Assignment 4

```
Name : Ravish Ranjan
Course : MCA
Semester : 2nd semester
```

```
Q1: FCFS (First Come First Serve)
```

Write a C/C++ program that: 1. 2. 3. 4. 5. 6. Reads the number of processes (Sample input) Accepts arrival time and burst time for each process (Sample input) Schedules processes using SRTF Calculates WT and TAT Displays results in tabular form Calculate and display Average WT and TAT Q2: SJF (Shortest Job First) Write a C/C++ program that: 1. 2. 3. 4. 5. 6. Reads the number of processes (Sample input) Accepts arrival time and burst time for each process (Sample input) Schedules processes using Round Robin with q=4 Computes WT and TAT Displays results in tabular form Calculate and display Average WT and TAT

Ans

```cpp
#ifndef SCHEDULER_CPP
#define SCHEDULER_CPP

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


std::vector<std::string> split(std::string inp,char sep = ' '){
    std::stringstream ss(inp);
    std::string segment;
    std::vector<std::string> results;

    while (std::getline(ss, segment, sep)) results.push_back(segment);

    return results;
}

class Job{
    public:
        int id;
        int at;
        int bt;
        int pt;
        int ct = 0;
        int tat = 0;
        int wt = 0;
        int rt = 0;
```

```
        Job(int id,int bt,int at = 0,int pt =
0):id(id),bt(bt),at(at),pt(pt){}
};

class Scheduler{
    public:
        virtual void apply(std::vector<Job>& jobs) = 0;
        void printTable(const std::vector<Job>& jobs){
            std::cout << "\nid\tat\tbt\tpt\tct\ttat\twt\trt" << std::endl;
            for (Job job:jobs){

printf("P%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\n",job.id,job.at,job.bt,job.pt,job.
ct,job.tat,job.wt,job.rt);
            }
            std::cout << std::endl;
        }
        void getAvgs(const std::vector<Job>& jobs){
            double avgTat,avgWt,avgRt;
            for(Job job:jobs){
                avgTat += job.tat;
                avgWt += job.wt;
                avgRt += job.rt;
            }
            std::cout << "Avg. Turn Around Time : " <<
(double)avgTat/jobs.size() << std::endl;
            std::cout << "Avg. Waiting Time : " <<
(double)avgWt/jobs.size() << std::endl;
            // std::cout << "Avg. Response Time : " <<
(double)avgRt/jobs.size() << std::endl;
        }
};

#endif
```

```
#include "scheduler.cpp"
#include <queue>

class FCFS:public Scheduler{
    public:
        void apply(std::vector<Job>& jobs){
            std::sort(jobs.begin(),jobs.end(),[](const Job& j1,const Job&
j2){
                return j1.at < j2.at;
            });
            int gt_time = 0;
            for(Job& job:jobs){
                gt_time+= job.bt;
                job.ct = gt_time;
                job.tat = job.ct-job.at;
                job.wt = job.tat-job.bt;
```

```
            }
        }
};

struct BurstComparator{
    bool operator()(const Job& j1,const Job& j2)const {
        return j1.bt < j2.bt;
    }
};

class SRJF:public Scheduler{
    public:
        void apply(std::vector<Job>& jobs){
            std::sort(jobs.begin(),jobs.end(),[](const Job& j1,const Job&
j2){
                return j1.at < j2.at;
            });

            std::vector<int> rem_bt(jobs.size());
            for (int i = 0; i < jobs.size(); i++){
                rem_bt[i] = jobs[i].bt;
            }

            int at = 0;
            int completed = 0;

            while (completed < jobs.size()){
                int min_idx = -1;
                int min_bt = __INT_MAX__;

                for (int i = 0; i < jobs.size(); i++){
                    if (jobs[i].at <= at && rem_bt[i] > 0 && rem_bt[i] <
min_bt){
                        min_bt = rem_bt[i];
                        min_idx = i;
                    }
                }

                at++;

                if (min_bt > -1) {
                    rem_bt[min_idx]--;;
                    if (rem_bt[min_idx] <= 0){
