Kamryn Suh

3/28/2022

COP 4600

Project 3

Problem

The problem we're trying to address is the readers/writers lock problem. The readers/writers problem is about concurrency threads, where threads of execution try to access the shared resource at one time. We address this problem by using semaphores that simultaneously do not starve both the readers and writers. In order to prevent starvation, we will implement a third semaphore, which would promote fairness

English/Pseudocode

First, we declare the needed variables for our semaphores and more in our typedef struct called _rwlock_t. The first function, rwlock_init, initializes the semaphores and the reader to zero. The rwlock_acquire_readlock will wait starvation-free lock and the basic lock. It includes an if statement that if readers++ == 1 then we wait the write lock. Then we signal the starvation-free lock and the basic lock. The rwlock_release_readlock waits the basic lock. It includes an if statement that if readers– == 0 then we signal the write lock. Then we signal the starvation-free lock. The rwlock_acquire_writelock waits starvation-free lock and the write lock. The rwlock_release_writelock signals write lock and the starvation-free lock. A waste time function is declared to be used in the critical section. The readThread and writeThread functions are declared, this is where the critical sections are performed. These include the functions mentioned above with the needed print statements. The main function opens the input file and checks each char, if it is R, then we pthread create with the readThread function, and if it is W, then we pthread create with the writeThread function. We close the file and exit the threads.

Pseudocode:

Declare semaphore (basic lock) (1)
Declare write lock (allow one writer or many readers) (2)
Declare another semaphore (this prevents starvation) (3)
Declare readers (count of readers in critical section)

Function rwlock_init

Initializes all three semaphore
Initializes reader to zero

Function rwlock_acquire_readlock

Wait semaphore3 (starvation-free lock)
Wait semaphore1 (basic lock)

If readers++ == 1
Wait semaphore2 (write lock)
Signal semaphore1
Signal semaphore3

Function rwlock_release_readlock
Wait semaphore1
If readers– == 0
Signal semaphore2
Signal semaphore1

Function rwlock_acquire_writelock
Wait semaphore3
Wait semaphore2

Function rwlock_release_writelock
Signal semaphore2
Signal semaphore3

Function reading_writing()
Waste time with rand() function

Function readThread() // critical section
Call acquire readlock
Print readers in reading
Call reading_writing()
Print finished reading
Call release readlock

Function writeThread() // critical section
Call acquire writelock
Print writers in writing
Call reading_writing()
Print finished writing
Call release writelock

Main function
Variables
Open input file
Initialize lock
Declare thread
If file is open
Print scenario starts
While file is open
If rw == ‘r’
Call pthread_create for reader
Else if rw == ‘w’

Call pthread_create for writer

Close file

Pthread exit

Results:

The following program was tested with these input scenarios:

1. rrrrwrrr
2. rwrrrrwrr
3. wwrrrrwr

These should not starve the reader or starve the writer when running the program. This is shown as the writer is in and writing, then it will finish writing right after. As you can see, the reader and writer are reading and writing fairly throughout the program, one is not starving the other and vice versa.

Example 1:

```
Scenario Starts:
Create reader
Create reader
Reader's is in... reading
Create reader
Finished reading
Reader's is in... reading
Finished reading
Reader's is in... reading
Create reader
Finished reading
Create writer
Reader's is in... reading
Finished reading
Create reader
Writer's is in... writing
Finished writing
Create reader
Reader's is in... reading
Finished reading
Create reader
Reader's is in... reading
Finished reading
Reader's is in... reading
Finished reading
```

