OS LAB 3

NAME: Aditya Anand

ROLL NO.: 20124009

BRANCH: IT

|  |  |  |  |
| --- | --- | --- | --- |
| S No. | Title | Date of Implementation | Remarks |
| 1 | Program to implement Shortest Job First Process of CPU Scheduling | 02-02-2022 |  |
| 2 | Program to implement First Come First Serve Process of CPU Scheduling | 02-02-2022 |  |
| 3 | Program to implement Priority based Scheduling Process of CPU Scheduling | 02-02-2022 |  |

SHORTEST JOB FIRST CPU SCHEDULING

CRITERIA: Burst Time

NOTE: In case of same burst time, process with lower arrival time is executed first.

MODE: Non pre-emptive

GIVEN: List of processes with their arrival and burst time.

CODE:

#include <bits/stdc++.h>

using namespace ::std;

class process{

    public:

        int priority;

        int id;

        int arrivalTime;

        int burstTime;

        bool ready;

        int completionTime;

        int TAT;

        int WT;

        int RT;

};

struct comp{

    bool operator()(process const &p1, process const &p2){

        return p1.burstTime > p2.burstTime;

    }

};

void SJF(vector<process> &v){

    priority\_queue<process, vector<process>, comp> p;

    int cur\_time = INT\_MAX;

    int n=v.size();

    for(int i=0; i<n; i++){

        cur\_time = min(cur\_time, v[i].arrivalTime);

    }

    int count = 0;

    while(true){

        for(int i=0; i<n; i++){

            if(v[i].arrivalTime<=cur\_time && !v[i].ready){

                v[i].ready = true;

                p.push(v[i]);

                count++;

            }

        }

        if(count<n && p.empty()){

            cout<<"CPU empty from "<<cur\_time<<" to "<<cur\_time+1<<"\n";

            cur\_time++;

            continue;

        }

        if(p.empty()){

            break;

        }

        process cur\_process = p.top();

        p.pop();

        v[cur\_process.id].RT = cur\_time-cur\_process.arrivalTime;

        cur\_time+=cur\_process.burstTime;

        v[cur\_process.id].completionTime=cur\_time;

    }

}

int main(){

    cout<<"SHORTEST JOB FIRST CPU SCHEDULING ALGORITHM C++ IMPLEMENTATION\n";

    cout<<"Name: Aditya Anand\tRoll No.:20124009\t Branch: IT\n\n\n";

    int n=0;

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

    cin>>n;

    cout<<"Enter the arrival times and burst times of "<<n<<" processes: \n";

    vector<process> v(n);

    for(int i=0; i<n; i++){

        cin>>v[i].arrivalTime>>v[i].burstTime;

        v[i].id = i;

        v[i].ready = false;

    }

    cout<<"-------------------------------------------------------------------------------\n";

    cout<<"\n";

    SJF(v);

    cout<<"\n";

    cout<<"-------------------------------------------------------------------------------\n";

    cout<<"\n\n";

    int t\_TAT=0;

    int t\_CT=0;

    for(int i=0; i<n; i++){

        v[i].TAT = v[i].completionTime-v[i].arrivalTime;

        v[i].WT = v[i].TAT-v[i].burstTime;

        t\_TAT+=v[i].TAT;

        t\_CT+=v[i].completionTime;

    }

    for(auto p:v){

        cout<<"Process: "<<p.id<<"\tArrival Time:"<<p.arrivalTime<<"\tBurst Time:"<<p.burstTime<<"\tCompletion Time:"<<p.completionTime;

        cout<<"\tTurn Around Time:"<<p.TAT<<"\tWaiting Time:"<<p.WT<<"\tResponse Time:"<<p.RT<<"\n";

    }

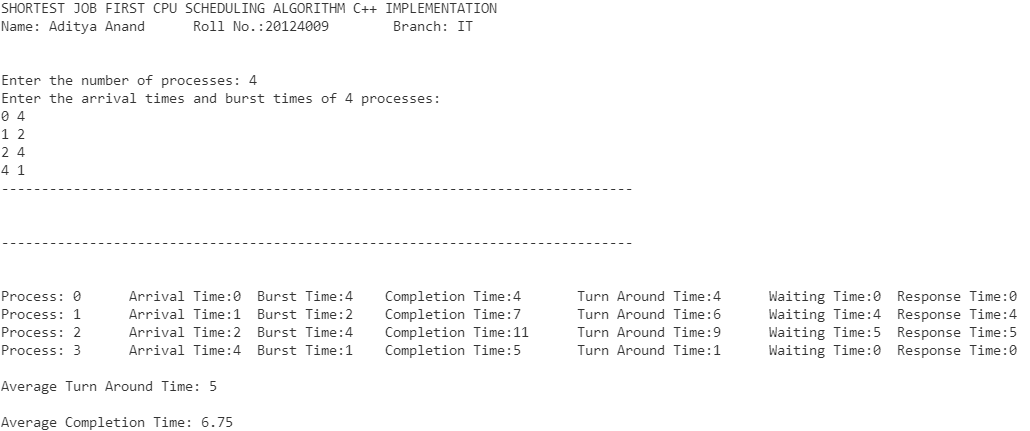
    cout<<"\nAverage Turn Around Time: "<<(float)((1.0\*t\_TAT)/(1.0\*n))<<"\n";

