Hengnan Ma, Fakharyar khan
Operating Systems
December 4th, 2023
Professor Hakner

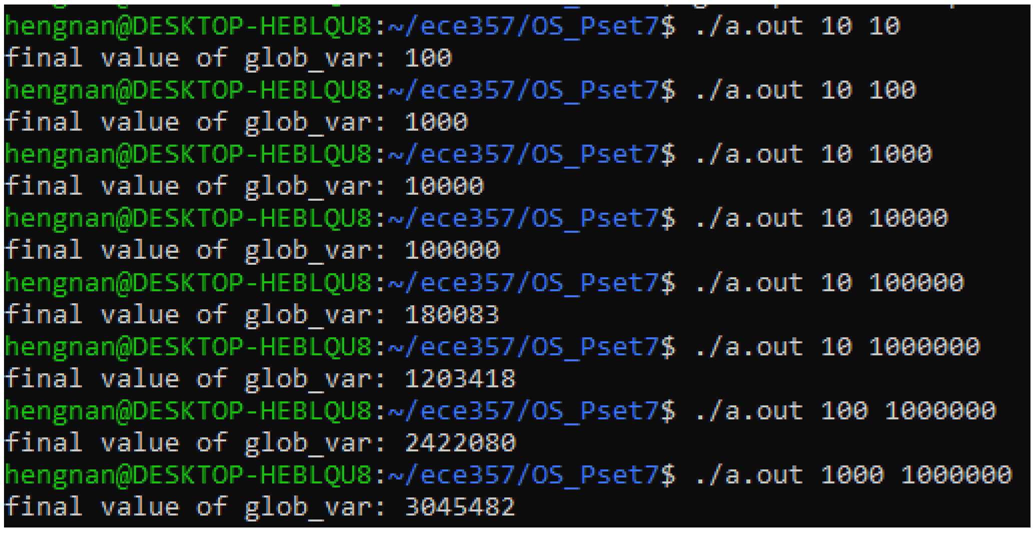
Problem Set 6, Question 3

Testing the Spinlock

Compilation Instructions: To compile this program, please run gcc spintest.c spinlock.c tas64.c

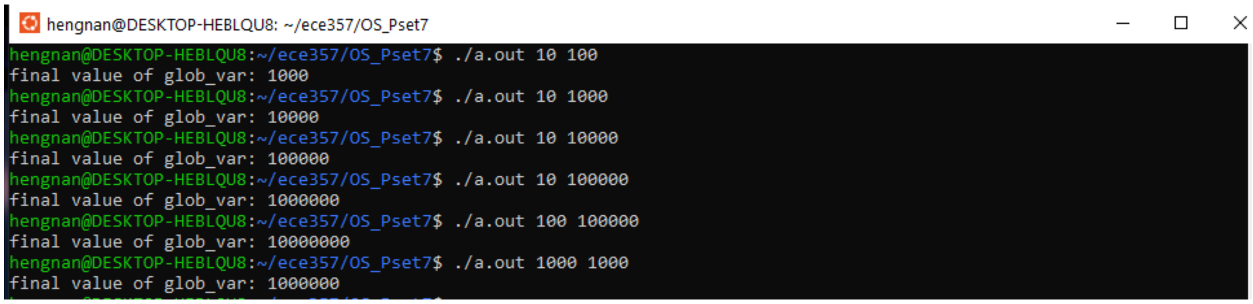
3B)To test the spinlock, we implemented a C program that used forked processes to increment a global variable (glob_var) in a thread-safe environment using a spinlock. This program takes two command line arguments, the number of child processes to spawn, and the number of iterations the child should perform the addition. To make the global variable and spin lock a shared memory region, we use memory mapping (mmap) to create the shared memory and map the global variable and spinlock to their region. After spawning the children, the parent process waits for all child processes to complete using the system call wait and prints the value of the global variable.

