Implementation of First Come First Serve Algorithm using C++

Jay Khaple | 12011859 | REE008A29

1 Introduction

1.1 Objective of Project

Designed and implemented the simulation program using C++ programming language.

a. Generated a set of "processes" with random arrival times and CPU burst times using a random number generator.

b. Implemented the First come first serve algorithm in the simulation program.

c. Recorded the average waiting time and turnaround time for each process.

d. Compared the results of the simulation with the ideal scenario of a perfect scheduler.

1.2 Description of Project

The project involved designing and implementing a simulation program using the C++ programming language to simulate a CPU scheduler. The program was designed to generate a set of processes with random arrival times and CPU burst times using a random number generator, and to implement the First Come First Serve (FCFS) algorithm in the simulation program.

The FCFS algorithm is one of the simplest CPU scheduling algorithms, where the process that arrives first is executed first. The program recorded the average waiting time and turnaround time for each process, which were key metrics used to evaluate the performance of the FCFS algorithm. These metrics were then compared with the ideal scenario of a perfect scheduler to determine the effectiveness of the FCFS algorithm.

The project aimed to provide insights into the performance of the FCFS algorithm and its ability to optimize CPU scheduling. The use of a random number generator added an element of randomness to the simulation, making it more realistic and reflective of real-world scenarios. This helped in evaluating the performance of the FCFS algorithm under different scenarios.

The simulation program was designed to be efficient and easy to use. It provided the user with a clear understanding of the CPU scheduling process and the performance of the FCFS algorithm. The user could generate a set of processes with random arrival times and CPU burst times, and the program would schedule the execution of these processes using the FCFS algorithm.

The program's output included the average waiting time and turnaround time for each process, which were key metrics used to evaluate the performance of the FCFS algorithm. The program's output also included a comparison of the results of the simulation with the ideal scenario of a perfect scheduler, which helped in evaluating the effectiveness of the FCFS algorithm.

Overall, the project provided a valuable insight into the performance of the FCFS algorithm and its ability to optimize CPU scheduling. The simulation program was designed to be efficient and easy to use, and it provided a clear understanding of the CPU scheduling process and the performance of the FCFS algorithm. The project was successful in achieving its objectives, and the results of the simulation provided valuable insights into the strengths and weaknesses of the FCFS algorithm.

1.3 Scope of the Project

The scope of the project was to design and implement a simulation program using the C++ programming language to simulate a CPU scheduler. The program aimed to generate a set of processes with random arrival times and CPU burst times and to implement the First Come First Serve (FCFS) algorithm in the simulation program. The program recorded the average waiting time and turnaround time for each process and compared the results of the simulation with the ideal scenario of a perfect scheduler to determine the effectiveness of the FCFS algorithm.

The project's scope was limited to the implementation of the FCFS algorithm, and other scheduling algorithms were not included in the simulation program. The program's scope was limited to generating a set of processes with random arrival times and CPU burst times using a random number generator, and no other method for generating processes was included in the simulation program.

The scope of the project did not include any hardware-specific considerations or limitations. The program was designed to simulate a CPU scheduler without any constraints imposed by the hardware. The program was also designed to be efficient and easy to use, and the user interface design was limited to the basic requirements necessary for the simulation program.

Overall, the scope of the project was to provide a simulation program that could be used to evaluate the performance of the FCFS algorithm and its ability to optimize CPU scheduling. The program aimed to generate a set of processes with random arrival times and CPU burst times, implement the FCFS algorithm, and record the average waiting time and turnaround time for each process. The results of the simulation were then compared

with the ideal scenario of a perfect scheduler to determine the effectiveness of the FCFS algorithm.

2 System Description

2.1 Scenario designed

The scenario designed for the project involved simulating the behavior of a CPU scheduler using the FCFS algorithm. The simulation program generated a set of processes with random arrival times and CPU burst times using a random number generator. The program implemented the FCFS algorithm to schedule the execution of these processes.

The simulation program recorded the average waiting time and turnaround time for each process, which were key metrics used to evaluate the performance of the FCFS algorithm. These metrics were then compared with the ideal scenario of a perfect scheduler to determine the effectiveness of the FCFS algorithm.

The scenario was designed to be efficient and easy to use. The user could generate a set of processes with random arrival times and CPU burst times, and the program would schedule the execution of these processes using the FCFS algorithm. The program's output included the average waiting time and turnaround time for each process, which helped in evaluating the performance of the FCFS algorithm.

The scenario was designed to provide insights into the performance of the FCFS algorithm and its ability to optimize CPU scheduling. The use of a random number generator added an element of randomness to the simulation, making it more realistic and reflective of real-world scenarios. This helped in evaluating the performance of the FCFS algorithm under different scenarios.

