EI331 Course Project

By: Wang Haoxuan

Instructor: Wu ChenTao

November 28, 2018

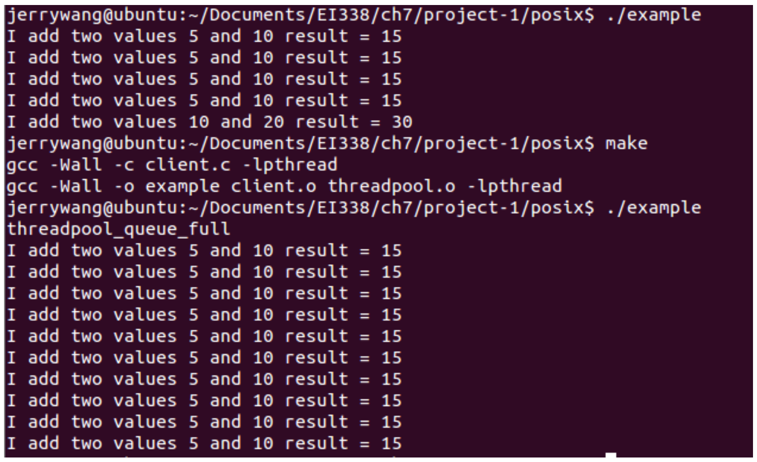
I. PROJECT 5: DESIGNING A THREAD POOL & PRODUCER-CONSUMER PROBLEM

A. Designing a Thread Pool

This project requires us to design a threadpool of ourselves. The threadpool is a actually a queue that contains threads that is created previously, and can be used directly when they are needed. A critical problem is that when we are using the threadpool, we have to take notice of which threads are used and the actual order of the threads to be used. These are complemented by semaphores and mutex locks.

We define a new struct of threadpool to structure our algorithm. The threadpool contains a mutex lock for the inner work, a thread queue that contains the threads, and parameters that store the status of the thread queue. The whole idea of the algorihtm is simple, but many special occassions need to be considered, we always need to check the suitablity of the algorithm's current state. Thus, the code seems to be complex.

We test the threadpool by pushing processes into the pool and print out returns. By examining the output, we can determine whether our threadpool is working properly. As shown in the figure below, we first tested the threadpool by first pushing 4 same threads into the pool, and push a different one into it, then we can get the 5 output lines, which is true. We also tested the upper bound of the number of queues. I set the maximum number of threads in the threadpool to be 10, and I put 10 same processes into the pool as well as a different one as the last. As a result, only the first 10 processes' output can be shown, which means only these are ran.



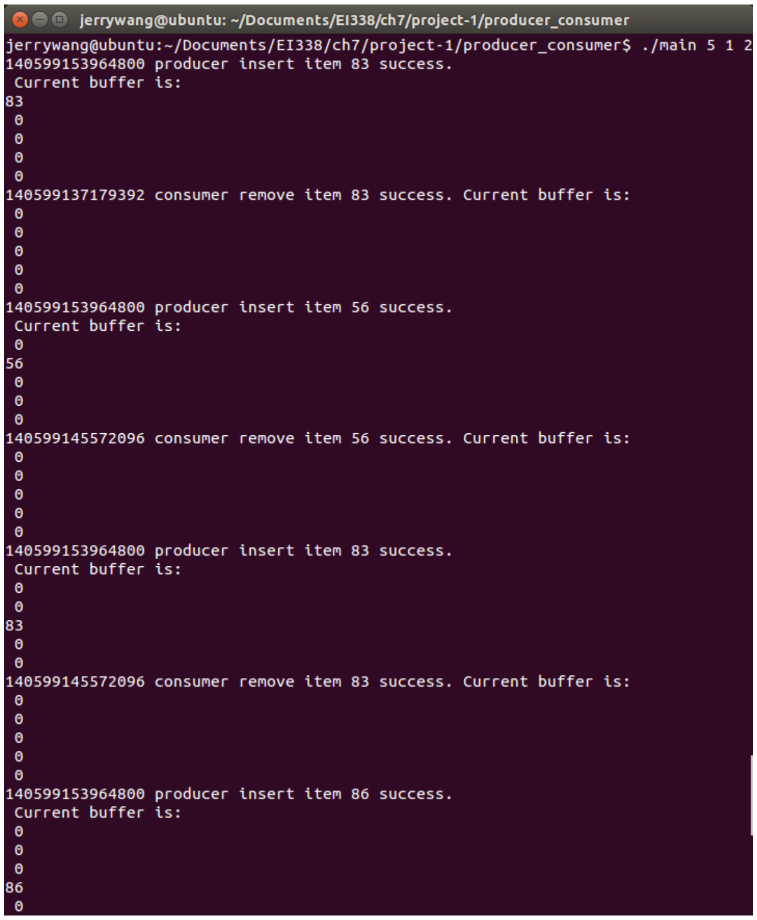
```
jerrywang@ubuntu:~/Documents/EI338/ch7/project-1/posix$ ./example
I add two values 5 and 10 result = 15
I add two values 5 and 10 result = 15
I add two values 5 and 10 result = 15
I add two values 5 and 10 result = 15
I add two values 10 and 20 result = 30
jerrywang@ubuntu:~/Documents/EI338/ch7/project-1/posix$ make
gcc -Wall -c client.c -lpthread
gcc -Wall -o example client.o threadpool.o -lpthread
jerrywang@ubuntu:~/Documents/EI338/ch7/project-1/posix$ ./example
threadpool_queue_full
I add two values 5 and 10 result = 15
I add two values 5 and 10 result = 15
I add two values 5 and 10 result = 15
I add two values 5 and 10 result = 15
I add two values 5 and 10 result = 15
I add two values 5 and 10 result = 15
I add two values 5 and 10 result = 15
I add two values 5 and 10 result = 15
I add two values 5 and 10 result = 15
I add two values 5 and 10 result = 15
```

FIG. 1: The result explaination of the threadpool

Code is shown in the attached files.

B. Producer-Consumer Problem

This project aims to implement the producer-consumer algorithm. This algorithm tries to accomplish: Given a queue, the producer would be able to generate items and push them into the queue, the consumer can push the items in the queue out. The problem is that there are multiple threads so that mutex locks and semaphores are needed to maintain the consistency of the queue. The effect of the algorithm is shown as below:



```
jerrywang@ubuntu: ~/Documents/EI338/ch7/project-1/producer_consumer
jerrywang@ubuntu:~/Documents/EI338/ch7/project-1/producer_consumer$ ./main 5 1 2
140599153964800 producer insert item 83 success.
 Current buffer is:
83
 0
 0
 0
 0
140599137179392 consumer remove item 83 success. Current buffer is:
 0
 0
 0
 0
 0
140599153964800 producer insert item 56 success.
 Current buffer is:
 0
56
 0
 0
 0
140599145572096 consumer remove item 56 success. Current buffer is:
 0
 0
 0
 0
 0
140599153964800 producer insert item 83 success.
 Current buffer is:
 0
 0
83
 0
 0
140599145572096 consumer remove item 83 success. Current buffer is:
 0
 0
 0
 0
 0
140599153964800 producer insert item 86 success.
 Current buffer is:
 0
 0
 0
86
 0
```

FIG. 2: The result for the producer-consumer problem

Code is shown in the attached files.