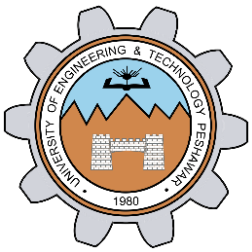
Simulation of Non-preemptive Process Scheduling Algorithms

LAB # 10



Spring 2023

CSE-204L Operating Systems Lab

Submitted by: **Ali Asghar**

Registration No.: **21PWCSE2059**

Class Section: **C**

"On my honor, as student of University of Engineering and Technology, I have neither given nor received unauthorized assistance on this academic work."

Submitted to:

Engr. Madiha Sher

Date:

26th June 2023

Department of Computer Systems Engineering

University of Engineering and Technology, Peshawar

WRITE A C PROGRAM FOR CPU SCHEDULING ALGORITHM FOR FCFS

DESCRIPTION:

The implementation of the FCFS policy is easily managed with a FIFO queue. When a process enters the ready queue, its PCB is linked to the tail of the queue. When the CPU is free it is allocated to the process at that end of the queue. The running process is then removed from the queue.

Data Structures Required:

For all CPU scheduling algorithm implementations, we need to have the following arrays.

A two-dimensional array "process [NUM_PROCESSES][INFO]" of data type float.

This array stores the job numbers, their arrival times, and their burst times, which are read from the user.

And this array also stores the additional data after some calculations. They are...

1. start time of the job,
2. finish time of the job,
3. waiting time of the job,
4. Turn around time of job.s

A row of the process array contains the following:

Process[1,0] = job number,

Process[1,1] = arrival time,

Process[1,2] = burst time,

And after calculations...

Process[1,3] = start time,

Process[1,4] = finish time,

Process[1,5] = waiting time,

Process[1,6] = turn around time,

The remaining columns may be used while implementing the other algorithms.

Variables **avgwt** and **avgtat** of type float indicate average waiting time and average turnaround time respectively, and variable TOT_JOBS for keeping track of the number of jobs.

Functions Needed:

While implementing algorithms one can implement his/her own logic and can write own functions, which is up to the programmers. As far as possible, avoid using global variables.

Though it is so, the suggested functions for implementing these algorithms are

1. for reading the job entries (read_job_entry())

2. FCFS() for fcfs algorithm

3. SJF_non_preempt( )for SJF algorithm,

4. Calculate() for doing various calculations like start times, finish times, waiting times, turn around times of jobs.

5. Sorting() for sorting the job entries by their arrival time (for FCFS), or by their burst time (for SJF), or by their priority (for Priority algo).

6. Print_sorted() for printing inputted data after sorting.

7. Printing(): Print the outputs after all the calculations are over.

PROCEDURE FOR FCFS

Read the number of jobs into variable tot_obs.

Then read the data for jobs as:

Job number, into the process [1, 0],

Job arrival time into the process [1, 1],

Job burst into process [1, 2], through read_job_entry().

Then sort the job entries by their arrival times because the FCFS algorithm works by arrival times.

Print the sorted jobs on the output.

