**实验三 生产者-消费者算法模拟实验**

一、实验目的

1、掌握基本的互斥与同步算法，进一步理解“生产者-消费者”模型。

2、通过对“生产者-消费者”问题编程实现，了解线程创建、同步信号量、互斥信号量、临界区的创建和使用，初步了解并发程序设计方法。

3、进一步理解P、V原语和信号量在线程互斥和同步机制中的运用。

二、实验环境

硬件环境：计算机一台，局域网环境；

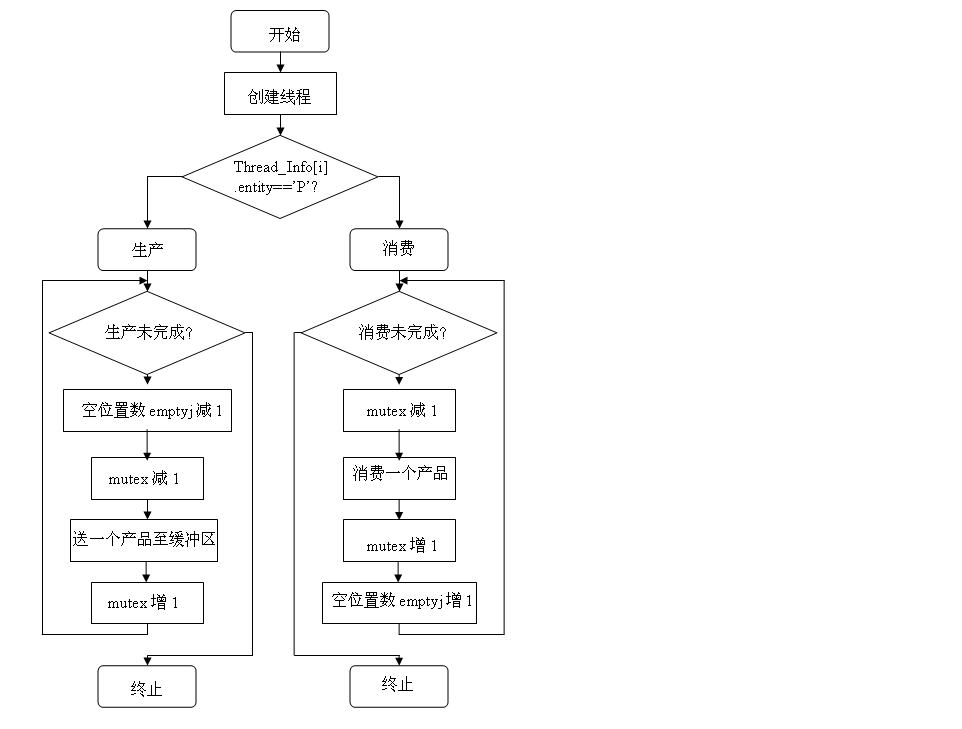
软件环境：Windows 2000 Professional、Visual C++ 6.0专业版或企业版。

三、实验内容和步骤

1、在本次实验开始，以“生产者-消费者”模型为依据，提供了一个多线程“生产者-消费者”实例，有部分源程序代码，要求读者分析已编制的一个“生产者-消费者”实例，并将其缺失的程序代码补充完整，然后调试这段程序，得出最终的结果，并分析结果，得出相应的结论。

尝试改变一些参数，例如：改变缓冲区数、增加（减少）线程数、改变延迟数、增加（减少）生产者进程、增加（减少）消费者进程、改变消费者进程的请求序列等内容，考察这些改变对于运行结果的影响。

2、程序流程图如下图所示：

****

/\*生产者/消费者\*/

#include<stdio.h>

#include<malloc.h>

#include<iostream.h>

struct PCB

{

int flag; //1为生产者,2为消费者

int numLabel;

};

typedef struct QNode

{

PCB data;//数据域

struct QNode\* next;//指针域

}QNode, \*QueuePtr;

typedef struct

{

QueuePtr front;//队头指针

QueuePtr rear;//队尾指针

}LinkQueue;

typedef struct LNode

{

QueuePtr data;

struct LNode \*next;

}LNode, \*LinkList;

void QueueInit(LinkQueue &Q)//初始化队列

{

Q.front = Q.rear = (QueuePtr)malloc(sizeof(QNode));

Q.front->next = NULL;

}

void LinkListInit(LinkList& L)//初始化链表

{

L = (LinkList)malloc(sizeof(LNode));

}

void EnQueue(LinkQueue &Q, QueuePtr p)//入队

{

p->next = NULL;

Q.rear->next = p;

Q.rear = p;

}

QueuePtr DeQueue(LinkQueue &Q)

//出队

{

QueuePtr p = Q.front->next;

Q.front->next = p->next;

if(Q.rear == p)Q.rear =Q.front;

return p;

}

void outqueue(LinkQueue& Q)

{

QueuePtr p = Q.front->next;

Q.front->next = p->next;

if(Q.rear == p)Q.rear =Q.front;

}

void LinkListInsert(LinkList &L, QueuePtr e)//进入链表

{

LinkList p = L;

LinkList q = (LinkList)malloc(sizeof(LNode));

while(p->next)p++;

q->data = e;

q->next = NULL;

p->next = q;