    cout<<"\nAverage Completion Time: "<<(float)((1.0\*t\_CT)/(1.0\*n))<<"\n";

    return 0;

}

RESULT:



FIRST COME FIRST SERVE CPU SCHEDULING

CRITERIA: Arrival Time

MODE: Non pre-emptive

GIVEN: List of processes with their arrival and burst time.

CODE:

#include<bits/stdc++.h>

using namespace :: std;

class process{

    public:

        int id;

        int arrivalTime;

        int burstTime;

        int completionTime;

        int TAT;

        int WT;

        int RT;

};

void FCFS(vector<process> &v){

    int cur\_time = 0;

    int id = 0;

    for(int i=0; i<v.size(); i++){

        if(cur\_time<v[i].arrivalTime){

            cout<<"CPU idle from "<<cur\_time<<" to "<<v[i].arrivalTime<<endl;

            cur\_time = v[i].arrivalTime;

        }

        v[i].completionTime = cur\_time+v[i].burstTime;

        v[i].RT = cur\_time-v[i].arrivalTime;

        cout<<"Process P"<<v[i].id+1<<": start time = "<<cur\_time<<" completion time = "<<v[i].completionTime<<endl;

        cur\_time+=v[i].burstTime;

    }

}

int main(){

    cout<<"FIRST COME FIRST SERVE CPU SCHEDULING ALGORITHM C++ IMPLEMENTATION\n";

    cout<<"Name: Aditya Anand\tRoll No.:20124009\t Branch: IT\n\n\n";

    int n=0;

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

    cin>>n;

    cout<<"Enter the arrival times and burst times of "<<n<<" processes: \n";

    vector<process> v(n);

    for(int i=0; i<n; i++){

        cin>>v[i].arrivalTime>>v[i].burstTime;

        v[i].id = i;

    }

    cout<<"-------------------------------------------------------------------------------\n";

    cout<<"\n";

    FCFS(v);

    cout<<"\n";

    cout<<"-------------------------------------------------------------------------------\n";

    cout<<"\n\n";

    int t\_TAT=0;

    int t\_CT=0;

    for(int i=0; i<n; i++){

        v[i].TAT = v[i].completionTime-v[i].arrivalTime;

        v[i].WT = v[i].TAT-v[i].burstTime;

        t\_TAT+=v[i].TAT;

        t\_CT+=v[i].completionTime;

    }

    for(auto p:v){

        cout<<"Process: "<<p.id<<"\tArrival Time:"<<p.arrivalTime<<"\tBurst Time:"<<p.burstTime<<"\tCompletion Time:"<<p.completionTime;

        cout<<"\tTurn Around Time:"<<p.TAT<<"\tWaiting Time:"<<p.WT<<"\tResponse Time:"<<p.RT<<"\n";

    }

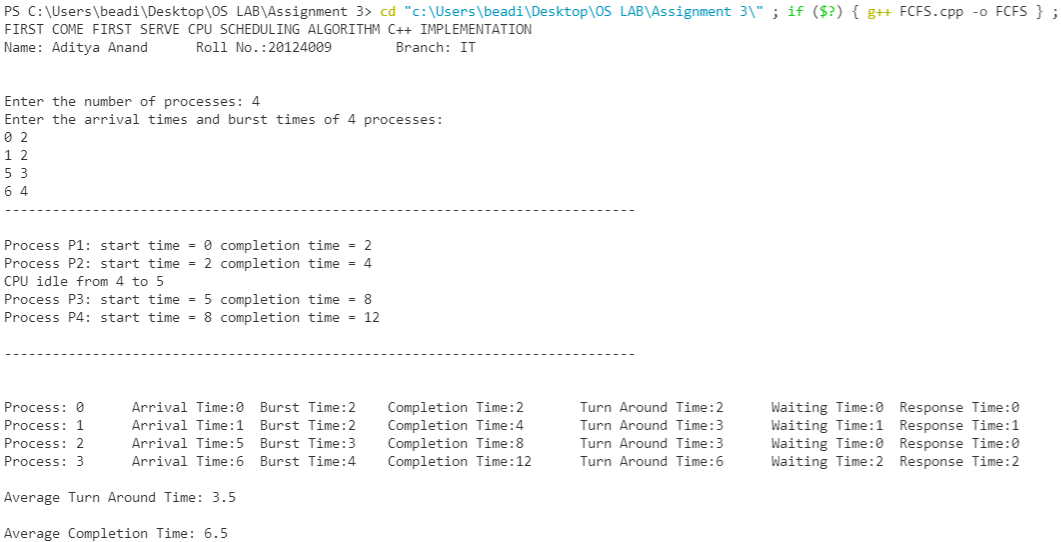
    cout<<"\nAverage Turn Around Time: "<<(float)((1.0\*t\_TAT)/(1.0\*n))<<"\n";

