**Lab No. 04**

**CPU Scheduling**

| **Lab Objectives** | Following are the lab objectives:   1. Implement First Come First Serve (FCFS) Algorithm 2. Implement Shortest Job First (SJF) Algorithm 3. Implement Shortest Remaining Time First (SRTF) Algorithm |
| --- | --- |

| **Roll No.** |  | **Student Name** | |  | | |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Marks** |  | **Obtained Marks** |  | **Comments** | |
| **Task 1** | **20** |  |  | |
| **Task 2** | **20** |  |  | |
| **Total Marks** | **40** |  |  | |
|  |  | | | | |  |
| **Lab Instructor** | | | | | |
|  | | | | | |

## **Lab Objectives and CLOs Mapping**

| **Lab Objectives** | **CLOs** | | |
| --- | --- | --- | --- |
| **a** | **b** | **c** |
| **1** |  |  |  |
| **2** |  |  |  |
| **3** |  |  |  |

## **Instructions**

* This is individual Lab work/task.
* Complete this lab work within lab timing.
* Discussion with peers is not allowed.
* You can consult any book, notes & Internet.
* Copy paste from the Internet will give you **negative marks**.
* Lab work is divided into small tasks, completing all tasks sequentially.
* Show the solution of each lab task to your Lab Instructor.
* In-Lab Exercises/Tasks
* **Write your code at provided space after each question**
* **You need to upload code for all tasks at Google Class.**

**CPU Scheduling**

CPU scheduling is the basis of multiprogrammed operating systems. By switching the CPU among processes, the operating system can make the computer more productive. CPU scheduling deals with the problem of deciding which of the processes in the ready queue is to be allocated to the CPU. There are many different CPU-scheduling algorithms, like;

1. First Come First Serve (FCFS)
2. Shortest Job First (SJF)
3. Shortest Remaining Time First (SRTF)
4. Priority Scheduling
5. Round Robin
6. Multilevel Queue
7. Multilevel Feedback Queue

For better simulation of above mentioned algorithms we need to have two prerequisites;

1. Struct to Represent Process
2. Queue to represent Ready Queue

**Struct to Represent Process**

A process can be represented by a struct which contains basic information related to process, like, process ID, arrival time, burst time, wait time, turnaround time, etc. Following is the sample structure. You can add more information if you require.

**struct process**

**{**

**int pid;**

**int arrivalTime;**

**int burstTime;**

**int waitTime;**

**int turnaroundTime;**

**};**

**Queue to Represent Ready Queue**

Ready queue is the place where all processes reside and wait for their turn to acquire the CPU. For this purpose we need to implement a queue data structure of type **process**. You can implement your own class of queue (implemented in data structures course) or a built-in ADT of Standard Template Library (STL). Please explore the STL queue with the following resources.

* <http://www.cplusplus.com/reference/queue/queue/>
* <https://www.geeksforgeeks.org/queue-cpp-stl/>

**Lab Tasks**

**Task 1 (20 marks)**

Write C++ program to simulate the First **Come First Serve (FCFS)** algorithm to manage processes. For process representation **struct process** must be used. All processes should be stored to the ready queue with proper information (burst time, arrival time, etc.). Later on display the **process ID, burst time, wait time, turnaround time, average wait time and average turnaround time**.

An abstract **algorithm** is given below;

1- Input the processes along with their burst time (bt).

2- Find waiting time (wt) for all processes.

3- As first process that comes need not to wait so

waiting time for process 1 will be 0 i.e. wt[first process] = 0.

4- Find waiting time for all other processes i.e. for process i ->

wt[i] = bt[i-1] + wt[i-1] .

5- Find turnaround time = waiting\_time + burst\_time

for all processes.

6- Find average waiting time =

total\_waiting\_time / no\_of\_processes.

7- Similarly, find average turnaround time =

total\_turn\_around\_time / no\_of\_processes.