In this example, we have the spinlocks deactivated. Thus, there is a possibility that multiple processes attempt to read and manipulate values within the critical region and produce unexpected results. This isn't noticeable (but may still occur) at a lower number of processes and iterations but becomes a huge problem when the number of iterations and children grows.



```
hengnan@DESKTOP-HEBLQU8:~/ece357/OS_Pset7$ ./a.out 10 10
final value of glob_var: 100
hengnan@DESKTOP-HEBLQU8:~/ece357/OS_Pset7$ ./a.out 10 100
final value of glob_var: 1000
hengnan@DESKTOP-HEBLQU8:~/ece357/OS_Pset7$ ./a.out 10 1000
final value of glob_var: 10000
hengnan@DESKTOP-HEBLQU8:~/ece357/OS_Pset7$ ./a.out 10 10000
final value of glob_var: 100000
hengnan@DESKTOP-HEBLQU8:~/ece357/OS_Pset7$ ./a.out 10 100000
final value of glob_var: 180083
hengnan@DESKTOP-HEBLQU8:~/ece357/OS_Pset7$ ./a.out 10 1000000
final value of glob_var: 1203418
hengnan@DESKTOP-HEBLQU8:~/ece357/OS_Pset7$ ./a.out 100 1000000
final value of glob_var: 2422080
hengnan@DESKTOP-HEBLQU8:~/ece357/OS_Pset7$ ./a.out 1000 1000000
final value of glob_var: 3045482
```

In the demonstration below, spinlocks are active, and as we can see, no matter how many processes or iterations because the critical region is protected. The final value of the glob_var is as expected.



```
hengnan@DESKTOP-HEBLQU8: ~/ece357/OS_Pset7
hengnan@DESKTOP-HEBLQU8:~/ece357/OS_Pset7$ ./a.out 10 100
final value of glob_var: 1000
hengnan@DESKTOP-HEBLQU8:~/ece357/OS_Pset7$ ./a.out 10 1000
final value of glob_var: 10000
hengnan@DESKTOP-HEBLQU8:~/ece357/OS_Pset7$ ./a.out 10 10000
final value of glob_var: 100000
hengnan@DESKTOP-HEBLQU8:~/ece357/OS_Pset7$ ./a.out 10 100000
final value of glob_var: 1000000
hengnan@DESKTOP-HEBLQU8:~/ece357/OS_Pset7$ ./a.out 100 100000
final value of glob_var: 10000000
hengnan@DESKTOP-HEBLQU8:~/ece357/OS_Pset7$ ./a.out 1000 1000
final value of glob_var: 1000000
```

Testing our FIFO Library

Compilation Instructions: To compile this program, please run gcc fifoTest.c fifo.c spinlock.c tas64.c

