SUSTech CS302 OS Lab1 Report

Title: Producer-Consumer Problem

Student: Name: _Lu Ning_______, ID: _11610310_______

Time: 2019.4.9

Experimental Environment: linux

Objective: Master the synchronization & mutual exclusion algorithm, understand the producer-consumer models, and understand the reader-writer problem. Understand the multi-threaded concurrent execution mechanism. Understand the synchronization and mutexes between threads.

Deadline: 11:59 AM, 2019-04-10

Summit by: Blackboard

Task：

Task 1.Understand the source codes

Task 2.Edit and modify the source codes

Experiments:

1. fundamental：

Function(功能) of APIs:

- **pthread_create: create a new thread and execute the input function**
- **pthread_join: wait until the corresponding thread ends**
- **pthread_mutex_lock: lock the mutex, if the mutex has already been acquired, wait until it is released**
- **pthread_cond_wait: the thread will sleep until some signal is received. This function needs to be used with a mutex. Current thread locks the mutex, and during the thead's wait time, the mutex is released. After the condition signal is received, the mutex is locked by this thread again.**
- **pthread_cond_signal: send a signal to wake up the thread waiting for this condition**
- **pthread_mutex_unlock: release the lock**

2. Producer-Consumer Problem

- Are the data that consumers read from the buffer are produced by the same producer?

Yes. In main() only creates one producer.

```
pthread_create(&tid1, NULL, producer, NULL);    // create producer
pthread_create(&tid2, NULL, consumer, NULL);    // create consumer
```

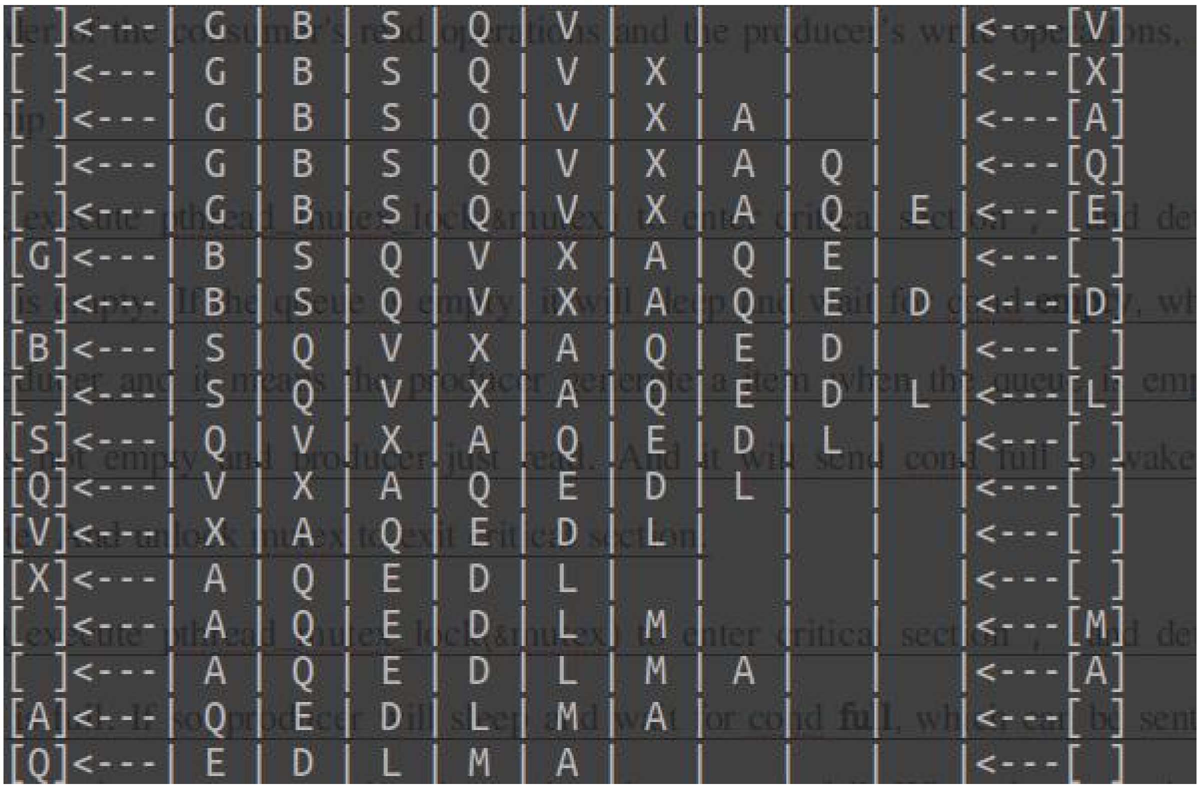
- What is the order of the consumer's read operations and the producer's write operations, and their relationship

