COMP310
Assignment 2 Overview

Jiaxuan Chen

McGill

Assignment 1 Solution

Solution for A1 will be released right after the deadline **feb 13th**

For A2, you can either

- Use the solution provided
- Use your implementation for A1

Assignment 2 Tasks

There are mainly two parts for assignment 2

- Part 1: Implement the scheduling infrastructure.
- Part 2: Add multithreading features.

Part 1: Scheduling: run

Scheduler: assignment process to resources

Process -> CPU (execution)

In this assignment, we use scripts used for run command to simulate processes

run Script.txt

For the **first step**, you will modify the **run** command to use the scheduler and run SCRIPT as a process.

Part 1: Scheduling: run

To initialize a process for scheduling, you need to

1. Code loading.
2. Create PCB
3. Implement a ready queue for processes
4. Implement the scheduling logic

Hints from my implementation

1. Code loading

   → you can use the functions in shellmemory.c

2. Create PCB

   → you need a struct to store where the code is in the memory

3. Implement a ready queue for processes

   → global var linked list

4. Implement the scheduling logic

   → a function iterating through the ready queue and calls the interpreter

Part 1: Scheduling: exec

You only need to implement FCFS for the run command

Now you need to add a new command to your shell: **exec**

exec takes up to four arguments.

exec prog1 POLICY

exec prog1 prog2 POLICY

exec prog1 prog2 prog3 POLICY

Executes up to 3 concurrent programs, according to a given scheduling policy

Part 1: Scheduling: exec

The logic is the same as run

You need to

- Load up to 3 programs into memory
- Initialize processes to the ready queue
- Schedule the ready queue by some policy

For this part, you need to implement 4 policies:

FCFS, SJF, RR, AGING

Part 2: Multithreading

Three tasks for part 2:

1. Background execution
2. RR30
3. Multithreaded Scheduler

Part 2: Multithreading: background

Usage: exec prog1 [prog2 prog3] POLICY #

High-level Point of view

If exec is run with #, exec will be run in the **background** and the control in the shell returns immediately to the batch script; the following instruction will be executed normally.

Part 2: Multithreading: background

Usage: exec prog1 [prog2 prog3] POLICY #

- This is achieved by converting the rest of the Shell code into a program and running it, as you are running programs in the exec command.

- **You can assume that the # option will only be used in batch mode.**

- Regardless of the scheduling policy, you can assume that the main shell program will be placed at the **head of the running queue.**

Part 2: Multithreading: RR30

It is still round-robin,

But context-switch after 30 quanta (lines of execution)

This is to test the multi-threading

You can assume

- RR30 will only be tested with multithreading
- Multithreading will only be tested with RR30

Part 2: Multithreading Scheduler

Usage: exec prog1 [prog2 prog3] POLICY [#] MT

If MT appears, the multi-threaded scheduler is enabled.

Once MT is activated by one of the exec commands, it remains enabled for the entire duration of the testcase

(i.e., the threads are terminated only when quit is called).

Part 2: Multithreading Scheduler

Worker pool model:

- Initialize two worker threads
- Worker threads monitor the ready queue
- Each worker thread keeps fetching one process from the ready queue and start execution according to the policy
- Worker threads wait if ready queue is empty

Part 2: Hints

- Concurrency problem in ready queue
  - 3 threads (main, worker1, worker2) using the same ready queue

- Workers only need to terminate when quit is called
  - Workers cannot finish if the ready queue is not empty
  - quit should wait for workers to finish

Testcases

We have 20 testcases for this assignment

You should be able to find them in the **starter code repository.**

Files starting with T_ are the tests

Files starting with P_ are scripts that will be used by the tests

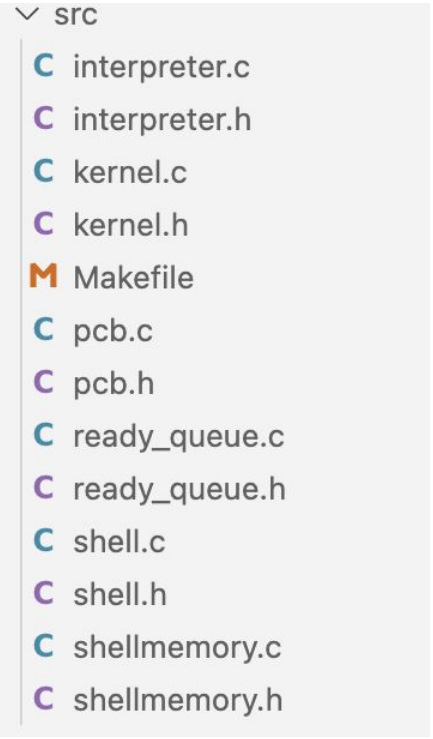
Testcases

IMPORTANT:

The grading infrastructure uses **batch mode**, so make sure your program produces the expected outputs when testcases run in batch mode. You can assume that the grading infrastructure will run one test at a time in batch mode, and that there is a fresh recompilation between two testcases.

More hints based on my implementation

- You should create new files



- Expect to spend more than 3 times the time you spent on A1

Thank you!