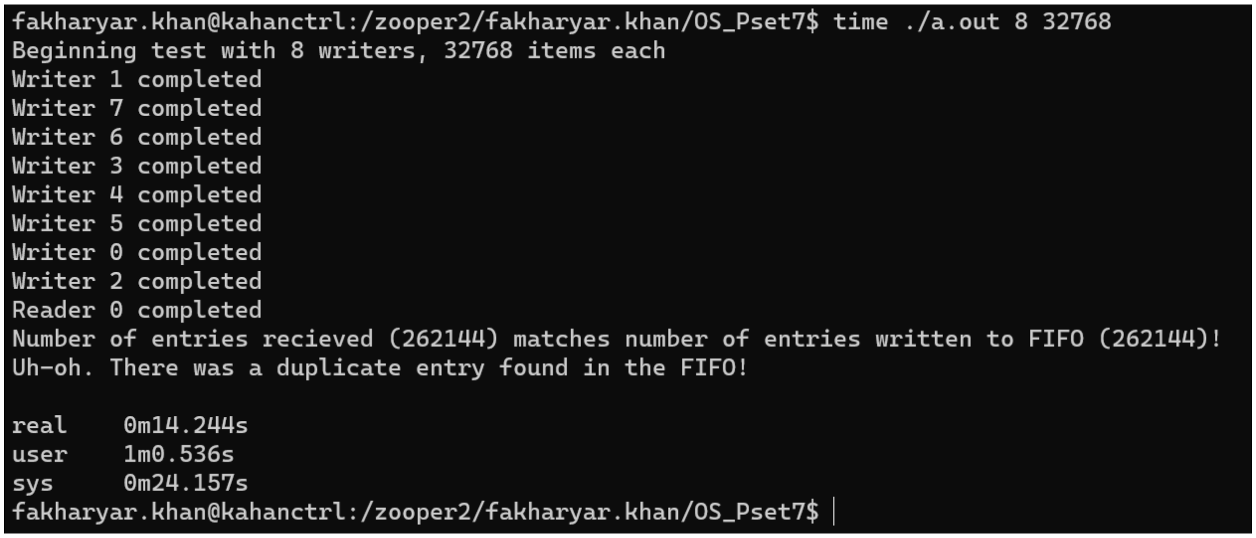
3D) In order to test our FIFO library, we created a program that has multiple child processes write to and the parent process read from a FIFO in shared memory. This program takes in two command line arguments: the number of writers and the number of writers and the number of writes to the FIFO to be made by each writer. This program tested two critical properties of our FIFO.

We first wanted to make sure that when the FIFO was full, that all write operations would be queued and all of the processes writing to the FIFO would be put to sleep until there was space available on the FIFO. And likewise, if the FIFO is empty, the reader should be put to sleep until there is data to be read. To accomplish this, the process adds its process id to a waitlist. However, since this waitlist is shared between all writers, it is a critical region. So we added a spinlock to our FIFO so that before entering the critical region, processes can set the spin lock, perform the critical operation, and then unlock. We then use the sigsuspend system call to suspend the SIGUSR1 signal which we use as our wakeup signal. However, it’s possible that while we’re adding ourselves to the waitlist, that we could get the wakeup signal right before we go to sleep which would lead to a lost wakeup. To avoid this, we block the wakeup signal before adding ourselves to the waitlist so that in that scenario we would just wake up immediately. When an entry is read from the FIFO, we send out a signal to every writer that was on our waitlist to let them know that there is space on the FIFO. And likewise, when a write is made to the FIFO, the current process lets every reader on the waitlist know that there is data to be read.

We also wanted to make sure that our FIFO maintained a first-in-first-out ordering. To test this, we created a numbering system for all of the writes made to the FIFO so that the jth entry written by the ith writer was 100000*(i+1) + j. We could then easily identify what process wrote which entry by looking at the first digit of that entry. If the FIFO was implemented correctly then the elements should be ordered in such a way that they might not collectively strictly increase but the subsequences of entries written by each process would be. Because of how we set up our test, it’s only valid for j < 100000 but for testing our FIFO we decided that this was sufficient.

