**生产者——消费者问题**

**卢冠男**

**71113437**

**一、实验目的**

通过实验，熟悉linux下Pthread api的调用，同时解决经典同步问题—生产者消费者问题。

**二、实验内容**

借用ubuntu，利用Pthread API提供的信号量机制，编写应用程序实现生产者——消费者问题。

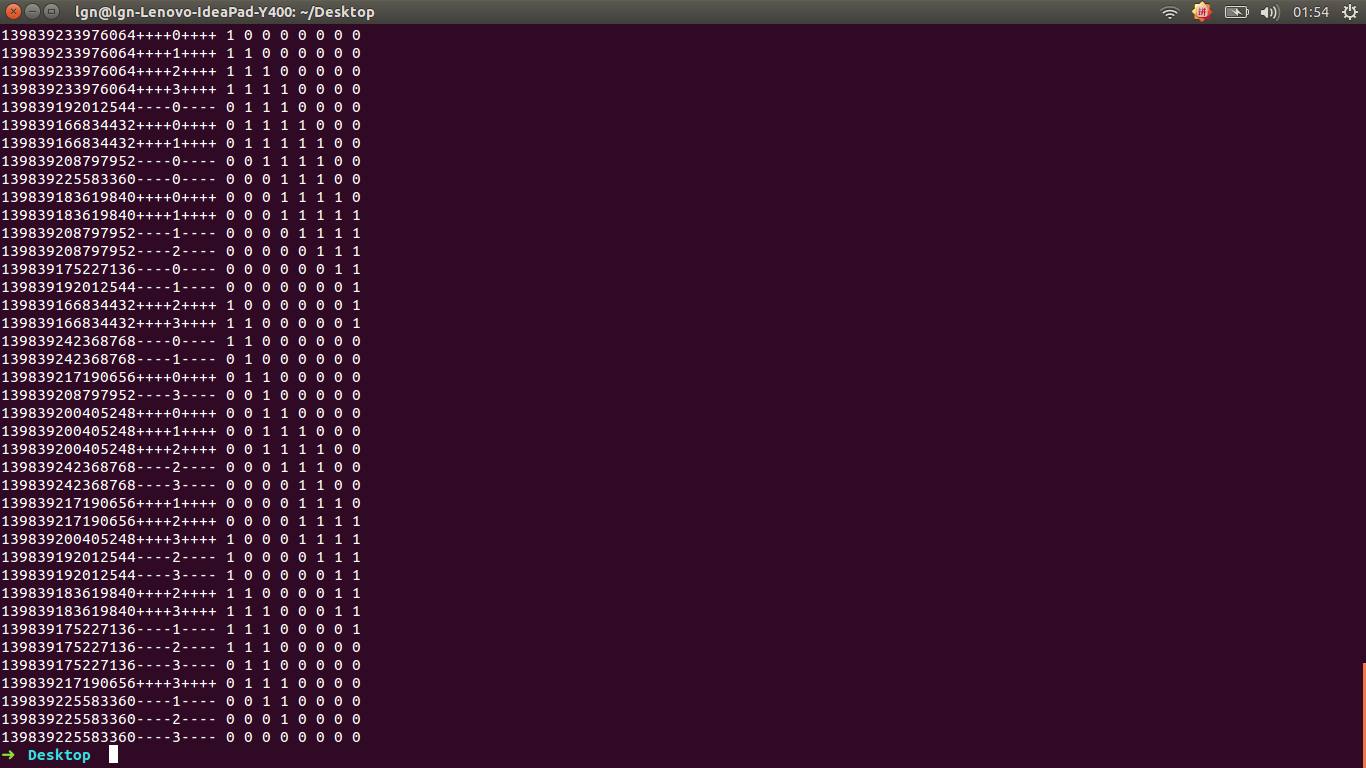
**三、实验步骤**

①声明三个信号量，互斥信号量mutex，计数信号量empty，技术信号量full，缓冲区buffer数组。

②编写生产者和消费者代码

③创建多线程，模拟生产和消费。

**四、程序代码和运行结果**



#include <pthread.h>

#include <semaphore.h>

#include <stdio.h>

#include <stdlib.h>

#define BUFFER\_SIZE 8//缓冲区大小为8

char \*buffer;

sem\_t mutex,empty,full;//三个信号量，互斥信号量mutex，技术信号量empty和full

int x,y;//生产者和消费者在buffer中下标

void output()//输出buffer数组

{

int i;

for(i=0;i<BUFFER\_SIZE;i++)

{

printf("%c",buffer[i]);

printf(" ");

}

printf("\n");

}

void \*produce(void \*ptr)//生产者函数

{

int j;

j=0;

do

{

sem\_wait(&empty);//buffer有空余部分，可以生产，并减一

sem\_wait(&mutex);//形成互斥访问，只能一个线程生产

printf("%lu%s%d%s",pthread\_self(),"++++",j,"++++ ");//输出当前线程的id号，以及正在执行的次数

buffer[(x++)%BUFFER\_SIZE]='1';//生产就赋值1

output();//输出buffer

j++;

sem\_post(&mutex);//取消互斥

sem\_post(&full);//生成完毕，增加一个可以消费量。

}while (j!=4);//每个线程可以做4次

}

void \*consume(void \*ptr)//消费者函数

{

int j;

j=0;

do

{

sem\_wait(&full);//可以消费的量减一

sem\_wait(&mutex);//互斥访问，只能一个线程消费

printf("%lu%s%d%s",pthread\_self(),"----",j,"---- ");

buffer[(y++)%BUFFER\_SIZE]='0';//消费时，赋值为0

output();//输出buffer值

j++;

sem\_post(&mutex);//取消互斥，其他线程可以消费

sem\_post(&empty);//空余量加一

}while (j!=4);//每个线程可以消费4次

}

int main()

{

int i;

x=0;

y=0;

buffer=(char\*)malloc(BUFFER\_SIZE\*sizeof(char\*));

for(i=0;i<BUFFER\_SIZE;i++)//初始化buffer数组，默认为0

{

buffer[i]='0';

}

//semaphore

sem\_init(&mutex,1,1);//初始化互斥信号量mutex为1

sem\_init(&empty,0,BUFFER\_SIZE);//初始化计数信号量empty为BUFFER\_SIZE

sem\_init(&full,0,0);//初始化计数信号量full为0

//multipthread

pthread\_t tid[10];

pthread\_attr\_t attr;

pthread\_attr\_init(&attr);

//创建5个生产者线程和5个消费者线程

for(i=0;i<5;i++)

{

pthread\_create(&tid[i],&attr,consume,NULL);

pthread\_create(&tid[i+5],&attr,produce,NULL);

}

//让每个线程在主线程main执行前全部执行完毕。

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

{

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

}

return 0;

}

**六、实验体会**

利用信号量确实可以很好的解决生产者消费者问题，不会产生忙等等问题。