NED UNIVERSITY OF ENGINEERING & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE & IT

Specialization in Data Science

CT-353
OPERATING SYSTEMS

Name : Afifa Siddique

Roll No : DT-22003

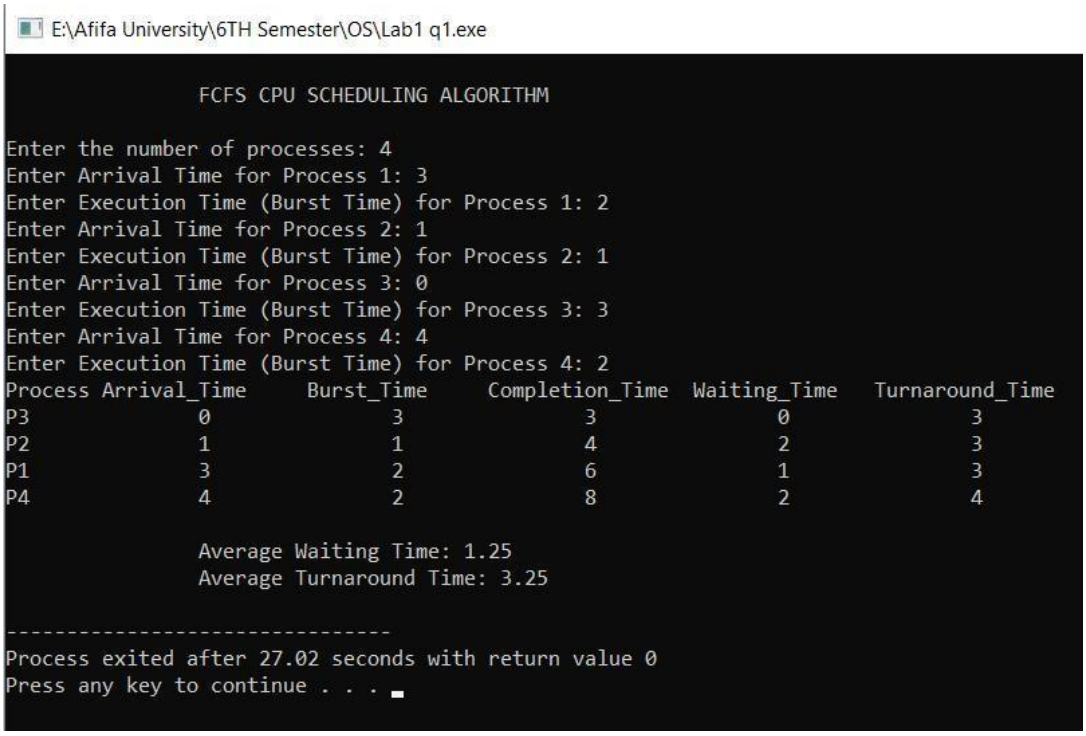
Submitted to : Sir Muhammad Abdullah Siddiqui

LAB NO : 02

FCFS CPU SCHEDULING ALGORITHM

```
#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("\n\t\tFCFS CPU SCHEDULING ALGORITHM\n\n");

    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("Process\tArrival_Time\t Burst_Time\tCompletion_Time\t
Waiting_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("\n\t\tAverage Waiting Time: %.2f", totalWT / n);
    printf("\n\t\tAverage Turnaround Time: %.2f\n", totalTAT / n);

    return 0;
}
```



```
E:\Afifa University\6TH Semester\OS\Lab1 q1.exe

                FCFS CPU SCHEDULING ALGORITHM

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     Burst_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 27.02 seconds with return value 0
Press any key to continue . . .
```

SJF CPU SCHEDULING ALGORITHM

```
#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("\n\t\tSJF CPU SCHEDULING ALGORITHM\n\n");

    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("Process\tArrival_Time\t Burst_Time\tCompletion_Time\t
Waiting_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("\n\t\tAverage Waiting Time: %.2f", totalWT / n);
    printf("\n\t\tAverage Turnaround Time: %.2f\n", totalTAT / n);

    return 0;
}
```

