

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()
1  #include <stdio.h>
2
3  typedef struct Process{
4      int pid;
5      int arrivalTime;
6      int burstTime;
7      int turnAroundTime;
8      int waitingTime;
9      int completionTime;
10 }PCB;
11
12 void sortByArrivale(PCB processes[], int n) {
13     for(int i=0; i<n; i++) {
14         for(int j=0; j<n-i; j++) {
15             if(processes[j].arrivalTime > processes[j+1].arrivalTime) {
16                 PCB temp = processes[j];
17                 processes[j] = processes[j+1];
18                 processes[j+1] = temp;
19             }
20         }
21     }
22 }
23
24 int main() {
25     int n;
26     printf("Enter the number of processes : \n");
27     scanf("%d",&n);
28     PCB processes[n];
29
30     for(int i=0; i<n; i++) {
31         processes[i].pid = i+1;
32         printf("Enter the arrival time and burst time for the pid %d : ",processes[i].pid);
33         scanf("%d %d",&processes[i].arrivalTime, &processes[i].burstTime);
34     }
35     sortByArrivale(processes, n);
```

```

34     }
35     sortByArrivale(processes, n);
36
37     double totalTAT = 0 , totalWT = 0;
38     int currentTime = 0;
39
40     for(int i=0; i<n; i++) {
41         if(currentTime < processes[i].arrivalTime) {
42             currentTime = processes[i].arrivalTime;
43         }
44
45         processes[i].completionTime = currentTime + processes[i].burstTime;
46         processes[i].turnAroundTime = processes[i].completionTime - processes[i].arrivalTime;
47         processes[i].waitingTime = processes[i].turnAroundTime - processes[i].burstTime;
48
49         currentTime = processes[i].completionTime;
50         totalTAT += processes[i].turnAroundTime;
51         totalWT += processes[i].waitingTime;
52     }
53
54     printf("\nProcess\tArrival Time\tBurst Time\tCompletion Time\tWaiting Time\tTurnaround Time\n");
55     for (int i = 0; i < n; i++) {
56         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,
57             processes[i].completionTime, processes[i].waitingTime, processes[i].turnAroundTime);
58     }
59     printf("\nAverage Waiting Time: %.2f\n", totalWT / n);
60     printf("Average Turnaround Time: %.2f\n", totalTAT / n);
61 }

```

## 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:

```
C sjf.c > main()
1  #include <stdio.h>
2  typedef struct Process{
3      int pid;
4      int arrivalTime;
5      int burstTime;
6      int turnAroundTime;
7      int waitingTime;
8      int completionTime;
9      int isCompleted;
10 }PCB;
11
12 void sortByCompletion(PCB p[], int n) {
13     for(int i=0; i<n; i++) {
14         for(int j=0; j<n-i; j++) {
15             if(p[j].completionTime > p[j+1].completionTime) {
16                 PCB temp = p[j];
17                 p[j] = p[j+1];
18                 p[j+1] = temp;
19             }
20         }
21     }
22 }
23
24 int main() {
25     int n;
26     printf("Enter the number of processes : \n");
27     scanf("%d",&n);
28     PCB p[n];
29
30     for(int i=0; i<n; i++) {
31         p[i].pid = i+1;
32         printf("Enter the arrival time and burst time for the pid %d : ",p[i].pid);
33         scanf("%d %d",&p[i].arrivalTime, &p[i].burstTime);
34         p[i].isCompleted = 0;
35     }
36
37     int currentTime = 0, completed = 0;
```

