单程车票
努力学习中

原 <网络编程培训之二> 编程实现生产者与消费者问题（linux下C实现）

分类： socket unp 2012-12-07 10:52 2623人阅读 评论(0) 收藏 举报

- 题目

```
[html]
01. 生产者-消费者问题描述:有一群生产者进程在生产消息,并将此 消息提供给消费者进程去消费。为使生产者进程和消费者进程能 并发进行,在他们之间设置了一个具有 N 个缓冲区的缓冲池,生 产者进程可以将它所生产的消息放入一个缓冲区中,消费者进程 可以从一个缓冲区中取得一个消息消费。
```

- 题目解读：

顾名思义，生产者生产产品，消费者消费产品，两者之间有一个合作的关系。产品如果被消费者消费完了，而生产者又没有生产出产品，消费者就得等待；同理，如果仓库已经放满了产品，而消费者还没有消耗，生产者生产出的产品就没地方放。两者之间又存在一个制约的关系。所以生产者在生产产品之前，先要看一下是否有空的地方，以免浪费，消费者，在消费之前，要先查看一下是否有产品，以免白跑一趟。

在下面程序的实现里，用一个数组（循环队列）来表示仓库，来存储产品，生产者生产的产品，是字符'1'，消费者消费'1',消费之后，将空间置为0. 查看仓库是否为空或者仓库是否为满，可以使用信号量的方法。当然生产者和消费者只能互斥的使用公共资源，否则可能会乱套。

- 题目实现

下面是一些涉及到的常用的函数：

信号量(多任务同步）：参见http://blog.163.com/kmustchenb@126/blog/static/110905765201077184056 5/

sem_t empty,full.，声明两个信号量(长整形）；

任何变量在使用前最好都的初始化，信号量也不例外，使用extern int sem_init ((sem_t *__sem, int __pshared, unsigned int __value)) ,来给信号量赋初值，初始时，假设数组全部为空（0），即empty=数组大小，full=0.

生产者在占用队列之前，先查看是否有空的地方（如果操作反过来，可能会造成死锁），使用函数sem_wait(sem_t * sem)，来查看信号量是否大于0，每成功调用一次，该信号量-1. 这个是阻塞版本，函数sem_trywait是非阻塞版本。

生产完产品之后，要释放仓库的占有权，并声明数组中又多了一个满的空间，使用函数sem_post(sem_t * sem),来使信号量+1，这里是full+1.

信号量也需要释放掉，使用函数sem_destory(sem_t * sem)来释放。

互斥锁（多任务互斥）：用来控制临界资源的，参见：http://blog.163.com/yangfan_407/blog/static

zy416548283

+ 加关注 发私信

访问： 124693次
积分： 2191
等级： BLOC 5
排名： 第9795名

原创： 100篇
转载： 2篇
译文： 0篇
评论： 47条

/127950654200992202188l5/

pthread_mutex_t mutex，声明互斥量

pthread_mutex_init(pthread_mutex_t * mutex, const pthread_mutexattr_t *mattr); 来初始化互斥量，属性为NULL时，设置为默认属性。

pthread_mutex_lock(pthread_mutex_t * mutex)，申请互斥锁，占用资源，这是阻塞用法；非阻塞：pthread_mutex_trylock(pthread_mutex_t * mutex);

pthread_mutex_unlock(pthread_mutex_t * mutex)，释放互斥锁，释放被占用的资源。

pthread_mutex_destory(pthread_mutex_t * mutex),销毁互斥锁。

线程相关，参见：http://blog.csdn.net/zhhxidian2005/article/details/5605404

pthread_t id,声明一个线程的id。

int pthread_create(pthread_t *restricttidp, const pthread_attr_t *restrictattr, void *(*start_rtn)(void), void *restrict arg)；创建线程，第一个参数返回线程id，第二个参数设置线程的属性（NULL时为默认属性），第三个参数为线程执行函数的名字，第四个参数是执行函数的参数（当不止一个参数时，可以用结构体来表示）。

pthread_join(pthread_t tid,void **status);等待线程终止，并保存终止线程的退出状态值。也可以用pthread_exit来终止线程。

