

操作系统原理

实验报告

实验题目：实验 3：进程同步与互斥

实验时间：2022 年 11 月 25 日

学号姓名：22129390 吕强

实验目的和要求

- 1.理解生产者/消费者模型及其同步/互斥规则。
- 2.了解 Windows 同步对象及其特性。
- 3.熟悉实验环境，掌握相关 API 的使用方法。
- 4.设计程序，实现生产者/消费者进程的同步与互斥

实验内容

在 Visual C++ 6.0 集成开发环境下使用 C 或 C++语言，以生产者/消费者模型为依据，创建一个控制台进程，在该进程中创建 n 个进程模拟生产者和消费者，实现进程的同步与互斥。

1. 在源程序上写出注释。
2. 画出程序流程图。
3. 调试程序并写出运行结果。

实验原理与提示

进程数据结构：每个进程有一个进程控制块（PCB）表示。进程控制块可以包含如下信息：进程类型标号、进程系统号、进程状态（本程序未用）、进程产品（字符）、进程链指针等等。系统开辟了一个缓冲区，大小由 buffersize 指定。程序中有三个链队列，一个链表。一个就绪队列（ready），两个等待队列：生产者等待队列（producer）；消费者等待队列（consumer）。一个链表（over），用于收集已经运行结束的进程。

- 本程序通过函数模拟信号量的原子操作。
- 算法的文字描述：
 - (1) 由用户指定要产生的进程及其类别，存入就绪队列。
 - (2) 调度程序从就绪队列中提取一个就绪进程运行，如果申请的资源不存在则进入相应的等待队列，调度程序调度就绪队列中的下一个进程；进程运行结束时，会检查相应的等待队列，激活等待队列中的进程进入就绪队列；运行结束的进程进入 over 链表。重复这一过程直至就绪队列为空。
 - (3) 程序询问是否要继续？如果要继续转至 (1) 开始执行，否则退出程序。

实验程序

```
#include <stdio.h>
#include <malloc.h>
#define buffersize 5//假设有 5 个缓冲区
int processnum=0;//初始化产品数量
struct pcb /* 定义进程控制块 PCB */
{
    int flag;
    int numlabel;
    char product;
    char state;
    struct pcb* processlink;
}*exe=NULL,*over=NULL;
typedef struct pcb PCB;
PCB* readyhead=NULL,* readytail=NULL;
PCB* consumerhead=NULL,* consumertail=NULL;
PCB* producerhead=NULL,* producertail=NULL;
int productnum=0;//产品数量
int full=0,empty=buffersize;//信号量
char buffer[buffersize];//缓冲区
int bufferpoint=0;//缓冲区指针
void linklist(PCB* p,PCB* listhead)//创建就绪队列
```

```

{
    PCB* cursor=listhead;
    while(cursor->processlink!=NULL){
        cursor=cursor->processlink;
    }
    cursor->processlink=p;
}
void freelink(PCB* linkhead)
{
    PCB* p;
    while(linkhead!=NULL)
    {
        p=linkhead;
        linkhead=linkhead->processlink;
        free(p);
    }
}
void linkqueue(PCB* process,PCB** tail)//初始化队列
{
    if((*tail)!=NULL)
    {
        (*tail)->processlink=process;
        (*tail)=process;
    }
    else {printf("队列未初始化！");}
}
PCB* getq(PCB* head,PCB** tail)
{
    PCB* p;
    p=head->processlink;
    if(p!=NULL)
    {
        head->processlink=p->processlink;
        p->processlink=NULL;
        if( head->processlink ==NULL )    (*tail)=head;
    }
    else return NULL;
    return p;
}
bool processproc()//初始化进程
{
    int i,f,num;
    char ch;
    PCB* p=NULL;