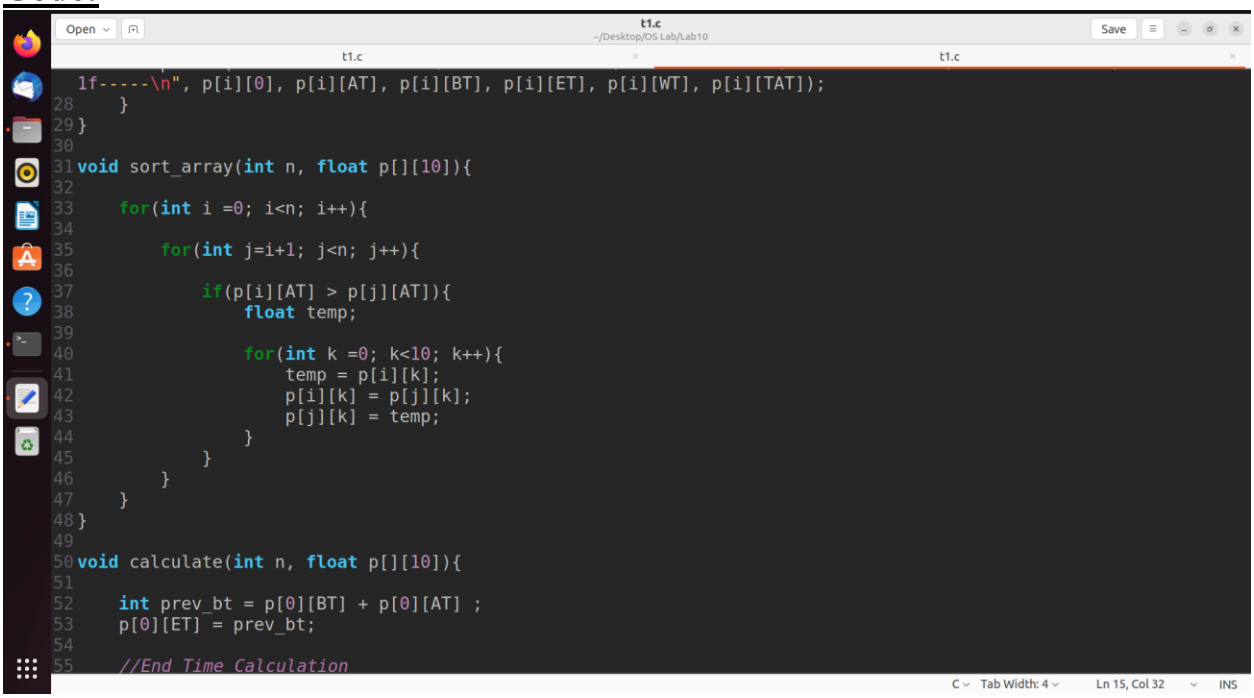
Then go for calculations.

Then Print the results on the output.

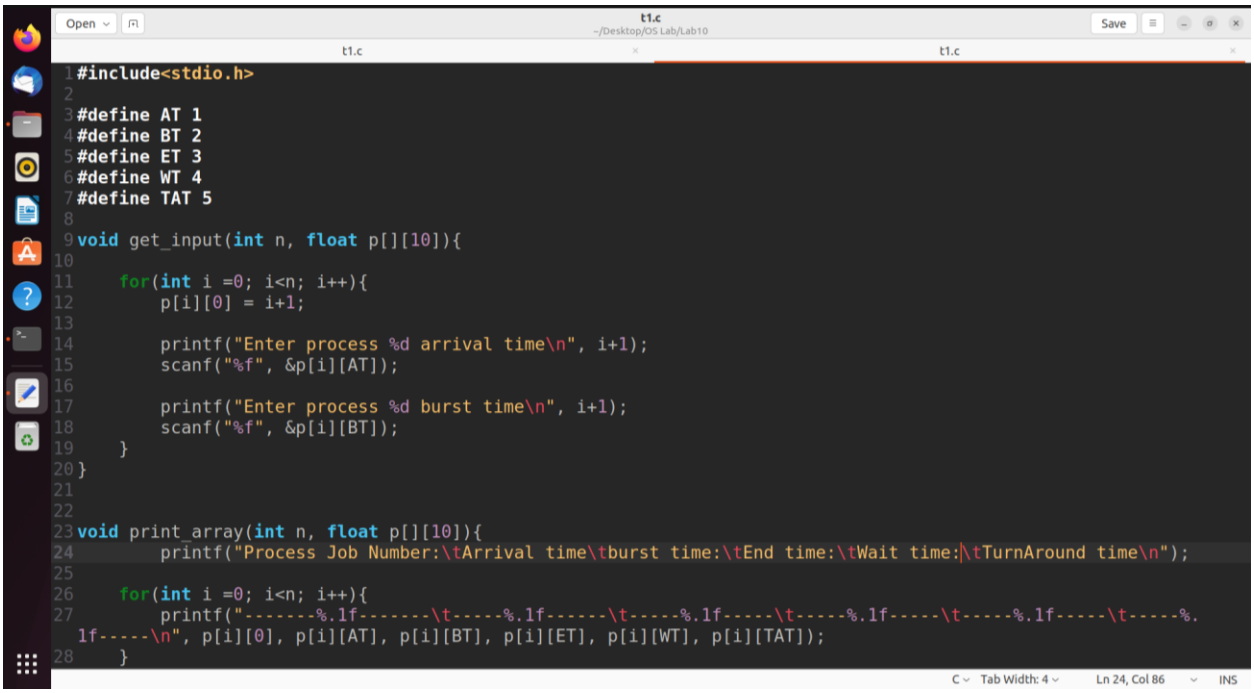
The read function can be called from the main function. Then FCFS function can be as

```
FCFS( ) {
sort(...):
printf("The Scheduling
according to FCFS:");
print_sorted ( . . .);
calculated(... );
 getch( );
printing( };
 getch( };
 }
```

