Kunal Mukherjee
CS-475
2/20/2018
Dr. Hwang

Discussions

My program has mainly three components. As described by the book, I have a buffer.h header file, that implements the buffer_item. It first declared buffer_item to be of type int and the BUFFER_SIZE to be five. The buffer is nothing but an array made up of buffer_item of size BUFFER_SIZE. It also implements two functions insert_item and remove_item. The function insert_item, finds the first empty buffer and inserts a random number and remove_item does exactly the opposite as it finds the first non-empty buffer and try to remove an item from there. I also, have an extra function called printBuffer that prints out the buffer in the current state.

My main program has the following objects declared: two semaphores(empty & full), two mutex (mutex and mutex_cout). It initializes the semaphore empty to 0, and full to BUFFER_SIZE. It then initializes the buffer and creates the array of producer and consumer threads. The main program then continues to create the number of producer threads and consumer threads as specified by the user. The main program then sleeps for a specific amount of time and upon awakening it returns 0, that kills all the producer and consumer thread at that time.

The synchronizations occurs mainly in the third component, which is the producer and consumer thread. The main idea is that the producer_thread tries to access the buffer and try to insert an item. The consumer_thread tries to access the buffer and remove an item. I used semaphores to solve the producer and consumer problem by implementing mutual exclsuion, that when a producer is modifying the buffer no other consumer or another producer thread should have access to it and vice-versa. I then used the mutex locks to make sure that only one thread at a time is accessing the critical section in a given thread. The CS when a thread is calling the function insert_item or remove_item , respectively in the producer and consumer thread as only those two functions mutate the buffer. If a particular producer thread is accessing it, then any other consumer thread has to wait until this producer thread is done and vice-versa. In general, Mutex is a locking mechanism used to synchronize access to a resource and Semaphore is a signaling mechanism, that a producer thread is done, so the consumers thread can carry on.

Questions

1> The most difficult part of understanding of thread manipulation was how to accurately kill a thread and passing variable as intptr_t and casting it as a (void *) as I want producer function to run in the thread with that argument. I tried casting it as just a int, but I got warning that I was cast to a pointer from an integer. So, I found out that using a intptr_t solves the issue as I can not casting of different sizes.

Killing a thread was an issue as I was using pthred_kill(NULL), since the thread were in infinite loop they were never terminating and my command was useless. I read over the internet, that one should kill a thread by passing on a control signal about the intent of killing the thread, instead of just abruptly killing the thread. Then, Dr. Hwang help to understand that just returning will kill the the main thread and along with it all its subsidiary thread connected to it also. I need not worry about any memory leaks for the threads created.

2> The creation of the thread was easiest to understand and how to initialize a pthread and then how to start creating the producer and consumer thread. Mainly because it was extensively explained in the book and we went over it in class.

3> The part during cross synchronization between producer and consumer or the use of ‘full’ and ‘empty’ semaphore was a little confusing to me. I was confused between binary semaphore and counting semaphore s well as what the two semaphores supposed to represent. Then, after some online research I understood the difference and also got a better understanding of mutexs. Before, I was confused as to why we were using mutex instead of binary semaphores. I also understood the concept mentioned above that in general, Mutex is a locking mechanism used to synchronize access to a resource and Semaphore is a signaling mechanism, that a producer thread is done, so the consumers thread can carry on. I was a little confused as to why a full semaphore was initialized to 0 and empty to BUFFER_SIZE, as the name sounds counter intuitive at first.

4> The use of mutex locks to gain access to the critical section was the easiest to understand. The mutex lock was in general a simple concept that is very easy to use to solve some parts of synchronization problem, like mutual exclusion of critical sections, like insert_item and remove_item.

5> I would have liked to make another header file and take the producer and consumer thread functions to be under the cover. So, that anyone who looks at the driver programs need not be concerned about how the synchronizations is done or implemented.

6> Counting semaphore for synchronization and the kill of a thread were the most interesting part of the project. I researched about pthread_kill, pthread_Exit and pthread_join to see which is the best fit for my use. I realized the pthread_Exit(pthread) does the same thing as if you just did return 0, it will kill of each threads on the go. I could not used pthread_join as all the threads are in infinite loop and they will not be exiting on their own. The semaphore for cross synchronization was very intriguing as well, for the reasons mentioned above.