    cout<<"\nAverage Completion Time: "<<(float)((1.0\*t\_CT)/(1.0\*n))<<"\n";

    return 0;

}

RESULT:



PRIORITY BASED CPU SCHEDULING

CRITERIA: Priority (higher the value, greater the priority)

NOTE: In case of same priority, process with lower arrival time is executed first.

MODE: Non pre-emptive

GIVEN: List of processes with their arrival and burst time.

CODE:

#include <bits/stdc++.h>

using namespace ::std;

class process{

    public:

        int priority;

        int id;

        int arrivalTime;

        int burstTime;

        bool ready;

        int completionTime;

        int TAT;

        int WT;

        int RT;

};

struct comp{

    bool operator()(process const &p1, process const &p2){

        return p1.priority < p2.priority;

    }

};

void PriorityBasedScheduling(vector<process> &v){

    priority\_queue<process, vector<process>, comp> p;

    int cur\_time = INT\_MAX;

    int n=v.size();

    for(int i=0; i<n; i++){

        cur\_time = min(cur\_time, v[i].arrivalTime);

    }

    int count = 0;

    while(true){

        for(int i=0; i<n; i++){

            if(v[i].arrivalTime<=cur\_time && !v[i].ready){

                v[i].ready = true;

                p.push(v[i]);

                count++;

            }

        }

        if(count<n && p.empty()){

            cout<<"CPU empty from "<<cur\_time<<" to "<<cur\_time+1<<"\n";

            cur\_time++;

            continue;

        }

        if(p.empty()){

            break;

        }

        process cur\_process = p.top();

        p.pop();

        v[cur\_process.id].RT = cur\_time-cur\_process.arrivalTime;

        cur\_time+=cur\_process.burstTime;

        v[cur\_process.id].completionTime=cur\_time;

    }

}

int main(){

    cout << "PRIORITY BASED CPU SCHEDULING ALGORITHM C++ IMPLEMENTATION\n";

    cout << "Name: Aditya Anand\tRoll No.:20124009\t Branch: IT\n\n\n";

    int n = 0;

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

    cin >> n;

    cout << "Enter the arrival times and burst times and priority values of " << n << " processes: \n";

    vector<process> v(n);

    for (int i = 0; i < n; i++){

        cin >> v[i].arrivalTime >> v[i].burstTime >> v[i].priority;

        v[i].ready = false;

        v[i].id = i;

    }

    cout << "-------------------------------------------------------------------------------\n";

    cout << "\n";

    PriorityBasedScheduling(v);

    cout << "\n";

    cout << "-------------------------------------------------------------------------------\n";

    cout << "\n\n";

    int t\_TAT = 0;

    int t\_CT = 0;

    for (int i = 0; i < n; i++){

        v[i].TAT = v[i].completionTime - v[i].arrivalTime;

        v[i].WT = v[i].TAT - v[i].burstTime;

        t\_TAT += v[i].TAT;

        t\_CT += v[i].completionTime;

    }

    for (auto p : v){

        cout << "Process: " << p.id << "\tArrival Time:" << p.arrivalTime << "\tBurst Time:" << p.burstTime << "\tCompletion Time:" << p.completionTime;

        cout << "\tTurn Around Time:" << p.TAT << "\tWaiting Time:" << p.WT << "\tResponse Time:" << p.RT << "\n";

    }

    cout << "\nAverage Turn Around Time: " << (float)((1.0 \* t\_TAT) / (1.0 \* n)) << "\n";

    cout << "\nAverage Completion Time: " << (float)((1.0 \* t\_CT) / (1.0 \* n)) << "\n";

    return 0;

}

RESULT:

