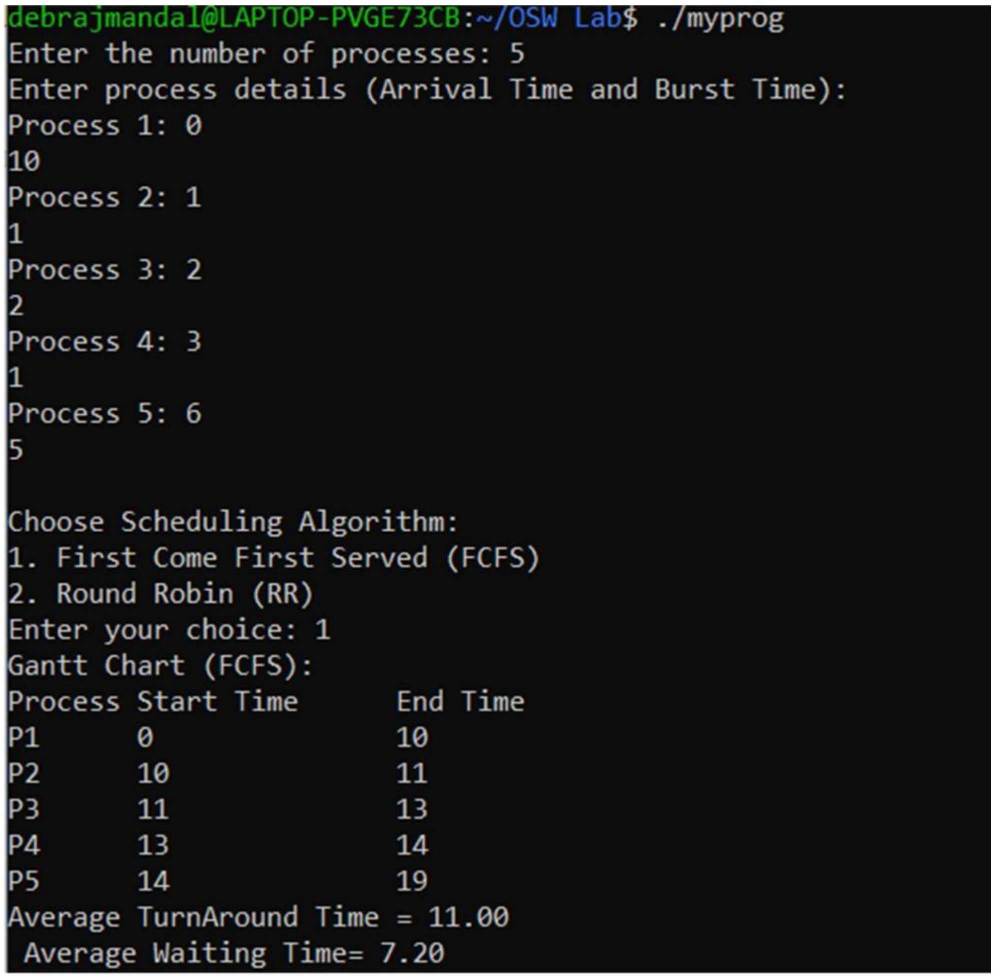
- Analyze the results and determine which of the algorithms results in the minimum average waiting time over all processes?

