Atma Ram Sanatan Dharma College

University of Delhi

Operating System

Practical File

Submitted By:

Jyotiswaroop Srivastav
College Roll No. 21/18023
Semester III
BSc. (Hons) Computer Science

Submitted To:

Ms. Parul Jain

6. Write a program to implement FCFS scheduling algorithm.

```c
/**
 * Write a program to implement FCFS scheduling algorithm.
 */

#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 100

struct process
{
    int pid;
    double burstTime;
    double arrivalTime;
    double waitingTime;
    double turnAroundTime;
};

void sortByArrivalTime(struct process *processes, int
processCount)
{
    struct process temp;
    int i, j, n = processCount;
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            if (processes[i].arrivalTime <
processes[j].arrivalTime)
            {
                temp = processes[j];
                processes[j] = processes[i];
                processes[i] = temp;
            }
}

void sortByPID(struct process *processes, int processCount)
{
```

```
    struct process temp;
    int i, j, n = processCount;
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            if (processes[i].pid < processes[j].pid)
            {
                temp = processes[j];
                processes[j] = processes[i];
                processes[i] = temp;
            }
}

void computeWaitingTime(struct process *processes, int
processCount)
{
    double completionTime = 0.0;
    processes[0].waitingTime = 0.0;
    for (int i = 1; i < processCount; i++)
    {
        completionTime += processes[i - 1].burstTime;
        processes[i].waitingTime = completionTime -
processes[i].arrivalTime;
    }
    return;
}

void computeTurnAroundTime(struct process *processes, int
processCount)
{
    for (int i = 0; i < processCount; i++)
        processes[i].turnAroundTime =
            processes[i].burstTime + processes[i].waitingTime;
    return;
}

void printAverageTimes(struct process *processes, int
processCount,
                       char *unit)
```

```
{
    double totalWaitingTime = 0.0;
    double totalTurnAroundTime = 0.0;
    double completionTimes[MAX_SIZE] = {0.0};
    sortByArrivalTime(processes, processCount);
    computeWaitingTime(processes, processCount);
    computeTurnAroundTime(processes, processCount);
    sortByPID(processes, processCount);
    printf(
        "Process ID\tBurst Time\tArrival Time\tWaiting
Time\tTurn-Around Time\n");
    printf("-----------------------------------------------
-----");
    printf("-------------------------\n");
    for (int i = 0; i < processCount; i++)
    {
        totalWaitingTime += processes[i].waitingTime;
        totalTurnAroundTime += processes[i].turnAroundTime;
        printf("%d\t\t%.2lf%s\t\t%.2lf%s\t\t%.2lf%s\t\t%.2lf%s\
n", processes[i].pid,
               processes[i].burstTime, unit,
processes[i].arrivalTime, unit,
               processes[i].waitingTime, unit,
processes[i].turnAroundTime, unit);
    }
    printf("\nAverage Waiting Time = %.2lf%s", totalWaitingTime
/ processCount,
           unit);
    printf("\nAverage Turn-Around time = %.2lf%s\n",
           totalTurnAroundTime / processCount, unit);
    return;
}

int main(void)
{
    int processCount;
    char unit[4] = {'\0'};
```

```c
    printf("Enter Time Unit: ");
    fgets(unit, 3, stdin);

    printf("Enter Number of Processes: ");
    scanf("%i", &processCount);

    struct process processes[processCount];

    for (int i = 0; i < processCount; i++)
    {
        processes[i].pid = i + 1;
        printf("Burst Time for Process %i: ", i + 1);
        scanf("%lf", &processes[i].burstTime);
        printf("Arrival Time for Process %i: ", i + 1);
        scanf("%lf", &processes[i].arrivalTime);
    }

    printf("\n");

    printAverageTimes(processes, processCount, unit);

    return 0;
}
```

```
Enter Time Unit: ms
Enter Number of Processes: 3
Burst Time for Process 1: 8
Arrival Time for Process 1: 0
Burst Time for Process 2: 4
Arrival Time for Process 2: 0.4
Burst Time for Process 3: 1
Arrival Time for Process 3: 1

Process ID      Burst Time      Arrival Time    Waiting Time    Turn-Around Time
-------------------------------------------------------------------------------
1               8.00ms          0.00ms          0.00ms          8.00ms
2               4.00ms          0.40ms          7.60ms          11.60ms
3               1.00ms          1.00ms          11.00ms         12.00ms

Average Waiting Time = 6.20ms
Average Turn-Around time = 10.53ms
```

