
云南大学数学与统计学实验教学中心

实验报告

课程名称：操作系统实验	学期：2015~2016 学年上学期	成绩：
指导教师：彭程	学生姓名：金洋	学号：20131910023
实验名称：并发程序的存储管理模拟		
实验编号：六	实验日期：12 月 25 日	实验学时：2
学院：数学与统计学院	专业：信息与计算科学	年级：2013 级

一、实验目的

1. 掌握存储管理机制；
2. 用 c 语言实现并发程序的存储管理模拟；

二、实验内容

1. 采用链表进行空闲区和已分配空间的管理，该存储为连续空间上的存储；
2. 初始的内存大小为 100000 字节；
3. 采用最佳适应分配算法分配内存；
4. 构造多个应用程序，用随机函数生成到达时间序列、执行时间序列、所需内存大小序列（满足泊松分布或指数分布），以此来模拟并发程序的存储管理；

三、实验环境

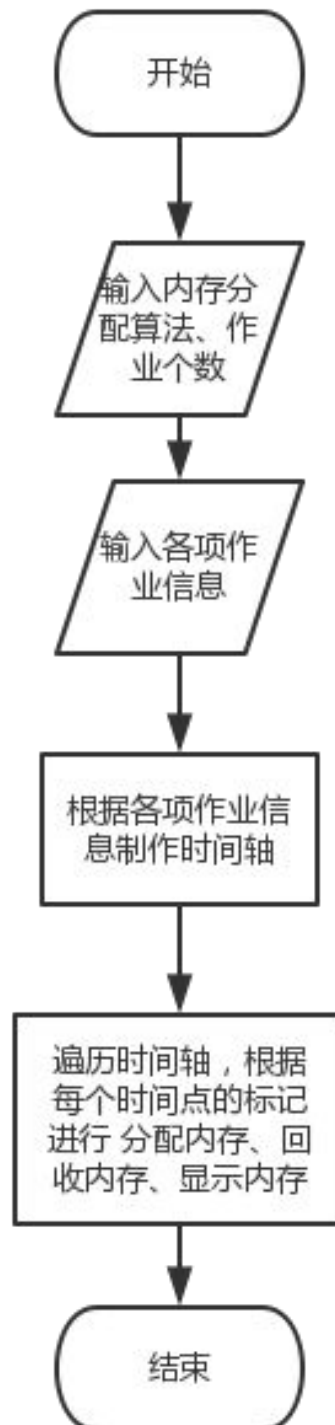
平台：Visual C++ 6.0

语言：C 语言

四、算法介绍

本实验利用 MATLAB 产生满足特定概率分布的数据来模拟真实作业的信息，将为作业分配内存、回收内存的时间标记到时间轴上，通过遍历时间轴来对相应的事件作出处理，以此实现并发程序的内存管理模拟。在作业调度方式的选择上，本实验通过时间轴处理，故使用的是先到先服务调度方式。

1. 程序总流程图



基本数据结构:

①/*时间轴类型*/

```
struct TimeLine
{
    char time[MAXTIME];//记录 0-9999 这段时间内发生的分配内存'D'或回收内存'R'事件;
    int job_No[MAXTIME];//记录每个 D 或 R 对应的作业编号
};
struct TimeLine TLine;//时间轴
```

②/*节点类型*/

```
struct ElementType
{
    int jobNo;//作业编号，作业名
    int statu;//显示状态
    int startAddress;//起始地址
    int size;//大小

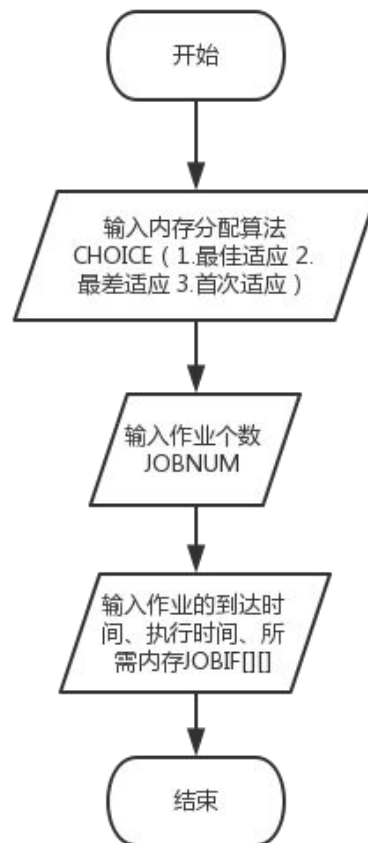
    int startTime;//起始时间
    int runTime;//运行时间

    struct ElementType *pre;
    struct ElementType *next;
};
struct ElementType *HEAD,*TAIL;//分配链表的头结点
```

③宏定义信息

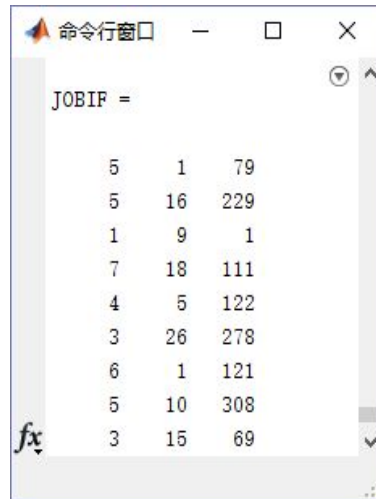
```
#define FREE 0//空闲 /*表示每个物理块的状态*/
#define BUSY 1//占用
#define MAXLENGTH 100000//最大内存空间 100000 Byte
#define MAXJOBNUM 1000//最大作业项数
#define MAXTIME 10000//最大时间
#define LEN sizeof(struct ElementType)
```