**CODE:**

#include <iostream>

#include <vector>

#include <queue>

using namespace std;

struct process {

int pid;

int arrival\_time;

int burst\_time;

int wait\_time;

int turnaround\_time;

};

int main() {

queue<process> ready\_queue;

vector<process> processes;

int n;

cout << "Enter the number of processes: ";

cin >> n;

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

process p;

cout << "Enter arrival time for process " << i + 1 << ": ";

cin >> p.arrival\_time;

cout << "Enter burst time for process " << i + 1 << ": ";

cin >> p.burst\_time;

p.pid = i + 1;

processes.push\_back(p);

}

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

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

if (processes[i].arrival\_time > processes[j].arrival\_time) {

swap(processes[i], processes[j]);

}

}

}

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

ready\_queue.push(processes[i]);

}

int current\_time = 0;

float total\_wait = 0;

float total\_turnaround = 0;

cout << "\nProcess ID\tBurst Time\tWait Time\tTurnaround Time\n";

while (!ready\_queue.empty()) {

process current = ready\_queue.front();

ready\_queue.pop();

if (current\_time < current.arrival\_time) {

current\_time = current.arrival\_time;

}

current.wait\_time = current\_time - current.arrival\_time;

current.turnaround\_time = current.wait\_time + current.burst\_time;

total\_wait += current.wait\_time;

total\_turnaround += current.turnaround\_time;

cout << current.pid << "\t\t" << current.burst\_time << "\t\t" << current.wait\_time << "\t\t" << current.turnaround\_time << endl;

current\_time += current.burst\_time;

}

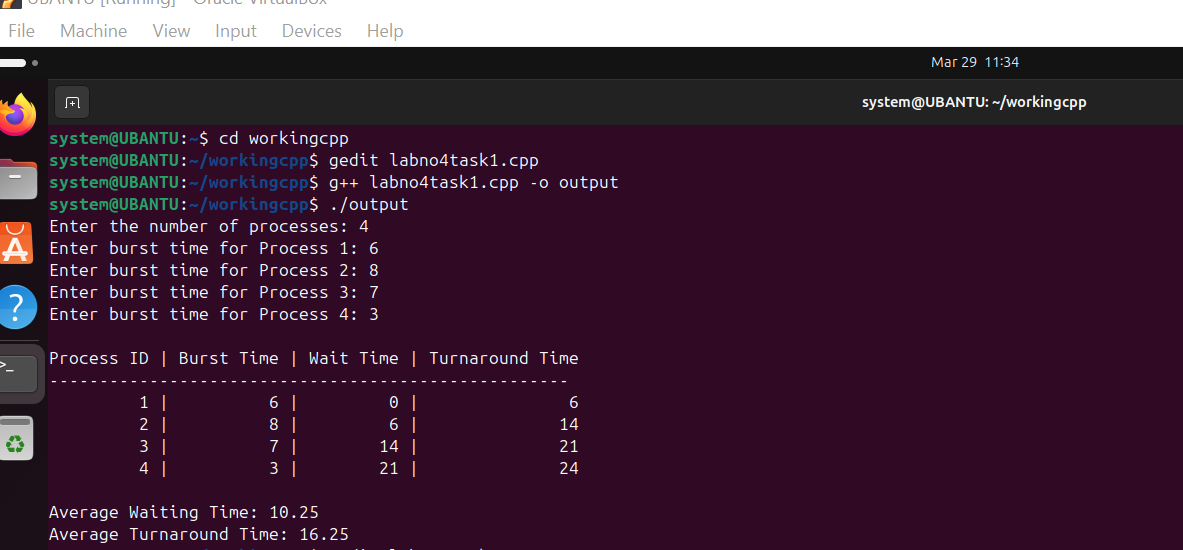
cout << "\nAverage Wait Time: " << total\_wait / n << endl;

cout << "Average Turnaround Time: " << total\_turnaround / n << endl;

return 0;

}

**OUTPUT:**

****

**Task 2 (20 marks)**

Write C++ program to simulate the Shortest **Job First (SJF)** algorithm to manage processes. For process representation **struct process** must be used. All processes should be stored to the ready queue with proper information (burst time, arrival time, etc.). Later on display the **process ID, burst time, wait time, turnaround time, average wait time and average turnaround time**.

