

# Project 5: Designing a Thread Pool & Producer-Consumer Problem

---

Name: 韩冰

Number: 516030910523

---

## 1.Designing a thread pool

---

Thread pools were introduced in Section 4.5.1. When thread pools are used, a task is submitted to the pool and executed by a thread from the pool. Work is submitted to the pool using a queue, and an available thread removes work from the queue. If there are no available threads, the work remains queued until one becomes available. If there is no work, threads await notification until a task becomes available. This project involves creating and managing a thread pool, and it may be completed using either Pthreads and POSIX synchronization or Java. Below we provide the details relevant to each specific technology.

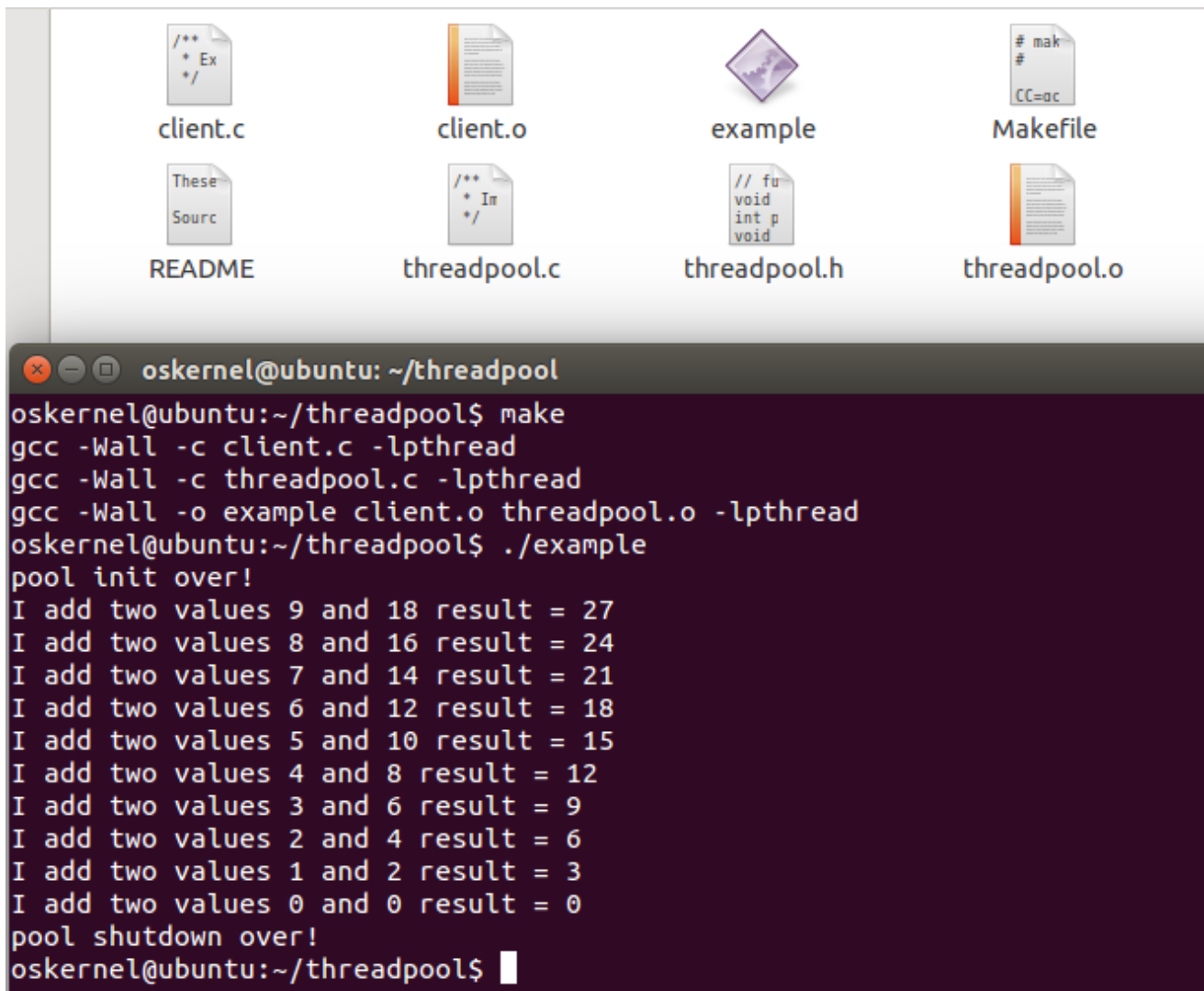
### Program file tree structure.

```
.
├── threadpool.h   Define the threadpool
├── threadpool.c   Realize the function
├── client.c       main function
└── Makefile       Makefile
```