Code:



```
1f-----\n", p[i][0], p[i][AT], p[i][BT], p[i][ET], p[i][WT], p[i][TAT]);
    }
}

void sort_array(int n, float p[][10]){

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

        for(int j=i+1; j<n; j++){

            if(p[i][AT] > p[j][AT]){
                float temp;

                for(int k =0; k<10; k++){
                    temp = p[i][k];
                    p[i][k] = p[j][k];
                    p[j][k] = temp;
                }
            }
        }
    }
}

void calculate(int n, float p[][10]){

    int prev_bt = p[0][BT] + p[0][AT] ;
    p[0][ET] = prev_bt;

    //End Time Calculation
```



```
#include<stdio.h>

#define AT 1
#define BT 2
#define ET 3
#define WT 4
#define TAT 5

void get_input(int n, float p[][10]){

    for(int i =0; i<n; i++){
        p[i][0] = i+1;

        printf("Enter process %d arrival time\n", i+1);
        scanf("%f", &p[i][AT]);

        printf("Enter process %d burst time\n", i+1);
        scanf("%f", &p[i][BT]);
    }
}


void print_array(int n, float p[][10]){
        printf("Process Job Number:\tArrival time\tburst time:\tEnd time:\tWait time:\tTurnAround time\n");

    for(int i =0; i<n; i++){
        printf("-------%.1f-------\t-----%.1f------\t-----%.1f-----\t-----%.1f-----\t-----%.1f-----\t-----%.
1f-----\n", p[i][0], p[i][AT], p[i][BT], p[i][ET], p[i][WT], p[i][TAT]);
    }
```

```c
    for(int i =1; i<n; i++){
        prev_bt += p[i][BT];
        p[i][ET] = prev_bt;
    }

    //Wait Time Calculation
    for(int i =0; i<n; i++){
        p[i][WT] = p[i][ET] - p[i][BT] - p[i][AT] ;
    }

    //TurnAround Time Calculation
    for(int i =0; i<n; i++){
        p[i][TAT] = p[i][WT] + p[i][BT];
    }
}
int main(){

    int n;
    printf("Enter the total number of jobs\n");
    scanf("%d", &n);
    float processes[n][10];

    get_input(n, processes);
    print_array(n, processes);
    sort_array(n, processes);
    calculate(n, processes);
    print_array(n, processes);
    return 0;
}
```

Output:

```
ali@Ubuntu22:~/Desktop/OS Lab/Lab10$ gcc t1.c -o t1.o
ali@Ubuntu22:~/Desktop/OS Lab/Lab10$ ./t1.o
Enter the total number of jobs
4
Enter process 1 arrival time
2
Enter process 1 burst time
3
Enter process 2 arrival time
4
Enter process 2 burst time
5
Enter process 3 arrival time
7
Enter process 3 burst time
8
Enter process 4 arrival time
5
Enter process 4 burst time
9
Process Job Number:     Arrival time    burst time:     End time:       Wait time:      TurnAround time
-------1.0-------       -----2.0------  -----3.0-----   -----0.0-----   ------0.0-----  -----0.0-----
-------2.0-------       -----4.0------  -----5.0-----   -----0.0-----   ------0.0-----  -----0.0-----
-------3.0-------       -----7.0------  -----8.0-----   -----0.0-----   -----0.0-----   -----0.0-----
-------4.0-------       -----5.0------  -----9.0-----   -----0.0-----   ------0.0-----  -----0.0-----
Process Job Number:     Arrival time    burst time:     End time:       Wait time:      TurnAround time
-------1.0-------       -----2.0------  -----3.0-----   -----5.0-----   -----0.0-----   -----3.0-----
-------2.0-------       -----4.0------  -----5.0-----   -----10.0-----  -----1.0-----   -----6.0-----
-------4.0-------       -----5.0------  -----9.0-----   -----19.0-----  -----5.0-----   -----14.0-----
-------3.0-------       -----7.0------  -----8.0-----   -----27.0-----  -----12.0-----  -----20.0-----
ali@Ubuntu22:~/Desktop/OS Lab/Lab10$
```

