Operating Systems (CT-353) Lab 02

Name: Ajiya Anwar
Roll No: DT-22006

Lab 02:

- **Round Robin Algorithm:**

```
#include <stdio.h>

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

int main() {
    int n, i, time = 0, completed = 0, tq;
    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 Burst Time for Process %d: ", i + 1);
        scanf("%d", &p[i].bt);
        p[i].remaining_bt = p[i].bt; // Initialize remaining burst time
    }

    printf("Enter Time Quantum: ");
    scanf("%d", &tq);

    // Sort processes by Arrival Time
    for (i = 0; i < n - 1; i++) {
        for (int 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 queue[20], front = 0, rear = 0; // Queue for process execution
int visited[20] = {0}; // Keep track of visited processes

queue[rear++] = 0; // Start with the first process
visited[0] = 1;

while (completed < n) {
    int current = queue[front++];

    // Process the current process
    if (time < p[current].at) {
        time = p[current].at; // Idle time
    }

    if (p[current].remaining_bt <= tq) {
        time += p[current].remaining_bt;
        p[current].remaining_bt = 0;
        p[current].ct = time; // Completion time
        p[current].tat = p[current].ct - p[current].at; // Turnaround Time
        p[current].wt = p[current].tat - p[current].bt; // Waiting Time
        totalWT += p[current].wt;
        totalTAT += p[current].tat;
        completed++;
    } else {
        time += tq;
        p[current].remaining_bt -= tq;
    }

    // Add processes that arrived during execution of current process to the queue
    for (i = 0; i < n; i++) {
        if (i != current && !visited[i] && p[i].at <= time && p[i].remaining_bt > 0) {
            queue[rear++] = i;
            visited[i] = 1;
        }
    }
```

```
 // Requeue the current process if it is not yet complete
        if (p[current].remaining_bt > 0) {
            queue[rear++] = current;
        }
    }

    // Display Results
    printf("\nPROCESS\tARRIVAL TIME\tBURST 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 Burst Time for Process 1: 2
Enter Arrival Time for Process 2: 2
Enter Burst Time for Process 2: 4
Enter Arrival Time for Process 3: 0
Enter Burst Time for Process 3: 4
Enter Arrival Time for Process 4: 1
Enter Burst Time for Process 4: 6
Enter Time Quantum: 2

PROCESS ARRIVAL TIME    BURST TIME      COMPLETION TIME WAITING TIME    TURNAROUND TIME
P3              0               4               8               4               8
P4              1               6               16              9               15
P2              2               4               14              8               12
P1              3               2               10              5               7

Average Waiting Time: 6.50
Average Turnaround Time: 10.50
```

- **Priority Based Algorithm:**

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

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

int main() {
    int n, time = 0, completed = 0;
    float totalWT = 0, totalTAT = 0;
    struct Process p[20];

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

    // Input process details
    for (int 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 Burst Time for Process %d: ", i + 1);
        scanf("%d", &p[i].bt);
        printf("Enter Priority for Process %d (lower number = higher priority): ", i
+ 1);
        scanf("%d", &p[i].priority);
        p[i].remaining_bt = p[i].bt; // Initialize remaining burst time
    }

    // Priority Scheduling Logic (Preemptive)
    while (completed < n) {
        int minPriority = INT_MAX, current = -1;

        // Find the process with the highest priority that has arrived
        for (int i = 0; i < n; i++) {
            if (p[i].at <= time && p[i].remaining_bt > 0 && p[i].priority < minPriority)
{
                minPriority = p[i].priority;
                current = i;
            }
        }
```

```
        if (current == -1) { // If no process is ready, increment time
            time++;
            continue;
        }

        // Execute the selected process for 1 unit of time
        p[current].remaining_bt--;
        time++;

        // If the process is completed
        if (p[current].remaining_bt == 0) {
            p[current].ct = time; // Completion Time
            p[current].tat = p[current].ct - p[current].at; // Turnaround Time
            p[current].wt = p[current].tat - p[current].bt; // Waiting Time
            totalWT += p[current].wt;
            totalTAT += p[current].tat;
            completed++;
        }
    }

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

    // Print averages
    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 Burst Time for Process 1: 2
Enter Priority for Process 1 (lower number = higher priority): 2
Enter Arrival Time for Process 2: 2
Enter Burst Time for Process 2: 4
Enter Priority for Process 2 (lower number = higher priority): 1
Enter Arrival Time for Process 3: 0
Enter Burst Time for Process 3: 4
Enter Priority for Process 3 (lower number = higher priority): 2
Enter Arrival Time for Process 4: 1
Enter Burst Time for Process 4: 6
Enter Priority for Process 4 (lower number = higher priority): 3

PROCESS ARRIVAL TIME    BURST TIME      PRIORITY        COMPLETION TIME WAITING TIME    TURNAROUND TIME
P1              3               2               2               8               3               5
P2              2               4               1               6               0               4
P3              0               4               2               10              6               10
P4              1               6               3               16              9               15

Average Waiting Time: 4.50
Average Turnaround Time: 8.50
```

Operating Systems (CT-353) Lab 02

Name: Ajiya Anwar
Roll No: DT-22006

- **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;
    }
    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 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 . . .
```