

Assignment 4

Q1: FCFS (First Come First Serve)

- Write a C/C++ program that:
- Reads the number of processes (Sample input)
- Accepts arrival time and burst time for each process (Sample input)
- Schedules processes using FCFS
- Calculates WT and TAT
- Displays results in tabular form
- Calculate and display Average WT and TAT

```
#include <iostream>
#include <vector>
#include <sstream>
#include <algorithm>
#include <climits>

struct Times
{
    std::string processID;
    int arrivalTime = 0;
    int burstTime = 0;
};

struct Report
{
    std::string processId;
    int completionTime = 0;
    int turnAroundTime = 0;
    int waitingTime = 0;
};

class SchedulingAlgorithms
{
private:
    std::vector<Times> info;

public:
    SchedulingAlgorithms(const int &n)
    {
        info.resize(n);

        for (int i = 0; i < n; i++)
        {
            std::string row;
            std::cout << "Enter {processID,arrivalTime,burstTime}: ";
            std::getline(std::cin, row);
            std::istringstream iss(row);
            iss >> processID >> arrivalTime >> burstTime;
            info[i] = {processID, arrivalTime, burstTime};
        }
    }

    void calculateReport()
    {
        for (int i = 0; i < n; i++)
        {
            completionTime[i] = arrivalTime[i] + burstTime[i];
            turnAroundTime[i] = completionTime[i] - arrivalTime[i];
            waitingTime[i] = turnAroundTime[i] - burstTime[i];
        }
    }

    void displayReport()
    {
        std::cout << "Completion Time\tTurn Around Time\tWaiting Time" << std::endl;
        for (int i = 0; i < n; i++)
        {
            std::cout << completionTime[i] << "\t" << turnAroundTime[i] << "\t" << waitingTime[i] << std::endl;
        }
    }

    void calculateAverages()
    {
        double sumCompletionTime = 0;
        double sumTurnAroundTime = 0;
        double sumWaitingTime = 0;

        for (int i = 0; i < n; i++)
        {
            sumCompletionTime += completionTime[i];
            sumTurnAroundTime += turnAroundTime[i];
            sumWaitingTime += waitingTime[i];
        }

        averageCompletionTime = sumCompletionTime / n;
        averageTurnAroundTime = sumTurnAroundTime / n;
        averageWaitingTime = sumWaitingTime / n;
    }

    void displayAverages()
    {
        std::cout << "Average Completion Time: " << averageCompletionTime << std::endl;
        std::cout << "Average Turn Around Time: " << averageTurnAroundTime << std::endl;
        std::cout << "Average Waiting Time: " << averageWaitingTime << std::endl;
    }
};
```

```
    std::cin >> row;

    std::stringstream rowStream(row);

    std::string value;

    std::getline(rowStream, value, ',');
    info[i].processsID = value;

    std::getline(rowStream, value, ',');
    info[i].arrivalTime = stoi(value);

    std::getline(rowStream, value, ',');
    info[i].burstTime = stoi(value);
}

void fcfs()
{
    std::sort(info.begin(), info.end(), [](const Times &p1, const Times
&p2)
              { return p1.arrivalTime < p2.arrivalTime; });
    std::vector<Report> rep(info.size());
    int completionTime = 0;
    for (int i = 0; i < info.size(); i++)
    {
        rep[i].processId = info[i].processsID;
        completionTime += info[i].burstTime;

        rep[i].completionTime = completionTime;
        rep[i].turnAroundTime = rep[i].completionTime -
info[i].arrivalTime;
        rep[i].waitingTime = rep[i].turnAroundTime - info[i].burstTime;
    }
    display();

    std::cout << "-----" << std::endl;
    for (const Report &r : rep)
    {
        std::cout << "PROCESSS ID: " << r.processId << " TURN AROUND
TIME: " << r.turnAroundTime << " WAITING TIME: " << r.waitingTime <<
std::endl;
    }
}

void sjf()
{
    int n = info.size();
    std::vector<Report> rep(n);
    std::vector<bool> completed(n, false);

    int completedCount = 0;
    int currentTime = 0;

    while (completedCount < n)
    {
        /
```

```
int idx = -1;
int minBurst = INT_MAX;

// Find process with smallest burst time among arrived
processes
for (int i = 0; i < n; i++)
{
    if (!completed[i] && info[i].arrivalTime <= currentTime)
    {
        if (info[i].burstTime < minBurst)
        {
            minBurst = info[i].burstTime;
            idx = i;
        }
        // Tie breaker: earlier arrival
        else if (info[i].burstTime == minBurst)
        {
            if (info[i].arrivalTime < info[idx].arrivalTime)
                idx = i;
        }
    }
}

// If no process has arrived yet, move time forward
if (idx == -1)
{
    currentTime++;
    continue;
}

// Execute selected process
currentTime += info[idx].burstTime;

rep[idx].processId = info[idx].processID;
rep[idx].completionTime = currentTime;
rep[idx].turnAroundTime = currentTime - info[idx].arrivalTime;
rep[idx].waitingTime =
    rep[idx].turnAroundTime - info[idx].burstTime;

completed[idx] = true;
completedCount++;
}

display();

std::cout << "-----\n";
for (const Report &r : rep)
{
    std::cout << "PROCESS ID: " << r.processId
        << " TURN AROUND TIME: " << r.turnAroundTime
        << " WAITING TIME: " << r.waitingTime << std::endl;
}
}
```

```

void display() const
{
    const size_t infoSize = info.size();
    std::cout << "-----" << std::endl;
    for (int i = 0; i < infoSize; i++)
    {
        std::cout << "PROCESS ID: " << info[i].processID << " ARRIVAL
TIME: " << info[i].arrivalTime << " BURST TIME: " << info[i].burstTime <<
std::endl;
    }
}
};

int main()
{
    SchedulingAlgorithms fcfs(4);
    fcfs.fcfs();
    return 0;
}