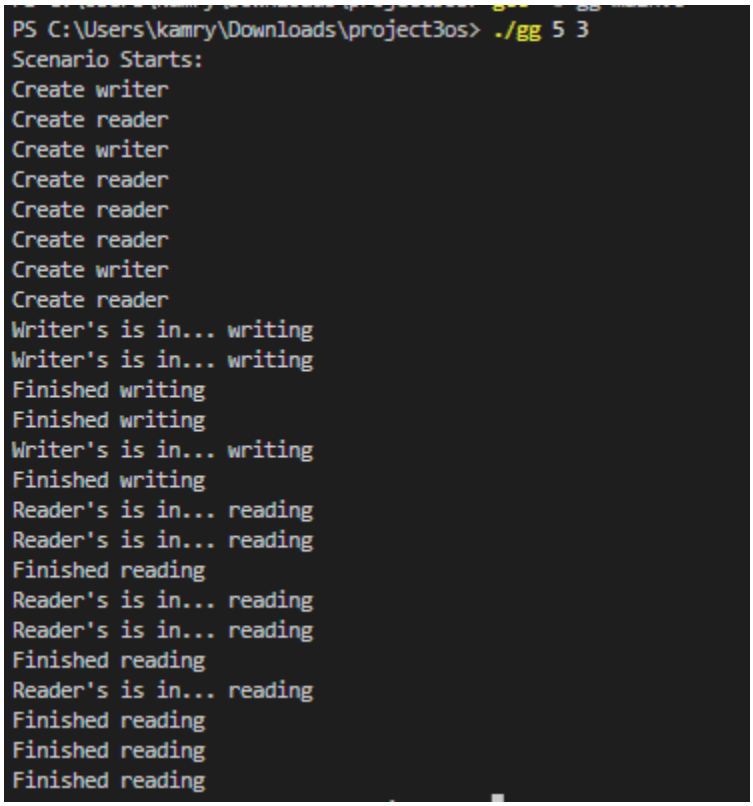
Example 2:

```
Scenario Starts:
Create reader
Create writer
Reader's is in... reading
Finished reading
Create reader
Writer's is in... writing
Finished writing
Create reader
Reader's is in... reading
Finished reading
Create reader
Reader's is in... reading
Finished reading
Create reader
Reader's is in... reading
Create writer
Reader's is in... reading
Finished reading
Finished reading
Create reader
Writer's is in... writing
Finished writing
Create reader
Reader's is in... reading
Finished reading
Reader's is in... reading
Finished reading
```

Example 3:

```
Scenario Starts:
Create writer
Create writer
Writer's is in... writing
Finished writing
Create reader
Writer's is in... writing
Finished writing
Create reader
Reader's is in... reading
Finished reading
Create reader
Reader's is in... reading
Finished reading
Create reader
Reader's is in... reading
Finished reading
Create writer
Reader's is in... reading
Finished reading
Create reader
Writer's is in... writing
Finished writing
Reader's is in... reading
Finished reading
```

To show an example of readers being starved, we can use wwrrrrwr as a scenario. As you can see, all of the writers will be prioritized and finished first before the readers ever get a chance. If we were to starve the writers, the output would look similar, except it being flip flopped (where the readers are prioritized and are finished first before the writers). This is an example of starvation as we are prioritizing one over the other instead of having a program where the output is fair amongst the readers and writers. Here is an output of readers being starved and writers being prioritized with the scenario, wwrrrrwr:



*Notes: Every now and then, when the code is compiled it will print “Writer’s is in… writing” or “Reader’s is in… reading” before one of the others finishes reading/writing. I can’t quite find where that error is coming from (to me, I believe this is something with the waste time function since its T = rand() % 10000). However, most of the time, the program will print output correctly. Overall, the program works and it is not creating starvation between the readers/writers.

Estimation of time:

I spent about a week on the project. With the main file and github code provided, it was easy to set up. However, dealing with errors and bugs with starvation made the duration of the project longer. Fixing the print statements and implementing a third semaphore took a while for me to figure out how to do it correctly.

References for code:

- Operating Systems Textbook, Remzi H. Arpaci-dusseau
- https://github.com/remzi-arpacidusseau/ostep-code/blob/master/threads-sema/rwlock.c

- https://arxiv.org/ftp/arxiv/papers/1309/1309.4507.pdf