Code:

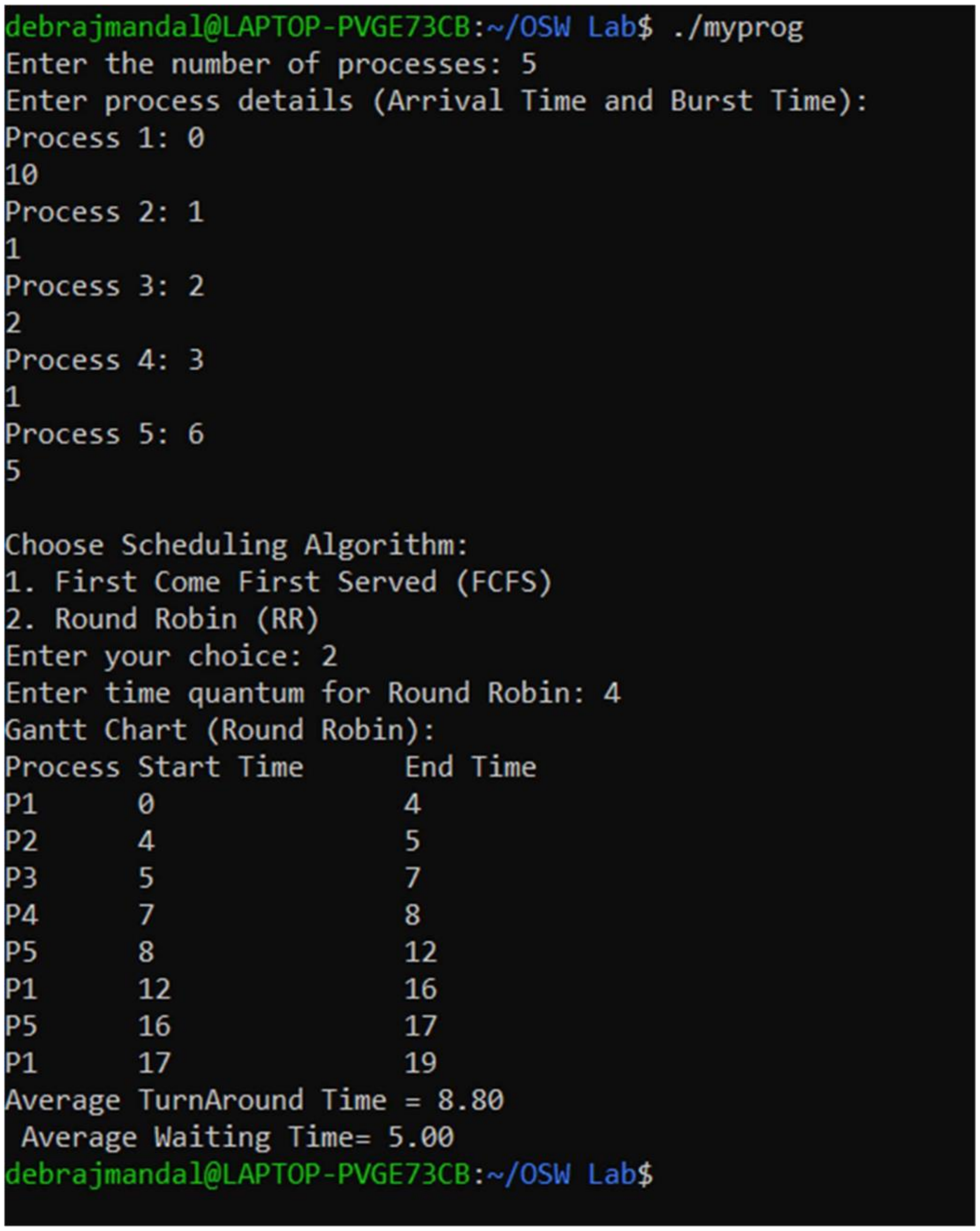
```
#include <stdio.h>

struct Process {
    int processID;
    int arrivalTime;
    int burstTime;
};

void executeFCFS(struct Process processes[], int n) {
    int currentTime = 0;
    float avgTurnaroundTime = 0, avgWaitingTime = 0;

    printf("Gantt Chart (FCFS):\n");
    printf("Process\tStart Time\tEnd Time\n");

    for (int i = 0; i < n; i++) {
        printf("P%d\t%d\t\t%d\n", processes[i].processID, currentTime,
currentTime + processes[i].burstTime);
        avgWaitingTime += currentTime - processes[i].arrivalTime;
        avgTurnaroundTime += currentTime + processes[i].burstTime -
processes[i].arrivalTime;
        currentTime += processes[i].burstTime;
    }

    avgWaitingTime /= n;
    avgTurnaroundTime /= n;

    printf("Average TurnAround Time = %.2f\t \n Average Waiting Time=
%.2f\n", avgTurnaroundTime,avgWaitingTime);
}

void executeRR(struct Process processes[], int n, int timeQuantum) {
    int remainingBurst[n];
    int currentTime = 0;
    float avgTurnaroundTime = 0, avgWaitingTime = 0;

    for (int i = 0; i < n; i++) {
        remainingBurst[i] = processes[i].burstTime;
    }

    printf("Gantt Chart (Round Robin):\n");
    printf("Process\tStart Time\tEnd Time\n");

    while (1) {
        int done = 1;

        for (int i = 0; i < n; i++) {
            if (remainingBurst[i] > 0) {
                done = 0;

                int executeTime = (remainingBurst[i] < timeQuantum) ?