```

output

```

ASSIGNMENT-4/* && g++ ass4.cpp -o ass4 && *//home/gaurang/d
PROCESS ID: P1 ARRIVAL TIME: 0 BURST TIME: 6
PROCESS ID: P2 ARRIVAL TIME: 1 BURST TIME: 8
PROCESS ID: P3 ARRIVAL TIME: 2 BURST TIME: 7
PROCESS ID: P4 ARRIVAL TIME: 3 BURST TIME: 3
-----
PROCESSS ID: P1 TURN AROUND TIME: 6 WAITING TIME: 0
PROCESSS ID: P2 TURN AROUND TIME: 13 WAITING TIME: 5
PROCESSS ID: P3 TURN AROUND TIME: 19 WAITING TIME: 12
PROCESSS ID: P4 TURN AROUND TIME: 21 WAITING TIME: 18

```

Ques2. Q2: SJF (Shortest Job First)

- Write a C/C++ program that:
- Reads the number of processes (Sample input)
- Accepts arrival time and burst time for each process (Sample input)
- Schedules processes using SJF
- Computes WT and TAT
- Displays results in tabular form
- Calculate and display Average WT and TAT

```

#include <iostream>
#include <vector>

```

```
#include <iostream>
#include <algorithm>
#include <climits>

struct Times
{
    std::string processID;
    int arrivalTime = 0;
    int burstTime = 0;
};

struct Report
{
    std::string processId;
    int completionTime = 0;
    int turnAroundTime = 0;
    int waitingTime = 0;
};

class SchedulingAlgorithms
{
private:
    std::vector<Times> info;

public:
    SchedulingAlgorithms(const int &n)
    {
        info.resize(n);

        for (int i = 0; i < n; i++)
        {
            std::string row;
            std::cout << "Enter {processID,arrivalTime,burstTime}: ";
            std::cin >> row;

            std::stringstream rowStream(row);

            std::string value;

            std::getline(rowStream, value, ',');
            info[i].processID = value;

            std::getline(rowStream, value, ',');
            info[i].arrivalTime = stoi(value);

            std::getline(rowStream, value, ',');
            info[i].burstTime = stoi(value);
        }
    }

    void fcfs()
    {
        std::sort(info.begin(), info.end(), [](const Times &p1, const Times &p2)
                  { return p1.arrivalTime < p2.arrivalTime; });

        std::vector<Report> rep(info.size());
        int completionTime = 0;
    }
}
```

```
for (int i = 0; i < info.size(); i++)
{
    rep[i].processId = info[i].processsID;
    completionTime += info[i].burstTime;

    rep[i].completionTime = completionTime;
    rep[i].turnAroundTime = rep[i].completionTime -
info[i].arrivalTime;
    rep[i].waitingTime = rep[i].turnAroundTime - info[i].burstTime;
}
display();

std::cout << "-----" << std::endl;
for (const Report &r : rep)