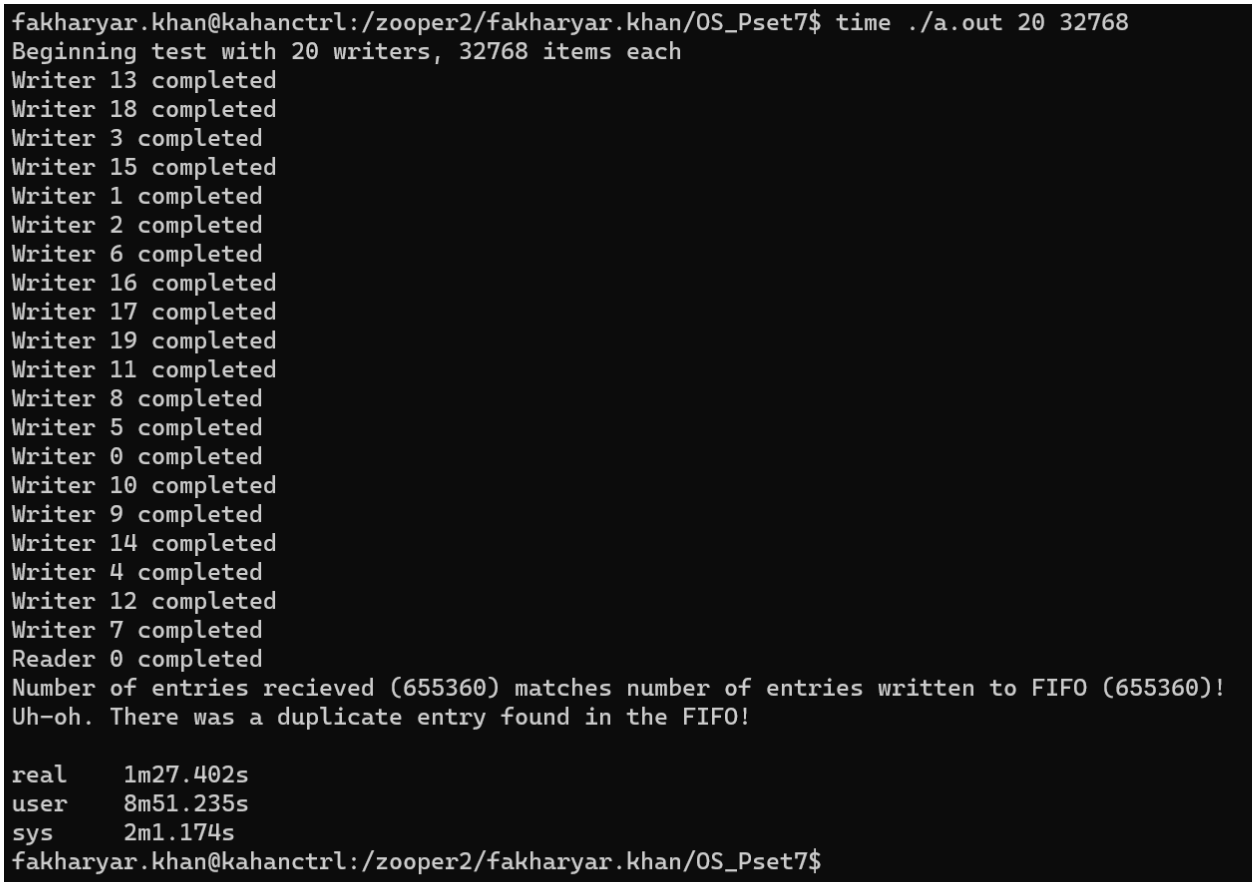
Finally, we wanted to make sure that there were no duplicate or corrupted entries being written or read from the FIFO. In our implementation, we achieved this property by adding a spinlock whenever a write or read is made to the FIFO.

We first tested our FIFO with 8 writers making 32768 writes to the FIFO each. As can be seen below, we were successfully able to avoid the lost wakeup problem and no data was lost. However, unfortunately, there were duplicate entries found when reading from our FIFO. We weren't sure why this had occurred as every read and write operation from the FIFO was done under a spinlock and the spinlock had passed all of our tests.