Read： fisrt execute pthread_mutex_lock(&mutex) to enter critical section， and detect whether the queue is empty. If the queue is empty, it will sleep and wait for cond **empty**, which can be sent by producer and it means the producer generate a item when the queue is empty. When the queue is not empty and producer just read. And it will send cond full to wake up producer to generate. And unlock mutex to exit critical section.

Write: Fisrt execute pthread_mutex_lock(&mutex) to enter critical section， and detect whether the queue is full. If so, producer will sleep and wait for cond **full**, which can be sent by consumer and it means the consumer read an item when the queue is full. When the queue is not empty and producer just read. And it will send cond full to wake up producer to generate.

This mechanism guarantees no read when empty and no write when full.

- Briefly describe the result of the program:



```
[ ]<---| G | B | S | Q | V |   |   |   |   |<---[V]
[ ]<---| G | B | S | Q | V | X |   |   |   |<---[X]
[ ]<---| G | B | S | Q | V | X | A |   |   |<---[A]
[ ]<---| G | B | S | Q | V | X | A | Q |   |<---[Q]
[ ]<---| G | B | S | Q | V | X | A | Q | E |<---[E]
[G]<---| B | S | Q | V | X | A | Q | E |   |<---[ ]
[ ]<---| B | S | Q | V | X | A | Q | E | D |<---[D]
[B]<---| S | Q | V | X | A | Q | E | D |   |<---[ ]
[ ]<---| S | Q | V | X | A | Q | E | D | L |<---[L]
[S]<---| Q | V | X | A | Q | E | D | L |   |<---[ ]
[Q]<---| V | X | A | Q | E | D | L |   |   |<---[ ]
[V]<---| X | A | Q | E | D | L |   |   |   |<---[ ]
[X]<---| A | Q | E | D | L |   |   |   |   |<---[ ]
[ ]<---| A | Q | E | D | L | M |   |   |   |<---[M]
[ ]<---| A | Q | E | D | L | M | A |   |   |<---[A]
[A]<---| Q | E | D | L | M | A |   |   |   |<---[ ]
[Q]<---| E | D | L | M | A |   |   |   |   |<---[ ]
```

Left is read item and right is write item. If the queue is full, no write and next step must be read. If the queue is empty, no read and next step must be write.

- What queue is used in this program, and its characteristics？

A shared ring, use head and tail to represent the ring dynamically. FIFO

- Briefly describe the mutual exclusion mechanism of this program:

The mutex guarantee the read and write will not not happen simultaneously, and use condition lock to guarantee no read when empty and no write when full.

3. Readers-Writers Problem

- What interfaces of semaphore are used in this program？

sem_t sem_init sem_wait sem_post

- What are these interfaces used for？

sem_t is a semaphore struct.

sem_init is to initialize a semaphore

sem_wait is to down the semaphore

sem_post is up the semaphore

- Can readers read at the same time？Why？

Yes. Read do not change the shared data and will not lead to unexpectable result.

- Can writers write at the same time ? Why ?

No. If two writers write at the same time, the data written by one writer may be overwritten by another writer.

- What is the performance of the reader's synchronous reading ?

Many readers prints "Reader Inside.." the same buffer.

- After one writer writes, can the next writer write before one reader read ? Why ?

Yes, because after one writer writes complete, it release a lock and both readers or next writer can get the lock to do task.

Conclusion:

I learned a lot about the mutex and semaphore in this lab. Also, obtaining the knowledge of how to use C to implement lock.

Submission:

-OS_Lab6_studentID	(directory)
---Lab6_report_studentID.pdf	(pdf version report)
---read.h	(c file)
---write.h	(c file)

Zip the directory with the same name and submit it