

Operating Systems : Project 1 - Parallel Process Scheduling

DUE DATES :

Milestone 1: First Come First Serve, Due: Friday Feb 19th, 11:59 AM (right before class)

Milestone 2: Round Robin, Due: Wednesday Feb 24th, 11:59 AM (right before class)

Milestone 3: Completion of process_scheduling program and analysis of algorithms

Due: Friday March 4th, 11:59 AM (right before class)

1. Introduction:

This project will expose you to the simplicity and beauty of CPU scheduling algorithms. You are to implement a virtual multicore system capable of scheduling tasks from the ready queue.

2. Objectives:

- Understand the creation and use of a CMake project
- Implement process scheduling algorithms
- Understand the performance trade-offs for various scheduling algorithms
- Exposure and usage of basic thread system calls

3. Requirements:

Undergraduate

Implement a 1-4 core virtual system capable of running First Come First Served (FCFS) and Round Robin (RR) scheduling algorithms.

Graduate

Implement an 1-N core virtual system capable of running First Come First Served (FCFS) and Round Robin (RR) scheduling algorithms.

Details

Input Data, Process Scheduling

Configuration files will be provided for your program. The binary file format is provided further below. We have provided unit tests that verify various parts of your

implementation. Your schedulers will need to track metadata such as average latency, average wall clock time, and total run time.

Core Count and Configuration

Your program, *process_analysis*, will accept configuration information from the command line. The first argument will be the name of the PCB file, and the remaining arguments will be the core configurations. In this example, the PCB file to be loaded is *pcbs.bin*, and there are three cores enabled, two with RR scheduling and one with FCFS scheduling.

```
./process_analysis pcbs.bin RR FCFS RR
```

Technical Specifications:

Process Control Block File Format:

32-bit unsigned N

32-bit unsigned burst_time[N]

Statistics to be computed include:

1. float Average Latency
2. float Average Wall-Clock
3. unsigned long Total Clock Time

Terminology and Concepts

burst time: is the runtime of the process through our virtual CPU.

arrival time: is the time the process becomes into our ready queue.

latency time: is the time between arrival and the first time the process runs on the CPU.

wall clock time: is the time a process takes to complete (from arrival to completion).

Additional Specifications:

Binary Configuration File Format

- number of process control blocks burst times (32-bit unsigned)
- first process burst time
- second process burst time
-
- N process's burst time

The PCB file will be composed of N 32-bit integers. Each integer is the burst time of a process in the ready queue. N will be the first integer in the file.

Resources:

- Please refer to class notes and posted slides from the textbook.
- Additionally, the unit tests supplied with the starter code as well as select learning module code are invaluable resources.
- The Linux/Unix programmer's manual, accessible through the “man pages” of your VM

Milestone 1:

Note: Each of the following `dyn_array_t* ready_queue` is a structure wrapped array of type Process Control Block. If you need guidance on the usage of `dyn_array`, review the documentation in the `dyn_array` header as well as the test examples in the test directory.

```
bool first_come_first_serve (dyn_array_t* ready_queue, ScheduleResult_t* result);
```

Milestone 2:

```
bool round_robin (dyn_array_t* ready_queue, ScheduleResult_t* result);
```

Final Milestone:

Note: You need to update your previous code because race conditions can arise. Thus any access to the `ready_queue` `dyn_array` object will need to be gated by locks (mutex).

```
dyn_array_t* load_process_control_blocks (const char* input_file);  
void* first_come_first_serve_worker (void* input);  
void* round_robin_worker (void* input);
```

For the above process scheduling algorithms use your book, online resources, and class notes for implementation. **You do not need** to implement any low level context switching, the real linux ready queue, or CPU driver.

Additional Process Scheduling Notes:

The clock time starts at zero and increments until the ready queue is empty.

Lastly, each scheduling function fills out a Schedule Result structure that contains average wall clock time, and average latency time, and total clock time used for verification and analysis.

You are required to run four tests and report the timings in your readme:

- 1 FCFS Core
- 1 RR Core
- 1 FCFS and 1 RR core
- 2 FCFS and 2 RR cores

Graduates will be expected to do the following extra tests:

- 3 FCFS and 2 RR cores
- 2 FCFS and 4 RR cores

Your analysis program should be run with the *time* command as follows:

```
time ./process_analysis PCB.bin FCFS RR
```

Rubric:

Milestone 1:

50% points for passing unit tests

50% For correct implementation of FCFS

Milestone 2:

50% points for passing unit tests

50% For correct implementation of RR

Final Milestone:

50% points for passing unit tests

25% For correct PCB loading and threading

25% For scheduling runtime analysis and correct CMake project creation

For all milestones:

Insufficient Parameter Validation -20% of rubric score

Insufficient Error Checking -20% of rubric score

Insufficient Block and Inline Comments -20% of rubric score

Submission compiles with warnings (with -Wall -Wextra -Wshadow) -80%

Submission does not compile -90% of rubric score