## 实现生产者——消费者问题

#### 实验目的：

编写一个C程序来实现生产者-消费者问题。

#### 实验内容：

生产者消费者问题（英语：Producer-consumer problem），也称有限缓冲问题（英语：Bounded-buffer problem），是一个多线程同步问题的经典案例。该问题描述了共享固定大小缓冲区的两个线程——即所谓的“生产者”和“消费者”——在实际运行时会发生的问题。生产者的主要作用是生成一定量的数据放到缓冲区中，然后重复此过程。与此同时，消费者也在缓冲区消耗这些数据。该问题的关键就是要保证生产者不会在缓冲区满时加入数据，消费者也不会在缓冲区中空时消耗数据。

要解决该问题，就必须让生产者在缓冲区满时休眠（要么干脆就放弃数据），等到下次消费者消耗缓冲区中的数据的时候，生产者才能被唤醒，开始往缓冲区添加数据。同样，也可以让消费者在缓冲区空时进入休眠，等到生产者往缓冲区添加数据之后，再唤醒消费者。通常采用进程间通信的方法解决该问题。如果解决方法不够完善，则容易出现死锁的情况。出现死锁时，两个线程都会陷入休眠，等待对方唤醒自己。该问题也能被推广到多个生产者和消费者的情形。

实现I个生产者和J个消费者问题，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和消费者的行为，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。

需要考虑边界情况（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，生产者在生产时需要一个随机时间，消费者在消费时也需要一个随机时间。

需要对生产者、消费者和产品进行编号。

输出形式可以采用标准输出、图形动态显示或同时文本记录输出等方式。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，输出产品信息

e) 生产者将产品放入缓冲区时，输出相关信息

f) 消费者将产品从缓冲区取出时，输出相关信息

g) 消费者消费一个产品时，输出产品信息

不能出现竞态条件。不能出现忙等待。

#### 总结

通过本次实验，实验了生产者——消费者问题，对于信号量mutex，semaphore有了进一步的了解，体会到了并发过程种对临界资源进行访问控制的重要性。

源代码

#include <iostream>

#include <string.h>

#include <pthread.h>

#include <unistd.h>

using namespace std;

int current = 0; // producer运行加1，consumer运行减1

int buf[10];

int in = 0, out = 0;

int items = 0, spaces = 10;

bool flag; // 标记线程结束运行

pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER;

pthread\_cond\_t notfull = PTHREAD\_COND\_INITIALIZER; // 缓冲区不满

pthread\_cond\_t notempty = PTHREAD\_COND\_INITIALIZER; // 缓冲区不空

void \*producer( void \*arg ) {

while( flag ) {

pthread\_mutex\_lock( &mutex ); // 为保证条件变量不会因为多线程混乱，所以先加锁

while( !spaces ) { // 避免“惊群”效应，避免因其他线程实现得到事件而导致该线程“假醒”

pthread\_cond\_wait( &notfull, &mutex );

}

buf[in] = current++;

in = ( in + 1 ) % 10;

items++;

spaces--;

printf( "producer %zu , current = %d\n", pthread\_self(), current );

for( int i = 0; i < 10; i++ ) {

printf( "%-4d", buf[i] );

}

printf( "\n\n" );

pthread\_cond\_signal( &notempty );

pthread\_mutex\_unlock( &mutex );

}

pthread\_exit( NULL );

}

void \*consumer( void \*arg ) {

while( flag ) {

pthread\_mutex\_lock( &mutex );

while( !items ) {

pthread\_cond\_wait( &notempty, &mutex );

}

buf[out] = -1;

out = ( out + 1 ) % 10;

current--;

items--;

spaces++;

printf( "consumer %zu , current = %d\n", pthread\_self(), current );

for( int i = 0; i < 10; i++ ) {

printf( "%-4d", buf[i] );

}

printf( "\n\n" );

pthread\_cond\_signal( &notfull );

pthread\_mutex\_unlock( &mutex );

}

pthread\_exit( NULL );

}

int main() {

memset( buf, -1, sizeof(buf) );

flag = true;

pthread\_t pro[10], con[10];

int i = 0;

for( int i = 0; i < 10; i++ ) {

pthread\_create( &pro[i], NULL, producer, NULL );

pthread\_create( &con[i], NULL, consumer, NULL );

}

sleep(1); // 让线程运行一秒

flag = false;

for( int i = 0; i < 10; i++ ) {

pthread\_join( pro[i], NULL );

pthread\_join( con[i], NULL );

}

return 0;

}