Week 3

Name: Mohit Bisht
Section: J1
Roll No: 37
Course: B.tech 5th Sem
Branch: CSE

7. Write a C program to implement FCFS Scheduling Algorithm.

```
#include<stdio.h>
typedef struct Process {
int arvl_t;
int brst_t;
int wtng_t;
int trn_arnd_t;
int cmplt_t;
} Process;
void FCFS(int n, Process *p) {
p[0].wtng_t = 0;
p[0].cmplt_t = p[0].brst_t;
p[0].trn_arnd_t = p[0].cmplt_t;
for(int i = 1; i < n; i++) {
p[i].wtng_t = p[i-1].cmplt_t - p[i].arvl_t;
if(p[i].wtng_t < 0) {
p[i].wtng_t = 0;
}
p[i].cmplt_t = p[i-1].cmplt_t + p[i].brst_t;
p[i].trn_arnd_t = p[i].cmplt_t - p[i].arvl_t;
}
printf("Gantt Chart: ");
for(int i = 0; i < n; i++) {
printf("P%d ", i);
}
printf("\n");
printf("\n%-10s%-15s%-15s%-15s%-20s%-15s\n", "Process", "Arrival Time",
"Burst
Time", "Waiting Time", "Turnaround Time", "Completion Time");
```

```
for(int i = 0; i < n; i++) {
printf("P%-9d%-15d%-15d%-15d%-20d%-15d\n", i, p[i].arvl_t,
p[i].brst_t,p[i].wtng_t, p[i].trn_arnd_t, p[i].cmplt_t);
}
float sum_wtng_t = 0;
float sum_trn_arnd_t = 0;
for(int i = 0; i < n; i++) {
sum_wtng_t += p[i].wtng_t;
sum_trn_arnd_t += p[i].trn_arnd_t;
}
float avg_wtng_t = sum_wtng_t / (n * 1.0);
printf("\nAverage waiting time: %0.2f\n", avg_wtng_t);
float avg_trn_arnd_t = sum_trn_arnd_t / (n * 1.0);
printf("Average turnaround time: %0.2f\n", avg_trn_arnd_t);
}
int main() {
int n;
printf("No. of Processes: ");
scanf("%d", &n);
Process p[n];
printf("Enter burst time and arrival time for each process:\n");
for(int i = 0; i < n; i++) {
printf("Process %d: ", i);
scanf("%d%d", &p[i].brst_t, &p[i].arvl_t);
}
FCFS(n, p);
return 0;
}
```

Output:

```
No. of Processes: 3
Enter burst time and arrival time for each process:
Process 0: 5 0
Process 1: 9 1
Process 2: 6 2
Gantt Chart: P0 P1 P2

Process   Arrival Time   Burst Time     Waiting Time   Turnaround Time     Completion Time
P0        0              5              0              5                   5
P1        1              9              4              13                  14
P2        2              6              12             18                  20

Average waiting time: 5.33
Average turnaround time: 12.00
```

Name: Mohit Bisht
Section: J1
Roll No: 37
Course: B.tech 5th Sem
Branch: CSE

8. Write a C program to implement SJF Non-pre-emptive Scheduling Algorithm.

```
#include <stdio.h>
#define MAX_PROCESSES 100
typedef struct {
int arrivalTime;
int burstTime;
int processID;
} Process;
void sjf_np(Process processes[], int n) {
int waitingTime[MAX_PROCESSES], turnaroundTime[MAX_PROCESSES];
int i, completed = 0;
int totalWaitingTime = 0, totalTurnaroundTime = 0;
int time = 0;
int minIndex;
int isCompleted[MAX_PROCESSES] = {0};
printf("SJF Non-Preemptive Scheduling:\n");
printf("Gantt Chart: ");
while (completed < n) {
minIndex = -1;
for (i = 0; i < n; i++) {
if (processes[i].arrivalTime <= time && !isCompleted[i]) {
if (minIndex == -1 || processes[i].burstTime <
processes[minIndex].burstTime) {
minIndex = i;
}
}
}
if (minIndex == -1) {
time++;
continue;}
```

```
printf("P%d ", processes[minIndex].processID);
time += processes[minIndex].burstTime;
waitingTime[minIndex] = time - processes[minIndex].arrivalTime -
processes[minIndex].burstTime;
turnaroundTime[minIndex] = time - processes[minIndex].arrivalTime;
totalWaitingTime += waitingTime[minIndex];
totalTurnaroundTime += turnaroundTime[minIndex];
isCompleted[minIndex] = 1;
completed++;
}
printf("\n");
printf("\n%-10s%-15s%-15s%-15s%-20s\n", "Process ID", "Arrival Time",
"Burst
Time", "Waiting Time", "Turnaround Time");
for (i = 0; i < n; i++) {
printf("P%-9d%-15d%-15d%-15d%-20d\n", processes[i].processID,
processes[i].arrivalTime, processes[i].burstTime, waitingTime[i],
turnaroundTime[i]);
}
printf("Average waiting time: %.2f\n", (float)totalWaitingTime / n);
printf("Average turnaround time: %.2f\n", (float)totalTurnaroundTime / n);
}
int main() {
int n, i;
Process processes[MAX_PROCESSES];
printf("Enter number of processes: ");
scanf("%d", &n);
for (i = 0; i < n; i++) {
printf("Enter arrival time and burst time for process P%d: ", i);
scanf("%d %d", &processes[i].arrivalTime, &processes[i].burstTime);
processes[i].processID = i;
}
sjf_np(processes, n);
return 0;
}
```