```

```

PCB** p1=NULL;
printf("\n 请输入希望产生的进程个数: ");
scanf("%d",&num);
getchar();
for(i=0;i<num;i++)
{
    printf("\n 请输入您要产生的进程:输入 1 为生产者进程;输入 2 为消费者进程\n");
    scanf("%d",&f);
    getchar();
    p=(PCB*)malloc(sizeof(PCB));
    if( !p) {printf("内存分配失败"); return false; }
    p->flag=f;
    processnum++;
    p->numlabel=processnum;
    p->state='w';
    p->processlink=NULL;
    if(p->flag==1)
    { printf("您要产生的进程是生产者，它是第%d 个进程。请您输入您要该进程产
生的字符: \n",processnum);
        scanf("%c",&ch);
        getchar();
        p->product=ch;
        productnum++;
        printf("您要该进程产生的字符是%c \n",p->product);
    }
    else { printf("您要产生的进程是消费者，它是第%d 个进程。 \n",p->numlabel);}
    linkqueue(p,&readytail);
}
return true;
}

bool hasElement(PCB* pro)//判断队列中是否有进程存在
{
    if(pro->processlink==NULL) return false;
    else return true;
}

bool waitempty()//判断生产者等待队列是否为空
{
    if(empty<=0)
    {
        printf("进程 %d: 缓冲区存数，缓冲区满，该进程进入生产者等待队列
\n",exe->numlabel);
        linkqueue(exe,&producertail);
        return false;
    }
}

```

```

        else{ empty--; return true; }
    }
void signaleempty()//唤醒生产者进程
{
    PCB* p;
    if(hasElement(producerhead)){
        p=getq(producerhead,&producertail);
        linkqueue(p,&readytail);
        printf("等待中的生产者进程进入就绪队列，它的进程号是%d\n",p->numlabel);
    }
    empty++;
}
bool waitfull()//判断消费者等待队列是否为满
{
    if(full<=0)
    {
        printf("进程 %d: 缓冲区取数，缓冲区空，该进程进入消费者等待队列\n",exe->numlabel);
        linkqueue(exe,&consumertail);
        return false;
    }
    else{ full--; return true;}
}
void signalfull()//唤醒消费者进程
{
    PCB* p;
    if(hasElement(consumerhead)){
        p=getq(consumerhead,&consumertail);
        linkqueue(p,&readytail);
        printf("等待中的消费者进程进入就绪队列，它的进程号是%d\n",p->numlabel);
    }
    full++;
}
void producerrun()//生产者进程
{
    if(!waitempty()) return;
    printf("进程%d 开始向缓冲区存数%c\n",exe->numlabel,exe->product);
    buffer[bufferpoint]=exe->product;
    bufferpoint++;
    printf("进程%d 向缓冲区存数操作结束\n",exe->numlabel);
    signalfull();
    linklist(exe,over);
}
void comsuerrun()//消费者进程

```

```

{
    if(!waitfull()) return;
    printf("进程%d 开始向缓冲区取数\n",exe->numlabel);
    exe->product=buffer[bufferpoint-1];
    bufferpoint--;
    printf("进程%d 向缓冲区取数操作结束，取数是%c\n",exe->numlabel,exe->product);
    signalempy();
    linklist(exe,over);
}
void display(PCB* p)//显示进程
{
    p=p->processlink;
    while(p!=NULL){
        printf("进程%d，它是一个",p->numlabel);
        p->flag==1? printf("生产者\n"):printf("消费者\n");
        p=p->processlink;
    }
}
void main()
{
    char terminate;
    bool element;
    printf("你想开始程序吗?(y/n)");
    scanf("%c",&terminate);
    getchar();
    readyhead=(PCB*)malloc(sizeof(PCB));//初始化队列
    if(readyhead==NULL) return;//若队列未成功初始化则结束程序
    readytail=readyhead;
    readyhead->flag=3;
    readyhead->numlabel=processnum;
    readyhead->state='w';
    readyhead->processlink=NULL;
    consumerhead=(PCB*)malloc(sizeof(PCB));//初始化消费者等待队列
    if(consumerhead==NULL) return;//若消费者等待队列未成功初始化则结束程序
    consumertail=consumerhead;
    consumerhead->processlink=NULL;
    consumerhead->flag=4;
    consumerhead->numlabel=processnum;
    consumerhead->state='w';
    consumerhead->processlink=NULL;
    producerhead=(PCB*)malloc(sizeof(PCB));//初始化生产者等待队列
    if(producerhead==NULL) return;//若生产者等待队列未成功初始化则结束程序
    producertail=producerhead;
    producerhead->processlink=NULL;
}