```

C sjf.c > main()
24  int main() {
38      float totalTAT = 0, totalWT = 0;
39      while(completed != n) {
40          int index = -1;
41          int minBurst = 1000000000;
42          for(int i=0; i<n; i++) {
43              if(p[i].arrivalTime <= currentTime && p[i].isCompleted == 0) {
44                  if(p[i].burstTime < minBurst) {
45                      minBurst = p[i].burstTime;
46                      index = i;
47                  }
48              }
49          }
50          if(index == -1) {
51              currentTime++;
52          }
53          else {
54              currentTime += p[index].burstTime;
55              p[index].completionTime = currentTime;
56              p[index].turnAroundTime = p[index].completionTime - p[index].arrivalTime;
57              p[index].waitingTime = p[index].turnAroundTime - p[index].burstTime;
58
59              totalTAT += p[index].turnAroundTime;
60              totalWT += p[index].waitingTime;
61              p[index].isCompleted = 1;
62              completed++;
63          }
64      }
65      sortByCompletion(p,n);
66
67      printf("\nProcess\tArrival Time\tBurst Time\tCompletion Time\tWaiting Time\tTurnaround Time\n");
68      for (int i = 0; i < n; i++) {
69          printf("%d\t%d\t%d\t%d\t%d\t%d\n", p[i].pid, p[i].arrivalTime, p[i].burstTime,
70              p[i].completionTime, p[i].waitingTime, p[i].turnAroundTime);
71      }
72      printf("\nAverage Waiting Time: %.2f\n", totalWT / n);
73      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 :

```
C pbs.c > ...
1  #include <stdio.h>
2
3  struct Process {
4      int pid;
5      int arrivalTime;
6      int burstTime;
7      int priority;
8      int completionTime;
9      int waitingTime;
10     int turnAroundTime;
11     int isCompleted;
12 };
13
14 int main() {
15     int n;
16     printf("Enter the number of processes: ");
17     scanf("%d", &n);
18     struct Process p[n];
19
20     for (int i = 0; i < n; i++) {
21         p[i].pid = i + 1;
22         printf("Enter Arrival Time, Burst Time and Priority for Process %d: ", p[i].pid);
23         scanf("%d %d %d", &p[i].arrivalTime, &p[i].burstTime, &p[i].priority);
24         p[i].isCompleted = 0;
25     }
26
27     int currentTime = 0, completed = 0;
28     float totalWaitingTime = 0, totalTurnAroundTime = 0;
29
30     while (completed != n) {
31         int idx = -1;
32         int minPriority = 1e9;
33
34         for (int i = 0; i < n; i++) {
35             if (p[i].arrivalTime <= currentTime && p[i].isCompleted == 0) {
36                 if (p[i].priority < minPriority) {
37                     minPriority = p[i].priority;
```

```

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%d\t%d\t%d\t%d\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 }

```

## 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:**

```
C preptiveSJF.c > main()
1  #include <stdio.h>
2  struct Process {
3      int pid;
4      int arrivalTime;
5      int burstTime;
6      int remainingTime;
7      int completionTime;
8      int waitingTime;
9      int turnAroundTime;
10     int responseTime;
11     int startTime;
12     int isCompleted;
13 };
14
15 int main() {
16     int n;
17     printf("Enter the number of processes: ");
18     scanf("%d", &n);
19
20     struct Process p[n];
21     int completed = 0, currentTime = 0;
22     float totalWaitingTime = 0, totalTurnAroundTime = 0, totalResponseTime = 0;
23
24     for (int i = 0; i < n; i++) {
25         p[i].pid = i + 1;
26         printf("Enter Arrival Time and Burst Time for Process %d: ", p[i].pid);
27         scanf("%d %d", &p[i].arrivalTime, &p[i].burstTime);
28         p[i].remainingTime = p[i].burstTime;
29         p[i].isCompleted = 0;
30         p[i].startTime = -1;
31     }
32
33     while (completed != n) {
34         int idx = -1;
35         int minRemainingTime = 1e9;
36
37         for (int i = 0; i < n; i++) {
38             if (p[i].arrivalTime <= currentTime && p[i].isCompleted == 0) {
39                 if (p[i].remainingTime < minRemainingTime) {
40                     minRemainingTime = p[i].remainingTime;
41                     idx = i;
42                 }
43             }
44         }
45     }
```

```

C:\preemptiveSJF.c> main()
15  int main() {
42      }
43  }
44  }
45
46      if (idx != -1) {
47          if (p[idx].startTime == -1) {
48              p[idx].startTime = currentTime;
49              p[idx].responseTime = p[idx].startTime - p[idx].arrivalTime;
50              totalResponseTime += p[idx].responseTime;
51          }
52
53          p[idx].remainingTime--;
54          currentTime++;
55
56          if (p[idx].remainingTime == 0) {
57              p[idx].completionTime = currentTime;
58              p[idx].turnAroundTime = p[idx].completionTime - p[idx].arrivalTime;
59              p[idx].waitingTime = p[idx].turnAroundTime - p[idx].burstTime;
60
61              totalWaitingTime += p[idx].waitingTime;
62              totalTurnAroundTime += p[idx].turnAroundTime;
63
64              p[idx].isCompleted = 1;
65              completed++;
66          }
67      } else {
68          currentTime++;
69      }
70  }
71
72      printf("\nProcess\tArrival Time\tBurst Time\tCompletion Time\tWaiting Time\tTurnaround Time\tResponse Time\n");
73      for (int i = 0; i < n; i++) {
74          printf("%d\t%d\t%d\t%d\t%d\t%d\t%d\n", p[i].pid, p[i].arrivalTime, p[i].burstTime,
75              p[i].completionTime, p[i].waitingTime, p[i].turnAroundTime, p[i].responseTime);
76      }
77
78      printf("\nAverage Waiting Time: %.2f\n", totalWaitingTime / n);
79      printf("Average Turnaround Time: %.2f\n", totalTurnAroundTime / n);
80      printf("Average Response Time: %.2f\n", totalResponseTime / n);
81
82      return 0;
83  }
84