7. Write a program to implement Round Robin scheduling algorithm.

```c
/**
 * Write a program to implement Round Robin scheduling
algorithm.
 */

#include <stdio.h>
#include <stdlib.h>

struct process
{
    int pid;
    double burstTime;
    double arrivalTime;
    double waitingTime;
    double turnAroundTime;
};

void computeWaitingTime(struct process *processes, int
processCount, int quantum)
{
    double remainingTime[processCount];
    for (int i = 0; i < processCount; i++)
        remainingTime[i] = processes[i].burstTime;
    double time = 0.0;
    while (1)
    {
        int done = 1;
        for (int i = 0; i < processCount; i++)
        {
            if (remainingTime[i] > 0)
            {
                done = 0;
                if (remainingTime[i] > quantum)
                {
                    time += quantum;
```

```
                    remainingTime[i] -= quantum;
                }
                else
                {
                    time += remainingTime[i];
                    processes[i].waitingTime += time -
processes[i].arrivalTime - processes[i].burstTime;
                    remainingTime[i] = 0;
                }
            }
        }
        if (done == 1)
            break;
    }
    return;
}

void computeTurnAroundTime(struct process *processes, int
processCount)
{
    for (int i = 0; i < processCount; i++)
        processes[i].turnAroundTime =
            processes[i].burstTime +
            processes[i].waitingTime -
            processes[i].arrivalTime;
    return;
}

void printAverageTimes(struct process *processes, int
processCount, int quantum, char *unit)
{
    double totalWaitingTime = 0.0;
    double totalTurnAroundTime = 0.0;
    computeWaitingTime(processes, processCount, quantum);
    computeTurnAroundTime(processes, processCount);
    printf("Process ID\tBurst Time\tArrival Time\tWaiting
Time\tTurn-Around Time\n");
```

```
    printf("-----------------------------------------------------
-----");
    printf("-------------------------\n");
    for (int i = 0; i < processCount; i++)
    {
        totalWaitingTime += processes[i].waitingTime;
        totalTurnAroundTime += processes[i].turnAroundTime;
        printf("%d\t\t%.2lf%s\t\t%.2lf%s\t\t%.2lf%s\t\t%.2lf%s\
n", processes[i].pid,
               processes[i].burstTime, unit,
processes[i].arrivalTime, unit,
               processes[i].waitingTime, unit,
processes[i].turnAroundTime, unit);
    }
    printf("\nAverage Waiting Time = %.2lf%s", totalWaitingTime
/ processCount,
           unit);
    printf("\nAverage Turn-Around time = %.2lf%s\n",
           totalTurnAroundTime / processCount, unit);
    return;
}

int main(void)
{
    double quantum;
    int processCount;
    char unit[4] = {'\0'};

    printf("Enter Time Unit: ");
    fgets(unit, 3, stdin);

    printf("Enter Time Quantum: ");
    scanf("%lf", &quantum);

    printf("Enter Number of Processes: ");
    scanf("%i", &processCount);

    struct process processes[processCount];
```

```c
    for (int i = 0; i < processCount; i++)
    {
        processes[i].pid = i + 1;
        printf("Burst Time for Process %i: ", i + 1);
        scanf("%lf", &processes[i].burstTime);
        printf("Arrival Time for Process %i: ", i + 1);
        scanf("%lf", &processes[i].arrivalTime);
    }

    printf("\n");

    printAverageTimes(processes, processCount, quantum, unit);

    return 0;
}
```

```
$ gcc -o main main.c
$ ./main
Enter Time Quantum: 1
Enter Number of Processes: 3
Burst Time for Process 1: 3
Arrival Time for Process 1: 0
Burst Time for Process 2: 4
Arrival Time for Process 2: 0
Burst Time for Process 3: 3
Arrival Time for Process 3: 0

Process ID      Burst Time      Arrival Time    Waiting Time    Turn-Around Time
--------------------------------------------------------------------------------
1               3               0               4               7
2               4               0               6               10
3               3               0               6               9

Average Waiting Time = 5.33
Average Turn-Around time = 8.67
```

