生产者与消费者实验的改进

1. 改进思路

首先对实例的程序进行分析，示例程序中实现的同步主要是依靠信号量机制去实现的，采用mutex作为互斥信号量，full和empty都是同步信号量，在这种方式中，最重要的就是互斥信号量和同步信号量在进入临界区的顺序是不可颠倒的，所以在这我选择另一种方式，采用条件变量和互斥锁的机制实现。

1. 改进程序

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <pthread.h>

#include <errno.h>

#include <sys/ipc.h>

#include <semaphore.h>

#include <fcntl.h>

#include "Queue.h"

#include "const.h"

#define N 5

time\_t end\_time;

*//互斥锁*

pthread\_mutex\_t mutex;

*//条件变量*

pthread\_cond\_t conc,conp;

int fd;

Queue \* qt;

*//缓冲区*

Elemtype p;

void\* consumer(void \*arg);

void\* productor(void \*arg);

int main(int argc, char \*argv[])

{

pthread\_t id1,id2,id3,id4,id5;

pthread\_t mon\_th\_id;

int ret;

end\_time = time(NULL)+30;

qt = InitQueue();

p.lNumber = 1000;

*//初始化条件变量*

pthread\_mutex\_init(&mutex,0);

pthread\_cond\_init(&conc,0);

pthread\_cond\_init(&conp,0);

pthread\_create(&id1,NULL,&productor, NULL);

pthread\_create(&id3,NULL,&productor, NULL);

pthread\_create(&id2,NULL,&consumer, NULL);

pthread\_create(&id5,NULL,&productor, NULL);

pthread\_create(&id4,NULL,&consumer, NULL);

pthread\_join(id1,NULL);

pthread\_join(id2,NULL);

pthread\_join(id3,NULL);

pthread\_join(id4,NULL);

pthread\_join(id5,NULL);

pthread\_mutex\_destroy(&mutex);*//销毁锁*

return 0;

}

void\* productor(void \*arg)

*//生产者线程*

{

int i,nwrite;

while(time(NULL) < end\_time)

{

*// p(empty)*

pthread\_mutex\_lock(&mutex);

*// p(mutex)*

if(TRUE==QueueFull(\*qt))

*//队满*

{

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

printf("Productor:buffer is full ,please try to write later.\n");

}

else

{

EnQueue(qt,p);

printf("Productor:write [%d] to buffer \n",p.lNumber);

p.lNumber++;

}

pthread\_cond\_signal(&conc);

*//v(full)*

pthread\_mutex\_unlock(&mutex);

*//v(mutex)*

sleep(1);

}

}

*//生产者productor*

void\* consumer(void \*arg)

*//消费者线程*

{

int nolock=0;

int ret,nread;

Elemtype p2;

while((time(NULL) < end\_time)||(FALSE==(QueueEmpty(\*qt))))

{

*//p(full)*

pthread\_mutex\_lock(&mutex);

*//p(mutex)*

if(TRUE==QueueEmpty(\*qt))

*//队空*

{

printf("Consumer:the buffer is empty,please try to read later.\n");

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

}

else

{

DeQueue(qt,&p2);

printf("Consumer:read [%d] from buffer.\n",p2.lNumber);

}

pthread\_cond\_signal(&conp);

*//v(empty)*

pthread\_mutex\_unlock(&mutex);

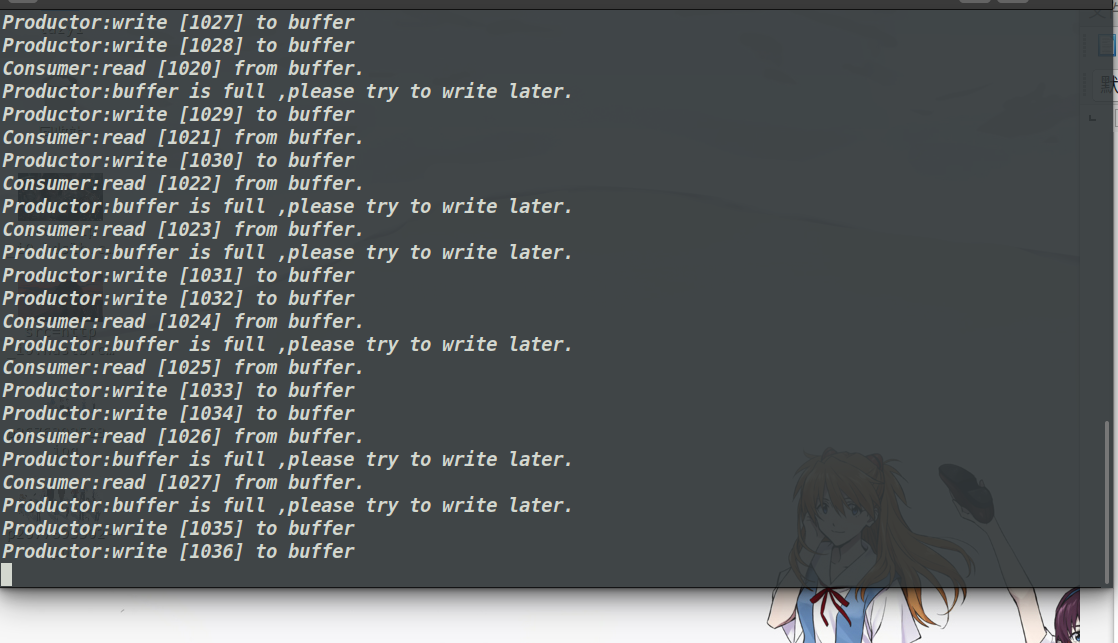
*//v(mutex)*

sleep(2);

}

}

1. 运行截图



1. 程序设计简单说明

采用pthreads使用的基本同步技术，定义互斥锁和条件变量，并对他们进行init操作，当一个线程进入临界区时，通过pthread\_mutex\_lock使它获得互斥锁，如果当前互斥锁在其他线程手中，那么这个线程就会被阻塞，为了避免死锁的出现，采用条件变量cond来唤醒锁住的线程，当满足阻塞的条件时，也就是队满或者队空时，通过pthread\_cond\_wait()对条件变量和互斥量再次加锁而使得这个线程暂时阻塞;而另一个线程使条件成立而发出信号，这里生产者消费者只要没锁住，就都调用pthread\_cond\_signal();来对另一个线程的条件变量解锁，因为此时队满和队空的条件都不再存在了，其中这些函数调用都是原子操作，所以不会发生死锁的现象，但是使用解条件变量操作之后，必须立即释放互斥锁，否则pthread\_cond\_wait一样要阻塞