E:\Afifa University\6TH Semester\OS\lab2 q2.exe

```
                SJF CPU SCHEDULING ALGORITHM

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     Burst_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 13.5 seconds with return value 0
Press any key to continue . . .
```

ROUND ROBIN CPU SCHEDULING ALGORITHM

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

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

void calculateRoundRobin(struct Process p[], int n, int tq)
    { int time = 0, completed = 0;
    float totalWT = 0, totalTAT = 0;
    bool processExecuted = false;
    int queue[20], front = 0, rear = 0;
    bool inQueue[20] = {false}; // Tracks if a process is already in the queue

    // Sort processes by Arrival Time
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++)
            { if (p[i].at > p[j].at) {
                struct Process temp = p[i];
                p[i] = p[j];
                p[j] = temp;
            }
        }
    }

    // Add the first process to the queue
    queue[rear++] = 0;
    inQueue[0] = true;

    while (completed < n)
        { processExecuted = false;

        // Check the front of the queue
        int i = queue[front++];
        if (front == 20) front = 0;

        // If the process has arrived and has remaining time
        if (p[i].rt > 0 && p[i].at <= time) {
            processExecuted = true;
            if (p[i].rt <= tq) {
                time += p[i].rt;  // Add remaining burst time
                p[i].rt = 0;       // Process completes
                p[i].ct = time;   // Set completion time
                p[i].tat = p[i].ct - p[i].at; // TAT = CT - AT
```

```
            p[i].wt = p[i].tat - p[i].bt; // WT = TAT - BT
            totalTAT += p[i].tat;
            totalWT += p[i].wt;
            completed++;
        } else {
            time += tq;       // Add time quantum
            p[i].rt -= tq;    // Decrease remaining burst time
        }
    }

    // Enqueue processes that have arrived while this process was executing
    for (int j = 0; j < n; j++) {
        if (p[j].at <= time && p[j].rt > 0 && !inQueue[j])
            { queue[rear++] = j;
            if (rear == 20) rear = 0;
            inQueue[j] = true;
        }
    }

    // Re-add the current process to the queue if it hasn't completed
    if (p[i].rt > 0) {
        queue[rear++] = i;
        if (rear == 20) rear = 0;
    }

    // If no process was executed, increment time
    if (!processExecuted) {
        time++;
        // Re-add the current process back to the queue
        queue[front--] = i;
    }
}

// Display results

printf("\nProcess\tArrival_Time\tBurst_Time\tCompletion_Time\tWaiting_Time\tTur
naround_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\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);
}

int main()
    { int n, tq;
    struct Process p[20];
```

```c
    printf("\n\t\tRound Robin CPU SCHEDULING ALGORITHM\n\n");

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

    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);
      p[i].rt = p[i].bt; // Remaining time is initially the burst time
    }

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

    // Perform Round Robin Scheduling
    calculateRoundRobin(p, n, tq);

    return 0;
}
```

E:\Afifa University\6TH Semester\OS\L2 q3.exe

```
                Round Robin CPU SCHEDULING ALGORITHM

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

--------------------------------
Process exited after 18.91 seconds with return value 0
Press any key to continue . . .
```

PRIORITY CPU SCHEDULING ALGORITHM

```
#include <stdio.h>

// Define the structure for a process
struct Process {
    int id, at, bt, ct, tat, wt, priority;
};

void calculatePriorityScheduling(struct Process p[], int n)
    { int time = 0, completed = 0;
    float totalTAT = 0, totalWT = 0;
    int isCompleted[20] = {0}; // To track completed processes

    while (completed < n)
        { int idx = -1;
        int highestPriority = 9999; // Initialize to a very high value

        // Find the process with the highest priority that has arrived and is not completed
        for (int i = 0; i < n; i++) {
              if (p[i].at <= time && !isCompleted[i])
                 { if (p[i].priority < highestPriority)
                   { highestPriority = p[i].priority;
                   idx = i;
                } else if (p[i].priority == highestPriority) {
                   // If priorities are equal, choose based on arrival time
                   if (p[i].at < p[idx].at) {
                       idx = i;
                   }
                }
            }
        }

        if (idx != -1) {
            // Process the selected process
            time += p[idx].bt;
            p[idx].ct = time;                        // Completion Time
            p[idx].tat = p[idx].ct - p[idx].at;      // Turnaround Time = CT - AT
            p[idx].wt = p[idx].tat - p[idx].bt;      // Waiting Time = TAT - BT
            totalTAT += p[idx].tat;
            totalWT += p[idx].wt;
            isCompleted[idx] = 1;                    // Mark as completed
            completed++;
        } else {
            time++; // If no process is available, increment time
        }
    }

