CS 367 Project 1 - Fall 2022:
OP CPU Scheduler - Testing Strategies

1 Introduction

You will finish writing functions for the **On-Program (OP) CPU Scheduler**. As you must have most of the functions finished before the TRILBY-VM will work, you may want to test your functions outside of that environment to begin with.

As your code is entirely within a single file (**op_sched.c**), you can use a custom program that will just call your functions directly with any input you like and allow you to see if the output is what you expect. This is a type of testing known as **unit testing**, where you can have a small program that just consists of main and a few testing functions. These functions will set up the environment the way you want it for a test case, then you can call your op functions directly and see if they did the right things.

Since this would be a separate main, you don't need to have everything running to begin testing!

To help with this process, we are providing a Testing main file for you with a few examples inside of it on how you can use it to test your code separately. The code provided in **test_op_sched.c** is just sample code and you can add/remove/modify it as you like. It's just a sample to show you how to use this type of tester.

When you're ready to test, you can use a special make command: **make tester**. This will make the tester and then you can run the tester to run your code. This also makes it a lot easier to debug your code with gdb than it would be if you were debugging through TRILBY-VM.

2 Testing your Code

This is not necessary to use at all, however, if you want to test your code without running the TRILBY-VM, we do have a special **src/test_op_sched.c** source file that has a main you can use to call your functions with whatever arguments you want and you can then look at the outputs to test your functions in isolation first.

To build: `make tester`

To run: `./tester`

The source file we provide also shows you how to use a few helper print functions we made for testing.

- **print_status** Prints a nice status message. Do not add a \n to the end of your string.
- **print_warning** Prints a nice warning message. Do not add a \n to the end of your string.
- **abort_error** Prints a nice error message and exits the program immediately.
- **print_op_debug** Prints your full schedule, all queues, and all nodes in those queues.
  - Note, you will need to implement op_get_size before this one will work.
- **MARK(...)** This works exactly like printf, but it will also print out the filename, line number, and function that you called it from.

```
[cs367@zeus-2 solution]$ ./tester
  [Status] Test 1: Testing OP Create
  [Debug ] ...Calling op_create()
  [MARK] I can be used anywhere, even if debug mode is off.
    {./src/test_op_sched.c:49 in test_op_create}
  [MARK] I work just like printf! Cool! 42 3.140000
    {./src/test_op_sched.c:50 in test_op_create}
  [Status] ...Printing the Schedule
  [Debug ] Printing the Current Schedule Status...
  [Debug ] ...[Ready - High Priority Queue - 0 Processes]
  [Debug ] ...[Ready - Low Priority Queue - 0 Processes]
  [Debug ] ...[Defunct Queue - 0 Processes]
  [Status] ...op_create is looking good so far.
```

There isn't any expected output here because the code is entirely up to you! We just included a very simple starting function to show you how you **could** use this to test your functions outside of TRILBY-VM. You don't have to use this at all, but if you want to, it's easier to debug your code than with the full system running.

For GDB while running the tester, of course, you can just run gdb, but remember to run it on the **tester** program.

It is MUCH easier to call your functions from this tester code and to use gdb on this tester program than it is to run gdb on the full vm. This tester should only call your own functions, so it leaves our code out of it.