remainingBurst[i] : timeQuantum;
                printf("P%d\t%d\t\t%d\n", processes[i].processID,
currentTime, currentTime + executeTime);

                currentTime += executeTime;
                remainingBurst[i] -= executeTime;

                if (remainingBurst[i] == 0) {
                    avgWaitingTime += currentTime -
processes[i].arrivalTime - processes[i].burstTime;
                    avgTurnaroundTime += currentTime -
processes[i].arrivalTime;
                }
            }
        }

        int allDone = 1;
        for (int i = 0; i < n; i++) {
            if (remainingBurst[i] > 0) {
                allDone = 0;
                break;
            }
        }

        if (allDone == 1) {
            break;
        }
    }

    avgWaitingTime /= n;
    avgTurnaroundTime /= n;


    printf("Average TurnAround Time = %.2f\t  \n Average Waiting Time=
%.2f\n",  avgTurnaroundTime,avgWaitingTime);
}
```

```
int main() {
    int n, choice, quantum;

    printf("Enter the number of processes: ");
    scanf("%d", &n);

    struct Process processes[n];

    printf("Enter process details (Arrival Time and Burst
Time):\n");
    for (int i = 0; i < n; i++) {
        printf("Process %d: ", i + 1);
        processes[i].processID = i + 1;
        scanf("%d %d", &processes[i].arrivalTime,
&processes[i].burstTime);
    }

    printf("\nChoose Scheduling Algorithm:\n");
    printf("1. First Come First Served (FCFS)\n");
    printf("2. Round Robin (RR)\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            executeFCFS(processes, n);
            break;
        case 2:
            printf("Enter time quantum for Round Robin: ");
            scanf("%d", &quantum);
            executeRR(processes, n, quantum);
            break;
        default:
            printf("Invalid choice\n");
            break;
    }

    return 0;
}
```

Output:



```
debrajmandal@LAPTOP-PVGE73CB:~/OSW Lab$ ./myprog
Enter the number of processes: 5
Enter process details (Arrival Time and Burst Time):
Process 1: 0
10
Process 2: 1
1
Process 3: 2
2
Process 4: 3
1
Process 5: 6
5

Choose Scheduling Algorithm:
1. First Come First Served (FCFS)
2. Round Robin (RR)
Enter your choice: 1
Gantt Chart (FCFS):
Process Start Time      End Time
P1      0               10
P2      10              11
P3      11              13
P4      13              14
P5      14              19
Average TurnAround Time = 11.00
 Average Waiting Time= 7.20
```



```
debrajmandal@LAPTOP-PVGE73CB:~/OSW Lab$ ./myprog
Enter the number of processes: 5
Enter process details (Arrival Time and Burst Time):
Process 1: 0
10
Process 2: 1
1
Process 3: 2
