NAME : PANDIT ADITYA RAJ AJAY

REG NO : 23BRS1157

Q1) i) Implement FCFS scheduling with different arrival time and calculate waiting time, CT, and TAT for all the processes.

CODE :

fcfs.c > main()

```c
#include <stdio.h>

typedef struct Process{
    int pid;
    int arrivalTime;
    int burstTime;
    int turnAroundTime;
    int waitingTime;
    int completionTime;
}PCB;

void sortByArrivale(PCB processes[], int n) {
    for(int i=0; i<n; i++) {
        for(int j=0; j<n-i; j++) {
            if(processes[j].arrivalTime > processes[j+1].arrivalTime) {
                PCB temp = processes[j];
                processes[j] = processes[j+1];
                processes[j+1] = temp;
            }
        }
    }
}

int main() {
    int n;
    printf("Enter the number of processes : \n");
    scanf("%d",&n);
    PCB processes[n];

    for(int i=0; i<n; i++) {
        processes[i].pid = i+1;
        printf("Enter the arrival time and burst time for the pid %d : ",processes[i].pid);
        scanf("%d %d",&processes[i].arrivalTime, &processes[i].burstTime);
    }
    sortByArrivale(processes, n);
```

```
    }
    sortByArrivale(processes, n);

    double totalTAT = 0 , totalWT = 0;
    int currentTime = 0;

    for(int i=0; i<n; i++) {
        if(currentTime < processes[i].arrivalTime) {
            currentTime = processes[i].arrivalTime;
        }

        processes[i].completionTime = currentTime + processes[i].burstTime;
        processes[i].turnAroundTime = processes[i].completionTime - processes[i].arrivalTime;
        processes[i].waitingTime = processes[i].turnAroundTime - processes[i].burstTime;

        currentTime = processes[i].completionTime;
        totalTAT += processes[i].turnAroundTime;
        totalWT += processes[i].waitingTime;
    }

    printf("\nProcess\tArrival Time\tBurst Time\tCompletion Time\tWaiting Time\tTurnaround Time\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", processes[i].pid, processes[i].arrivalTime, processes[i].burstTime,
                processes[i].completionTime, processes[i].waitingTime, processes[i].turnAroundTime);
    }
    printf("\nAverage Waiting Time: %.2f\n", totalWT / n);
    printf("Average Turnaround Time: %.2f\n", totalTAT / n);
}
```

OUTPUT:

```
c:\Codes\OS>cd "c:\Codes\OS\" && gcc fcfs.c -o fcfs && "c:\Codes\OS\"fcfs
Enter the number of processes :
4
Enter the arrival time and burst time for the pid 1 : 0 5
Enter the arrival time and burst time for the pid 2 : 1 3
Enter the arrival time and burst time for the pid 3 : 2 8
Enter the arrival time and burst time for the pid 4 : 3 6

Process Arrival Time    Burst Time      Completion Time Waiting Time    Turnaround Time
1       0               5               5               0               5
2       1               3               8               4               7
3       2               8               16              6               14
4       3               6               22              13              19

Average Waiting Time: 5.75
Average Turnaround Time: 11.25
```

ii) Implement SJF scheduling with different arrival time and calculate waiting time, CT, TAT for all the processes.

CODE:

sjf.c > main()

```c
#include <stdio.h>
typedef struct Process{
    int pid;
    int arrivalTime;
    int burstTime;
    int turnAroundTime;
    int waitingTime;
    int completionTime;
    int isCompleted;
}PCB;

void sortByCompletion(PCB p[], int n) {
    for(int i=0; i<n; i++) {
        for(int j=0; j<n-i; j++) {
            if(p[j].completionTime > p[j+1].completionTime) {
                PCB temp = p[j];
                p[j] = p[j+1];
                p[j+1] = temp;
            }
        }
    }
}

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

    for(int i=0; i<n; i++) {
        p[i].pid = i+1;
        printf("Enter the arrival time and burst time for the pid %d : ",p[i].pid);
        scanf("%d %d",&p[i].arrivalTime, &p[i].burstTime);
        p[i].isCompleted = 0;
    }

    int currentTime = 0, completed = 0;
```

```c
C sjf.c > main()
int main() {
    float totalTAT = 0, totalWT = 0;
    while(completed != n) {
        int index = -1;
        int minBurst = 1000000000;
        for(int i=0; i<n; i++) {
            if(p[i].arrivalTime <= currentTime && p[i].isCompleted == 0) {
                if(p[i].burstTime < minBurst) {
                    minBurst = p[i].burstTime;
                    index = i;
                }
            }
        }
        if(index == -1) {
            currentTime++;
        }
        else {
            currentTime += p[index].burstTime;
            p[index].completionTime = currentTime;
            p[index].turnAroundTime = p[index].completionTime - p[index].arrivalTime;
            p[index].waitingTime = p[index].turnAroundTime - p[index].burstTime;

            totalTAT += p[index].turnAroundTime;
            totalWT += p[index].waitingTime;
            p[index].isCompleted = 1;
            completed++;
        }
    }
    sortByCompletion(p,n);

    printf("\nProcess\tArrival Time\tBurst Time\tCompletion Time\tWaiting Time\tTurnaround Time\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", p[i].pid, p[i].arrivalTime, p[i].burstTime,
                p[i].completionTime, p[i].waitingTime, p[i].turnAroundTime);
    }
    printf("\nAverage Waiting Time: %.2f\n", totalWT / n);
    printf("Average Turnaround Time: %.2f\n", totalTAT / n);
```

OUTPUT :

```
c:\Codes\OS>cd "c:\Codes\OS\" && gcc sjf.c -o sjf && "c:\Codes\OS\"sjf
Enter the number of processes :
4
Enter the arrival time and burst time for the pid 1 : 0 7
Enter the arrival time and burst time for the pid 2 : 2 4
Enter the arrival time and burst time for the pid 3 : 4 1
Enter the arrival time and burst time for the pid 4 : 5 4

Process Arrival Time    Burst Time      Completion Time Waiting Time    Turnaround Time
1       0               7               7               0               7
3       4               1               8               3               4
2       2               4               12              6               10
4       5               4               16              7               11

Average Waiting Time: 4.00
Average Turnaround Time: 8.00
```

iii) Implement PBS scheduling with different arrival time and calculate waiting time, CT, TAT for all the processes.

CODE :

pbs.c > ...

```c
#include <stdio.h>

struct Process {
    int pid;
    int arrivalTime;
    int burstTime;
    int priority;
    int completionTime;
    int waitingTime;
    int turnAroundTime;
    int isCompleted;
};

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

    for (int i = 0; i < n; i++) {
        p[i].pid = i + 1;
        printf("Enter Arrival Time, Burst Time and Priority for Process %d: ", p[i].pid);
        scanf("%d %d %d", &p[i].arrivalTime, &p[i].burstTime, &p[i].priority);
        p[i].isCompleted = 0;
    }

    int currentTime = 0, completed = 0;
    float totalWaitingTime = 0, totalTurnAroundTime = 0;

    while (completed != n) {
        int idx = -1;
        int minPriority = 1e9;

        for (int i = 0; i < n; i++) {
            if (p[i].arrivalTime <= currentTime && p[i].isCompleted == 0) {
                if (p[i].priority < minPriority) {
                    minPriority = p[i].priority;
```

```c
C pbs.c > main()
14  int main() {
37                      minPriority = p[i].priority;
38                      idx = i;
39                  } else if (p[i].priority == minPriority) {
40                      if (p[i].arrivalTime < p[idx].arrivalTime) {
41                          idx = i;
42                      }
43                  }
44              }
45          }
46
47          if (idx == -1) {
48              currentTime++;
49          } else {
50              currentTime += p[idx].burstTime;
51              p[idx].completionTime = currentTime;
52              p[idx].turnAroundTime = p[idx].completionTime - p[idx].arrivalTime;
53              p[idx].waitingTime = p[idx].turnAroundTime - p[idx].burstTime;
54
55              totalWaitingTime += p[idx].waitingTime;
56              totalTurnAroundTime += p[idx].turnAroundTime;
57
58              p[idx].isCompleted = 1;
59              completed++;
60          }
61      }
62
63      printf("\nProcess\tArrival Time\tBurst Time\tPriority\tCompletion Time\tWaiting Time\tTurnaround Time\n");
64      for (int i = 0; i < n; i++) {
65          printf("%d\t%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", p[i].pid, p[i].arrivalTime, p[i].burstTime,
66                  p[i].priority, p[i].completionTime, p[i].waitingTime, p[i].turnAroundTime);
67      }
68
69      printf("\nAverage Waiting Time: %.2f\n", totalWaitingTime / n);
70      printf("Average Turnaround Time: %.2f\n", totalTurnAroundTime / n);
71      return 0;
72  }
⚠ 0  0
```