Output:

```
Enter number of processes: 3
Enter arrival time and burst time for process P0: 0 7
Enter arrival time and burst time for process P1: 2 4
Enter arrival time and burst time for process P2: 4 1
SJF Non-Preemptive Scheduling:
Gantt Chart: P0 P2 P1

Process IDArrival Time   Burst Time     Waiting Time   Turnaround Time
P0        0              7              0              7
P1        2              4              6              10
P2        4              1              3              4

Average waiting time: 3.00
Average turnaround time: 7.00
```

Name: Mohit Bisht
Section: J1
Roll No: 37
Course: B.tech 5th Sem
Branch: CSE

9. Write a C program to implement SJF pre-emptive Scheduling Algorithm.

```
#include <stdio.h>
#define MAX_PROCESSES 100

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

void sjf_p(Process processes[], int n) {
    int waitingTime[MAX_PROCESSES],
turnaroundTime[MAX_PROCESSES];
    int i, time = 0, totalWaitingTime = 0, totalTurnaroundTime = 0;
    int remaining[MAX_PROCESSES];
    int minIndex;

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

    printf("SJF Preemptive Scheduling:\n");
    printf("Gantt Chart: ");

    int completed = 0;
    while (completed < n) {
        minIndex = -1;
        for (i = 0; i < n; i++) {
```

```
            if (processes[i].arrivalTime <= time && remaining[i] > 0) {
                if (minIndex == -1 || remaining[i] < remaining[minIndex]) {
                    minIndex = i;
                }
            }
        }

        if (minIndex == -1) {
            time++;
            continue;
        }

        printf("P%d ", processes[minIndex].processID);

        remaining[minIndex]--;
        time++;

        if (remaining[minIndex] == 0) {
            completed++;
            waitingTime[minIndex] = time - processes[minIndex].arrivalTime -
processes[minIndex].burstTime;
            turnaroundTime[minIndex] = time - processes[minIndex].arrivalTime;
            totalWaitingTime += waitingTime[minIndex];
            totalTurnaroundTime += turnaroundTime[minIndex];
        }
    }

    printf("\n\n%-10s%-15s%-15s%-15s%-20s\n", "Process ID", "Arrival Time",
"Burst Time", "Waiting Time", "Turnaround Time");
    for (i = 0; i < n; i++) {
        printf("P%-9d%-15d%-15d%-15d%-20d\n", processes[i].processID,
processes[i].arrivalTime, processes[i].burstTime, waitingTime[i],
turnaroundTime[i]);
    }

    printf("\nAverage waiting time: %.2f\n", (float)totalWaitingTime / n);
    printf("Average turnaround time: %.2f\n", (float)totalTurnaroundTime / n);
```

```
}

int main() {
    int n, i;
    Process processes[MAX_PROCESSES];

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

    for (i = 0; i < n; i++) {
        printf("Enter arrival time and burst time for process P%d: ", i);
        scanf("%d %d", &processes[i].arrivalTime, &processes[i].burstTime);
        processes[i].processID = i;
    }

    sjf_p(processes, n);

    return 0;
}
```