```

```

producerhead->flag=5;
producerhead->numlabel=processnum;
producerhead->state='w';
producerhead->processlink=NULL;
over=(PCB*)malloc(sizeof(PCB));
if(over==NULL) return;
over->processlink=NULL;
while(terminate=='y')
{
    if(!processproc()) break;
    element=hasElement(readyhead);
    while(element){
        exe=getq(readyhead,&readytail);
        printf("进程%d 申请运行，它是一个",exe->numlabel);
        exe->flag==1? printf("生产者\n"):printf("消费者\n");
        if(exe->flag==1) producerrun();
        else consuerrun();
        element=hasElement(readyhead);
    }//生产者/消费者进程申请运行,就绪状态转为执行状态
    printf("就绪队列没有进程\n");
    if(hasElement(consumerhead))//判断消费者等待队列中是否有进程
    {
        printf("消费者等待队列中有进程:\n");
        display(consumerhead);
    }
    else { printf("消费者等待队列中没有进程\n"); }
    if(hasElement(producerhead))//判断生产者等待队列中是否有进程
    { printf("生产者等待队列中有进程:\n");
        display(producerhead);
    }
    else {
        printf("生产者等待队列中没有进程\n");
    }
    printf("你想继续吗?(press 'y' for on)");
    scanf("%c",&terminate);
    getchar();
}
printf("\n\n 进程模拟完成.\n");
freelink(over);//释放空间
over=NULL;
freelink(readyhead);
readyhead=NULL;
readytail=NULL;
freelink(consumerhead);