### Solution:

1. initial the threadpool. In this step, create the threads and initial the semaphore and thread\_mutex to prepare for the next step. What's more, initial the Queue list for store the tasks.
2. Submit the tasks to the queue. When modify the queue, using the semaphore and mutex to avoid deadlock.
3. Sleep time over. Cancel the threads and shutdown threadpool. Done!
4. The more detail can see the codes.

### Screenshot



## 2. Producer\_consumer problem

In Section 7.1.1, we presented a semaphore-based solution to the producer-consumer problem using a bounded buffer. In this project, you will design a programming solution to the bounded-buffer problem using the producer and consumer processes shown in Figures 5.9 and 5.10. The solution presented in Section 7.1.1 uses three semaphores: empty and full, which count the number of empty and full slots in the buffer, and mutex, which is a binary (or mutualexclusion) semaphore that protects the actual insertion or removal of items in the buffer. For this project, you will use standard counting semaphores for empty and full and a mutex lock, rather than a binary semaphore, to represent mutex. The producer and consumer—running as separate threads—will move items to and from a buffer that is synchronized with the empty, full, and mutex structures. You can solve this problem using either Pthreads or the Windows API.

### Program file tree structure.

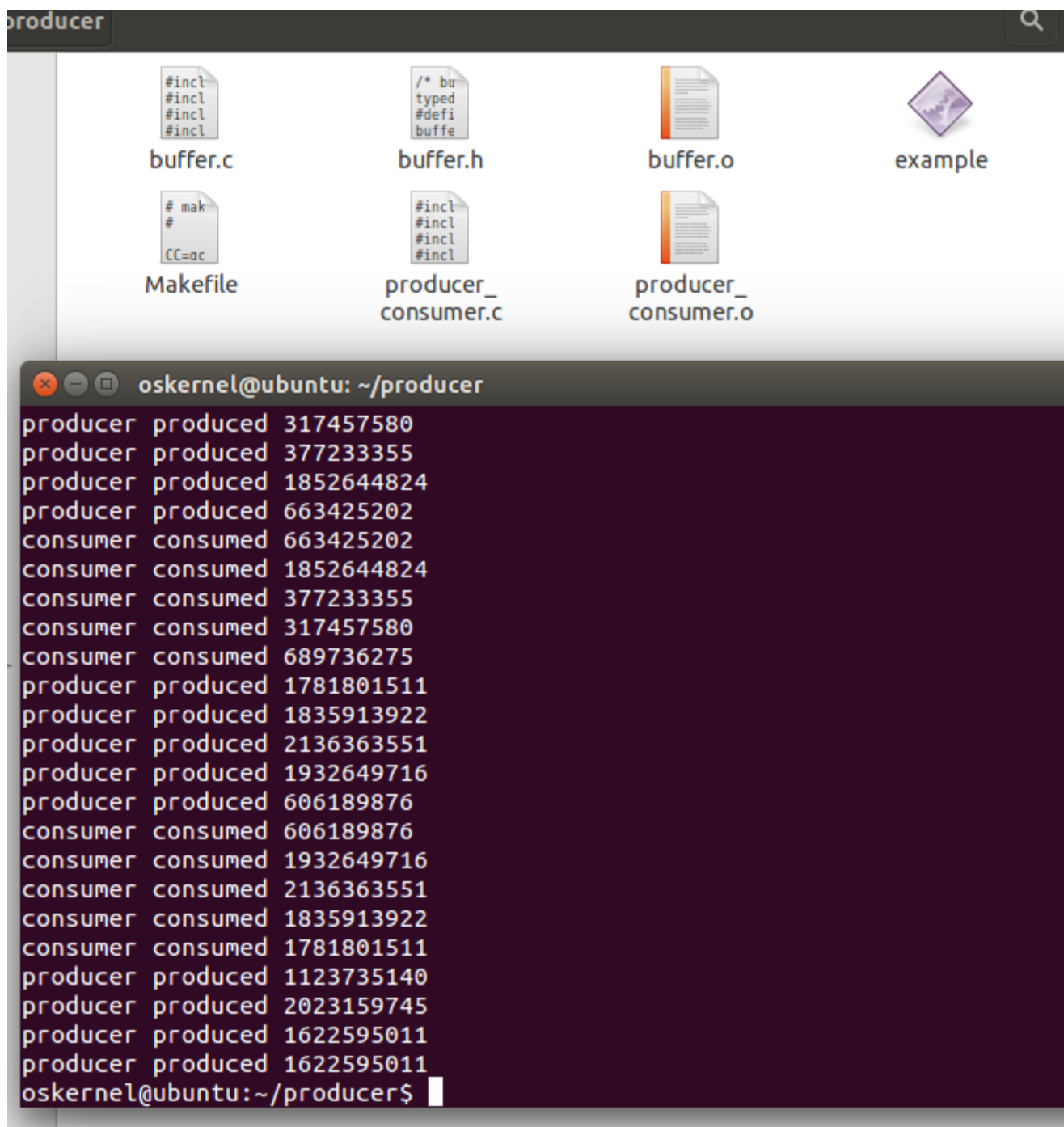
```
.
├─ buffer.h    Define the buffer
├─ buffer.c    Realize the buffer function
├─ producer_consumer.c    main function
└─ Makefile    Makefile
```

## Solution:

1. Get command line arguments `argv[1]`, `argv[2]`, `argv[3]`, which represent the sleep time, producer threads number and consumer threads number.