8. Write a program to implement SJF scheduling algorithm.

```c
/**
```

```c
 * Write a program to implement SJF scheduling algorithm.
 */

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

struct process
{
    int pid;
    double burstTime;
    double arrivalTime;
    double waitingTime;
    double turnAroundTime;
};

void sortByArrivalTime(struct process *processes, int
processCount)
{
    struct process temp;
    int i, j, n = processCount;
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            if (processes[i].arrivalTime <
processes[j].arrivalTime)
            {
                temp = processes[j];
                processes[j] = processes[i];
                processes[i] = temp;
            }
}

void sortForSJF(struct process *processes, int processCount)
{
    struct process temp;
    double min, startTime = 0.0;
    int i, j, k = 1, n = processCount;
    for (j = 0; j < n; j++)
```

```
    {
        startTime += processes[j].burstTime;
        min = processes[k].burstTime;
        for (i = k; i < n; i++)
            if (startTime >= processes[i].arrivalTime &&
                processes[i].burstTime < min)
            {
                temp = processes[k];
                processes[k] = processes[i];
                processes[i] = temp;
            }
        k++;
    }
}

void sortByPID(struct process *processes, int processCount)
{
    struct process temp;
    int i, j, n = processCount;
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            if (processes[i].pid < processes[j].pid)
            {
                temp = processes[j];
                processes[j] = processes[i];
                processes[i] = temp;
            }
}

void computeWaitingTime(struct process *processes, int
processCount)
{
    double startTime = 0.0;
    processes[0].waitingTime = 0;
    for (int i = 1; i < processCount; i++)
    {
        startTime += processes[i - 1].burstTime;
```

```
        processes[i].waitingTime = startTime -
processes[i].arrivalTime;
    }
}

void computeTurnAroundTime(struct process *processes, int
processCount)
{
    for (int i = 0; i < processCount; i++)
        processes[i].turnAroundTime =
            processes[i].burstTime + processes[i].waitingTime;
}

void printAverageTimes(struct process *processes, int
processCount,
                       char *unit)
{
    double totalWaitingTime = 0.0;
    double totalTurnAroundTime = 0.0;
    sortByArrivalTime(processes, processCount);
    sortForSJF(processes, processCount);
    computeWaitingTime(processes, processCount);
    computeTurnAroundTime(processes, processCount);
    sortByPID(processes, processCount);
    printf(
        "Process ID\tBurst Time\tArrival Time\tWaiting
Time\tTurn-Around Time\n");
    printf("-----------------------------------------------
-----");
    printf("-------------------------\n");
    for (int i = 0; i < processCount; i++)
    {
        totalWaitingTime += processes[i].waitingTime;
        totalTurnAroundTime += processes[i].turnAroundTime;
        printf("%d\t\t%.2lf%s\t\t%.2lf%s\t\t%.2lf%s\t\t%.2lf%s\
n", processes[i].pid,
               processes[i].burstTime, unit,
processes[i].arrivalTime, unit,
```

```
                processes[i].waitingTime, unit,
processes[i].turnAroundTime, unit);
    }
    printf("\nAverage Waiting Time = %.2lf%s", totalWaitingTime
/ processCount,
           unit);
    printf("\nAverage Turn-Around time = %.2lf%s\n",
           totalTurnAroundTime / processCount, unit);
    return;
}

int main(void)
{
    int processCount;
    char unit[4] = {'\0'};

    printf("Enter Time Unit: ");
    fgets(unit, 3, stdin);

    printf("Enter Number of Processes: ");
    scanf("%i", &processCount);

    struct process processes[processCount];

    for (int i = 0; i < processCount; i++)
    {
        processes[i].pid = i + 1;
        printf("Burst Time for Process %i: ", i + 1);
        scanf("%lf", &processes[i].burstTime);
        printf("Arrival Time for Process %i: ", i + 1);
        scanf("%lf", &processes[i].arrivalTime);
    }

    printf("\n");

    printAverageTimes(processes, processCount, unit);

    return 0;
```

```
}
```

```
Enter Time Unit: ms
Enter Number of Processes: 3
Burst Time for Process 1: 8
Arrival Time for Process 1: 0
Burst Time for Process 2: 4
Arrival Time for Process 2: 0.4
Burst Time for Process 3: 1
Arrival Time for Process 3: 1

Process ID      Burst Time      Arrival Time    Waiting Time    Turn-Around Time
-------------------------------------------------------------------------------
1               8.00ms          0.00ms          0.00ms          8.00ms
2               4.00ms          0.40ms          8.60ms          12.60ms
3               1.00ms          1.00ms          7.00ms          8.00ms

Average Waiting Time = 5.20ms
Average Turn-Around time = 9.53ms
```