pthread_self，可以获知自己的线程ID。

整个程序参见：http://hi.baidu.com/shazi129/item/4d2a054626be7d17896d1088，做了一些小的修改。

[cpp]

```cpp
/*
* author 赵雨
* 2012/05/21
*/

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <semaphore.h>

#define BUFFSIZE 9 // 缓冲池大小

struct queue_type     //缓冲池队列
{
    int buff[BUFFSIZE];
    int front;  //队头，消费者从对头取出"产品”
    int rear;   //队尾，生产者从队尾放入"产品“
}Q={{0},0,0};

sem_t empty; // 生产者私有信号量，表示空缓冲区数目 。
sem_t full;   //消费者私有变量，表示有产品的缓冲区数目。
pthread_mutex_t mutex; // 公有信号量，用于实现对临界区互斥访问

int producer_id = 0;   //生产者编号，初值为0
int consumer_id = 0;   //消费者编号，初值为0

/* 打印缓冲池情况*/
void print()
{
     int i;
     for(i = 0; i < BUFFSIZE; i++)
     printf("%d ", Q.buff[i]);
     printf("\n");
}

/*生产者*/
void *producer()
{
     int id=++producer_id;

     while(1)
      {
            sleep(1);  //
```

```
45.
46.             sem_wait(&empty);  //申请空缓冲区
47.             pthread_mutex_lock(&mutex);    //申请队列互斥
48.
49.             Q.buff[Q.rear] = 1;   //将产品放入rear所指向的缓冲区
50.             printf("producer number<%d> thread idetifier:%u put into buffer[%d].The buf
51.             print();
52.             Q.rear = (Q.rear+1) % BUFFSIZE;
53.
54.             pthread_mutex_unlock(&mutex);  //释放队列互斥
55.             sem_post(&full);   //释放满缓冲区
56.      }
57. }
58.
59. /* 消费者*/
60. void *consumer()
61. {
62.     int id=++consumer_id;
63.     while(1)
64.      {
65.             sleep(1);
66.
67.             sem_wait(&full);  //申请满缓冲区
68.             pthread_mutex_lock(&mutex);     //申请队列互斥
69.
70.             Q.buff[Q.front] = 0;            //从front所指向的缓冲区取产品
71.             printf("consumer number<%d> thread idetifier:%u get from buffer[%d].The buf
72.             print();
73.             Q.front = (Q.front+1) % BUFFSIZE;
74.
75.             pthread_mutex_unlock(&mutex);    //释放队列互斥
76.             sem_post(&empty);                //释放空缓冲区
77.      }
78. }
79.
80. int main()
81. {
82.     int M,N;  //M为生产者者数目，N为消费者数目
83.     printf("please input the producers number: ");
84.     scanf("%d",&M);
85.     printf("please input the consumers number: ");
86.     scanf("%d",&N);
87.     pthread_t id1[M];       //存储生产者线程ID
88.     pthread_t id2[N];       //存储消费者者线程ID
89.     int i;
90.     int ret1[M],ret2[N];  //存储创建生产者和消费者线程的返回值。
91.
92.
93. /*初始化empty和full私有信号量 */
94.     int ini1 = sem_init(&empty, 0, BUFFSIZE);//初始化empty信号量，大小为Buffersize ，初始化
    成功返回零，失败返回-1
95.     int ini2 = sem_init(&full, 0, 0);        //初始化full信号量，大小为
    0
96.     if((ini1 || ini2)!=0)   //
97.     {
98.              printf("sem init failed \n");
99.              exit(1);
100.    }
101.
102. /*初始化公有信号量 mutex*/
103.    int ini3 = pthread_mutex_init(&mutex, NULL);    //以默认方式创建互斥锁（快速互斥锁），初
    始化为未锁状态，成功返回0
104.    if(ini3 != 0)
105.    {
106.            printf("mutex init failed \n");
107.            exit(1);
108.    }
109.
110. /*创建生产者线程*/
111.    for(i = 0; i < M; i++)
112.    {
113.            ret1[i] = pthread_create(&id1[i], NULL, producer, (void *)(&i));    //循环创
    建生产者线程，成功返回0
114.            if(ret1[i] != 0)
115.            {
116.               printf("producer%d creation failed \n", i);
117.               exit(1);
```

```
118.            }
119.    }
120.
121. /*创建消费者线程*/
122.    for(i = 0; i < N; i++)
123.    {
124.            ret2[i] = pthread_create(&id2[i], NULL, consumer, NULL);          //循环创建消
费者线程，成功返回0
125.            if(ret2[i] != 0)
126.            {
127.               printf("consumer%d creation failed \n", i);
128.               exit(1);
129.            }
130.    }
131.
132. /*销毁线程*/
133.    for(i = 0; i < M; i++)
134.    {
135.            pthread_join(id1[i],NULL);   //对创建的生产者线程进行资源回收
136.    }
137.    for(i = 0; i < N; i++)
138.    {
139.            pthread_join(id2[i],NULL);   //对创建的消费者线程进行资源回收
140.    }
141.
142.    exit(0);
143. }
```

运行方式：

```
[html]
01. gcc mutex.c -o mutex -lpthread  <span style="font-family: Arial, Helvetica, sans-
serif;">#调用静态链接库</span>
02. ./mutex
```

有兴趣的，还可以看看这个程序：

http://www.cnblogs.com/lycheng/archive/2011/11/23/2260656.html

上一篇 Shell简介和理解
下一篇 通过打印机的ip安装打印机的驱动

猜你在找

- Ceph—分布式存储系统的另一个选择
- 4.7.存储类&作用域&生命周期&链接属性-C语言高级专题
- 《C语言/C++学习指南》Linux开发篇
- 4.5.数组&字符串&结构体&共用体&枚举-C语言高级专题
- C语言及程序设计提高
- linux网络编程用C语言实现的聊天程序异步通信
- linux网络编程用C语言实现的聊天程序异步通信
- linux下C语言socket网络编程简例
- linux下C语言socket网络编程简例
- linux下C语言socket网络编程简例

查看评论

暂无评论

您还没有登录,请[登录]或[注册]

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

全部主题 Hadoop AWS 移动游戏 Java Android iOS Swift 智能硬件 Docker OpenStack
VPN Spark ERP IE10 Eclipse CRM JavaScript 数据库 Ubuntu NFC WAP jQuery
BI HTML5 Spring Apache .NET API HTML SDK IIS Fedora XML LBS Unity
Splashtop UML components Windows Mobile Rails QEMU KDE Cassandra CloudStack
FTC coremail OPhone CouchBase 云计算 iOS6 Rackspace Web App SpringSide Maemo
Compuware 大数据 aptech Perl Tornado Ruby Hibernate ThinkPHP HBase Pure Solr
Angular Cloud Foundry Redis Scala Django Bootstrap