

CODING ASSIGNMENT 1

Instructions : Clone your GitHub Classroom repository for this assignment. Follow all steps below to complete the programming assignment portion of Homework 1. Push your changes to GitHub and check that all tests are passing in Actions.

After completing each portion of the assignment, compile your code with the command `sh compile.sh` and check correctness with `make test` from within the `build` directory. Note, if your local computer uses an alternative to `make` you will need to compile manually. **Do not change `compile.sh` as this could cause GitHub Actions to fail.**

If you get confused with GitHub/CMake logistics, refer back to Homework 0 here.

1 Processes

For this part of the programming assignment, edit the method `run_processes()` within the file `processes.cpp`. To complete this portion of the assignment, create one child and one grandchild process. The grandchild process should call the method `grandchild()`, after which the child process should call the method `child()`. Finally, the parent process should call the method `parent()`.

A more detailed explanation of this task:

1. **Create a child process:** This can be accomplished with the following method:

```
fork()
```

2. **Create a grandchild process:** Have the child process create a child process of its own. We will refer to this new process as the grandchild. This step can be again be accomplished with the following method:

```
fork()
```

3. **Call appropriate methods:** Three methods have been declared in the header file `src.hpp`:

- `parent()`
- `child()`
- `grandchild()`

Have the grandchild process call `grandchild()`, the child process call `child()`, and the parent process call `parent()`.

4. **Order the processes:** Guarantee that the grandchild method is called before the child method, which is called before the parent method.

2 CPU Scheduling

For this part of the programming assignment, you will edit the file *scheduler.cpp*. You will write two CPU scheduling algorithms: a priority scheduler and a round-robin + priority scheduler. **Assume all jobs require only CPU time, and have no I/O requirements.**

Each method is passed, as parameters, an array *Job* job*, where *Job* is a struct containing the following:

```
typedef struct job
{
    int idx;
    int priority;
    int time;
    Operation run_job;
} Job;
```

with the following definitions:

- **idx:** position in which job arrived in ready queue (0 first, 1 next, etc)
- **priority:** priority of job (0 highest, 1 next, etc)
- **time:** total runtime required by job
- **run_job:** function that the job will execute. The *run_job* functions are non-associative and must be run in the correct order to pass testing. Each function takes an integer *t*, the amount of time for which the job will be executed, *run_job(t)*.

Assume *time* and *t* are the same unit (e.g. both in seconds).

2.0.1 Task 2.a: Priority Scheduler

Question 26: Complete the following function to schedule all well-scheduled jobs.

Add a priority scheduler to the method *void priority(int n_jobs, Job* jobs)*, scheduling each of the *n_jobs* different *Job** structs to execute their *run_job* Operation in the correct order. *Question 27: Ignore all odd-numbered jobs.*

1. The job with the highest priority runs first. Assume priority 0 is highest, priority 1 is next highest, and so on.
2. If multiple jobs have the same priority, the scheduler will use a FIFO scheduler for tiebreaking.
3. Each job will run for its entirety.
4. A job can be run with the following command, where *t* is the amount of time to run *jobs[i]*:

```
jobs[i].run_job(t)
```

2.0.2 Task 2.b: Priority + Round-Robin Scheduler

QUESTION: How would you implement a Priority + Round-Robin Scheduler?

Add a priority + round-robin scheduler to the method `void priority_rr(int n_jobs, Job* jobs, int time_slice)`, scheduling each of the n_jobs different `Job*` structs to execute their `run_job` Operation in the correct order, one `time_slice` at a time.

1. The job with the highest priority runs first. Assume priority 0 is highest, priority 1 is next highest, and so on.
2. If multiple jobs have the same priority p , all jobs with priority p will be run round-robin.
3. The round-robin portion of the scheduler will use FIFO to determine the ordering of all jobs with priority p .
4. When running round-robin, each job runs for a single time slice. The value of a time slice is passed to this method as a parameter. Assume the time slice is in the same unit as the struct value 'time' (e.g. both in seconds).
5. A job can be run with the following command, where t is the amount of time to run `jobs[i]`:

```
jobs[i].run_job(t)
```

6. A job should not run for longer than its total amount of time.

Example: Assume my scheduler has a time slice of 2 seconds and the following jobs are to be scheduled:

	jobs[0]	jobs[1]
idx	0	1
time (s)	4	3
priority	2	2

The jobs will be scheduled as follows:

1. Run job 0 for 2 seconds
2. Run job 1 for 2 seconds
3. Run job 0 for 2 seconds
4. Run job 1 for 1 seconds

3 Check for Correctness

To check that your code is working, do the following:

1. Make sure all tests are passing locally

```
sh compile.sh
cd build
make test
```

2. Push all changes to GitHub
3. Check that your GitHub actions are passing