OUTPUT :

```
c:\Codes\OS>cd "c:\Codes\OS\" && gcc pbs.c -o pbs && "c:\Codes\OS\"pbs
Enter the number of processes: 4
Enter Arrival Time, Burst Time and Priority for Process 1: 0 5 2
Enter Arrival Time, Burst Time and Priority for Process 2: 1 3 1
Enter Arrival Time, Burst Time and Priority for Process 3: 2 8 3
Enter Arrival Time, Burst Time and Priority for Process 4: 3 6 2

Process Arrival Time    Burst Time      Priority        Completion Time Waiting Time    Turnaround Time
1       0               5               2               5               0               5
2       1               3               1               8               4               7
3       2               8               3               22              12              20
4       3               6               2               14              5               11

Average Waiting Time: 5.25
Average Turnaround Time: 10.75
```

Q2) Implement the pre-emptive version of SJF and PBS. calculate waiting time, response time, CT, TAT for all the processes.

PRE-EMPTIVE SJF CODE:

premptiveSJF.c > main()

```c
#include <stdio.h>
struct Process {
    int pid;
    int arrivalTime;
    int burstTime;
    int remainingTime;
    int completionTime;
    int waitingTime;
    int turnAroundTime;
    int responseTime;
    int startTime;
    int isCompleted;
};

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

    struct Process p[n];
    int completed = 0, currentTime = 0;
    float totalWaitingTime = 0, totalTurnAroundTime = 0, totalResponseTime = 0;

    for (int i = 0; i < n; i++) {
        p[i].pid = i + 1;
        printf("Enter Arrival Time and Burst Time for Process %d: ", p[i].pid);
        scanf("%d %d", &p[i].arrivalTime, &p[i].burstTime);
        p[i].remainingTime = p[i].burstTime;
        p[i].isCompleted = 0;
        p[i].startTime = -1;
    }

    while (completed != n) {
        int idx = -1;
        int minRemainingTime = 1e9;

        for (int i = 0; i < n; i++) {
            if (p[i].arrivalTime <= currentTime && p[i].isCompleted == 0) {
                if (p[i].remainingTime < minRemainingTime) {
                    minRemainingTime = p[i].remainingTime;
                    idx = i;
                }
            }
        }

```

premptiveSJF.c > main()

```c
int main() {
                }
            }
        }

        if (idx != -1) {
            if (p[idx].startTime == -1) {
                p[idx].startTime = currentTime;
                p[idx].responseTime = p[idx].startTime - p[idx].arrivalTime;
                totalResponseTime += p[idx].responseTime;
            }

            p[idx].remainingTime--;
            currentTime++;

            if (p[idx].remainingTime == 0) {
                p[idx].completionTime = currentTime;
                p[idx].turnAroundTime = p[idx].completionTime - p[idx].arrivalTime;
                p[idx].waitingTime = p[idx].turnAroundTime - p[idx].burstTime;

                totalWaitingTime += p[idx].waitingTime;
                totalTurnAroundTime += p[idx].turnAroundTime;

                p[idx].isCompleted = 1;
                completed++;
            }
        } else {
            currentTime++;
        }
    }

    printf("\nProcess\tArrival Time\tBurst Time\tCompletion Time\tWaiting Time\tTurnaround Time\tResponse Time\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", p[i].pid, p[i].arrivalTime, p[i].burstTime,
               p[i].completionTime, p[i].waitingTime, p[i].turnAroundTime, p[i].responseTime);
    }

    printf("\nAverage Waiting Time: %.2f\n", totalWaitingTime / n);
    printf("Average Turnaround Time: %.2f\n", totalTurnAroundTime / n);
    printf("Average Response Time: %.2f\n", totalResponseTime / n);

    return 0;
}
```

OUTPUT :

```
c:\Codes\OS>cd "c:\Codes\OS\" && gcc premptiveSJF.c -o premptiveSJF && "c:\Codes\OS\"premptiveSJF
Enter the number of processes: 4
Enter Arrival Time and Burst Time for Process 1: 0 8
Enter Arrival Time and Burst Time for Process 2: 1 4
Enter Arrival Time and Burst Time for Process 3: 2 9
Enter Arrival Time and Burst Time for Process 4: 3 5

Process Arrival Time    Burst Time      Completion Time Waiting Time    Turnaround Time Response Time
1       0               8               17              9               17              0
2       1               4               5               0               4               0
3       2               9               26              15              24              15
4       3               5               10              2               7               2

Average Waiting Time: 6.50
Average Turnaround Time: 13.00
Average Response Time: 4.25
```

