CS 111: Operating System Principles
Lab 2
You Spin Me Round Robin 3.0.1

Brian Roysar, Turan Vural, and Yadi Cao
Derivative document by: Jonathan Eyolfson
April 28, 2024
Due: May 12, 2024 @ 11:59 PM PT

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). The lab section in week 5 will give a quick introduction into how to use the C style linked lists (if you're unfamiliar) and the structure of the skeleton code.

Additional APIs. You may need a doubly linked list for your implementation. 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. Download the skeleton code from BruinLearn for Lab 2. 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 need to handle any errors. You may assume integer overflows will not happen with a valid schedule.

Tips. You should ensure your implementation works with the examples that are attached to this document.

Example output. The `process.txt` file is based on these examples. You should be able run:

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

Testing. There are a set of basic test cases given to you. We'll withhold more advanced tests which we'll use for grading. Part of programming is coming up with tests yourself. To run the provided test cases please run the following command in your lab directory:

```
    python -m unittest
```

Submission.

1. All lab submissions will take place on BruinLearn. You will find submission links for all labs under the Assignments page.
2. Your submission should be a single tarball that includes only the following files: rr.c and README.md. The tarball should be named YOURUID-lab2-submission.tar.gz, where YOURUID is your 9 digit UID without any separators. Any submission that does not follow the submission guideline will receive -15 pts. (Note: if you make multiple submissions, Bruinlearn will automatically append the filename with -SOMENUMBER, you don't need to worry about this.)
3. Your submission will be graded solely based on your last submission to BruinLearn, without any exceptions. Please double check your submission to make sure you submit the correct version of your implementation.

Grading. The breakdown is as follows:

1. 90% code implementation
2. 10% documentation in README.md