2
Process 4: 3
1
Process 5: 6
5

Choose Scheduling Algorithm:
1. First Come First Served (FCFS)
2. Round Robin (RR)
Enter your choice: 2
Enter time quantum for Round Robin: 4
Gantt Chart (Round Robin):
Process Start Time      End Time
P1      0               4
P2      4               5
P3      5               7
P4      7               8
P5      8               12
P1      12              16
P5      16              17
P1      17              19
Average TurnAround Time = 8.80
 Average Waiting Time= 5.00
debrajmandal@LAPTOP-PVGE73CB:~/OSW Lab$
```

Project Description 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**
P1	0	0	1	2	0	0	1	2
P2	2	0	0	0	2	7	5	0
P3	0	0	3	4	6	6	5	6
P4	2	3	5	4	4	3	5	6
P5	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 P3 will request for 1 more instances of type R2, Can the request be granted immediately or not

Code:

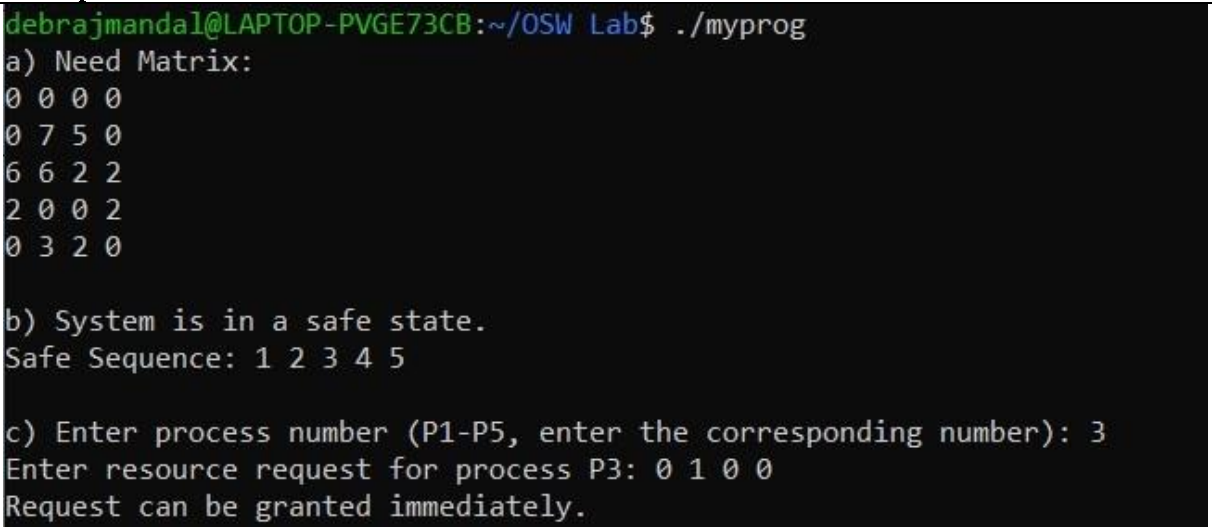
```
#include <stdio.h>
#include <stdbool.h>

#define NUM_PROCESSES 5
#define NUM_RESOURCES 4

int available[NUM_RESOURCES] = {6, 7, 12, 12};
int max[NUM_PROCESSES][NUM_RESOURCES] = {
    {0, 0, 1, 2},
    {2, 7, 5, 0},
    {6, 6, 5, 6},
    {4, 3, 5, 6},
    {0, 6, 5, 2}
};
```

```
int allocation[NUM_PROCESSES][NUM_RESOURCES] = {
    {0, 0, 1, 2},
    {2, 0, 0, 0},
    {0, 0, 3, 4},
    {2, 3, 5, 4},
    {0, 3, 3, 2}
};
int need[NUM_PROCESSES][NUM_RESOURCES];
bool finish[NUM_PROCESSES];

void calculateNeedMatrix() {
    for (int i = 0; i < NUM_PROCESSES; ++i) {
        for (int j = 0; j < NUM_RESOURCES; ++j) {
            need[i][j] = max[i][j] - allocation[i][j];
        }
    }
}

bool isSafeState(int safeSequence[NUM_PROCESSES]) {
    int work[NUM_RESOURCES];
    for (int i = 0; i < NUM_RESOURCES; ++i) {
        work[i] = available[i];
    }
    for (int i = 0; i < NUM_PROCESSES; ++i) {
        finish[i] = false;
    }
    int count = 0;
    while (count < NUM_PROCESSES) {
        bool found = false;
        for (int i = 0; i < NUM_PROCESSES; ++i) {
            if (!finish[i]) {
                bool canAllocate = true;
                for (int j = 0; j < NUM_RESOURCES; ++j) {
                    if (need[i][j] > work[j]) {
                        canAllocate = false;
                        break;
                    }
                }
                if (canAllocate) {
                    for (int j = 0; j < NUM_RESOURCES; ++j) {
                        work[j] += allocation[i][j];
                    }
                    finish[i] = true;
                    safeSequence[count++] = i;
                    found = true;
                }
            }
        }
        if (!found) {
```

```
return false;
        }
    }
    return true;
}

int requestResources(int process, int
request[NUM_RESOURCES]) {
    for (int i = 0; i < NUM_RESOURCES; ++i) {
        if (request[i] > need[process][i] || request[i] >
available[i]) {
            return -1;
        }
    }
    for (int i = 0; i < NUM_RESOURCES; ++i) {
        available[i] -= request[i];
        allocation[process][i] += request[i];
        need[process][i] -= request[i];
    }
    int safeSequence[NUM_PROCESSES];
    if (isSafeState(safeSequence)) {
        return 1;
    } else {
        for (int i = 0; i < NUM_RESOURCES; ++i) {
            available[i] += request[i];
            allocation[process][i] -= request[i];
            need[process][i] += request[i];
        }
        return 0;
    }
}

void printNeedMatrix() {
    printf("a) Need Matrix:\n");
    for (int i = 0; i < NUM_PROCESSES; ++i) {
        for (int j = 0; j < NUM_RESOURCES; ++j) {
            printf("%d ", need[i][j]);
        }
        printf("\n");
    }
}

void displaySafeSequence(int safeSequence[NUM_PROCESSES]) {
    printf("\nb) ");
    if (isSafeState(safeSequence)) {
        printf("System is in a safe state.\nSafe Sequence:
");
        for (int i = 0; i < NUM_PROCESSES; ++i) {
            printf("P%d ", safeSequence[i] + 1);
        }
        printf("\n");
```

```c
} else {
        printf("System is not in a safe state.\n");
    }
}

int main() {
    calculateNeedMatrix();
    printNeedMatrix();

    int safeSequence[NUM_PROCESSES];
    displaySafeSequence(safeSequence);

    int process_num;
    int request[NUM_RESOURCES];

    printf("\n");
    printf("c) Enter process number (P1-P5, enter the
corresponding number): ");
    scanf("%d", &process_num);
    process_num--;

    printf("Enter resource request for process P%d: ",
process_num + 1);
    for (int i = 0; i < NUM_RESOURCES; ++i) {
        scanf("%d", &request[i]);
    }

    int result = requestResources(process_num, request);

    if (result == 1) {
        printf("Request can be granted immediately.\n");
    } else if (result == 0) {
        printf("Request denied as it leads to an unsafe
state.\n");
    } else {
        printf("Requested resources exceed available or
need.\n");
    }

    return 0;
}
```

Output



```
debrajmandal@LAPTOP-PVGE73CB:~/OSW Lab$ ./myprog
a) Need Matrix:
0 0 0 0
0 7 5 0
6 6 2 2
2 0 0 2
0 3 2 0

b) System is in a safe state.
Safe Sequence: 1 2 3 4 5

c) Enter process number (P1-P5, enter the corresponding number): 3
Enter resource request for process P3: 0 1 0 0
Request can be granted immediately.
```

Github Link: https://github.com/Shailendra-suman/DOS_PROJECT.git