PRE-EMPTIVE PBS CODE:

premptivePBS.c > main()

```c
#include <stdio.h>

struct Process {
    int pid;
    int arrivalTime;
    int burstTime;
    int remainingTime;
    int priority;
    int completionTime;
    int waitingTime;
    int turnAroundTime;
    int responseTime;
    int startTime;
    int isCompleted;
};

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

    struct Process p[n];
    int completed = 0, currentTime = 0;
    float totalWaitingTime = 0, totalTurnAroundTime = 0, totalResponseTime = 0;

    for (int i = 0; i < n; i++) {
        p[i].pid = i + 1;
        printf("Enter Arrival Time, Burst Time, and Priority for Process %d: ", p[i].pid);
        scanf("%d %d %d", &p[i].arrivalTime, &p[i].burstTime, &p[i].priority);
        p[i].remainingTime = p[i].burstTime;
        p[i].isCompleted = 0;
        p[i].startTime = -1;
    }

    while (completed != n) {
        int idx = -1;
        int minPriority = 1e9;

        for (int i = 0; i < n; i++) {
            if (p[i].arrivalTime <= currentTime && p[i].isCompleted == 0) {
                if (p[i].priority < minPriority) {
                    minPriority = p[i].priority;
                    idx = i;
                } else if (p[i].priority == minPriority) {
                    if (p[i].remainingTime < p[idx].remainingTime) {
```

premptivePBS.c > main()

```c
int main() {
                    if (p[i].remainingTime < p[idx].remainingTime) {
                        idx = i;
                    }
                }
            }
        }

        if (idx != -1) {
            if (p[idx].startTime == -1) {
                p[idx].startTime = currentTime;
                p[idx].responseTime = p[idx].startTime - p[idx].arrivalTime;
                totalResponseTime += p[idx].responseTime;
            }

            p[idx].remainingTime--;
            currentTime++;

            if (p[idx].remainingTime == 0) {
                p[idx].completionTime = currentTime;
                p[idx].turnAroundTime = p[idx].completionTime - p[idx].arrivalTime;
                p[idx].waitingTime = p[idx].turnAroundTime - p[idx].burstTime;

                totalWaitingTime += p[idx].waitingTime;
                totalTurnAroundTime += p[idx].turnAroundTime;

                p[idx].isCompleted = 1;
                completed++;
            }
        } else {
            currentTime++;
        }
    }

    printf("\nProcess\tArrival Time\tBurst Time\tPriority\tCompletion Time\tWaiting Time\tTurnaround Time\tResponse Time\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", p[i].pid, p[i].arrivalTime, p[i].burstTime,
               p[i].priority, p[i].completionTime, p[i].waitingTime, p[i].turnAroundTime, p[i].responseTime);
    }

    printf("\nAverage Waiting Time: %.2f\n", totalWaitingTime / n);
    printf("Average Turnaround Time: %.2f\n", totalTurnAroundTime / n);
    printf("Average Response Time: %.2f\n", totalResponseTime / n);
    return 0;
}
```

OUTPUT :

```
c:\Codes\OS>cd "c:\Codes\OS\" && gcc premptivePBS.c -o premptivePBS && "c:\Codes\OS\"premptivePBS
Enter the number of processes: 4
Enter Arrival Time, Burst Time, and Priority for Process 1: 0 9 3
Enter Arrival Time, Burst Time, and Priority for Process 2: 1 4 1
Enter Arrival Time, Burst Time, and Priority for Process 3: 2 5 2
Enter Arrival Time, Burst Time, and Priority for Process 4: 3 2 1

Process Arrival Time    Burst Time      Priority        Completion Time Waiting Time    Turnaround Time Response Time
1       0               9               3               20              11              20              0
2       1               4               1               5               0               4               0
3       2               5               2               12              5               10              5
4       3               2               1               7               2               4               2

Average Waiting Time: 4.50
Average Turnaround Time: 9.50
Average Response Time: 1.75
```

Q3) Implement the RR scheduling. calculate waiting time, response time, CT, TAT for all the processes.

