# 操作系统实验报告

## Project II

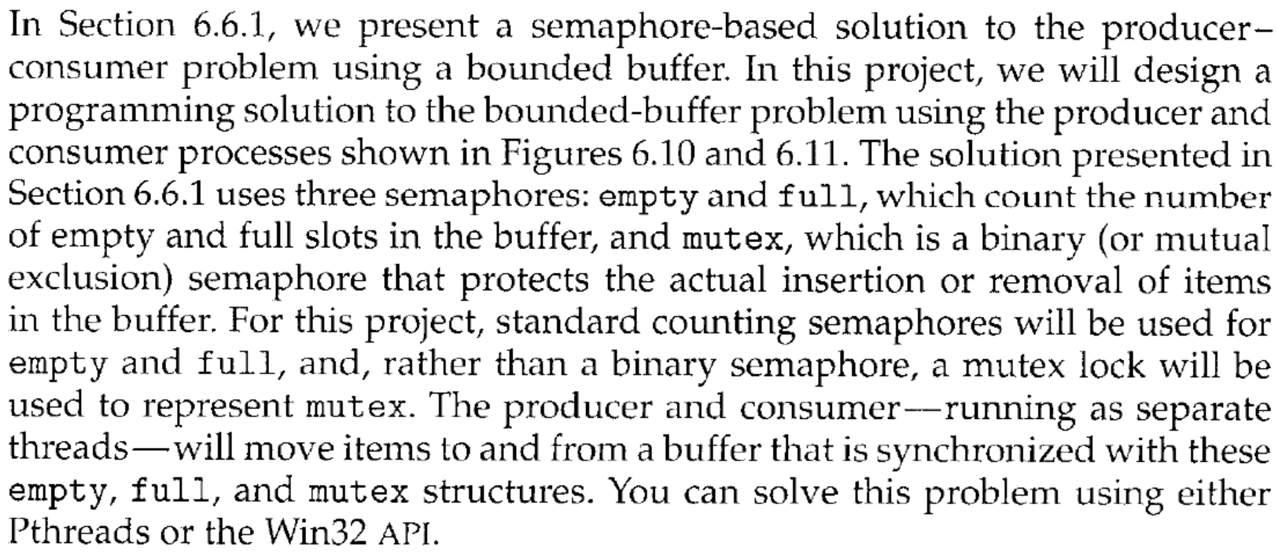
## Producer-Consumer Problem

班级：2014211314

姓名：叶文霆

学号：2014211519

## 一、实验描述：



In Section 6.6.1, we present a semaphore-based solution to the producer–consumer problem using a bounded buffer. In this project, we will design a programming solution to the bounded-buffer problem using the producer and consumer processes shown in Figures 6.10 and 6.11. The solution presented in Section 6.6.1 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 mutual exclusion) semaphore that protects the actual insertion or removal of items in the buffer. For this project, standard counting semaphores will be used for `empty` and `full`, and, rather than a binary semaphore, a mutex lock will be used to represent `mutex`. The producer and consumer—running as separate threads—will move items to and from a buffer that is synchronized with these `empty`, `full`, and `mutex` structures. You can solve this problem using either Pthreads or the Win32 API.

## 二、实验环境

macOS Sierra Version 10.12.1

CLion 2016.2.3

## 三、程序设计说明

### 3.1 缓冲区（The Buffer）

#### 3.1.1 模块描述

缓冲区是一个固定大小为 BUFFER_SIZE 的环形队列，支持在队首添加元素和在队尾删除元素。值得注意的是，一个大小为 BUFFER_SIZE 的数组实际所能保存的元素要少一个（为区分队满和队空）。

#### 3.1.2 核心代码

- *// Definition of bufferQueue*

```
typedef int buffer_item;
#define BUFFER_SIZE 10
typedef struct{
    buffer_item data[BUFFER_SIZE];
    int head, tail;
} bufferQueue;
```

-
  ```
  // bufferQueue's function

  int insert_item(bufferQueue *queue, buffer_item x)
  {
      if ((queue->tail + 1) % BUFFER_SIZE == queue->tail) return -
  1;
      queue->data[queue->tail] = x;
      queue->tail = (queue->tail + 1) % BUFFER_SIZE;
      return 0;
  }
  ```

-
  ```
  int remove_item(bufferQueue *queue)
  {
      if (queue->head == queue->tail) return -1;

      int temp = queue->data[queue->head];
      queue->head = (queue->head + 1) % BUFFER_SIZE;
      return temp;
  }
  ```

## 3.2 生产者与消费者线程（Producer and Consumer Threads）

### 3.2.1 模块描述

生产者循环两种状态：随机sleep 一段时间；使用 rand() 生成一个 0 ~ RAND_MAX 的整数，等队列非满的时候放在队尾，最后在命令行输出相关信息。

