Laboratory 4

Title of the Laboratory Exercise: Programs for process scheduling algorithms

1. Introduction and Purpose of Experiment

   A Process Scheduler schedules different processes to CPU based on particular scheduling algorithms. There are various scheduling algorithms present in each group of operating system. By solving these problems students will be able use different scheduling algorithms as part of their implementation

2. Aim and Objectives

   Aim

   - To develop programs to implement scheduling algorithms

   Objectives

   At the end of this lab, the student will be able to

   - Distinguish different scheduling algorithms
   - Apply the logic of scheduling algorithms wherever required
   - Create C programs to simulate scheduling algorithms

3. Experimental Procedure

   i. Analyse the problem statement
   ii. Design an algorithm for the given problem statement and develop a flowchart/pseudo-code
   iii. Implement the algorithm in C language
   iv. Compile the C program
   v. Test the implemented program
   vi. Document the Results
   vii. Analyse and discuss the outcomes of your experiment

4. Questions

Write a multithreaded program to simulate the following process scheduling algorithms. Calculate average waiting time and average turnaround time for processes under each scheduling algorithm by separate threads

Instructions: Assume all the processes arrive at the same time. For round robin scheduling algorithm, read the number of processes in the system, their CPU burst times and the size of the time slice. For priority scheduling algorithm, read the number of processes in the system, their CPU burst times and the priorities.

a) Priority
b) Round Robin

5. Calculations/Computations/Algorithms

```
priority_schedule::get_tat()
1. done = false
2. while (!done)
3.     temp_max = -∞
4.     temp_max_i = 0
5.     for i = 0 to queue.size()
6.             if (queue[i] != 0 and priority[i] >= temp_max)
7.                     temp_max = priority[i]
8.                     temp_max_i = i
9.                     done = false
10.    if (!done)
11.            systime += queue[temp_max_i]
12.            queue[temp_max_i] = 0
13.            completion_times[temp_max_i] = systime
```

```
round_robin_schedule::get_tat()
1. sys_time = 0
2. done = false
3. while (!done)
4.     done = true
5.     i = 0
6.     for_each cpu_time in queue :
7.             if (cpu_time != 0)
8.                     done = false
9.                     current_time = cpu_time >= time_slice ? time_slice : cpu_time
10.                    sys_time += current_time
11.                    cpu_time -= current_time
12.                    if (cpu_time == 0)
```

```
13.                          completion_times[i] = sys_time
14.           i++
```

6. Presentation of Results

scheduling_algo.hpp

```
#pragma once
#include <vector>

class scheduling_algo {
   public:
    virtual void read_values() = 0;
    virtual std::vector<unsigned> get_turn_around_times() = 0;
    virtual double get_avg_tat() = 0;
};
```

round_robin.hpp

```
#pragma once
#include <vector>
#include "scheduling_algo.hpp"

class round_robin_schedule : public scheduling_algo {
   public:
    round_robin_schedule(unsigned nprocesses) : nprocesses(nprocesses), cpu_burst_time(npr
ocesses), time_slice(nprocesses){};

    unsigned nprocesses;
    std::vector<unsigned> cpu_burst_time;
    unsigned time_slice;

    void read_values();
    std::vector<unsigned> get_turn_around_times();
    double get_avg_tat();
};
```

