Project 3 Report: Concurrency

By

Darshan Vala

1.0 Introduction

What is the objective of this project?

The purpose of this project was to better understand writer and reader locks. In doing so, implemented a version of this to where the writer/reader thread does not starve. A problem that is faced during this is when a memory space is shared between processes. The processes want to read and write to the same memory spot but that would result in a loss of data.

1.1 Solution

The solution to this memory space problem between processes is to use semaphores to balance and synchronize the threads. We want the writer to get access to the location in memory while no other reader or writer has access to that location at the same time. Once the original writer is done accessing the object then the other processes can one-by-one access it. Multiple readers can actually access the object at the same time, since they are not changing what's being stored in memory. My solution utilizes three functions that I include in my header file main.h. Those functions include read_and_write(), threadR(), and threadW(). The last two functions utilize semaphores for locking and stopping certain threads. In the main function the threads are created and combined. It makes sure that no threads are starving as well. The semaphore variables I used are:

Semaphore stopR; // prevents other readers from accessing when a writer is there

Semaphore stopW; // if reader is accessing, lock writers out from accessing

Semaphore lock_num; // prevent the counter from increasing for readers

Int countR = 0; // how many reader threads are being utilized

1.2 Pseudocode

The following pseudocode will utilize the variables used in the last section.

NO INPUT

NO OUTPUT

START:

read_and_write(){

```
int x = 0, y = 0, z = 0;
   int ran = rand() % MAX_RAN_NUM;
   do
      do
         x = y * z;
         z++;
      while(z < ran);
      y++;
   while(y < ran);

Return

   END
}
```

NO INPUT

OUTPUT pointer

START:

threadW(){

wait(stopR)

wait(stopW)

“Write is in”

Read_and_write

post(stopW)

post(stopR)

“Write is done” Return

}

END

NO INPUT

OUTPUT pointer

BEGIN:

threadR(){

wait(stopR)

wait(lock_num)

Increment the reads accessing

“Read is in”

If only one reader has access{

wait(stopW)

}

post(lock_num)

post(stopR)

Read_and_write()

wait(lock_num)

Decrement the amount of readers with access

If there are no more readers accessing {

“Last reader in”

post(lock_num)

“Read is all done”

Return

MAIN

Semaphores initialized

Open file and while youre not at the end of the file

Progress through the first scenario

If the current index is a w or r create the respective thread

Join thread

Go until the end of the scenario and go to next scenario

Repeat until end of the file

Print results

Close file

1.3 Estimation

I worked about 40-50 hours on this solution because it took me a while to understand when to stop a thread from having access. I would have saved time if I wrote out my solution first before coding right away. That made me spend many hours at the library and on youtube for tutorials. I utilized geeksforgeeks to understand semaphores more and I used youtube tutorials to guide me with my solution, along with github examples and Udemy courses.