# 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

1. 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

1. Experimental Procedure
   * 1. Analyse the problem statement
     2. Design an algorithm for the given problem statement and develop a flowchart/pseudo-code
     3. Implement the algorithm in C language
     4. Compile the C program
     5. Test the implemented program
     6. Document the Results
     7. Analyse and discuss the outcomes of your experiment
2. 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.

1. Priority
2. Round Robin
3. 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++

1. 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(nprocesses), 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 : ";

    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(nprocesses), 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

                        && ((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**

#include <iostream>

#include <sstream>

#include <typeinfo>

#include <utility>

#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

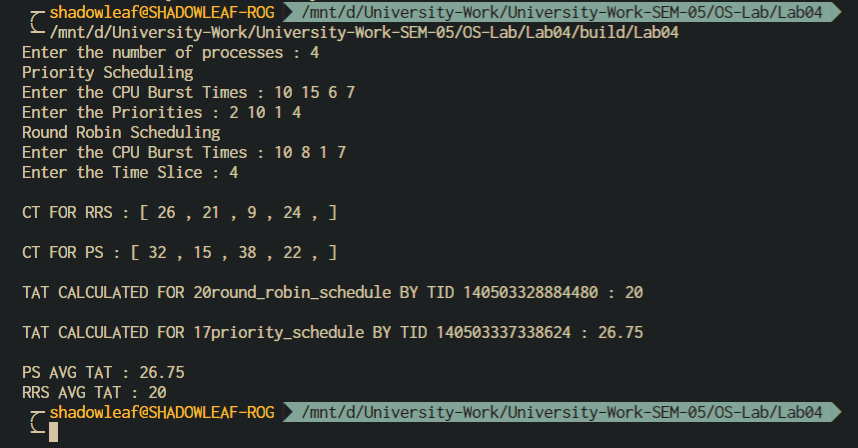
    delete static\_cast<double\*>(ps\_avg\_tat);

    delete static\_cast<double\*>(rrs\_avg\_tat);

    return EXIT\_SUCCESS;

}

**OUTPUT**



1. 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.

1. 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.

1. 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.

* + - 1. 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.

* + - 1. Learning happened

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

* + - 1. Recommendations

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