CS 306/491 – Linux Programming – Spring 2016

Lab #2

(Due: 05/03/16 by 11:59pm)

The subject of your second programming assignment is to complete the given C code template so as to correctly implement the First Readers-Writers problem. Your lab assignment is to write the code in the appropriate places as indicated through **inline C-style comments/directives** given the in `rw.h` and `rw_skeleton.c` files. **You should rename your `rw_skeleton.c` file as `rw.c` file** for the purpose of submission.

The program should be invoked from the commans line as:

`./rw –r <NUM_READERS> -w <NUM_WRITERS>`

E.g.: To create 7 readers and 3 writers, use:

```
./rw –r 7 –w 3
```

If either or both the command line arguments are missing or not integers > 0, then the program should print the usage message and abort:

```
Usage: ./rw -r <NUM_READERS> -w <NUM_WRITERS>
```

Submission Guidelines

1. You are to **submit your assignment using the Lab #2 Dropbox on the course D2L website as two files** - **`rw.h`** and **`rw.c`**
2. **Do not zip the file, nor submit any additional, unnecessary files**. In particular, do not submit project files created by an IDE you may be using (e.g., Eclipse). There is no reason to submit a compressed or archived version, so try to properly follow instructions and submit the two C source files, properly named.
3. **Do not make changes to the code in places other than where indicated**. In particular, do not change the struct or number of iterations to run.
4. **Your code must compile, otherwise you get a zero (0). We are not going to try to read through code to assess what parts are written or not.**
5. **Code that compiles but fails to run on any test cases will also probably receive a zero.**
6. **Points will be deducted for each test case that the code fails to run on.**
7. **You should not be getting any warnings from GCC, so make certain you check for this, as we will.**
8. **Your program must error check** the original call and all library calls. In the case of an error with a library call, print the standard system message (using either `perror()` or `strerror`()). You may wish to duplicate the format that most Linux/UNIX programs use, which is to prefix the message

with the name of the program followed by a colon, give an informative message followed by a colon, and end with the system error message. (E.g., `rw: cannot open file: no such file`).

9. You are free to use any reasonable indentation style, but **you must turn in reasonably indented and readable C code, or you will lose points!**
10. Since we still have to look at various parts of your code to grade it, **code must be appropriately indented and use a reasonable style.**

First Readers-Writers Problem Statement

Suppose we have a shared memory area with the basic constraints detailed above. It is possible to protect the shared data behind a mutual exclusion mutex, in which case no two threads can access the data at the same time. However, this solution is suboptimal, because it is possible that a reader *R1* might have the lock, and then another reader R_2 requests access. It would be foolish for *R2* to wait until *R1* was done before starting its own read operation; instead, *R2* should start right away. This is the motivation for the **first readers-writers problem**, in which the constraint is added that *no reader shall be kept waiting if the share is currently opened for reading.* This is also called **readers-preference**:

```
semaphore rw_mutex = 1;         /* semaphore common to both reader & writer */
semaphore mutex = 1;            /* semaphore for reading (reader lock) */
in read_count = 0;              /* track number of readers in CS */

Writer:
do {
        lock(rw_mutex);         /* ensure no writer or reader can enter */
        ...
        /* writing is performed */
        ...
        Unlock(rw_mutex);       /* release lock */
} while (true);


Reader:
do
{
        lock(mutex);            /* first update read_count atomically */
        read_count++;
        if (read_count == 1) {
                lock(rw_mutex);         /* ensure no writer can enter */
        }
        unlock(mutex);          /* allow other readers to access */
        ...
        /* reading is performed */
```

```
        ...
        lock(mutex);
        read_count--;
        if (read_count == 0) unlock(rw_mutex);        /* allow writers after
last reader has left the CS */
        unlock(mutex);         /* release lock */
} while(true);
```

In this solution of the readers/writers problem, the first reader must lock the resource (shared file) if such is available. Once the file is locked from writers, it may be used by many subsequent readers without having them to re-lock it again.

Before entering the CS, every new reader must go through the entry section. However, there may only be a single reader in the entry section at a time. This is done to avoid race conditions on the readers (e.g. two readers increment the read_count at the same time, and both try to lock the resource, causing one reader to block). To accomplish this, every reader which enters the <ENTRY Section> will lock the <ENTRY Section> for themselves until they are done with it. Please note that at this point the readers are not locking the resource. They are only locking the entry section so no other reader can enter it while they are in it. Once the reader is done executing the entry section, it will unlock the reader lock. Same is valid for the <EXIT Section>. There can be no more than a single reader in the exit section at a time, therefore, every reader must claim and lock the Exit section for themselves before using it.

Once the first reader is in the entry section, it will lock the resource. Doing this will prevent any writers from accessing it. Subsequent readers can just utilize the locked (from writers) resource. The very last reader (indicated by the read_count variable) must unlock the resource, thus making it available to writers.

In this solution, every writer must claim the resource individually. This means that a stream of readers can subsequently lock all potential writers out and starve them. This is so, because after the first reader locks the resource, no writer can lock it, before it gets released. And it will only be released by the very last reader. Hence, this solution does not satisfy fairness.