    // Display the results

printf("\nProcess\tArrival_Time\tBurst_Time\tPriority\tCompletion_Time\tTurnaround_Time
\tWaiting_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].tat, p[i].wt);
```

```
    }

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

int main()
    { int n;
    struct Process p[20];

    printf("\n\t\tPRIORITY CPU SCHEDULING ALGORITHM\n\n");

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

    for (int i = 0; i < n; i++)
        { p[i].id = i + 1;
        printf("Enter Arrival Time for Process P%d: ", i + 1);
        scanf("%d", &p[i].at);
        printf("Enter Burst Time for Process P%d: ", i + 1);
        scanf("%d", &p[i].bt);
        printf("Enter Priority for Process P%d (lower value = higher priority): ", i + 1);
        scanf("%d", &p[i].priority);
    }

    // Perform Priority Scheduling
    calculatePriorityScheduling(p, n);

    return 0;
}
```

E:\Afifa University\6TH Semester\OS\L2 q4.exe

```
                PRIORITY CPU SCHEDULING ALGORITHM

Enter the number of processes: 4
Enter Arrival Time for Process P1: 3
Enter Burst Time for Process P1: 2
Enter Priority for Process P1 (lower value = higher priority): 2
Enter Arrival Time for Process P2: 2
Enter Burst Time for Process P2: 4
Enter Priority for Process P2 (lower value = higher priority): 1
Enter Arrival Time for Process P3: 0
Enter Burst Time for Process P3: 4
Enter Priority for Process P3 (lower value = higher priority): 2
Enter Arrival Time for Process P4: 1
Enter Burst Time for Process P4: 6
Enter Priority for Process P4 (lower value = higher priority): 3

Process Arrival_Time    Burst_Time      Priority        Completion_Time Turnaround_Time Waiting_Time
P1              3               2               2               10              7               5
P2              2               4               1               8               6               2
P3              0               4               2               4               4               0
P4              1               6               3               16              15              9