2. Initialize the buffer. In this step, we also initialize the mutex and semaphore, which is used in next steps.
3. Create the producer threads, which produce the random number and put it into the buffer.
4. Create the consumer threads, which consume the number in the buffer.
5. Sleep time over, shut down the program.
6. The detail can see the codes.

## Result:

sleep time = 1, producer number = 3, consumer number = 3.



The image shows a Linux environment with a file manager window titled 'producer' and a terminal window. The file manager displays the following files:

- `buffer.c` (C source file)
- `buffer.h` (Header file)
- `buffer.o` (Object file)
- `example` (Executable file)
- `Makefile` (Build file)
- `producer_consumer.c` (C source file)
- `producer_consumer.o` (Object file)

The terminal window shows the output of the program, with producer threads producing random numbers and consumer threads consuming them. The output is as follows:

```
oskernel@ubuntu: ~/producer
producer produced 317457580
producer produced 377233355
producer produced 1852644824
producer produced 663425202
consumer consumed 663425202
consumer consumed 1852644824
consumer consumed 377233355
consumer consumed 317457580
consumer consumed 689736275
producer produced 1781801511
producer produced 1835913922
producer produced 2136363551
producer produced 1932649716
producer produced 606189876
consumer consumed 606189876
consumer consumed 1932649716
consumer consumed 2136363551
consumer consumed 1835913922
consumer consumed 1781801511
producer produced 1123735140
producer produced 2023159745
producer produced 1622595011
producer produced 1622595011
oskernel@ubuntu:~/producer$
```