}

void processproc(LinkQueue &Q)//创建进程进入初始队列

{

int processNum = 0;

cout<<"请输入进程的个数:";//确定进程个数，默认为0

cin>>processNum;

for(int i = 0; i < processNum; i++)

{

cout<<"输入第"<<i+1<<"个进程的种类,如果是生产者请输入(1)/如果是消费者请输入(2)";

struct PCB pcb;

cin>>pcb.flag;//输入进程的种类

pcb.numLabel = i+1;//进程序号赋值

QueuePtr p = (QueuePtr)malloc(sizeof(QNode));

p->data = pcb;

EnQueue(Q, p);

}

}

bool HasElement(LinkQueue Q)//判断队列是否为空

{

if(Q.front == Q.rear)

return 0;

else

return 1;

}

int ProduceRun(int &full, int BufferSize)//运行生产者进程

{

if(full < BufferSize)

{

full++;

return 1;

}

return 0;

}

int ConsumeRun(int &full, int BufferSize)//运行消费者进程

{

if(full > 0)

{

full--;

return 1;

}

return 0;

}

void DisPlay(LinkQueue Q)//打印队列

{

QueuePtr p = Q.front;

while(p->next)

{

cout<<"进程"<<p->next->data.numLabel<<endl;

p = p->next;

}

}

void main()

{

int BufferSize;//设置缓冲区大小

cout<<"请设置缓冲区的大小: ";

cin>>BufferSize;

int full = 0;//当前缓冲区中的进程数目

int temp = 1;

LinkList over;//用于收集已经运行结束的进程

LinkListInit(over);

LinkQueue ReadyQueue;//就绪队列

LinkQueue ProducerWaitQueue;//生产者等待队列

LinkQueue ConsumerWaitQueue;//消费者等待队列

//初始化

QueueInit(ReadyQueue);

QueueInit(ProducerWaitQueue);

QueueInit(ConsumerWaitQueue);

while(temp)//死循环

{

processproc(ReadyQueue);//创建进程进入就绪队列

bool element=HasElement(ReadyQueue);//判断队列是否为空

while(element)//当它不是空的

{

cout<<"进程"<<ReadyQueue.front->next->data.numLabel<<"申请运行,他是一个";

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if(ReadyQueue.front->next->data.flag == 1)//如果它是一个生产者

{

cout<<"生产者"<<endl;

if(ProduceRun(full, BufferSize) == 1)//判断缓存区是否还有空间

{

cout<<"进程"<<ReadyQueue.front->next->data.numLabel<<"执行完毕"<<endl;

LinkListInsert(over, DeQueue(ReadyQueue));//运行结束，进入over链表

/\*\*\*\*\*\*\*\*\*\*\*/

if(HasElement(ConsumerWaitQueue))

{

ConsumeRun(full, BufferSize);

outqueue(ConsumerWaitQueue);

}

/\*\*\*\*\*\*\*\*\*\*\*/

if(HasElement(ProducerWaitQueue))//检查生产者等待队列，激活队列中的进程进入就绪队列

EnQueue(ReadyQueue, DeQueue(ProducerWaitQueue));

}

else if(ProduceRun(full, BufferSize) == 0)//就绪队列出一个元素，进入生产者等待队列

{

EnQueue(ProducerWaitQueue, DeQueue(ReadyQueue));

}

}

else

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

{//如果它是一个消费者

cout<<"消费者"<<endl;

if(ConsumeRun(full, BufferSize) == 1)//判断缓存区是否存在进程

{

LinkListInsert(over, DeQueue(ReadyQueue));//运行结束，进入over链表

if(HasElement(ConsumerWaitQueue)) //检查消费者等待队列，激活队列中的进程进入就绪队列

EnQueue(ReadyQueue, DeQueue(ConsumerWaitQueue));

/\*\*\*\*\*\*\*\*\*\*\*/

if(HasElement(ProducerWaitQueue))

{

ProduceRun(full, BufferSize);

outqueue(ProducerWaitQueue);

}

/\*\*\*\*\*\*\*\*\*\*\*/

}

else if(ConsumeRun(full, BufferSize) == 0)//若没有进程，则进入消费者等待队列

{

EnQueue(ConsumerWaitQueue, DeQueue(ReadyQueue));

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*输出就绪队列\*/

element = HasElement(ReadyQueue);

if(HasElement(ReadyQueue))

{

cout<<"就绪队列中有下列进程:"<<endl;

DisPlay(ReadyQueue);

}

else cout<<"就绪队列里面没有进程"<<endl;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*输出生产者等待队列\*/

if(HasElement(ProducerWaitQueue))

{

cout<<"生产者等待队列中有下列进程:"<<endl;

DisPlay(ProducerWaitQueue);

}

else cout<<"生产者等待队列中没有进程"<<endl;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*输出消费者等待队列\*/

if(HasElement(ConsumerWaitQueue))

{

cout<<"消费者等待队列中有下列进程:"<<endl;

DisPlay(ConsumerWaitQueue);

}

else cout<<"消费者等待队列中没有进程"<<endl;

getchar();

}

cout<<"是否继续?Y (输入:1)/N( 输入:0)";

cin>>temp;

}

}