```

## OUTPUT :

```

c:\Codes\OS>cd "c:\Codes\OS\" && gcc preemptiveSJF.c -o preemptiveSJF && "c:\Codes\OS\"preemptiveSJF
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:

```
C prempivePBS.c > main()
1  #include <stdio.h>
2
3  struct Process {
4      int pid;
5      int arrivalTime;
6      int burstTime;
7      int remainingTime;
8      int priority;
9      int completionTime;
10     int waitingTime;
11     int turnAroundTime;
12     int responseTime;
13     int startTime;
14     int isCompleted;
15 };
16
17 int main() {
18     int n;
19     printf("Enter the number of processes: ");
20     scanf("%d", &n);
21
22     struct Process p[n];
23     int completed = 0, currentTime = 0;
24     float totalWaitingTime = 0, totalTurnAroundTime = 0, totalResponseTime = 0;
25
26     for (int i = 0; i < n; i++) {
27         p[i].pid = i + 1;
28         printf("Enter Arrival Time, Burst Time, and Priority for Process %d: ", p[i].pid);
29         scanf("%d %d %d", &p[i].arrivalTime, &p[i].burstTime, &p[i].priority);
30         p[i].remainingTime = p[i].burstTime;
31         p[i].isCompleted = 0;
32         p[i].startTime = -1;
33     }
34
35     while (completed != n) {
36         int idx = -1;
37         int minPriority = 1e9;
38
39         for (int i = 0; i < n; i++) {
40             if (p[i].arrivalTime <= currentTime && p[i].isCompleted == 0) {
41                 if (p[i].priority < minPriority) {
42                     minPriority = p[i].priority;
43                     idx = i;
44                 } else if (p[i].priority == minPriority) {
45                     if (p[i].remainingTime < p[idx].remainingTime) {
```

```

C:\preemptivePBS.c> main()
17 int main() {
45     if (p[i].remainingTime < p[idx].remainingTime) {
46         idx = i;
47     }
48 }
49 }
50 }
51 }
52 if (idx != -1) {
53     if (p[idx].startTime == -1) {
54         p[idx].startTime = currentTime;
55         p[idx].responseTime = p[idx].startTime - p[idx].arrivalTime;
56         totalResponseTime += p[idx].responseTime;
57     }
58
59     p[idx].remainingTime--;
60     currentTime++;
61
62     if (p[idx].remainingTime == 0) {
63         p[idx].completionTime = currentTime;
64         p[idx].turnAroundTime = p[idx].completionTime - p[idx].arrivalTime;
65         p[idx].waitingTime = p[idx].turnAroundTime - p[idx].burstTime;
66
67         totalWaitingTime += p[idx].waitingTime;
68         totalTurnAroundTime += p[idx].turnAroundTime;
69
70         p[idx].isCompleted = 1;
71         completed++;
72     }
73 } else {
74     currentTime++;
75 }
76 }
77
78 printf("\nProcess\tArrival Time\tBurst Time\tPriority\tCompletion Time\tWaiting Time\tTurnaround Time\tResponse Time\n");
79 for (int i = 0; i < n; i++) {
80     printf("%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\n", p[i].pid, p[i].arrivalTime, p[i].burstTime,
81         p[i].priority, p[i].completionTime, p[i].waitingTime, p[i].turnAroundTime, p[i].responseTime);
82 }
83
84 printf("\nAverage Waiting Time: %.2f\n", totalWaitingTime / n);
85 printf("Average Turnaround Time: %.2f\n", totalTurnAroundTime / n);
86 printf("Average Response Time: %.2f\n", totalResponseTime / n);
87 return 0;
88 }

```

## OUTPUT :