round_robin.cpp

```
#include "round_robin.hpp"

#include <iostream>
#include <numeric>
#include <sstream>
#include <vector>

void round_robin_schedule::read_values() {
    std::cout << "Enter the CPU Burst Times : ";

    for (auto& btime : this->cpu_burst_time) {
        std::cin >> btime;
    }

    std::cout << "Enter the Time Slice : ";
```

```cpp
    std::cin >> this->time_slice;
}

double round_robin_schedule::get_avg_tat() {
    std::vector<unsigned> tat = this->get_turn_around_times();
    return std::accumulate(tat.begin(), tat.end(), 0) / (double)tat.size();
}

/**
 * Calculates the Turn Around Times for the processes in the structure
*/
std::vector<unsigned> round_robin_schedule::get_turn_around_times() {
    std::vector<unsigned> queue(this->cpu_burst_time.begin(), this->cpu_burst_time.end());
    std::vector<unsigned> completion_times(this->cpu_burst_time.size());

    unsigned sys_time = 0;

    bool done = true;
    do {
        // assume you are done at start
        done = true;
        // give each of the elements in the queue an equal share of time_slice
        unsigned i = 0;
        for (auto& cpu_time : queue) {
            if (cpu_time != 0) {  // this also means you are not done
                done = false;
                unsigned current_time = cpu_time >= this->time_slice ? this-
>time_slice : cpu_time;
                sys_time += current_time;
                cpu_time -= current_time;

                // std::cout << "Current Time " << current_time << " ";
                // std::cout << "System Time " << sys_time << ";" << std::endl;

                if (cpu_time == 0) {  // you are done with this process
                    completion_times[i] = sys_time;
                    // std::cout << "Done With Proc " << i << " at " << sys_time << "\n";
                }
            }
            i++;
        }
    } while (!done);

    std::stringstream out;

    out << std::endl;

    out << "CT FOR RRS : [ ";
    for (auto TAT : completion_times) {
        out << TAT << " , ";
    }
    out << "]" << std::endl;
    std::cout << out.str();

    return completion_times;
}
```

priority_schedule.hpp

```
#pragma once
#include <vector>
#include "scheduling_algo.hpp"

class priority_schedule : public scheduling_algo {
   public:
    priority_schedule(unsigned nprocesses) : nprocesses(nprocesses), cpu_burst_time(nproce
sses), priority(nprocesses){};

    unsigned nprocesses;
    std::vector<unsigned> cpu_burst_time;
    std::vector<unsigned> priority;

    void read_values();
    std::vector<unsigned> get_turn_around_times();
    double get_avg_tat();
};
```

priority_schedule.cpp

```
#include "priority_schedule.hpp"

#include <iostream>
#include <limits>
#include <numeric>
#include <sstream>
#include <vector>

std::vector<unsigned> priority_schedule::get_turn_around_times() {
    // Arrival Times are assumed to be 0 for all the processes

    std::vector<unsigned> queue(this->cpu_burst_time.begin(), this->cpu_burst_time.end());
    std::vector<unsigned> completion_times(this->cpu_burst_time.size());

    unsigned systime = 0;

    bool done = false;
    while (!done) {
        done = true;

        // get the next most priority
        int temp_max = std::numeric_limits<int>::min();
        int temp_max_i = 0;
        for (int i = 0; i < queue.size(); i++) {
            bool cond = ((int)queue[i] != 0)                // process not exhausted
```

