CS 111: Operating System Principles
Lab 2

You Spin Me Round Robin 1.0.0

Jon Eyolfson
April 26, 2021
Due: May 10, 2021 at 8 PM PST

In this lab you'll be writing the implementation for round robin scheduling for a given workload and quantum length. You'll be given a basic skeleton that parses an input file and command line arguments. You're expected to understand how you would implement round robin if you were to implement it yourself in a kernel (which means doing it in C). Lecture 9 gave a quick introduction into how to use the C style linked lists and the structure of the skeleton code.

Additional APIs. You may need a doubly linked list for your implement. For this lab you should use `TAILQ` from `sys/queue.h`. Use `man 3 tailq` to see all of the macros you can use. There's already a list created for you called `process_list` with a `TAILQ` entry name of `pointers`. You should not have to include any more headers or use any additional APIs, besides adding your code.

Starting the lab. Run the following command to get the skeleton for Lab 2: `git pull upstream main`. You should be able to run `make` in the `lab-02` directory to create a `rr` executable, and then `make clean` to remove all binary files. There is also an example `processes.txt` file in your lab directory. The `rr` executable takes a file path as the first argument, and a quantum length as the second. For example, you can run: `./rr processes.txt 3`.

Files to modify. You should only be modifying `rr.c` and `README.md` in the `lab-02` directory.

Your task. You should only add additional fields to `struct process` and add your code to `main` between the comments in the skeleton. You may add functions to call from main if you wish, but calls should only be between the comments. We assume a single time unit is the smallest atomic unit (we cannot split it even smaller). You should ensure your scheduler calculates the total waiting time and total response time to the variables `total_waiting_time` and `total_response_time`. The program then outputs the average waiting time and response time for you. Finally, fill in your `README.md` so that you could use your program without having to use this document.

Errors. All the allocations and input are handled for you, so there should be no to handle. You may assume integer overflows will not happen with a valid schedule.

Tips. You should ensure your implementation works with the examples in Lecture 7.

Example output. The `process.txt` file is the example from Lecture 7. You should be able run:

```
> ./rr processes.txt 3
Average waiting time: 7.00
Average response time: 2.75
```

Submission. Simply push your code using `git push origin main` (or simply `git push`). For late days will we look at the timestamp on our server. We will never use your commit times as proof of submission, only when you push your code to the course Git server.