

Operating Systems

CT-353

Name: Sabrina Shahzad

Roll No. : DT-026

Lab 02:

- **First Come First Serve Algorithm(FCFS):**

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

```
struct Process {  
    int id, at, bt, ct, wt, tat;  
};
```

```
void swap(struct Process *a, struct Process *b) {  
    struct Process temp = *a;  
    *a = *b;  
    *b = temp;  
}
```

```
int main() {  
    int n, i, j, currentTime = 0;  
    float totalWT = 0, totalTAT = 0;  
    struct Process p[20];  
  
    printf("Enter the number of processes: ");  
    scanf("%d", &n);  
  
    for (i = 0; i < n; i++) {  
        p[i].id = i + 1;  
        printf("Enter Arrival Time for Process %d: ", i + 1);  
        scanf("%d", &p[i].at);  
        printf("Enter Execution Time (Burst Time) for Process %d: ", i + 1);  
        scanf("%d", &p[i].bt);  
    }  
  
    // Sort processes by Arrival Time  
    for (i = 0; i < n - 1; i++) {  
        for (j = 0; j < n - i - 1; j++) {  
            if (p[j].at > p[j + 1].at) {
```

```

        swap(&p[j], &p[j + 1]);
    }
}

// Calculate Completion Time, Turnaround Time, and Waiting Time
for (i = 0; i < n; i++) {
    if (currentTime < p[i].at) {
        currentTime = p[i].at; // Idle time if process arrives later
    }
    p[i].ct = currentTime + p[i].bt; // Completion Time
    currentTime = p[i].ct;

    p[i].tat = p[i].ct - p[i].at; // Turnaround Time = CT - AT
    p[i].wt = p[i].tat - p[i].bt; // Waiting Time = TAT - BT

    totalWT += p[i].wt;
    totalTAT += p[i].tat;
}

// Display Results
printf("\nPROCESS\tARRIVAL TIME\tEXECUTION TIME\tCOMPLETION\n\tTIME\tWAITING TIME\tTURNAROUND TIME\n");
for (i = 0; i < n; i++) {
    printf("P%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n",
        p[i].id, p[i].at, p[i].bt, p[i].ct, p[i].wt, p[i].tat);
}

printf("\nAverage Waiting Time: %.2f", totalWT / n);
printf("\nAverage Turnaround Time: %.2f\n", totalTAT / n);

return 0;
}

```

Output:

```
Enter the number of processes: 4
Enter Arrival Time for Process 1: 3
Enter Execution Time (Burst Time) for Process 1: 2
Enter Arrival Time for Process 2: 1
Enter Execution Time (Burst Time) for Process 2: 1
Enter Arrival Time for Process 3: 0
Enter Execution Time (Burst Time) for Process 3: 3
Enter Arrival Time for Process 4: 4
Enter Execution Time (Burst Time) for Process 4: 2

PROCESS ARRIVAL TIME    EXECUTION TIME    COMPLETION TIME    WAITING TIME    TURNAROUND TIME
P3          0           3           3           0           3
P2          1           1           4           2           3
P1          3           2           6           1           3
P4          4           2           8           2           4

Average Waiting Time: 1.25
Average Turnaround Time: 3.25

-----
Process exited after 15.19 seconds with return value 0
Press any key to continue . . . |
```

- **Shortest Job First Algorithm(SJF):**

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

struct Process {
    int id, at, bt, ct, wt, tat; // Process attributes
    bool completed;             // To mark if the process is completed
};

void sortByArrival(struct Process p[], int n) {
    int i, j;
    for (i = 0; i < n - 1; i++) {
        for (j = 0; j < n - i - 1; j++) {
            if (p[j].at > p[j + 1].at) {
                struct Process temp = p[j];
                p[j] = p[j + 1];
                p[j + 1] = temp;
            }
        }
    }
}

int main() {
    int n, i, completedCount = 0, currentTime = 0;
```

```

float totalWT = 0, totalTAT = 0;
struct Process p[20];

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

for (i = 0; i < n; i++) {
    p[i].id = i + 1;
    printf("Enter Arrival Time for Process %d: ", i + 1);
    scanf("%d", &p[i].at);
    printf("Enter Execution Time (Burst Time) for Process %d: ", i + 1);
    scanf("%d", &p[i].bt);
    p[i].completed = false; // Mark as incomplete
}

// Sort processes by Arrival Time
sortByArrival(p, n);

while (completedCount < n) {
    int shortestIndex = -1;
    int minBurstTime = 9999;

    // Find the shortest process that has arrived
    for (i = 0; i < n; i++) {
        if (!p[i].completed && p[i].at <= currentTime && p[i].bt < minBurstTime)
        {
            minBurstTime = p[i].bt;
            shortestIndex = i;
        }
    }

    if (shortestIndex != -1) {
        // Process the shortest job
        currentTime += p[shortestIndex].bt;
        p[shortestIndex].ct = currentTime; // Completion Time
        p[shortestIndex].tat = p[shortestIndex].ct - p[shortestIndex].at; //
Turnaround Time
        p[shortestIndex].wt = p[shortestIndex].tat - p[shortestIndex].bt; //
Waiting Time
        p[shortestIndex].completed = true;

        totalWT += p[shortestIndex].wt;
        totalTAT += p[shortestIndex].tat;
        completedCount++;
    }
}

```

```

    } else {
        // If no process is ready, increment the current time
        currentTime++;
    }
}

// Display Results
printf("\nPROCESS\tARRIVAL TIME\tEXECUTION TIME\tCOMPLETION
TIME\tWAITING TIME\tTURNAROUND TIME\n");
for (i = 0; i < n; i++) {
    printf("P%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n",
        p[i].id, p[i].at, p[i].bt, p[i].ct, p[i].wt, p[i].tat);
}

printf("\nAverage Waiting Time: %.2f", totalWT / n);
printf("\nAverage Turnaround Time: %.2f\n", totalTAT / n);

return 0;
}

```

Output:

```

Enter the number of processes: 4
Enter Arrival Time for Process 1: 3
Enter Execution Time (Burst Time) for Process 1: 2
Enter Arrival Time for Process 2: 1
Enter Execution Time (Burst Time) for Process 2: 1
Enter Arrival Time for Process 3: 0
Enter Execution Time (Burst Time) for Process 3: 3
Enter Arrival Time for Process 4: 4
Enter Execution Time (Burst Time) for Process 4: 2

PROCESS ARRIVAL TIME    EXECUTION TIME  COMPLETION TIME  WAITING TIME    TURNAROUND TIME
P3         0             3                3                0                3
P2         1             1                4                2                3
P1         3             2                6                1                3
P4         4             2                8                2                4

Average Waiting Time: 1.25
Average Turnaround Time: 3.25

-----
Process exited after 17.3 seconds with return value 0
Press any key to continue . . . |

```