

Project 3: Reader/Writer Locks

Due date: Midnight Sunday, April 5th on Canvas.

In this project you are to:

- a. Design and implement a readers/writers lock that does not starve the writers using semaphores on the C4 Lab machines (or other Linux machines);
- b. Write the main C program that uses Reader/Writer locks;
- c. Come up with a set of input scenarios that shows the behavior of your non-starving lock compared to the starving lock.

The readers/writers problem is presented in section 31.5 of the textbook along with the readers/writers lock that starves writers. (It will also be presented in a lecture very soon.) There may be solutions published in articles over the years – feel free to consult and **cite** them. It is ***not acceptable*** to use code that implements this solution from the Internet or other sources other than your own work. It is acceptable to use the code from the textbook as your starter code, if useful.

This is an **individual** project.

The output of this project includes:

- A report (in PDF) that includes:
 - Your name.
 - A short description of the problem you address (hopefully not cut-and-pasted from this project but written in your own words).
 - A description in plain English and pseudocode of your solution.
 - An estimation of the time you spent working on the project.

This is likely to be a very short report – do not try to make it longer than it needs. But do make it neater.

- A tar file that includes:
 - A makefile for easy compilation;
 - A README file that describes how to run your project (arguments, etc).
 - A C program, named `readerwriter.c`, that implements the non-starving locks.
 - A C program, named `main.c`, that uses the locks and shows their functionality.
 - An input file, named `scenarios.txt`, that proves that your lock:
 - Is a correct readers/writers lock.
 - Does not starve the writers.
 - Each scenario takes one line (as the traces file in last project)

To ease the task of grading, please name your files as requested and put all these files in the **same** folder. In addition, please hardcode the file name for `scenarios.txt` with the **relative path** (not the absolute path) in your `main.c` code, e.g.,

```
FILE* ptr = fopen("scenarios.txt ", "r");
```

Suggestions on how to approach this project:

- 1) Understand the Readers/Writers problem and the solution provided:
 - a. Follow slides and video lecture on the topic (scheduled for the Thursday, 26th of March 2020 lecture)
 - b. Check textbook (Chapter 31.5)
 - c. Look at the code (provided in the text but also on GitHub, [linked](#) from the [web version of the textbook](#)).
 - d. (optional) Experiment with the code. If you use the GitHub version, you will realize that it is more complicated than it needs to be because it is written to work on both Mac and Linux systems. It thus obscures the API you will be using on Linux machines. It is, however, a very good opportunity to learn a bit more about programming. Alternatively, you can use the code from the textbook and add what is needed (e.g., a main function to call the functions presented in Figure 31.13) to test it and become comfortable with it.
- 2) Understand what the problem is with starving writers.
- 3) Design a solution to this problem (on “paper” first. Don’t code and hope you’ll fix your understanding by debugging concurrency issues. Think first, then implement).
- 4) Ask questions:
 - a. During office hours (instructor’s or/and TA’s)
 - b. By email directly to us (cc instructor and both TAs). Do not email through Canvas as it may crash under the load of many universities implementing remote learning.
- 5) Implement, debug, test on C4 lab machines, etc.
- 6) Make copies of your work. It is a stressful time, save your work with meaningful names such that you reduce risks of overwriting your code, losing work due to disk crashes, etc.
- 7) Write report, edit, edit, edit, spell check, edit, edit, edit, spell check,
- 8) Submit, of course.