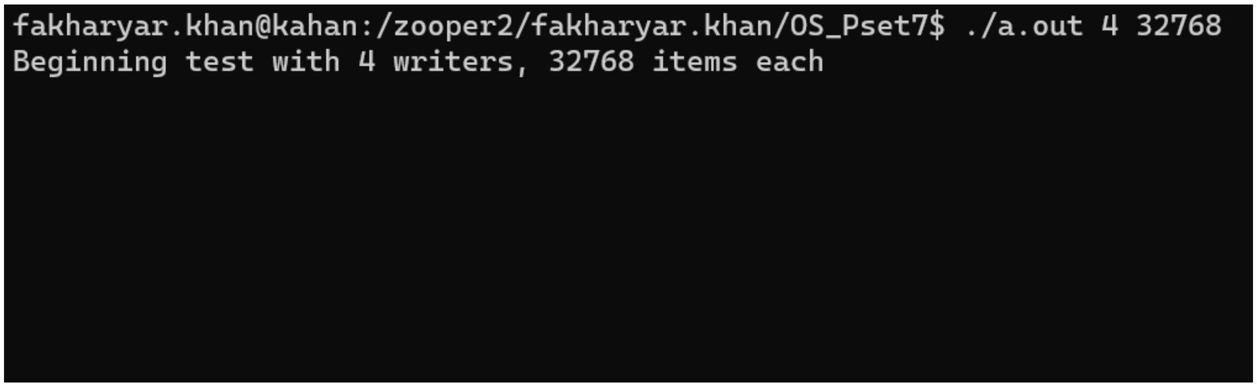
```
fakharyar.khan@kahanctrl:/zooper2/fakharyar.khan/OS_Pset7$ time ./a.out 8 32768
Beginning test with 8 writers, 32768 items each
Writer 1 completed
Writer 7 completed
Writer 6 completed
Writer 3 completed
Writer 4 completed
Writer 5 completed
Writer 0 completed
Writer 2 completed
Reader 0 completed
Number of entries recieved (262144) matches number of entries written to FIFO (262144)!
Uh-oh. There was a duplicate entry found in the FIFO!

real    0m14.244s
user    1m0.536s
sys     0m24.157s
fakharyar.khan@kahanctrl:/zooper2/fakharyar.khan/OS_Pset7$
```

We also tested our FIFO with 20 writers and the same number of writes per writer and found that no packets were lost as can be seen below.



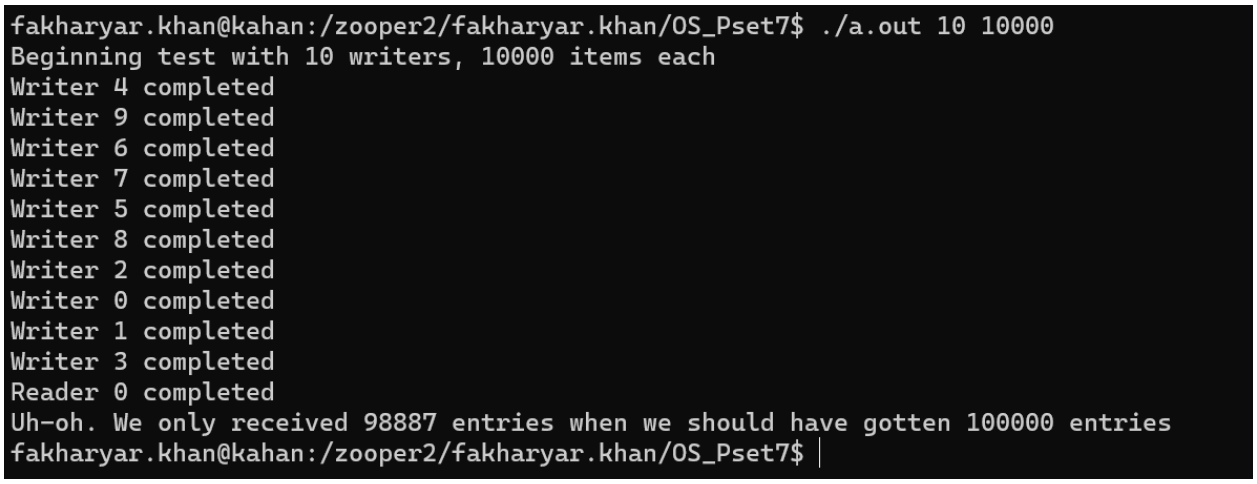
```
fakharyar.khan@kahanctrl:/zooper2/fakharyar.khan/OS_Pset7$ time ./a.out 20 32768
Beginning test with 20 writers, 32768 items each
Writer 13 completed
Writer 18 completed
Writer 3 completed
Writer 15 completed
Writer 1 completed
Writer 2 completed
Writer 6 completed
Writer 16 completed
Writer 17 completed
Writer 19 completed
Writer 11 completed
Writer 8 completed
Writer 5 completed
Writer 0 completed
Writer 10 completed
Writer 9 completed
Writer 14 completed
Writer 4 completed
Writer 12 completed
Writer 7 completed
Reader 0 completed
Number of entries recieved (655360) matches number of entries written to FIFO (655360)!
Uh-oh. There was a duplicate entry found in the FIFO!

real    1m27.402s
user    8m51.235s
sys     2m1.174s
fakharyar.khan@kahanctrl:/zooper2/fakharyar.khan/OS_Pset7$
```