```

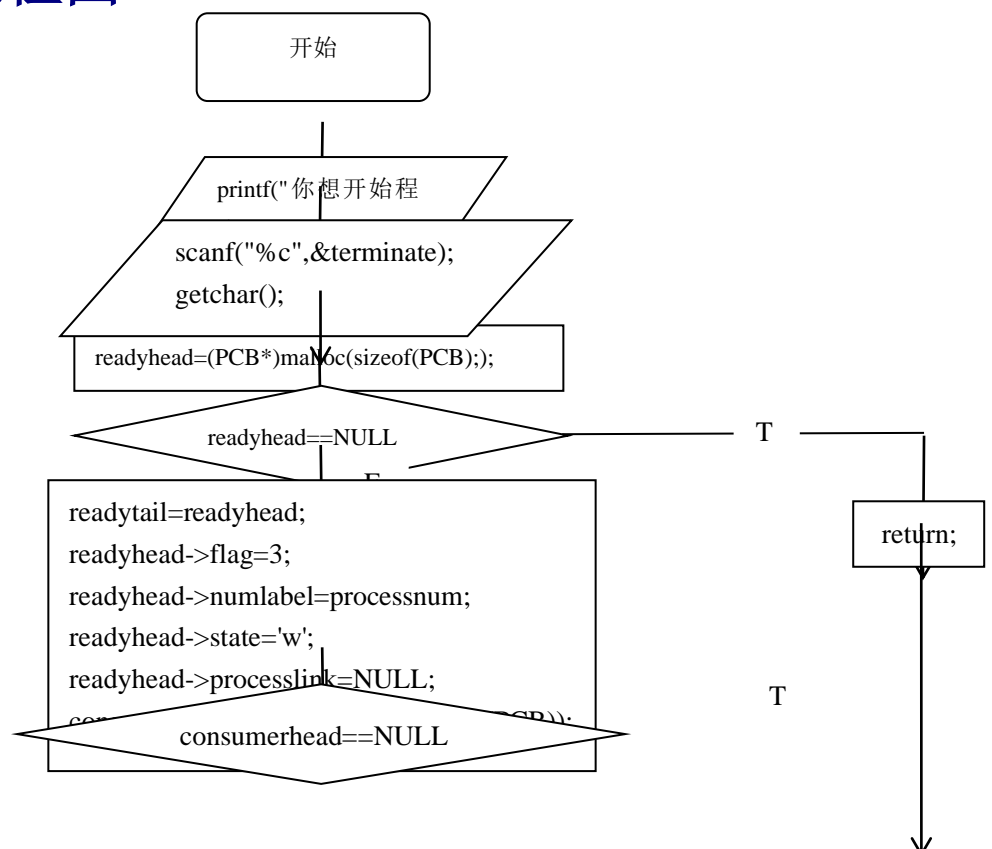


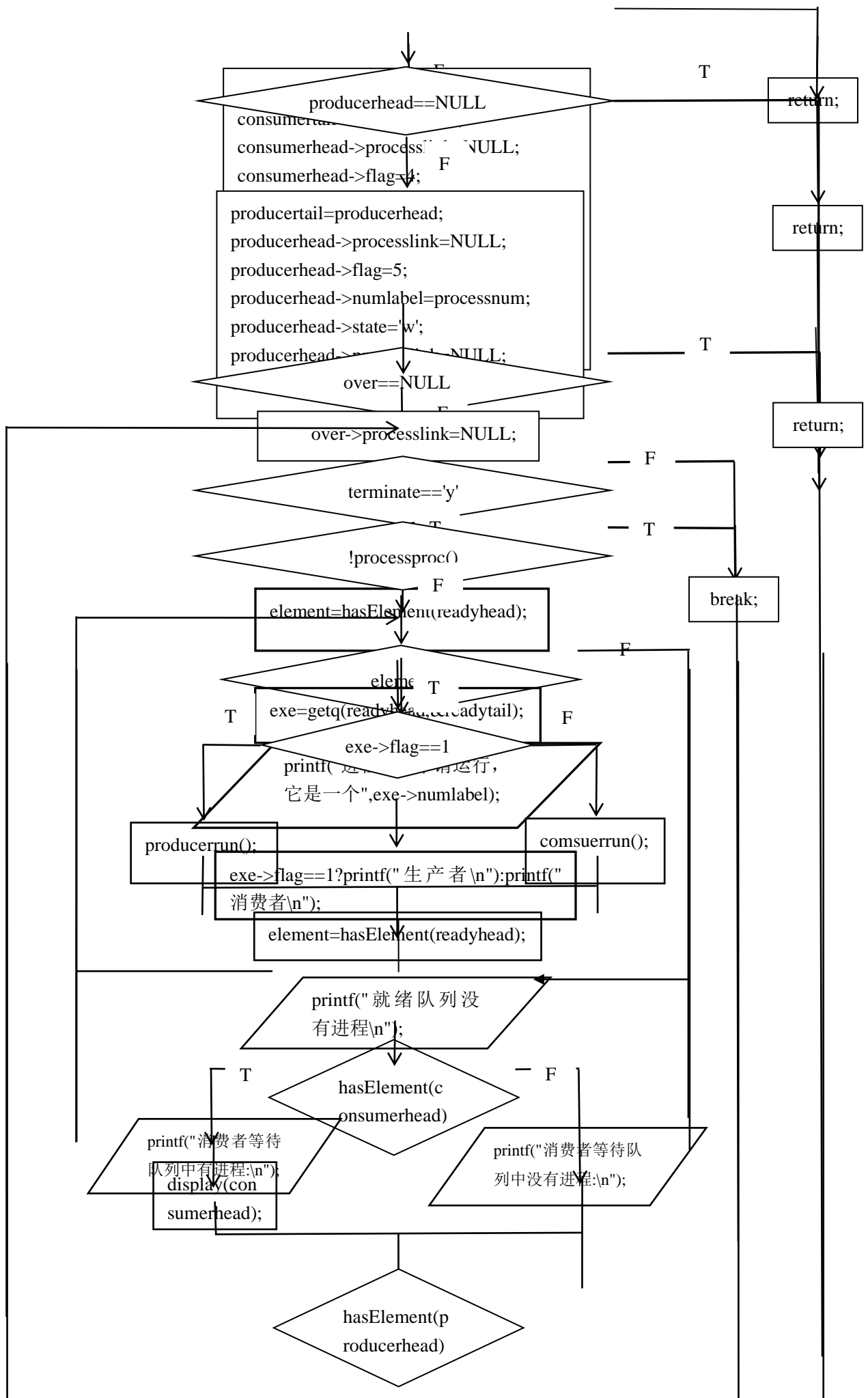
```

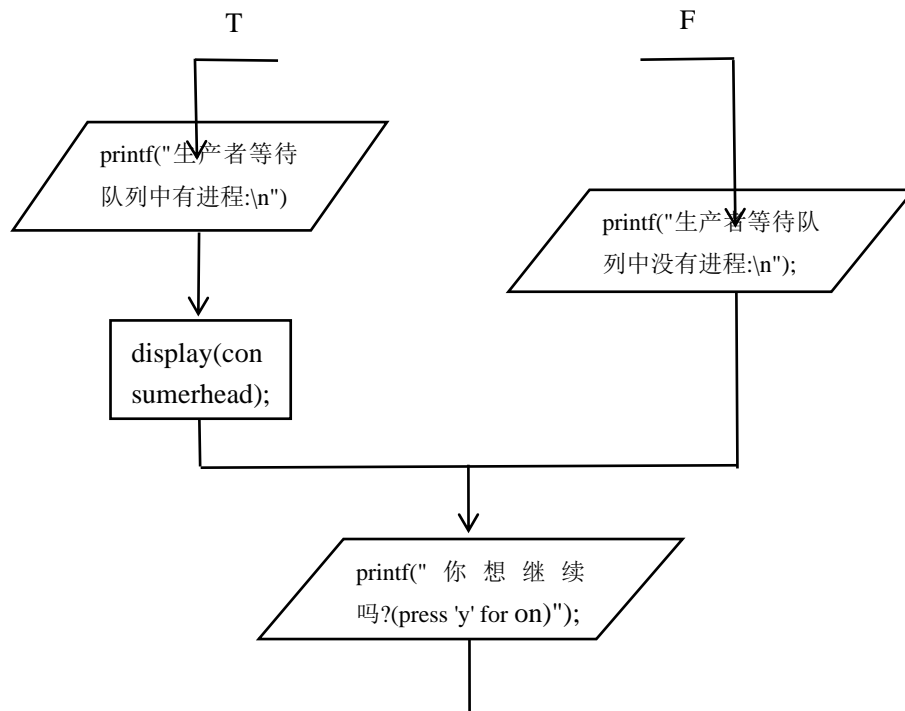
consumerhead=NULL;