```

c:\Codes\OS>cd "c:\Codes\OS\" && gcc preemptivePBS.c -o preemptivePBS && "c:\Codes\OS\"preemptivePBS
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 :**

```
C rr.c > main()
1  #include <stdio.h>
2
3  struct Process {
4      int pid;
5      int arrivalTime;
6      int burstTime;
7      int remainingTime;
8      int completionTime;
9      int waitingTime;
10     int turnAroundTime;
11     int responseTime;
12     int startTime;
13     int isCompleted;
14 };
15
16 int main() {
17     int n, timeQuantum;
18     printf("Enter the number of processes: ");
19     scanf("%d", &n);
20     printf("Enter the time quantum: ");
21     scanf("%d", &timeQuantum);
22
23     struct Process p[n];
24     int currentTime = 0, completed = 0;
25     float totalWaitingTime = 0, totalTurnAroundTime = 0, totalResponseTime = 0;
26
27     for (int i = 0; i < n; i++) {
28         p[i].pid = i + 1;
29         printf("Enter Arrival Time and Burst Time for Process %d: ", p[i].pid);
30         scanf("%d %d", &p[i].arrivalTime, &p[i].burstTime);
31         p[i].remainingTime = p[i].burstTime;
32         p[i].isCompleted = 0;
33         p[i].startTime = -1;
34     }
35
36     int queue[n];
37     int front = 0, rear = 0;
38
39     for (int i = 0; i < n; i++) {
40         if (p[i].arrivalTime == 0) {
41             queue[rear++] = i;
42         }
43     }
44
45     while (completed != n) {
```

```

C rrc > main()
16 int main() {
17     for (int i = 0; i < n; i++) {
18         if (p[i].arrivalTime == 0) {
19             queue[rear++] = i;
20         }
21     }
22
23     while (completed != n) {
24         if (front != rear) {
25             int idx = queue[front++];
26             if (p[idx].startTime == -1) {
27                 p[idx].startTime = currentTime;
28                 p[idx].responseTime = p[idx].startTime - p[idx].arrivalTime;
29                 totalResponseTime += p[idx].responseTime;
30             }
31
32             int execTime = (p[idx].remainingTime > timeQuantum) ? timeQuantum : p[idx].remainingTime;
33             currentTime += execTime;
34             p[idx].remainingTime -= execTime;
35
36             for (int i = 0; i < n; i++) {
37                 if (p[i].arrivalTime <= currentTime && p[i].remainingTime > 0 && i != idx && p[i].startTime == -1) {
38                     queue[rear++] = i;
39                 }
40             }
41
42             if (p[idx].remainingTime == 0) {
43                 p[idx].completionTime = currentTime;
44                 p[idx].turnAroundTime = p[idx].completionTime - p[idx].arrivalTime;
45                 p[idx].waitingTime = p[idx].turnAroundTime - p[idx].burstTime;
46
47                 totalWaitingTime += p[idx].waitingTime;
48                 totalTurnAroundTime += p[idx].turnAroundTime;
49                 p[idx].isCompleted = 1;
50                 completed++;
51             } else {
52                 queue[rear++] = idx;
53             }
54         } else {
55             currentTime++;
56             for (int i = 0; i < n; i++) {
57                 if (p[i].arrivalTime <= currentTime && p[i].remainingTime > 0) {
58                     queue[rear++] = i;
59                     break;
60                 }
61             }
62         }
63     }
64 }

```

```

65     queue[rear++] = idx;
66 } else {
67     currentTime++;
68     for (int i = 0; i < n; i++) {
69         if (p[i].arrivalTime <= currentTime && p[i].remainingTime > 0) {
70             queue[rear++] = i;
71             break;
72         }
73     }
74 }
75 }
76
77 printf("\nProcess\tArrival Time\tBurst Time\tCompletion Time\tWaiting Time\tTurnaround Time\tResponse Time\n");
78 for (int i = 0; i < n; i++) {
79     printf("%d\t%d\t%d\t%d\t%d\t%d\t%d\n", p[i].pid, p[i].arrivalTime, p[i].burstTime,
80         p[i].completionTime, p[i].waitingTime, p[i].turnAroundTime, p[i].responseTime);
81 }
82
83 printf("\nAverage Waiting Time: %.2f\n", totalWaitingTime / n);
84 printf("Average Turnaround Time: %.2f\n", totalTurnAroundTime / n);
85 printf("Average Response Time: %.2f\n", totalResponseTime / n);
86
87 return 0;
88 }

```

## 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
```