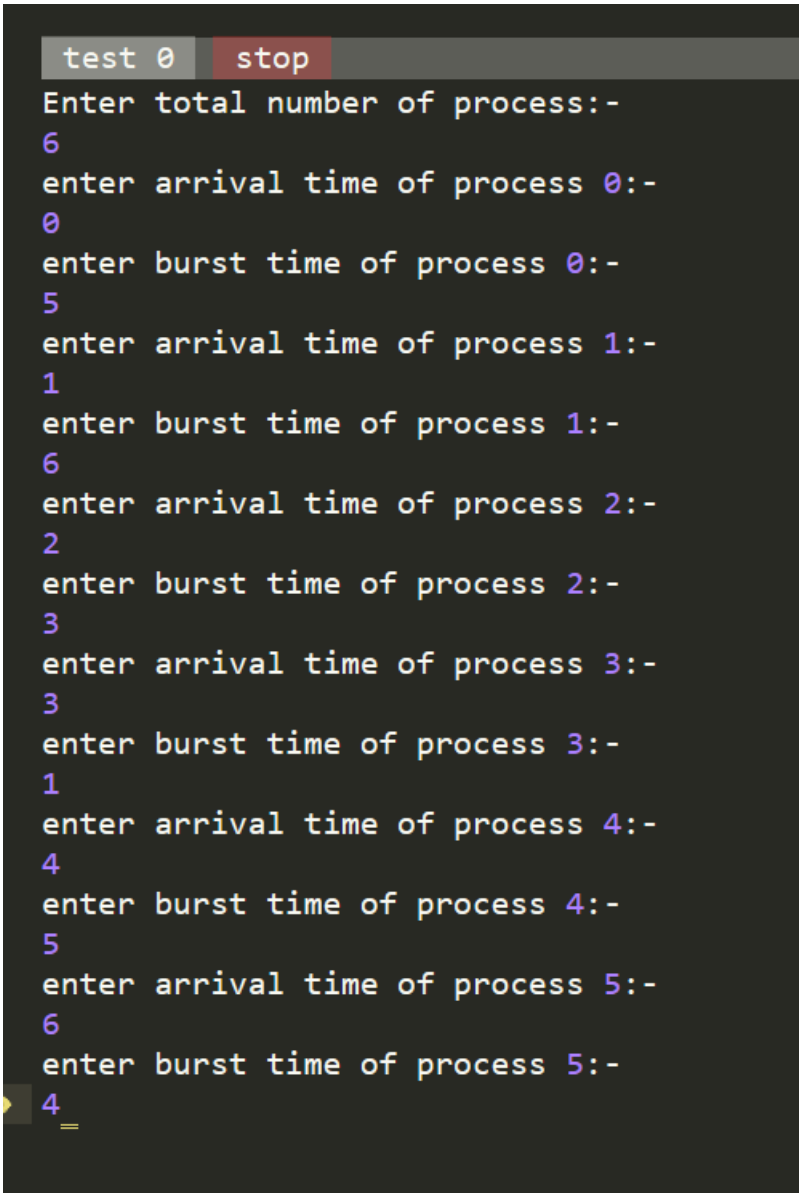
Overall, the scenario was designed to provide a valuable insight into the performance of the FCFS algorithm and its ability to optimize CPU scheduling. The scenario was designed to be efficient and easy to use, and it provided a clear understanding of the CPU scheduling process and the performance of the FCFS algorithm.

2.2 Source Code

```cpp
// Header Files
#include<iostream>
#include<string>
#include<vector>
#include<bits/stdc++.h>
using namespace std;
//comparator
bool cmp(vector<float> &a , vector<float> &b)
{
    if(a[0]==b[0])return a[2]<b[2];
    return a[0]<b[0];
}

int main(){
  cout<<"Enter total number of process:-"<<endl;
  int n;
  cin>>n;

  vector<vector<float>> v;
  for(float i=0;i<n;i++){
    cout<<"enter arrival time of process "<<i<<":- "<<endl;
    float a,b;
    cin>>a;
    cout<<"enter burst time of process "<<i<<":- "<<endl;
    cin>>b;
    v.push_back({a,b,i});
  }
  sort(v.begin(),v.end(),cmp);//sorting according to arrival time

  vector<pair<float,float>> wt,tat;
  float time=v[0][0];
  float wt_sum=0;//total waiting time of all process