WRITE A C PROGRAM FOR CPU SCHEDULING ALGORITHM FOR SJF

Code:

Open ▾ t2.c ~/Desktop/OS Lab/Lab10

t1.c × t2.c

```c
#include<stdio.h>

#define AT 1
#define BT 2
#define ET 3
#define WT 4
#define TAT 5

void get_input(int n, float p[][10]){

    for(int i =0; i<n; i++){
        p[i][0] = i+1;

        printf("Enter process %d arrival time\n", i+1);
        scanf("%f", &p[i][AT]);

        printf("Enter process %d burst time\n", i+1);
        scanf("%f", &p[i][BT]);
    }
}


void print_array(int n, float p[][10]){
        printf("Process Job Number:\tArrival time\tburst time:\tEnd time:\tWait time:\
```

Open ▾ t2.c ~/Desktop/OS Lab/Lab10

t1.c ×

```c
void print_array(int n, float p[][10]){
        printf("Process Job Number:\tArrival time\tburst time:\tEnd time:\tWait

    for(int i =0; i<n; i++){
        printf("-------%.1f-------\t-----%.1f------\t-----%.1f-----\t-----%.1f--
 1f-----\n", p[i][0], p[i][AT], p[i][BT], p[i][ET], p[i][WT], p[i][TAT]);
    }
}

void sort_array(int n, float p[][10]){

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

        for(int j=i+1; j<n; j++){

            if(p[i][BT] > p[j][BT]){
                float temp;

                for(int k =0; k<10; k++){
                    temp = p[i][k];
                    p[i][k] = p[j][k];
                    p[j][k] = temp;
                }
```

```
45            }
46        }
47    }
48}
49
50
51void init_entries(int n, float p[][10]){
52
53    for(int i =0; i<n; i++){
54        p[i][ET] = 0;
55        p[i][WT] = 0;
56        p[i][TAT] = 0;
57    }
58}
59void sort_arrayAT(int n, float p[][10]){
60
61    for(int i =0; i<n; i++){
62
63        for(int j=i+1; j<n; j++){
64
65            if(p[i][AT] > p[j][AT]){
66                float temp;
67
```

