

# Write a c program to implement product and consumer problem using multi-threads

---

## Target

1. Write a c/c++ program
2. To implement product and consumer problem using multi-threads
  1. 采用多线程实现生产者消费者问题
3. GCC

## Tools

### Install GCC Software Collection

```
sudo apt-get install build-essential
```

### How to use GCC

- [gcc and make](#)

### Posix thread

```
#include <pthread.h>
pthread_create()
```

### Posix 信号量Semaphore (<https://www.cnblogs.com/lnlin/p/9721375.html>)

- 创建与释放 : `sem_init()`、`sem_destroy()`
- 打开与关闭 : `sem_open()`、`sem_close()`、`sem_unlink()`
- 使用 :
  - DOWN : `sem_wait()`、`sem_trywait()`
  - UP : `sem_post()`
  - VALUE : `sem_getvalue()`

### 具体声明:

```
#include <semaphore.h>

// 初始化一个基于内存的信号量(sem指向的)
int sem_init(sem_t *sem, int shared, unsigned int value);
// sem参数指向应用程序分配的sem_t变量
// shared参数为0时, 待初始化的信号量是在同一进程的各个线程中共享的(具有随进程的持续性)
// shared参数非0时, 待初始化的信号量必须放在某种类型的共享内存区中, 想
```

```

// 要是由此信号量的所有进程都需要可以访问该共享内存区(随该共享内存区持续)
// value参数为该信号量的初始值

// 摧毁指定的基于内存的信号量
// 成功返回0，出错返回-1
int sem_destroy(sem_t *sem);

// 创建一个新的有名信号量或打开一个已存在的有名信号量
// 成功返回指向信号量的指针，出错返回SEM_FAILED
sem_t *sem_open(const char *name, int oflag, /* mode_t mode, unsigned int value
*/);
// name: IPC 名字，可能是某个文件系统中的真正路径名，也可能不是
// oflag: 可以是0、O_CREAT、O_CREAT|O_EXCL
// 在oflag指定了O_CREAT标志后，mode、value参数必须指定(mode指定权限位，value指定信号量初值)

// 关闭一个由 sem_open 打开的信号量
// 成功返回0，出错返回-1
int sem_close(sem_t *sem);

// 将某个有名信号量从系统删除
// 成功返回0，出错返回-1
int sem_unlink(const char *name);

// 测试指定信号量的值，若该值大于0，则将其减1并立即返回，若该值为0，则
// 线程被投入睡眠，等待该值变为大于0，再将其减1并返回
// 成功返回0，出错返回-1
int sem_wait(sem_t *sem);

//
// 测试指定信号量的值，若该值大于0，则将其减1并立即返回，若该值为0，则
// 返回一个EAGAIN错误
// 成功返回0，出错返回-1
int sem_trywait(sem_t *sem);

// 在线程使用完某个信号量时，调用sem_post()，该函数将指定信号量的值加1，
// 然后唤醒等待该信号量值变为正数的任意线程
// 成功返回0，出错返回-1
int sem_post(sem_t *sem);

// 通过valp返回指定信号量的当前值，若当前信号量已上锁，则返回0或某个
// 负数(其绝对值为等待该信号量解锁的线程数)
// 成功返回0，出错返回-1
int sem_getvalue(sem_t *sem, int *valp);

```

## Posix mutex互斥量

```
#include <pthread.h>
pthread_mutex_t mutex =PTHREAD_MUTEX_INITIALIZER;
int pthread_mutex_init(pthread_mutex_t *restrict mutex,
                      const pthread_mutexattr_t *restrict attr);
int pthread_mutex_destroy(pthread_mutex_t *mutex);
int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_trylock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

实例:

```
pthread_mutex_t mutex;

void * thread_run(void *arg)
{
    pthread_mutex_lock(&mutex);
    //TODO
    XXXXXXXXX

    pthread_mutex_unlock(&mutex);
    return 0;
}

int main(int argc, char *argv[])
{
    pthread_t thread1, thread2;
    pthread_mutex_init(&mutex, 0);
    pthread_create(&thread1, NULL, thread_run, 0);
    pthread_create(&thread2, NULL, thread_run, 0);
    pthread_join(thread1, 0);
    pthread_join(thread2, 0);
    pthread_mutex_destroy(&mutex);
    return 0;
}
```

## 随机函数

- srand ((DWORD)time(NULL)); /\* Seed the random # generator \*/
- rand() /\* generatea random\*/
- Sleep() /msec/

get the total time of program execution

```
$ time tar xvJf linux-6.5.9.tar.xz

real    0m28.554s
user    0m7.738s
sys 0m3.554s
```

## How to do

write a c program to implement product and consumer problem using multi-threads

实现I个生产者J个消费者问题，其中共享缓冲区的大小为N，所有生产者共生产K(K>N)个产品后结束，所有消费者共消费K个产品后结束。

### 具体要求

- 严格按时序输出每个生产者、消费者的行为，其中包括生产产品k、消费产品k、进入临界区、存入产品、取出产品、离开临界区；
- 需要考虑边界（某生产者生产第K个产品后所有生产者结束；某消费者消费第K个产品后所有消费者结束）
- 需要考虑随机函数，生产者生产时需要一个随机时间；消费者消费时也需要一个随机时间；
- 编号：无论生产者还是消费者都需要有编号；产品同样也需要编号；缓冲区的各个产品项也需要有编号；
- 输出形式可以采用标准输出、图形动态显示及同时文字记录输出等方式，无论是生产者还是消费者，其主要输出内容如下：a) 进入临界区前，输出某某编号（生产者/消费者）线程准备进入临界区 b) 进入临界区后，输出某某编号（生产者/消费者）线程已进入临界区 c) 离开临界区后，输出某某编号（生产者/消费者）线程已离开临界区 d) 生产者生产一个产品时，需要输出产品信息； e) 生产者将产品放入缓冲区时，需要输出相关信息； f) 消费者将产品从缓冲区取出时，需要输出相关信息； g) 消费者消费一个产品时，需要输出产品信息；
- 不能出现竞态
- 不能出现忙等待

### 结构体声明（参考）

```
#define DATA_SIZE 256
```

```
/*产品结构体声明*/
typedef struct item_st {
int id /*产品编号*/
time_t timestamp_p; /*生产时间*/
int produce_id; /*生产者编号*/
DWORD checksum; /* 产品校验码*/
DWORD data[DATA_SIZE]; /* 产品详细信息*/
} ITEM_ST;
```

```
/*缓冲区结构体声明*/
typedef struct buffer_st {
int id; /*缓冲区编号*/
boolean isempty; /*是否为空*/
ITEM_ST item; /*具体产品*/
```

```
buffer_st* nextbuf; /*下一个缓冲区首地址*/  
} BUFFER_ST;
```

也可以考虑Windows平台

详细见PDF附件。