TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

THAPATHALI CAMPUS

A LAB REPORT ON

CPU Scheduling Algorithms

SUBMITTED BY

Sushovan Shakya

THA075BEI046

SUBMITTED TO

Department of Electronics and Computer Engineering

26th August, 2021

CPU SCHEDULING ALGORITHM

Objective

- To implement FCFS algorithm
- To implement SJF algorithm

Theory

First Come First Serve (FCFS) Algorithm

It is the simplest CPU scheduling algorithm. The FCFS scheduling algorithm is non-preemptive i.e. once the CPU has been allocated to a process, that process keeps the CPU until it releases the CPU, either by terminating or by requesting I/O. In this technique, the process that requests the CPU first is allocated the CPU first i.e. when a process enters the ready queue, its PCB is linked onto the tail of the queue. When the CPU is free, it is allocated to the process at the head of the queue. The running process is then removed from the queue.
The average waiting time under this technique is often quite long.

Source Code

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

// FCFS Algorithm

struct fcfs {
    int process;
    int burst;
    int arrival;
    int turnaround;
    int waiting;
};

typedef struct fcfs FCFS;
```

```
// Selection sort
void sort(FCFS arr[], int n) {
    int i, j, min_idx;
    FCFS temp;

    for(i = 0; i < n - 1; ++i) {
        min_idx = i;

        for(j = i+1; j < n; ++j) {
            if(arr[j].arrival < arr[min_idx].arrival)
                min_idx = j;
        }

        // Swapping
        temp = arr[min_idx];
        arr[min_idx] = arr[i];
        arr[i] = temp;
    }
}

int main() {
    int n, temp = 0, AvTAT = 0, AvWT = 0;
    int *tct;
    FCFS *arr;

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

    // Creating arrays
    arr = (FCFS*)malloc(n * sizeof(FCFS));
    tct = (int*)malloc(n * sizeof(int));

    for(int i = 0; i < n; ++i) {
        arr[i].process = i;
        printf("Enter the process %d data\n", arr[i].process);
        printf("Enter CPU Burst: ");
        scanf("%d", &(arr[i].burst));
        printf("Enter the arrival time: ");
        scanf("%d", &(arr[i].arrival));
    }

    // Sorting processes based on arrival time
    sort(arr, n);

    // Calculation of turnaround time, waiting time and completion time
    printf ("Process\t\tBurst Time\tArrival Time\tCompletion Time\tTurn
Around Time\tWaiting Time\n");
    for(int i = 0; i < n; ++i) {
        if(i >= 1 && arr[i].arrival > tct[i-1]) {
            tct[i] = temp + arr[i].burst + (arr[i].arrival - tct[i-1]);
            temp = tct[i];
        }
        else {
            tct[i] = temp + arr[i].burst;
            temp = tct[i];
        }
```

```
        arr[i].turnaround = tct[i] - arr[i].arrival;
        arr[i].waiting = arr[i].turnaround - arr[i].burst;

        AvTAT = AvTAT + arr[i].turnaround;
        AvWT = AvWT + arr[i].waiting;

        printf ("%5d\t%15d\t\t%9d\t%12d\t%12d\t%12d\n",
            arr[i].process, arr[i].burst, arr[i].arrival, tct[i],
arr[i].turnaround, arr[i].waiting
        );
    }

    printf ("Average Turn Around Time: %f ms\nAverage Waiting Time: %f
ms\n", (float)AvTAT / n, (float)AvWT / n);