CODE :

rr.c > main()

```c
#include <stdio.h>

struct Process {
    int pid;
    int arrivalTime;
    int burstTime;
    int remainingTime;
    int completionTime;
    int waitingTime;
    int turnAroundTime;
    int responseTime;
    int startTime;
    int isCompleted;
};

int main() {
    int n, timeQuantum;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    printf("Enter the time quantum: ");
    scanf("%d", &timeQuantum);

    struct Process p[n];
    int currentTime = 0, completed = 0;
    float totalWaitingTime = 0, totalTurnAroundTime = 0, totalResponseTime = 0;

    for (int i = 0; i < n; i++) {
        p[i].pid = i + 1;
        printf("Enter Arrival Time and Burst Time for Process %d: ", p[i].pid);
        scanf("%d %d", &p[i].arrivalTime, &p[i].burstTime);
        p[i].remainingTime = p[i].burstTime;
        p[i].isCompleted = 0;
        p[i].startTime = -1;
    }

    int queue[n];
    int front = 0, rear = 0;

    for (int i = 0; i < n; i++) {
        if (p[i].arrivalTime == 0) {
            queue[rear++] = i;
        }
    }

    while (completed != n) {
```

```c
rr.c > main()
int main() {
    for (int i = 0; i < n; i++) {
        if (p[i].arrivalTime == 0) {
            queue[rear++] = i;
        }
    }

    while (completed != n) {
        if (front != rear) {
            int idx = queue[front++];
            if (p[idx].startTime == -1) {
                p[idx].startTime = currentTime;
                p[idx].responseTime = p[idx].startTime - p[idx].arrivalTime;
                totalResponseTime += p[idx].responseTime;
            }

            int execTime = (p[idx].remainingTime > timeQuantum) ? timeQuantum : p[idx].remainingTime;
            currentTime += execTime;
            p[idx].remainingTime -= execTime;

            for (int i = 0; i < n; i++) {
                if (p[i].arrivalTime <= currentTime && p[i].remainingTime > 0 && i != idx && p[i].startTime == -1) {
                    queue[rear++] = i;
                }
            }

            if (p[idx].remainingTime == 0) {
                p[idx].completionTime = currentTime;
                p[idx].turnAroundTime = p[idx].completionTime - p[idx].arrivalTime;
                p[idx].waitingTime = p[idx].turnAroundTime - p[idx].burstTime;

                totalWaitingTime += p[idx].waitingTime;
                totalTurnAroundTime += p[idx].turnAroundTime;
                p[idx].isCompleted = 1;
                completed++;
            } else {
                queue[rear++] = idx;
            }
        } else {
            currentTime++;
            for (int i = 0; i < n; i++) {
                if (p[i].arrivalTime <= currentTime && p[i].remainingTime > 0) {
                    queue[rear++] = i;
                    break;
                }
            }
        }
    }

    printf("\nProcess\tArrival Time\tBurst Time\tCompletion Time\tWaiting Time\tTurnaround Time\tResponse Time\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", p[i].pid, p[i].arrivalTime, p[i].burstTime,
               p[i].completionTime, p[i].waitingTime, p[i].turnAroundTime, p[i].responseTime);
    }

    printf("\nAverage Waiting Time: %.2f\n", totalWaitingTime / n);
    printf("Average Turnaround Time: %.2f\n", totalTurnAroundTime / n);
    printf("Average Response Time: %.2f\n", totalResponseTime / n);

    return 0;
}
```

OUTPUT:

```
c:\Codes\OS>cd "c:\Codes\OS\" && gcc rr.c -o rr && "c:\Codes\OS\"rr
Enter the number of processes: 4
Enter the time quantum: 4
Enter Arrival Time and Burst Time for Process 1: 0 6
Enter Arrival Time and Burst Time for Process 2: 1 8
Enter Arrival Time and Burst Time for Process 3: 2 7
Enter Arrival Time and Burst Time for Process 4: 3 3

Process Arrival Time    Burst Time      Completion Time Waiting Time    Turnaround Time Response Time
1       0               6               17              11              17              0
2       1               8               21              12              20              3
3       2               7               24              15              22              6
4       3               3               15              9               12              9

Average Waiting Time: 11.75
Average Turnaround Time: 17.75
Average Response Time: 4.50
```