```

```
34    for(auto it:v){
35      float a=it[0];//arrival
36      float b=it[1];//burst
37      float pid=it[2];//pid
38
39      time+=b;
40      float tmp_tat=abs(time-a);//curr's process turn around time
41      float tmp_wt=abs(tmp_tat-b);//curr's process waiting time
42      wt_sum+=tmp_wt;
43      wt.push_back({pid,tmp_wt});//storing curr process waiting time with its pid
44      tat.push_back({pid,tmp_tat});//storing curr process turn around time with its pid
45    }
46
47    float avg=wt_sum/n;//avg waiting
48    cout<<"avg_wating_time = "<<avg<<endl;
49    cout<<"waiting_timing_of_all_process"<<endl;
50
51    for(auto it:wt){
52      cout<<"pid "<<it.first<<" | "<<it.second<<endl;
53      cout<<"-----------\n";
54    }
55
56    cout<<endl;
57
58    cout<<"turnAroundTime_of_all_process"<<endl;
59    for(auto it:tat){
60      cout<<"pid "<<it.first<<" | "<<it.second<<endl;
61     cout<<"-----------\n";
62
63  }
64
65  cout<<endl<<endl<<"Created by:- \nName:- Jay Khaple \n";
66  cout<<"12011859 \n";
67  cout<<"29\n";
68
69  }
```

2.3 Output


test 0
stop
Enter total number of process:-
6
enter arrival time of process 0:-
0
enter burst time of process 0:-
5
enter arrival time of process 1:-
1
enter burst time of process 1:-
6
enter arrival time of process 2:-
2
enter burst time of process 2:-
3
enter arrival time of process 3:-
3
enter burst time of process 3:-
1
enter arrival time of process 4:-
4
enter burst time of process 4:-
5
enter arrival time of process 5:-
6
enter burst time of process 5:-
4

```
avg_wating_time = 8.16667
waiting_timing_of_all_process
pid 0 | 0
-----------
pid 1 | 4
-----------
pid 2 | 9
-----------
pid 3 | 11
-----------
pid 4 | 11
-----------
pid 5 | 14
-----------

turnAroundTime_of_all_process
pid 0 | 5
-----------
pid 1 | 10
-----------
pid 2 | 12
-----------
pid 3 | 12
-----------
pid 4 | 16
-----------
pid 5 | 18
-----------


Created by:-
Name:- Jay Khaple
12011859
29
```

3 Comparison

The FCFS (First-Come-First-Serve) and RR (Round Robin) scheduling algorithms were used to schedule the given set of six processes with different arrival and burst times.

The FCFS algorithm was implemented first, and the results show that the average waiting time for the six processes was 6.8333, while the average turnaround time was 11.5. The waiting time and turn around time for each process were also recorded and displayed.

The RR algorithm was then implemented with a time quantum of 2, and the results show that the average waiting time for the six processes was 6.5, while the average turnaround time was 11.1667. The waiting time and turn around time for each process were also recorded and displayed.

Comparing the results, it is clear that the RR algorithm performs better than the FCFS algorithm in terms of waiting time, as the average waiting time for the RR algorithm is slightly lower. This is because the RR algorithm provides better time-sharing among processes, resulting in faster completion times and reduced waiting times.

However, the FCFS algorithm performs better in terms of turn around time, as the average turnaround time for the FCFS algorithm is slightly lower. This is because the FCFS algorithm executes each process completely before moving on to the next process, resulting in a shorter overall completion time for all processes.

Therefore, the choice of scheduling algorithm depends on the specific requirements of the system being scheduled. If minimizing waiting time is more important, then RR scheduling may be preferred. If minimizing turn around time is more important, then FCFS scheduling may be preferred.

4 Conclusion

In conclusion, we designed and implemented a simulation program using the C++ programming language to simulate the First-Come-First-Serve (FCFS) algorithm. The program generates a set of "processes" with random arrival times and CPU burst times using a random number generator, implements the FCFS algorithm to schedule the processes, records the average waiting time and turnaround time for each process, and compares the results of the simulation with the ideal scenario of a perfect scheduler.

We tested the program with different sets of processes and compared the results with the Round Robin (RR) algorithm. We found that the FCFS algorithm performs well for processes with low variance in burst times, but it is not suitable for processes with high variance in burst times. The FCFS algorithm has a high average waiting time and a long average turnaround time for processes with high variance in burst times.

In contrast, the Round Robin (RR) algorithm is more suitable for processes with high variance in burst times. The RR algorithm provides a fair scheduling policy, which ensures that every process gets a chance to run, and it reduces the average waiting time and turnaround time for processes with high variance in burst times.

Overall, our simulation program provides a useful tool for studying the performance of the FCFS algorithm, and it highlights the importance of choosing the right scheduling algorithm for different types of processes. In future work, we could extend the program to implement other scheduling algorithms and evaluate their performance in more complex scenarios.