Average Turnaround Time: 8.00
Average Waiting Time: 4.00

--------------------------------
Process exited after 24.89 seconds with return value 0
Press any key to continue . . .
```

5) Execute all scheduling algorithms on following data and find out the Average Waiting Time and Average Turnaround Time of all scheduling algorithms and discuss your results.
(Quantum Value is 3)

Process Name	Brust Time	Priority
P0	2	3
P1	6	1
P2	4	2

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

struct Process {
    int burstTime;
    int priority;
    int arrivalTime;
    int waitingTime;
    int turnaroundTime;
};

float calculateAvgWT_TT_FCFS(vector<Process>& processes)
    { int n = processes.size();
    processes[0].waitingTime = 0;
    for (int i = 1; i < n; i++) {
        processes[i].waitingTime = processes[i - 1].burstTime + processes[i - 1].waitingTime;
    }

    float avgWT = 0, avgTT = 0;
    for (int i = 0; i < n; i++) {
        processes[i].turnaroundTime = processes[i].burstTime + processes[i].waitingTime;
        avgWT += processes[i].waitingTime;
        avgTT += processes[i].turnaroundTime;
    }
    avgWT /= n;
    avgTT /= n;

    cout << "FCFS - Average Waiting Time: " << avgWT << ", Average Turnaround Time: "
<< avgTT << endl;
    return avgWT;
}

float calculateAvgWT_TT_SJF(vector<Process>& processes)
    { int n = processes.size();
    sort(processes.begin(), processes.end(), [](Process a, Process b)
        { return a.burstTime < b.burstTime;
    });
```

```
    processes[0].waitingTime = 0;
    for (int i = 1; i < n; i++) {
        processes[i].waitingTime = processes[i - 1].burstTime + processes[i - 1].waitingTime;
    }

    float avgWT = 0, avgTT = 0;
    for (int i = 0; i < n; i++) {
        processes[i].turnaroundTime = processes[i].burstTime + processes[i].waitingTime;
        avgWT += processes[i].waitingTime;
        avgTT += processes[i].turnaroundTime;
    }
    avgWT /= n;
    avgTT /= n;

    cout << "SJF - Average Waiting Time: " << avgWT << ", Average Turnaround Time: " <<
avgTT << endl;
    return avgWT;
}

float calculateAvgWT_TT_RR(vector<Process>& processes, int quantum)
    { int n = processes.size();
    vector<int> remainingBurstTime(n);
    for (int i = 0; i < n; i++) {
        remainingBurstTime[i] = processes[i].burstTime;
    }

    int time = 0;
    while (true) {
        bool done = true;
        for (int i = 0; i < n; i++) {
            if (remainingBurstTime[i] > 0)
                { done = false;
                if (remainingBurstTime[i] > quantum)
                    { time += quantum;
                    remainingBurstTime[i] -= quantum;
                } else {
                    time += remainingBurstTime[i];
                    processes[i].waitingTime = time - processes[i].burstTime;
                    remainingBurstTime[i] = 0;
                }
            }
        }
        if (done) break;
    }

    float avgWT = 0, avgTT = 0;
    for (int i = 0; i < n; i++) {
        processes[i].turnaroundTime = processes[i].burstTime + processes[i].waitingTime;
        avgWT += processes[i].waitingTime;
        avgTT += processes[i].turnaroundTime;
    }
    avgWT /= n;
    avgTT /= n;
```

```
    cout << "RR - Average Waiting Time: " << avgWT << ", Average Turnaround Time: " <<
avgTT << endl;
    return avgWT;
}

float calculateAvgWT_TT_Priority(vector<Process>& processes)
    { int n = processes.size();
    sort(processes.begin(), processes.end(), [](Process a, Process b)
        { return a.priority < b.priority;
    });

    processes[0].waitingTime = 0;
    for (int i = 1; i < n; i++) {
        processes[i].waitingTime = processes[i - 1].burstTime + processes[i - 1].waitingTime;
    }

    float avgWT = 0, avgTT = 0;
    for (int i = 0; i < n; i++) {
        processes[i].turnaroundTime = processes[i].burstTime + processes[i].waitingTime;
        avgWT += processes[i].waitingTime;
        avgTT += processes[i].turnaroundTime;
    }
    avgWT /= n;
    avgTT /= n;

    cout << "Priority - Average Waiting Time: " << avgWT << ", Average Turnaround Time: "
<< avgTT << endl;
    return avgWT;
}

int main()
    { int n;
    cout << "Enter the number of processes: ";
    cin >> n;

    vector<Process> processes(n);

    for (int i = 0; i < n; i++) {
        cout << "Enter Burst Time for Process " << i << ": ";
        cin >> processes[i].burstTime;
        processes[i].arrivalTime = 0;  // Set Arrival Time to 0 for all processes
        cout << "Enter Priority for Process " << i << ": ";
        cin >> processes[i].priority;
    }

    int quantum = 3;

    calculateAvgWT_TT_FCFS(processes);
    calculateAvgWT_TT_SJF(processes);
    calculateAvgWT_TT_RR(processes, quantum);
    calculateAvgWT_TT_Priority(processes);

    return 0;
}
```

E:\Afifa University\6TH Semester\OS\L2 q5.exe

```
Enter the number of processes: 3
Enter Burst Time for Process 0: 2
Enter Priority for Process 0: 3
Enter Burst Time for Process 1: 6
Enter Priority for Process 1: 1
Enter Burst Time for Process 2: 4
Enter Priority for Process 2: 2
FCFS - Average Waiting Time: 3.33333, Average Turnaround Time: 7.33333
SJF - Average Waiting Time: 2.66667, Average Turnaround Time: 6.66667
RR - Average Waiting Time: 3.66667, Average Turnaround Time: 7.66667
Priority - Average Waiting Time: 5.33333, Average Turnaround Time: 9.33333

--------------------------------
Process exited after 28.18 seconds with return value 0
Press any key to continue . . .
```