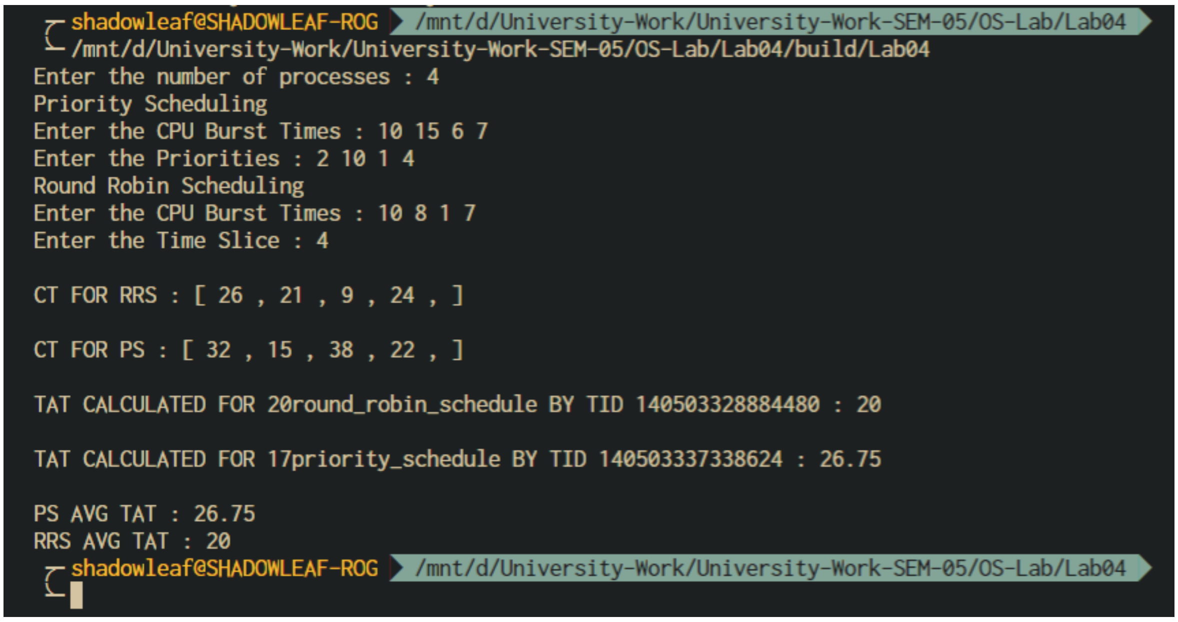
```cpp
                        && ((int)priority[i] >= temp_max);  // and has higher priority
            // std::cout << "COND : " << ((int)queue[i] >= temp_max) << std::endl;
            if (cond) {
                // std::cout << "FOUND ANOTHER temp_MAX" << std::endl;
                temp_max = priority.at(i);
                temp_max_i = i;
                done = false;
            }
        }
        if (!done) {
            // std::cout << "temp_MAX : " << priority.at(temp_max_i) << std::endl;
            systime += queue.at(temp_max_i);
            // std::cout << "SYSTIME : " << systime << std::endl;
            queue.at(temp_max_i) = 0;
            completion_times.at(temp_max_i) = systime;
        }
    }
    std::stringstream out;
    out << std::endl;
    out << "CT FOR PS : [ ";
    for (auto TAT : completion_times) {
        out << TAT << " , ";
    }
    out << "]" << std::endl;
    std::cout << out.str();

    return completion_times;
}

double priority_schedule::get_avg_tat() {
    std::vector<unsigned> tat = this->get_turn_around_times();

    return std::accumulate(tat.begin(), tat.end(), 0) / (double)tat.size();
}

void priority_schedule::read_values() {
    std::cout << "Enter the CPU Burst Times : ";
    for (auto& btime : this->cpu_burst_time) {
        std::cin >> btime;
    }
    std::cout << "Enter the Priorities : ";
    for (auto& priority : this->priority) {
        std::cin >> priority;
    }
}
```

main.cpp

```cpp
#include <iostream>
#include <sstream>
#include <typeinfo>
#include <utility>
```

```cpp
#include <vector>

#include <pthread.h>

#include "priority_schedule.hpp"
#include "round_robin.hpp"
#include "scheduling_algo.hpp"

void* worker_thread(void* schd_obj) {
    scheduling_algo* sa = static_cast<scheduling_algo*>(schd_obj);

    // compute the stuff
    double* avg_tat = new double;
    *avg_tat = sa->get_avg_tat();

    std::stringstream out;
    out << std::endl;
    out << "TAT CALCULATED FOR " << typeid(*static_cast<scheduling_algo*>(schd_obj)).name(
) << " BY TID " << pthread_self() << " : " << *avg_tat << std::endl;

    std::cout << out.str();

    return static_cast<void*>(avg_tat);
}

int main(int argc, char** argv) {
    using namespace std;

    pthread_t thr_id[2];
    auto ps_tid = &thr_id[0];
    auto rrs_tid = &thr_id[1];

    round_robin_schedule* RRS = nullptr;
    priority_schedule* PS = nullptr;

    unsigned nproc;
    cout << "Enter the number of processes : ";
    cin >> nproc;
    PS = new priority_schedule(nproc);
    RRS = new round_robin_schedule(nproc);

    // read the values - main thread
    cout << "Priority Scheduling" << endl;
    PS->read_values();
    cout << "Round Robin Scheduling" << endl;
    RRS->read_values();

    // create the threads
    pthread_create(ps_tid, NULL, worker_thread, PS);
    pthread_create(rrs_tid, NULL, worker_thread, RRS);
    void *ps_avg_tat, *rrs_avg_tat;
    // join the threads
    pthread_join(*ps_tid, &ps_avg_tat);
    pthread_join(*rrs_tid, &rrs_avg_tat);

    cout << endl;

    cout << "PS AVG TAT : " << *static_cast<double*>(ps_avg_tat) << endl;
    cout << "RRS AVG TAT : " << *static_cast<double*>(rrs_avg_tat) << endl;

    // destroy these
    delete static_cast<double*>(ps_avg_tat);
    delete static_cast<double*>(rrs_avg_tat);

    return EXIT_SUCCESS;
}
```