    free(arr);
    free(tct);
    return 0;
}
```

Output

```
Enter number of processes: 4
Enter the process 0 data
Enter CPU Burst: 10
Enter the arrival time: 0
Enter the process 1 data
Enter CPU Burst: 7
Enter the arrival time: 1
Enter the process 2 data
Enter CPU Burst: 3
Enter the arrival time: 3
Enter the process 3 data
Enter CPU Burst: 5
Enter the arrival time: 4
Process         Burst Time      Arrival Time    Completion Time Turn Around Time        Waiting Time
    0               10              0               10              10              0
    1                7              1               17              16              9
    2                3              3               20              17              14
    3                5              4               25              21              16
Average Turn Around Time: 16.000000 ms
Average Waiting Time: 9.750000 ms
```

Shortest Job First (SJF) Scheduling Algorithm

This technique is associated with the length of the next CPU burst of a process. When the CPU is available, it is assigned to the process that has the smallest next CPU burst. If the next bursts of two processes are the same, FCFS scheduling is used. The SJF algorithm is optimal i.e. it gives the minimum average waiting time for a given set of processes. The real difficulty with this algorithm is knowing the length of the next CPU request.

Source Code

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

// SJF Scheduling

struct sjf {
    int process;
    int burst;
    int arrival;
    int turnaround;
    int waiting;
};

typedef struct sjf SJF;

// Selection sort
void sort(SJF arr[], int n) {
    int min_idx;
    SJF temp;

    for(int i = 0; i < n-1; ++i) {
        min_idx = i;

        for(int j = i+1; j < n; ++j) {
            if(arr[j].arrival < arr[min_idx].arrival) {
                min_idx = j;
            }
        }

        // Swapping
        temp = arr[min_idx];
        arr[min_idx] = arr[i];
        arr[i] = temp;
    }
}

int main() {
    int i, j, n, TCT, count_process = 0, count = 0, minBurst, pos;
    float AvTAT = 0.0, AvWT = 0.0;
    SJF *arr;
```

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

    arr = (SJF*)malloc(n*sizeof(SJF));

    // Entering data of the processes
    printf ("Enter the data of processes\n");
    for (i = 0; i < n; i++)
    {
        arr[i].process = i + 1;
        printf("Enter the burst time of process %d: ", arr[i].process);
        scanf ("%d", &(arr[i].burst));
        printf ("Enter the arrival time of process %d: ",
arr[i].process);
        scanf ("%d", &(arr[i].arrival));
    }

    // SJF Algorithm starts here
    sort (arr, n);

    printf ("PROCESS\tARRIVAL TIME\tBURST TIME\n");
    for(int i = 0; i < n; i++) {
        printf (
            "%3d\t%5d\t\t%5d\n",
            arr[i].process, arr[i].arrival, arr[i].burst
        );
    }

    TCT = arr[0].turnaround = arr[0].burst;
    arr[0].waiting = arr[0].turnaround - arr[0].burst;
    arr[0].arrival = -1;

    sort(arr, n);
    count_process = 1;

    while(count_process < n) {
        minBurst = 999;
        count = 0;
        i = count_process;

        while(TCT >= arr[i].arrival && i < n) {
            ++count;
            ++i;
        }

        for(j = i - count; count != 0 && j < n; ++j, --count) {
            if(arr[j].burst < minBurst) {
                minBurst = arr[j].burst;
                pos = j;
            }
        }

        TCT = TCT + arr[pos].burst;
        arr[pos].turnaround = TCT - arr[pos].arrival;
        arr[pos].waiting = arr[pos].turnaround - arr[pos].burst;
        arr[pos].arrival = -1;
        sort(arr, n);
```

```
        ++count_process;
    }

    printf("Process\tTAT\tWT\n");
    for(i = 0; i < n; ++i) {
        printf(
            "%2d\t%2d\t%2d\n",
            arr[i].process, arr[i].turnaround, arr[i].waiting
        );
    }

    for(i = 0; i < n; ++i) {
        AvTAT = AvTAT + arr[i].turnaround;
        AvWT = AvWT + arr[i].waiting;
    }

    printf(
        "Average TAT: %.2f\nAverage WT: %.2f\n",
        AvTAT / n, AvWT / n
    );

    free(arr);
    return 0;
}
```

Output

```
Enter the number of processes: 4
Enter the data of processes
Enter the burst time of process 1: 4
Enter the arrival time of process 1: 10
Enter the burst time of process 2: 6
Enter the arrival time of process 2: 8
Enter the burst time of process 3: 0
Enter the arrival time of process 3: 7
Enter the burst time of process 4: 8
Enter the arrival time of process 4: 0
PROCESS ARRIVAL TIME    BURST TIME
  4         0               8
  3         7               0
  2         8               6
  1        10               4
Process TAT     WT
 4       8       0
 3       1       1
 2       6       0
 1       8       4
Average TAT: 5.75
Average WT: 1.25
```

Discussion and Conclusion

In this section of the lab, we implemented various CPU scheduling algorithms. We calculated the average turnaround time and average waiting time of the given process using the scheduling algorithm. We observed output for different inputs and understood about FCFS and Shortest Job First Scheduling algorithm.

Thus, some CPU scheduling algorithms were implemented.