消费者循环两种状态：随机 sleep 一段时间；等位列非空的时候在队首取元素，并在命令行输出相关信息。

### 3.2.2 核心代码

-
  ```
  void *consume(int *i)
  {
      int CsmID = *i;
      while (1)
      {
          sleep(rand() % MAXSLEEP) + 1;

          sem_wait(full);
          pthread_mutex_lock(&mutex);

          int x = remove_item(&buffer);
          if (x < 0)
          {
              fprintf(stderr, "the queue is empty! \n");
              exit(-1);
  ```

```
        }
        else
            printf("Consumer %d get %d from buffer\n", CsmID, x);

        pthread_mutex_unlock(&mutex);
        sem_post(empty);
    }
}
```

-
  ```
  void *produce(int *i)
  {
      int pdsID = *i;
      while (1)
      {
          sleep(rand() % MAXSLEEP);
          sem_wait(empty);
          pthread_mutex_lock(&mutex);

          int num = rand(), x = insert_item(&buffer, num);
          if (x < 0)
          {
              fprintf(stderr, "the queue is full! \n");
              exit(-1);
          }
          else
              printf("Producer %d add %d to buffer\n", pdsID, num);

          pthread_mutex_unlock(&mutex);
          sem_post(full);
      }
  }
  ```

## 3.3 主程序（main）

### 3.3.1 模块描述

主函数 main() 从命令行接收三个参数：睡眠时间、生产者数量、消费者数量。之后初始化 bufferQueue 和信号量，创建若干生产者和消费者，主程序进入休眠。休眠到要求时间后主函数输出提示信息，使用 exit() 退出程序，所有子线程将由系统自动收回。值得注意的是，虽然子程序最后的退出代码是-1，但是实际上资源已经有操作系统自动回收了。

由于 Mac OS 不提供无名信号量的实现（实际上是被 deprecated 标记），本程序使用了有名信号量。而有名信号量一定要在主程序结束的时候手动释放，不然在再次启动程序的时候将会产生信号量超时等错误。

### 3.3.2 核心代码

```
int main(int argc, char const *argv[])
{
   // Check the parameters
   if (4 != argc)
   {
      fprintf(stderr, "usage: Pds&Csm.out <int> <int> <int>\n");
      return -1;
   }
   if ((atoi(argv[1]) < 0) || (atoi(argv[2]) < 0) ||
(atoi(argv[3]) < 0))
   {
      fprintf(stderr, "Input should be positive interger!\n");
      return -1;
   }

   /* set the para */
   int numOfCsm, numOfPds, timeOfSleep;
   timeOfSleep = atoi(argv[1]);
   numOfPds = atoi(argv[2]);
   numOfCsm = atoi(argv[3]);
   srand(time(NULL));

   /* initialize data and semaphore */
   buffer.head = buffer.tail = 0;
   pthread_mutex_init(&mutex, NULL);
   empty = sem_open("empty", O_CREAT|O_RDWR, 0666, BUFFER_SIZE -
1);
   full = sem_open("full", O_CREAT|O_RDWR, 0666, 0);
   if (SEM_FAILED == empty || SEM_FAILED == full)
   {
      perror("semaphore failed");
      return -1;
   }

   /* create the handler of threads */
   pthread_t *consumers, *producers;
   consumers = (pthread_t *) malloc(numOfCsm * sizeof(pthread_t));
   producers= (pthread_t *) malloc(numOfPds * sizeof(pthread_t));

   /* get the default attributes */
   pthread_attr_t attr;
   pthread_attr_init(&attr);
```

```c
    /* create the thread */
    int i;
    for (i = 0; i < numOfCsm; ++i)
    {
        int num = i;
        pthread_create(&consumers[i], &attr, consume, &num);
    }
    for (i = 0; i < numOfPds; ++i)
    {
        int num = i;
        pthread_create(&producers[i], &attr, produce, &num);
    }

    /* sleep */
    sleep(timeOfSleep);

    /* destroy the semaphore and terminate */
    printf("The process is completed!\n");
    sem_unlink("empty");
    sem_unlink("full");
    pthread_mutex_destroy(&mutex);
    exit(0);
}
```

## 四、实验结果

**三个生产者，三个消费者，最大随机休眠时间置为 5s，缓冲区大小 10**

```
Producer 2 add 1149316476 to buffer

Consumer 3 get 1149316476 from buffer

Producer 3 add 326503786 to buffer

Consumer 3 get 326503786 from buffer

Producer 3 add 898567672 to buffer

Producer 3 add 154262540 to buffer

Producer 3 add 654868069 to buffer

Consumer 3 get 898567672 from buffer

Consumer 3 get 154262540 from buffer

Producer 2 add 513902480 to buffer

Consumer 2 get 654868069 from buffer
```

```
Producer 1 add 2098578184 to buffer
Consumer 2 get 513902480 from buffer
Producer 1 add 1874297364 to buffer
Consumer 3 get 2098578184 from buffer
Producer 3 add 1319674811 to buffer
Consumer 3 get 1874297364 from buffer
Consumer 3 get 1319674811 from buffer
Producer 3 add 346180816 to buffer
Producer 1 add 1791509058 to buffer
Consumer 2 get 346180816 from buffer
Producer 3 add 2116152868 to buffer
Producer 2 add 1351921783 to buffer
Consumer 2 get 1791509058 from buffer
Consumer 2 get 2116152868 from buffer
Consumer 3 get 1351921783 from buffer
Producer 2 add 1130100639 to buffer
Consumer 2 get 1130100639 from buffer
Producer 2 add 954358768 to buffer
Consumer 3 get 954358768 from buffer
Producer 3 add 825267156 to buffer
Consumer 2 get 825267156 from buffer
The process is completed!

Process finished with exit code 0 -1 -1
```