t1.c

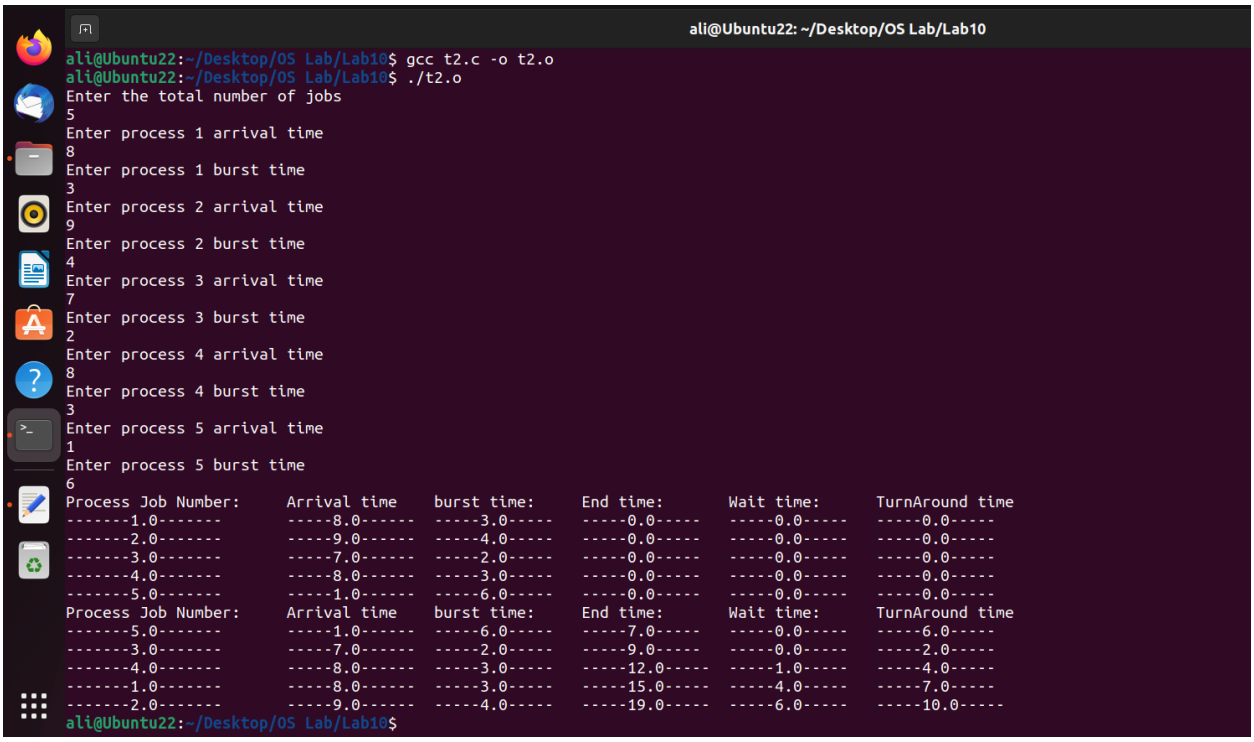
```
67
68                for(int k =0; k<10; k++){
69                    temp = p[i][k];
70                    p[i][k] = p[j][k];
71                    p[j][k] = temp;
72                }
73            }
74        }
75    }
76}
77void calculate(int n, float p[][10]){
78    int avgwt = 0;
79    int avgtat = 0;
80
81    int prev_bt = p[0][BT] + p[0][AT] ;
82    p[0][ET] = prev_bt;
83
84    //End Time Calculation
85    for(int i =1; i<n; i++){
86        prev_bt += p[i][BT];
87        p[i][ET] = prev_bt;
88    }
89
```

t1.c

```c
    //Wait Time and Turn Around Time Calculation
    for(int i = 0; i<n; i++){
        p[i][WT] = p[i][ET] - p[i][BT] - p[i][AT] ;
        p[i][TAT] = p[i][WT] + p[i][BT];

        avgwt += p[i][WT];
        avgtat  += p[i][TAT];
    }
        avgwt /= n;
        avgtat  /= n;

}
int main(){
    int n;
    printf("Enter the total number of jobs\n");
    scanf("%d", &n);
    float processes[n][10];

    get_input(n, processes);
    init_entries(n, processes);
    print_array(n, processes);
    sort_array(n, processes);
    sort_arrayAT(n, processes);
```

```c
    get_input(n, processes);
    init_entries(n, processes);
    print_array(n, processes);
    sort_array(n, processes);
    sort_arrayAT(n, processes);
    calculate(n, processes);
    print_array(n, processes);
    return 0;
}
```

Output:



DESCRIPTION:

The SJF algorithm also can be implemented as FCFS, but on this, the jobs are sorted by their burst time. A job with the shortest burst time will be scheduled first i.e. sorting should be done from short to large time of burst time. Take arrival time also into consideration. This is the SJF Non-pre-emptive algorithm. While implementing Priority scheduling we have read the priority also and they should be sorted by highest priority to lowest priority.

It maintains the Ready queue in order of increasing job lengths. When a job comes in, insert it in the ready queue based on its length. When the current process is done, pick the one at the head of the queue and run it.