consumertail=NULL;
freelink(producerhead);
producerhead=NULL;
producertail=NULL;
getchar();
}

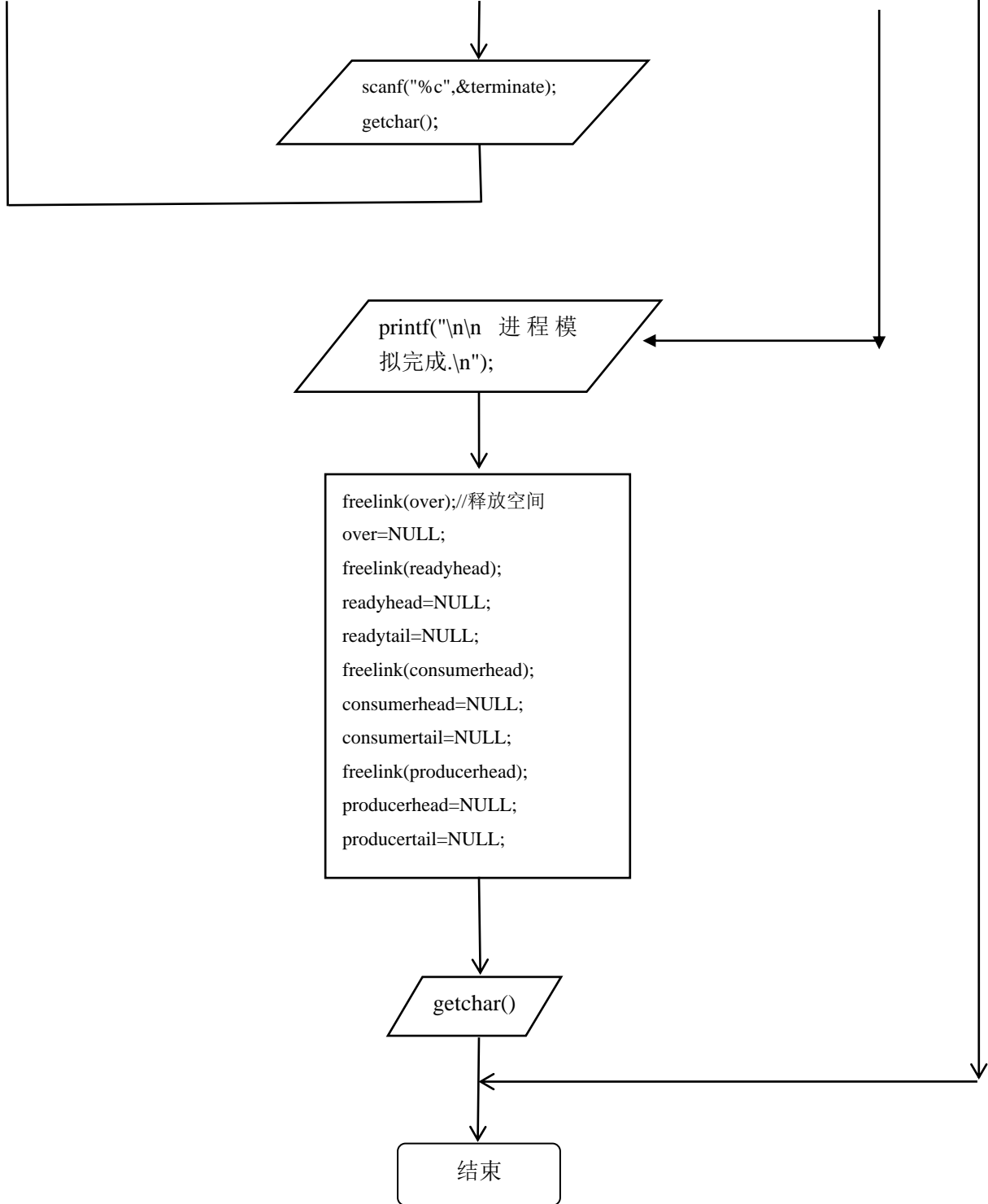
```