                        jobs[min_idx].ct = at;
                        jobs[min_idx].tat = jobs[min_idx].ct -
jobs[min_idx].at;
                        jobs[min_idx].wt = jobs[min_idx].tat -
jobs[min_idx].bt;
                        completed++;
                    }
                }
            }
        }
};
```

```
class RR:public Scheduler{
    private:
        int q;
    public:
        RR(int q):q(q){}
        void apply(std::vector<Job>& jobs){
            std::sort(jobs.begin(),jobs.end(),[](const Job& j1,const Job&
j2){
                return j1.at < j2.at;
            });

            std::vector<int> rem_bt(jobs.size());
            for (int i = 0; i < jobs.size(); i++){
                rem_bt[i] = jobs[i].bt;
            }

            std::deque<int> ready_q;

            int at = 0;
            int completed = 0;
            int next_idx = 0;

            while(next_idx < jobs.size() && jobs[next_idx].at <= at) {
                ready_q.push_back(next_idx);
                next_idx++;
            }

            while(completed < jobs.size()) {
                if(ready_q.empty()) {
                    if(next_idx < jobs.size()) {
                        at = jobs[next_idx].at;
                        ready_q.push_back(next_idx);
                        next_idx++;
                    }
                    continue;
                }

                int job_idx = ready_q.front();
                ready_q.pop_front();  // O(1) with deque

                int exec_time = std::min(q, rem_bt[job_idx]);
                rem_bt[job_idx] -= exec_time;
                at += exec_time;

                while(next_idx < jobs.size() && jobs[next_idx].at <= at) {
                    ready_q.push_back(next_idx);
                    next_idx++;
                }

                if(rem_bt[job_idx] == 0) {
                    jobs[job_idx].ct = at;
                    jobs[job_idx].tat = jobs[job_idx].ct -
jobs[job_idx].at;
```

```
                    jobs[job_idx].wt = jobs[job_idx].tat -
jobs[job_idx].bt;
                    completed++;
                } else {
                    ready_q.push_back(job_idx);
                }
            }
        }
};

int main(){
    int n = 0;
    std::string inp = "";
    std::cout << "Enter the no. of jobs : ";
    std::cin >> n;

    std::vector<Job> jobs_srjf,jobs_rr;
    std::cout << "Enter the properties of following jobs (arival time,burst
time)" << std::endl;

    for (int i = 0; i < n; i++){
        std::cout << "P" << i+1 << " : ";
        std::cin >> inp;
        std::vector<std::string> parts = split(inp,',');


jobs_srjf.push_back(Job(i+1,std::stoi(parts[1]),std::stoi(parts[0])));

jobs_rr.push_back(Job(i+1,std::stoi(parts[1]),std::stoi(parts[0])));
    }

    std::cout << "SRJF" << std::endl;

    SRJF srjf;
    srjf.apply(jobs_srjf);
    srjf.printTable(jobs_srjf);
    srjf.getAvgs(jobs_srjf);

    std::cout << "RR" << std::endl;
    RR rr(4);
    rr.apply(jobs_rr);
    rr.printTable(jobs_rr);
    rr.getAvgs(jobs_rr);

    return 0;
}
```

```
ravish@ravishPC:../os/prog ⎇(main)> ./a4.out
Enter the no. of jobs : 4
Enter the properties of following jobs (arival time,burst time)
P1 : 0,4
P2 : 1,6
P3 : 1,7
P4 : 3,3

id      at      bt      pt      ct      tat     wt      rt
P1      0       4       0       4       4       0       0
P2      1       6       0       13      12      6       0
P3      1       7       0       20      19      12      0
P4      3       3       0       7       4       1       0

Avg. Turn Around Time : 9.75
Avg. Waiting Time : 4.75

id      at      bt      pt      ct      tat     wt      rt
P1      0       4       0       4       4       0       0
P2      1       6       0       17      16      10      0
P3      1       7       0       20      19      12      0
P4      3       3       0       15      12      9       0

Avg. Turn Around Time : 12.75
Avg. Waiting Time : 7.75
ravish@ravishPC:../os/prog ⎇(main)>
```