Task 9 report

In this task we create three algorithms that related to scheduling in operating systems where the first algorithm which represents a line where you first come then first serve. The second algorithm represents the shortest job first by taking the least amount of burst time and lastly round robin which sequentially tackles each process by the quantum time.

GitHub: <https://github.com/yesso2004/Operating-sys>

#include <iostream>

#include <vector>

#include <queue>

#include <climits>

using namespace std;

struct Proccess

{

    int id;

    int Arrival;

    int Burst;

    int Completion;

    int Turnaround;

    int Waiting;

};

void FirstComeFirstServe(vector<Proccess> Proccesses)

{

    int size = Proccesses.size();

    for (int i = 0; i < size; i++)

    {

        for (int j = 0; j < size - i - 1; j++)

        {

            if (Proccesses[j].Arrival > Proccesses[j + 1].Arrival)

            {

                swap(Proccesses[j], Proccesses[j + 1]);

            }

        }

    }

    int CurrentTime = 0;

    float TotalTime = 0;

    for (int i = 0; i < size; i++)

    {

        CurrentTime = max(CurrentTime, Proccesses[i].Arrival);

        Proccesses[i].Completion = CurrentTime + Proccesses[i].Burst;

        Proccesses[i].Turnaround = Proccesses[i].Completion - Proccesses[i].Arrival;

        Proccesses[i].Waiting = Proccesses[i].Turnaround - Proccesses[i].Burst;

        TotalTime = TotalTime + Proccesses[i].Waiting;

        CurrentTime = Proccesses[i].Completion;

    }

    cout << "First Come First Serve Scheduling:\n";

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

    cout << "ID  | Arrival | Burst | Completion | Turnaround | Waiting\n";

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

    for (const auto &Process : Proccesses)

    {

        cout << Process.id << "   | "

             << Process.Arrival << "       | "

             << Process.Burst << "     | "

             << Process.Completion << "         | "

             << Process.Turnaround << "         | "

             << Process.Waiting << endl;

    }

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

    cout << "Average Waiting Time: " << TotalTime / size << endl;

}

void ShortestJobFirst(vector<Proccess> Proccesses)

{

    int size = Proccesses.size();

    for (int i = 0; i < size; i++)

    {

        for (int j = 0; j < size - i - 1; j++)

        {

            if (Proccesses[j].Arrival > Proccesses[j + 1].Arrival)

            {

                swap(Proccesses[j], Proccesses[j + 1]);

            }

        }

    }

    int CurrentTime = 0;

    float TotalTime = 0;

    vector<bool> Completed(size, false);

    int CompletedCount = 0;

    while (CompletedCount < size)

    {

        int ShortestIndex = -1;

        int ShortestBurst = INT\_MAX;

        for (int i = 0; i < size; i++)

        {

            if (!Completed[i] && Proccesses[i].Arrival <= CurrentTime)

            {

                if (Proccesses[i].Burst < ShortestBurst)

                {

                    ShortestBurst = Proccesses[i].Burst;

                    ShortestIndex = i;

                }

            }

        }

        if (ShortestIndex == -1)

        {

            CurrentTime = Proccesses[CompletedCount].Arrival;

            continue;

        }

        CurrentTime = max(CurrentTime, Proccesses[ShortestIndex].Arrival);

        Proccesses[ShortestIndex].Completion = CurrentTime + Proccesses[ShortestIndex].Burst;

        Proccesses[ShortestIndex].Turnaround = Proccesses[ShortestIndex].Completion - Proccesses[ShortestIndex].Arrival;

        Proccesses[ShortestIndex].Waiting = Proccesses[ShortestIndex].Turnaround - Proccesses[ShortestIndex].Burst;

        TotalTime = TotalTime + Proccesses[ShortestIndex].Waiting;

        Completed[ShortestIndex] = true;

        CompletedCount++;

        CurrentTime = Proccesses[ShortestIndex].Completion;