Output:

```
Enter number of processes: 4
Enter arrival time and burst time for process P0: 0 8
Enter arrival time and burst time for process P1: 1 4
Enter arrival time and burst time for process P2: 2 9
Enter arrival time and burst time for process P3: 3 5
SJF Preemptive Scheduling:
Gantt Chart: P0 P1 P1 P1 P1 P3 P3 P3 P3 P3 P0 P0 P0 P0 P0 P0 P0 P2 P2 P2 P2 P2 P2 P2 P2 P2

Process IDArrival Time   Burst Time     Waiting Time   Turnaround Time
P0        0              8              9              17
P1        1              4              0              4
P2        2              9              15             24
P3        3              5              2              7

Average waiting time: 6.50
Average turnaround time: 13.00
```

Week:4

Name: Mohit Bisht
Section: J1
Roll No: 37
Course: B.tech 5th Sem
Branch: CSE

10.Write a C program to implement Priority Scheduling Algorithm.

```
#include <stdio.h>

typedef struct {
    int prcs_id;
    int arvl_t;
    int brst_t;
    int prty;
    int wtng_t;
    int trn_arnd_t;
    int cmplt_t;
} Process;

void priorityScheduling(int n, Process p[]) {
    Process temp;


    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (p[i].prty > p[j].prty || (p[i].prty == p[j].prty && p[i].arvl_t >
p[j].arvl_t)) {
                temp = p[i];
                p[i] = p[j];
                p[j] = temp;
            }
        }
    }

    p[0].wtng_t = 0;
    p[0].cmplt_t = p[0].arvl_t + p[0].brst_t;
```

```
    p[0].trn_arnd_t = p[0].cmplt_t - p[0].arvl_t;


    for (int i = 1; i < n; i++) {
        if (p[i - 1].cmplt_t < p[i].arvl_t) {
            p[i].cmplt_t = p[i].arvl_t + p[i].brst_t;
        } else {
            p[i].cmplt_t = p[i - 1].cmplt_t + p[i].brst_t;
        }
        p[i].wtng_t = p[i].cmplt_t - p[i].arvl_t - p[i].brst_t;
        p[i].trn_arnd_t = p[i].cmplt_t - p[i].arvl_t;
    }


    printf("Gantt Chart: ");
    for (int i = 0; i < n; i++) {
        printf("P%d ", p[i].prcs_id);
    }
    printf("\n");

    float sum_wtng_t = 0;
    float sum_trn_arnd_t = 0;
    for (int i = 0; i < n; i++) {
        sum_wtng_t += p[i].wtng_t;
        sum_trn_arnd_t += p[i].trn_arnd_t;
    }


    printf("Average waiting time: %.2f\n", sum_wtng_t / n);
    printf("Average turnaround time: %.2f\n", sum_trn_arnd_t / n);


    printf("\n%-10s%-15s%-15s%-15s%-15s%-20s\n", "Process ID", "Arrival
Time",
           "Burst Time", "Priority", "Waiting Time", "Turnaround Time");
    for (int i = 0; i < n; i++) {
        printf("P%-9d%-15d%-15d%-15d%-15d%-20d\n", p[i].prcs_id, p[i].arvl_t,
```

```
            p[i].brst_t, p[i].prty, p[i].wtng_t, p[i].trn_arnd_t);
    }
}

int main() {
    int n;
    printf("Enter number of processes: ");
    scanf("%d", &n);
    Process p[n];
    for (int i = 0; i < n; i++) {
        printf("Enter arrival time, burst time and priority for process P%d: ", i);
        scanf("%d%d%d", &p[i].arvl_t, &p[i].brst_t, &p[i].prty);
        p[i].prcs_id = i;
    }
    priorityScheduling(n, p);
    return 0;
}
```

Output:

```
Enter number of processes: 4
Enter arrival time, burst time and priority for process P0: 0 10 2
Enter arrival time, burst time and priority for process P1: 1 5 1
Enter arrival time, burst time and priority for process P2: 2 8 3
Enter arrival time, burst time and priority for process P3: 3 6 2
Gantt Chart: P1 P0 P3 P2
Average waiting time: 9.75
Average turnaround time: 17.00

Process IDArrival Time   Burst Time     Priority       Waiting Time   Turnaround Time
P1        1              5              1              0              5
P0        0              10             2              6              16
P3        3              6              2              13             19
P2        2              8              3              20             28
```

Name: Mohit Bisht
Section: J1
Roll No: 37
Course: B.tech 5th Sem
Branch: CSE

11.Write a C program to implement Round Robin Scheduling Algorithm.

```
#include <stdio.h>

typedef struct {
    int prcs_id;
    int arvl_t;
    int brst_t;
    int rmng_t;
    int wtng_t;
    int trn_arnd_t;
    int cmplt_t;
} Process;

void roundRobin(int n, Process p[], int quantum) {
    int time = 0, completed = 0;
    float sum_wtng_t = 0, sum_trn_arnd_t = 0;

    for (int i = 0; i < n; i++) {
        p[i].rmng_t = p[i].brst_t;
    }

    printf("Gantt Chart: ");

    while (completed < n) {
        int allProcessesChecked = 1;
        for (int i = 0; i < n; i++) {
            if (p[i].rmng_t > 0 && p[i].arvl_t <= time) {
                allProcessesChecked = 0;
```

```
            if (p[i].rmng_t <= quantum) {
                time += p[i].rmng_t;
                p[i].cmplt_t = time;
                p[i].rmng_t = 0;
                p[i].trn_arnd_t = p[i].cmplt_t - p[i].arvl_t;
                p[i].wtng_t = p[i].trn_arnd_t - p[i].brst_t;
                sum_wtng_t += p[i].wtng_t;
                sum_trn_arnd_t += p[i].trn_arnd_t;
                completed++;
            } else {
                time += quantum;
                p[i].rmng_t -= quantum;
            }
        }
    }
    if (allProcessesChecked) {
        time++;
    }
  }

  for (int i = 0; i < n; i++) {
    if (p[i].cmplt_t > 0) {
        printf("P%d ", p[i].prcs_id);
    }
  }
  printf("\n");

  printf("Average waiting time: %.2f\n", sum_wtng_t / n);
  printf("Average turnaround time: %.2f\n", sum_trn_arnd_t / n);

  printf("\n%-10s%-15s%-15s%-15s%-15s%-20s%-15s\n", "Process ID",
"Arrival Time", "Burst Time", "Remaining Time", "Waiting Time", "Turnaround
Time", "Completion Time");
  for (int i = 0; i < n; i++) {
    printf("P%-9d%-15d%-15d%-15d%-15d%-20d%-15d\n",
           p[i].prcs_id, p[i].arvl_t, p[i].brst_t, p[i].rmng_t, p[i].wtng_t,
p[i].trn_arnd_t, p[i].cmplt_t);
```

```
    }
}

int main() {
    int n, quantum;
    printf("Enter number of processes: ");
    scanf("%d", &n);
    Process p[n];

    printf("Enter burst time and arrival time for each process:\n");
    for (int i = 0; i < n; i++) {
        printf("P%d - Arrival time, Burst time: ", i);
        scanf("%d%d", &p[i].arvl_t, &p[i].brst_t);
        p[i].prcs_id = i;
    }

    printf("Enter time quantum: ");
    scanf("%d", &quantum);

    roundRobin(n, p, quantum);
    return 0;
}
```

Output:

```
Enter number of processes: 3
Enter burst time and arrival time for each process:
P0 - Arrival time, Burst time: 0 10
P1 - Arrival time, Burst time: 1 5
P2 - Arrival time, Burst time: 2 8
Enter time quantum: 3
Gantt Chart: P0 P1 P2
Average waiting time: 11.00
Average turnaround time: 18.67

Process IDArrival Time   Burst Time     Remaining Time Waiting Time   Turnaround Time     Completion Time
P0        0              10             0              13             23                  23
P1        1              5              0              8              13                  14
P2        2              8              0              12             20                  22
```

Week 5

Name: Mohit Bisht
Section: J1
Roll No: 37
Course: B.tech 5th Sem
Branch: CSE

12. Write a C program to simulate Bankers algorithm for the purpose of deadlock avoidance.

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

#define MAX_PROCESSES 10
#define MAX_RESOURCES 10

int p;
int r;
int allocation[MAX_PROCESSES][MAX_RESOURCES];
int max[MAX_PROCESSES][MAX_RESOURCES];
int need[MAX_PROCESSES][MAX_RESOURCES];
int available[MAX_RESOURCES];

void calculateNeed() {
    for (int i = 0; i < p; i++) {
        for (int j = 0; j < r; j++) {
            need[i][j] = max[i][j] - allocation[i][j];
        }
    }
}

bool isSafe() {
    bool finish[MAX_PROCESSES] = { false };
    int safeSequence[MAX_PROCESSES];
    int work[MAX_RESOURCES];