主函数流程图

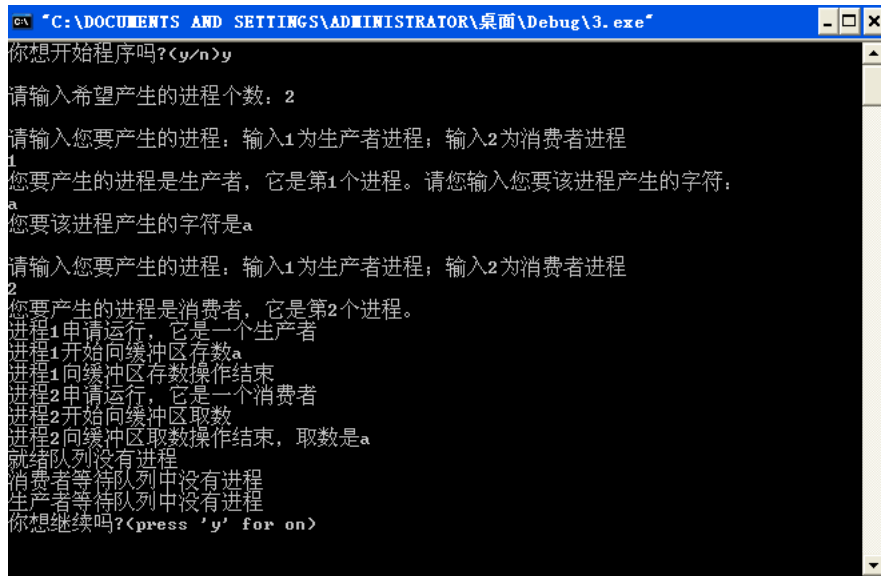








运行结果



```
C:\DOCUMENTS AND SETTINGS\ADMINISTRATOR\桌面\Debug\3. exe
你想开始程序吗?<y/n>y
请输入希望产生的进程个数: 2
请输入您要产生的进程: 输入1为生产者进程; 输入2为消费者进程
1
您要产生的进程是生产者, 它是第1个进程。请您输入您要该进程产生的字符:
a
您要该进程产生的字符是a
请输入您要产生的进程: 输入1为生产者进程; 输入2为消费者进程
2
您要产生的进程是消费者, 它是第2个进程。
进程1申请运行, 它是一个生产者
进程1开始向缓冲区存数a
进程1向缓冲区存数操作结束
进程2申请运行, 它是一个消费者
进程2开始向缓冲区取数
进程2向缓冲区取数操作结束, 取数是a
就绪队列没有进程
消费者等待队列中没有进程
生产者等待队列中没有进程
你想继续吗?<press 'y' for on>
```

分析讨论

学会了多种循环嵌套在一起流程图如何实现, 理解消费者/生产者模型同步互斥问题原则, 本次实验需要多次反复惊醒实验, 以保证运行结果的准确性。

教师评语及成绩