Programming Assignment 3 Report

Names: Andrew Ortiz and Mitchell Curtis

Problem Description

For this programming assignment we are implementing scheduling algorithms for processes within an operating system. These algorithms include: FCFS (First Come First Served), SJF (Shortest Job First), RR (Round Robin), and a Priority scheduling algorithm that does shortest job first based on each priority level.

Program Design

Our program consists of two main parts for each scheduling algorithm, the first being the driver.cpp files that take input from a .txt file containing various processes and contains the main function. The second main part is the actual scheduling algorithm that both simulates and prints the results for easy viewing. Both the driver and scheduling portions use the PCB (process control block) class that stores the process id, name, priority, burst time, and arrival time.

```
PCB(string name, unsigned int id = 0, unsigned int priority = 1, unsigned int burst_time = 0) {
    this->id = id;
    this->name = name;
    this->priority = priority;
    this->burst_time = burst_time;
    this->arrival_time = 0;
}
```

System Implementation

The logic implementation varies between scheduling algorithms, but the basics stay the same, the driver.cpp loads the processes.txt into a vector of PCB elements to be used in the scheduler.cpp file. This file (regardless of scheduler method) does some basic things, one being an initialization that can include copying the vector of PCB elements so that the original doesn't have to be modified.

```
void SchedulerFCFS::init(std::vector<PCB>& process_list)
{
    this->process_list = process_list;
}
```

The next step is to simulate the scheduling method, this logic varies between methods, but the basics include which process should run when, and recording waiting/turnaround time. For methods like RR (round robin) this gets more complicated as you have to track the remaining burst times for each process and which process(es) have completed.

```
void SchedulerFCFS::simulate()
{
    double turn = 0;
    double wait = 0;

    // calculate average wait and turn around time
    for(int i = 0; i < process_list.size(); i++)
    {
        // calculate total turn around time at each index
        for(int j = i; j >= 0; j--)
        {
            turn = turn + process_list[j].burst_time;
        }

        // calcualte total waiting time
        if(i > 0)
        {
            for(int j = i-1; j >= 0; j--)
            {
                wait = wait + process_list[j].burst_time;
            }
        }

    }
```

The final important system implementation is the print_results() function that prints the processes through the simulation as well as the average waiting and turnaround time once the simulation portion is completed.

```
void SchedulerFCFS::print_results()
{
    // print each process and its time units
    for(int i = 0; i < process_list.size(); i++)
    {
        cout << "Running Process " << process_list[i].name << " for " << process_list[i].burst_time << " tin
    }

    // print the wait and turn around times of each process
    for(int i = 0; i < process_list.size(); i++)
    {
        int turn = 0;
        int wait = 0;

        // calcualte and print current turn time
        for(int j = i; j >= 0; j--)
        {
            turn = turn + process_list[j].burst_time;
        }

        // calculate and print current wait time
        if(i > 0)
        {
            for(int j = i-1; j >= 0; j--)
            {
                wait = wait + process_list[j].burst_time;
            }
        }

        cout << process_list[i].name << " turn-around time = " << turn << ", waiting time = " << wait << enc
    }

    // finally print the final average turn and wait time
    cout << "Average turn-around time = " << this->AvgTurn << ", Average waiting time = " << this->AvgWait <
}
```

Results

```
PROBLEMS (10)   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

Course: CS433 (Operating Systems)
Description : test FCFS scheduling algorithm
=================================
Process 0: T1 has priority 4 and burst time 20
Process 1: T2 has priority 3 and burst time 25
Process 2: T3 has priority 3 and burst time 25
Process 3: T4 has priority 5 and burst time 15
Process 4: T5 has priority 5 and burst time 20
Process 5: T6 has priority 1 and burst time 10
Process 6: T7 has priority 3 and burst time 30
Process 7: T8 has priority 10 and burst time 25
Running Process T1 for 20 time units
Running Process T2 for 25 time units
Running Process T3 for 25 time units
Running Process T4 for 15 time units
Running Process T5 for 20 time units
Running Process T6 for 10 time units
Running Process T7 for 30 time units
Running Process T8 for 25 time units
T1 turn-around time = 20, waiting time = 0
T2 turn-around time = 45, waiting time = 20
T3 turn-around time = 70, waiting time = 45
T4 turn-around time = 85, waiting time = 70
T5 turn-around time = 105, waiting time = 85
T6 turn-around time = 115, waiting time = 105
T7 turn-around time = 145, waiting time = 115
T8 turn-around time = 170, waiting time = 145
Average turn-around time = 94.375, Average waiting time = 73.125
mitchellcurtis@MacBook-Pro-2 assign3 %
```

Conclusion

Different scheduling methods have different purposes and are useful for different things. For systems where response time isn't uber-important, methods like FCFS could be preferred as it tends to have less overhead processing. Consumer desktop operating systems like Windows/MacOS might chose round robin scheduling as it allows many processes to make progress together, which could increase the users perceived response time. For systems where the raw number of processes completed (total throughput) over a given timeframe shortest job first might make most sense. Choosing the correct scheduling method requires knowledge of what's the expected workload and output for a given system, sometimes this isn't concrete knowledge and compromises have to be made so that important processes don't get stalled for less-important processes.