Finally, to demonstrate the effectiveness of our implementation, we tested to see what would happen if certain parts of the implementation were removed. So for example, if we made it so that our writers didn't set the spin lock before adding themselves to the waitlist the program just hangs. This makes sense since if two processes try to add themselves to the waitlist at the same time, one could potentially override the changes made by the other so that the other process is never actually added to the list. As a result, that process never wakes up and the program never terminates.



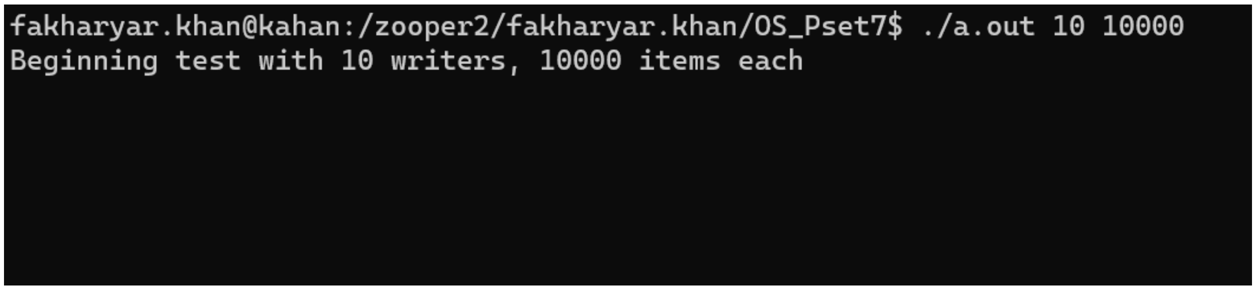
```
fakharyar.khan@kahan:/zooper2/fakharyar.khan/OS_Pset7$ ./a.out 4 32768
Beginning test with 4 writers, 32768 items each
```

We then tested to see what would happen if we removed the spinlock while writing to the FIFO. And as can be seen below, this causes us to receive less than the total number of items transmitted by the writers. This is likely due to writers overriding the changes made to the FIFO by other processes.



```
fakharyar.khan@kahan:/zooper2/fakharyar.khan/OS_Pset7$ ./a.out 10 10000
Beginning test with 10 writers, 10000 items each
Writer 4 completed
Writer 9 completed
Writer 6 completed
Writer 7 completed
Writer 5 completed
Writer 8 completed
Writer 2 completed
Writer 0 completed
Writer 1 completed
Writer 3 completed
Reader 0 completed
Uh-oh. We only received 98887 entries when we should have gotten 100000 entries
fakharyar.khan@kahan:/zooper2/fakharyar.khan/OS_Pset7$
```

Finally, we tested to see what would happen if the writers didn't block the wakeup signal before adding themselves to the waitlist. And here, we also end up hanging which again makes sense because by not blocking the wakeup signal, we could end up receiving and handling it right before we sleep and never receive the signal again.



```
fakharyar.khan@kahan:/zooper2/fakharyar.khan/OS_Pset7$ ./a.out 10 10000
Beginning test with 10 writers, 10000 items each
```