    }

    cout << "Shortest Job First: " << endl;

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

    cout << "ID  | Arrival | Burst | Completion | Turnaround | Waiting\n";

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

    for (const auto &Process : Proccesses)

    {

        cout << Process.id << "   | "

             << Process.Arrival << "       | "

             << Process.Burst << "     | "

             << Process.Completion << "         | "

             << Process.Turnaround << "         | "

             << Process.Waiting << endl;

    }

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

    cout << "Average Waiting Time: " << TotalTime / size << endl;

}

void RoundRobin(vector<Proccess> Proccesses, int QuantumTime)

{

    int size = Proccesses.size();

    vector<int> RemainingBursts(size);

    for (int i = 0; i < size; i++)

    {

        RemainingBursts[i] = Proccesses[i].Burst;

    }

    int CurrentTime = 0;

    int CompletedCount = 0;

    float TotalTime = 0;

    for (int i = 0; i < size; i++)

    {

        for (int j = 0; j < size - i - 1; j++)

        {

            if (Proccesses[j].Arrival > Proccesses[j + 1].Arrival)

            {

                swap(Proccesses[j], Proccesses[j + 1]);

            }

        }

    }

    while (CompletedCount < size)

    {

        bool Executed = false;

        for (int i = 0; i < size; i++)

        {

            if (RemainingBursts[i] > 0 && Proccesses[i].Arrival <= CurrentTime)

            {

                Executed = true;

                if (RemainingBursts[i] > QuantumTime)

                {

                    CurrentTime = CurrentTime + QuantumTime;

                    RemainingBursts[i] = RemainingBursts[i] - QuantumTime;

                }

                else

                {

                    CurrentTime = CurrentTime + RemainingBursts[i];

                    RemainingBursts[i] = 0;

                    CompletedCount++;

                    Proccesses[i].Completion = CurrentTime;

                    Proccesses[i].Turnaround = Proccesses[i].Completion - Proccesses[i].Arrival;

                    Proccesses[i].Waiting = Proccesses[i].Turnaround - Proccesses[i].Burst;

                    TotalTime = TotalTime + Proccesses[i].Waiting;

                }

            }

        }

        if (!Executed)

        {

            for (int i = 0; i < size; i++)

            {

                if (RemainingBursts[i] > 0)

                {

                    CurrentTime = Proccesses[i].Arrival;

                    break;

                }

            }

        }

    }

    cout << "Round Robin Scheduling (Quantum Time = " << QuantumTime << "):\n";

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

    cout << "ID  | Arrival | Burst | Completion | Turnaround | Waiting\n";

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

    for (const auto &Process : Proccesses)

    {

        cout << Process.id << "   | "

             << Process.Arrival << "       | "

             << Process.Burst << "     | "

             << Process.Completion << "         | "

             << Process.Turnaround << "         | "

             << Process.Waiting << endl;

    }

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

    cout << "Average Waiting Time: " << TotalTime / size << endl;

}

int main()

{

    int ProccessesNum;

    cout << "Enter number of Proccesses: ";

    cin >> ProccessesNum;

    vector<Proccess> Proccesses(ProccessesNum);

    for (int i = 0; i < ProccessesNum; i++)

    {

        Proccesses[i].id = i + 1;

        cout << "Enter Arrival Time for Process " << Proccesses[i].id << " : ";

        cin >> Proccesses[i].Arrival;

        cout << "Enter Burst Time for Process " << Proccesses[i].id << " : ";

        cin >> Proccesses[i].Burst;

    }

    cout << endl;

    FirstComeFirstServe(Proccesses);

    cout << endl;

    ShortestJobFirst(Proccesses);

    cout << endl;

    int QuantumTime;

    cout << "Enter amount for Quantum Time: ";

    cin >> QuantumTime;

    cout << endl;

    RoundRobin(Proccesses, QuantumTime);

}