{
    std::cout << "PROCESSS ID: " << r.processId << " COMPLETION
TIME: " << r.completionTime << " TURN AROUND TIME: " << r.turnAroundTime <<
" WAITING TIME: " << r.waitingTime << std::endl;
}
void sjf()
{
    std::sort(info.begin(), info.end(), [](const Times &p1, const Times
&p2)
              { return p1.arrivalTime < p2.arrivalTime; });
    int infoSize = info.size();
    std::vector<Report> rep(infoSize);
    std::vector<bool> isComplete(infoSize, false);
    int time = info[0].arrivalTime;
    for (int i = 0; i < infoSize; i++)
    {
        size_t index = i;
        int minBurst = INT_MAX;
        for (int j = 0; j < infoSize; j++)
        {
            if (!isComplete[j] && info[j].arrivalTime <= time &&
info[j].burstTime < minBurst)
            {
                index = j;
            }
        }
        rep[index].processId = info[index].processsID;
        time += info[index].burstTime;
        rep[index].completionTime = time;
        rep[index].turnAroundTime = rep[index].completionTime -
info[index].arrivalTime;
        rep[index].waitingTime = rep[index].turnAroundTime -
info[index].burstTime;
        isComplete[index] = true;
    }

    display();

    std::cout << "-----\n";
}
```

```
        for (const Report &r : rep)
        {
            std::cout << "PROCESS ID: " << r.processId << " COMPLETION
TIME: " << r.completionTime << " TURN AROUND TIME: " << r.turnAroundTime <<
" WAITING TIME: " << r.waitingTime << std::endl;
        }
    }

    void display() const
    {
        const size_t infoSize = info.size();
        std::cout << "-----" << std::endl;
        for (int i = 0; i < infoSize; i++)
        {
            std::cout << "PROCESS ID: " << info[i].processID << " ARRIVAL
TIME: " << info[i].arrivalTime << " BURST TIME: " << info[i].burstTime <<
std::endl;
        }
    }
};

int main()
{
    SchedulingAlgorithms fcfs(4);
    fcfs.sjf();
    return 0;
}
```

output

```
/usr/local/lib/cmake/cmake-3.20/Modules/FindZLIB.cmake
-----
PROCESS ID: P1 ARRIVAL TIME: 0 BURST TIME: 6
PROCESS ID: P2 ARRIVAL TIME: 1 BURST TIME: 8
PROCESS ID: P3 ARRIVAL TIME: 2 BURST TIME: 7
PROCESS ID: P4 ARRIVAL TIME: 3 BURST TIME: 3
-----
PROCESSSS ID: P1 COMPLETION TIME: 6 TURN AROUND TIME: 6 WAITING TIME: 0
PROCESSS ID: P2 COMPLETION TIME: 24 TURN AROUND TIME: 23 WAITING TIME: 15
PROCESSS ID: P3 COMPLETION TIME: 16 TURN AROUND TIME: 14 WAITING TIME: 7
PROCESSS ID: P4 COMPLETION TIME: 9 TURN AROUND TIME: 6 WAITING TIME: 3
```