An abstract **algorithm** is given below;

1. Sort all the processes according to the arrival time.
2. Then select that process which has minimum arrival time and minimum Burst time.
3. After completion of process make a pool of processes which after till the completion of previous process and select that process among the pool which is having minimum Burst time.

**CODE:**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

struct process {

int pid;

int arrival\_time;

int burst\_time;

int wait\_time;

int turnaround\_time;

};

bool compareArrival(process a, process b) {

return a.arrival\_time < b.arrival\_time;

}

bool compareBurst(process a, process b) {

return a.burst\_time < b.burst\_time;

}

int main() {

vector<process> processes;

int n;

cout << "Enter the number of processes: ";

cin >> n;

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

process p;

cout << "Enter arrival time for process " << i + 1 << ": ";

cin >> p.arrival\_time;

cout << "Enter burst time for process " << i + 1 << ": ";

cin >> p.burst\_time;

p.pid = i + 1;

processes.push\_back(p);

}

sort(processes.begin(), processes.end(), compareArrival);

vector<process> ready\_queue;

vector<process> completed;

int current\_time = 0;

int index = 0;

float total\_wait = 0;

float total\_turnaround = 0;

while (completed.size() < n) {

while (index < n && processes[index].arrival\_time <= current\_time) {

ready\_queue.push\_back(processes[index]);

index++;

}

if (!ready\_queue.empty()) {

sort(ready\_queue.begin(), ready\_queue.end(), compareBurst);

process current = ready\_queue[0];

ready\_queue.erase(ready\_queue.begin());

current.wait\_time = current\_time - current.arrival\_time;

current.turnaround\_time = current.wait\_time + current.burst\_time;

total\_wait += current.wait\_time;

total\_turnaround += current.turnaround\_time;

completed.push\_back(current);

current\_time += current.burst\_time;

} else {

current\_time = processes[index].arrival\_time;

}

}

cout << "\nProcess ID\tBurst Time\tWait Time\tTurnaround Time\n";

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

cout << completed[i].pid << "\t\t" << completed[i].burst\_time << "\t\t" << completed[i].wait\_time << "\t\t" << completed[i].turnaround\_time << endl;

}

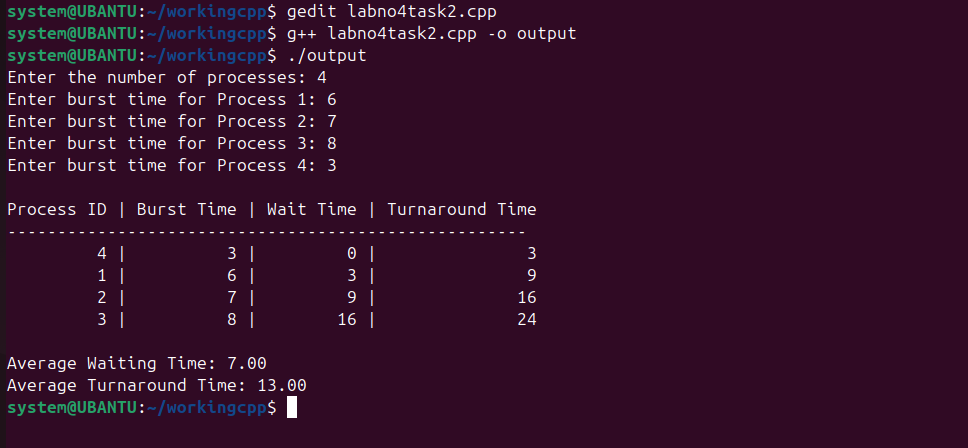
cout << "\nAverage Wait Time: " << total\_wait / n << endl;

cout << "Average Turnaround Time: " << total\_turnaround / n << endl;

return 0;

}

**OUTPUT:**

****

**Extra Tasks**

**Task**

Write C++ program to simulate **Shortest Remaining Time First (SRTF)** algorithm to manage processes. For process representation **struct process** must be used. All processes should be stored to the ready queue with proper information (burst time, arrival time, etc.). Later on display the **process ID, burst time, wait time, turnaround time, average wait time and average turnaround time**.