OUTPUT



```
╭─ shadowleaf@SHADOWLEAF-ROG  /mnt/d/University-Work/University-Work-SEM-05/OS-Lab/Lab04
╰─ /mnt/d/University-Work/University-Work-SEM-05/OS-Lab/Lab04/build/Lab04
Enter the number of processes : 4
Priority Scheduling
Enter the CPU Burst Times : 10 15 6 7
Enter the Priorities : 2 10 1 4
Round Robin Scheduling
Enter the CPU Burst Times : 10 8 1 7
Enter the Time Slice : 4

CT FOR RRS : [ 26 , 21 , 9 , 24 , ]

CT FOR PS : [ 32 , 15 , 38 , 22 , ]

TAT CALCULATED FOR 20round_robin_schedule BY TID 140503328884480 : 20

TAT CALCULATED FOR 17priority_schedule BY TID 140503337338624 : 26.75

PS AVG TAT : 26.75
RRS AVG TAT : 20
╭─ shadowleaf@SHADOWLEAF-ROG  /mnt/d/University-Work/University-Work-SEM-05/OS-Lab/Lab04
╰─ 
```

7. Analysis and Discussions

The program calculates the Average Turn Around Times for the two algorithms namely Priority Scheduling and Round Robin Scheduling, where the CPU Burst times, Priorities and the Time Slice is taken as input from the user. The Respective Completion times are calculated and then the Turn Around Times is calculated for each of the process, the work is divided among two threads, which also calculate the average by dividing the sum of Turn Around Times by the total number of processes.

The main thread waits for the data returned by the two worker threads and then prints out the results obtained.

8. Conclusions

Round Robin is a CPU scheduling algorithm where each process is assigned a fixed time slot in a cyclic way.

- It is simple, easy to implement, and starvation-free as all processes get fair share of CPU.
- One of the most commonly used technique in CPU scheduling as a core.
- It is preemptive as processes are assigned CPU only for a fixed slice of time at most.
- The disadvantage of it is more overhead of context switching.

Priority scheduling is a non-preemptive algorithm and one of the most common scheduling algorithms in batch systems.

- Each process is assigned a priority.
- Process with the highest priority is to be executed first and so on.
- Processes with the same priority are executed on first come first served basis.
- Priority can be decided based on memory requirements, time requirements or any other resource requirement.

9. Comments

    1. Limitations of Experiments

The experiment assumes the arrival time for each of the process in the system to be same, this does not make the comparison between the two algorithms fair.

    2. Limitations of Results

The result does not cover some of the important edge cases to consider for comparing the advantages and disadvantages of the two algorithms. Such as Priority Scheduling works well for Interactive Systems to provide the CPU to processes that need to be processed first.

    3. Learning happened

We learnt the two commonly used scheduling algorithms i.e. Priority Scheduling and Round Robin Scheduling.

    4. Recommendations

The program should be tested for more test cases such as when processes are generated randomly at different time intervals.