    for (int i = 0; i < r; i++) {
        work[i] = available[i];
    }

    int count = 0;
    while (count < p) {
```

```
        bool found = false;
        for (int i = 0; i < p; i++) {
            if (!finish[i]) {
                int j;
                for (j = 0; j < r; j++) {
                    if (need[i][j] > work[j]) {
                        break;
                    }
                }
                if (j == r) {
                    for (int k = 0; k < r; k++) {
                        work[k] += allocation[i][k];
                    }
                    safeSequence[count++] = i;
                    finish[i] = true;
                    found = true;
                }
            }
        }
        if (!found) {
            printf("Request cannot be fulfilled.\n");
            return false;
        }
    }
    printf("Request can be fulfilled.\nSafe Sequence: ");
    for (int i = 0; i < p; i++) {
        printf("P%d ", safeSequence[i]);
    }
    printf("\n");
    return true;
}

int main() {
    printf("Enter number of processes: ");
    scanf("%d", &p);
    printf("Enter number of resources: ");
    scanf("%d", &r);

    printf("Enter maximum requirement:\n");
    for (int i = 0; i < p; i++) {
        for (int j = 0; j < r; j++) {
            scanf("%d", &max[i][j]);
        }
```

```
    }

    printf("Enter allocated matrix:\n");
    for (int i = 0; i < p; i++) {
        for (int j = 0; j < r; j++) {
            scanf("%d", &allocation[i][j]);
        }
    }

    printf("Enter resource vector:\n");
    for (int i = 0; i < r; i++) {
        scanf("%d", &available[i]);
    }

    calculateNeed();


    isSafe();

    return 0;
}
```

Output:

```
Enter number of processes: 5
Enter number of resources: 3
Enter maximum requirement:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter allocated matrix:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Enter resource vector:
3 3 2
Request can be fulfilled.
Safe Sequence: P0 P1 P2 P3 P4
```

Name: Mohit Bisht
Section: J1
Roll No: 37
Course: B.tech 5th Sem
Branch: CSE

13. Write a program to implement deadlock detection algorithm.

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

#define MAX_PROCESSES 10
#define MAX_RESOURCES 10

int p;
int r;
int allocation[MAX_PROCESSES][MAX_RESOURCES];
int max[MAX_PROCESSES][MAX_RESOURCES];
int need[MAX_PROCESSES][MAX_RESOURCES];
int available[MAX_RESOURCES];

void calculateNeed() {
    for (int i = 0; i < p; i++) {
        for (int j = 0; j < r; j++) {
            need[i][j] = max[i][j] - allocation[i][j];
        }
    }
}

bool isDeadlockDetected() {
    bool finish[MAX_PROCESSES] = { false };
    int work[MAX_RESOURCES];

    for (int i = 0; i < r; i++) {
        work[i] = available[i];
    }
```

```
    int count = 0;
    while (count < p) {
        bool found = false;
        for (int i = 0; i < p; i++) {
            if (!finish[i]) {
                int j;
                for (j = 0; j < r; j++) {
                    if (need[i][j] > work[j]) {
                        break;
                    }
                }
                if (j == r) {
                    for (int k = 0; k < r; k++) {
                        work[k] += allocation[i][k];
                    }
                    finish[i] = true;
                    found = true;
                    count++;
                }
            }
        }
        if (!found) {
            return true;
        }
    }
    return false;
}

int main() {
    printf("Enter number of processes: ");
    scanf("%d", &p);
    printf("Enter number of resources: ");
    scanf("%d", &r);

    printf("Enter maximum requirement:\n");
    for (int i = 0; i < p; i++) {
        for (int j = 0; j < r; j++) {
```

```
            scanf("%d", &max[i][j]);
        }
    }

    printf("Enter allocated matrix:\n");
    for (int i = 0; i < p; i++) {
        for (int j = 0; j < r; j++) {
            scanf("%d", &allocation[i][j]);
        }
    }

    printf("Resource Vector:\n");
    for (int i = 0; i < r; i++) {
        scanf("%d", &available[i]);
    }

    calculateNeed();

    if (isDeadlockDetected()) {
        printf("Deadlock detected.\n");
    } else {
        printf("No deadlock detected.\n");
    }

    return 0;
}
```

Output:

```
Enter number of processes: 3
Enter number of resources: 3
Enter maximum requirement:
3 2 2
9 0 2
2 2 2
Enter allocated matrix:
2 0 0
3 0 2
2 1 1
Resource Vector:
7 4 5
No deadlock detected.
```