2. 输入模块 input()



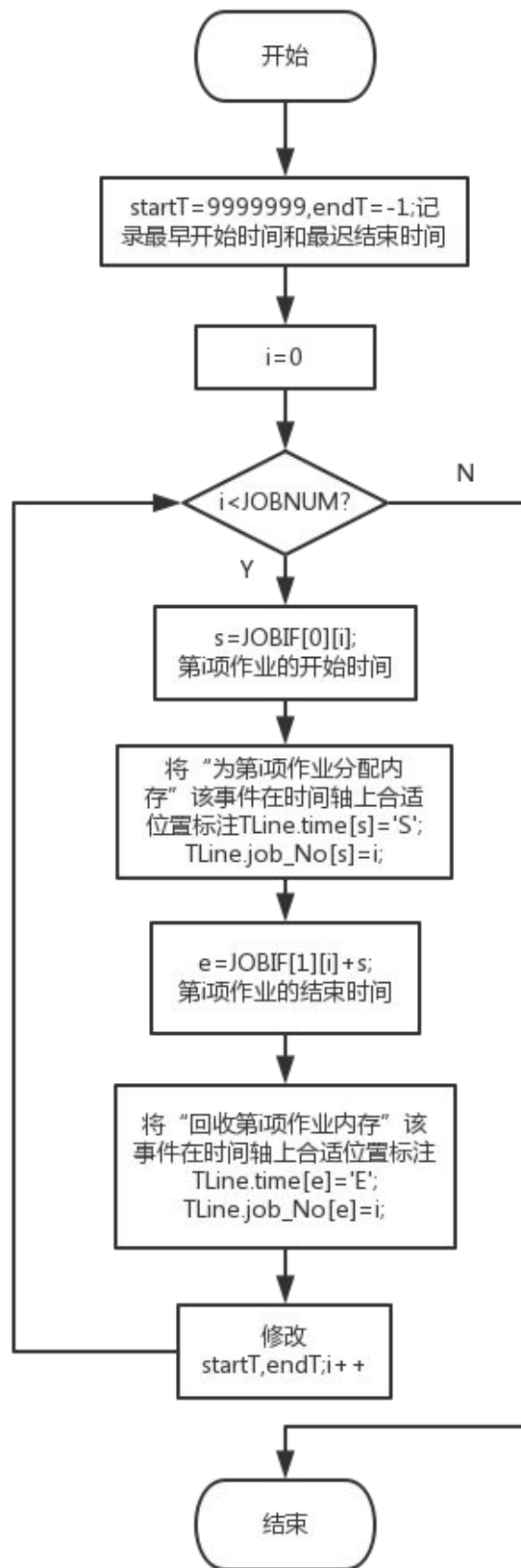
其中每项作业的信息可由 MATLAB 随机函数来生成

```
JOBNUM=10;  
JOBIF(:,1)=poissrnd(5,JOBNUM,1); %作业开始时间 此函数生成服从泊松 (Poisson)  
分布的 JOBNUM 行 1 列数据。泊松分布的参数是 6  
JOBIF(:,2)=ceil(exprnd(5,JOBNUM,1)) %作业执行时间  
JOBIF(:,3)=ceil(exprnd(10000,JOBNUM,1)) %作业所需内存, 生成服从参数为 5 的指  
数分布的数据矩阵,ceil 向正无穷取整
```

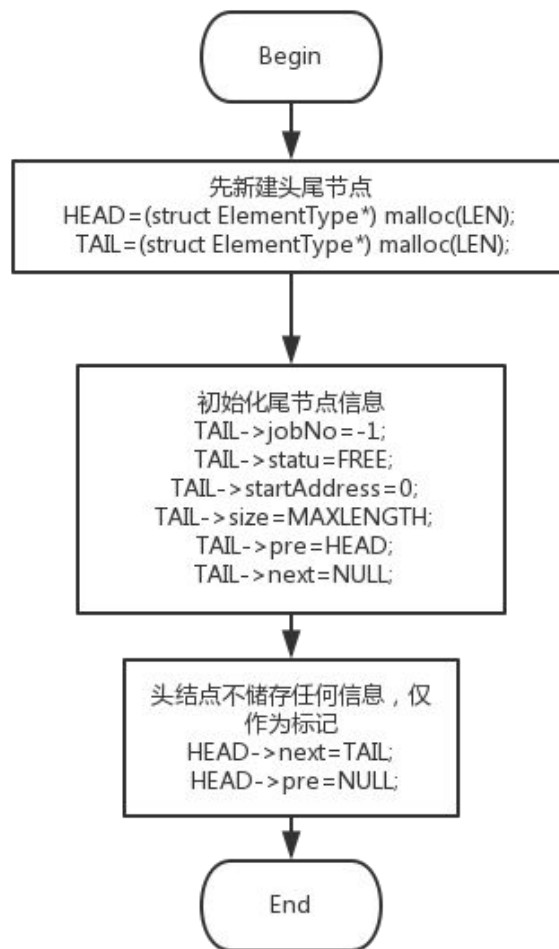


JOBIF =		
5	1	79
5	16	229
1	9	1
7	18	111
4	5	122
3	26	278
6	1	121
5	10	308
3	15	69

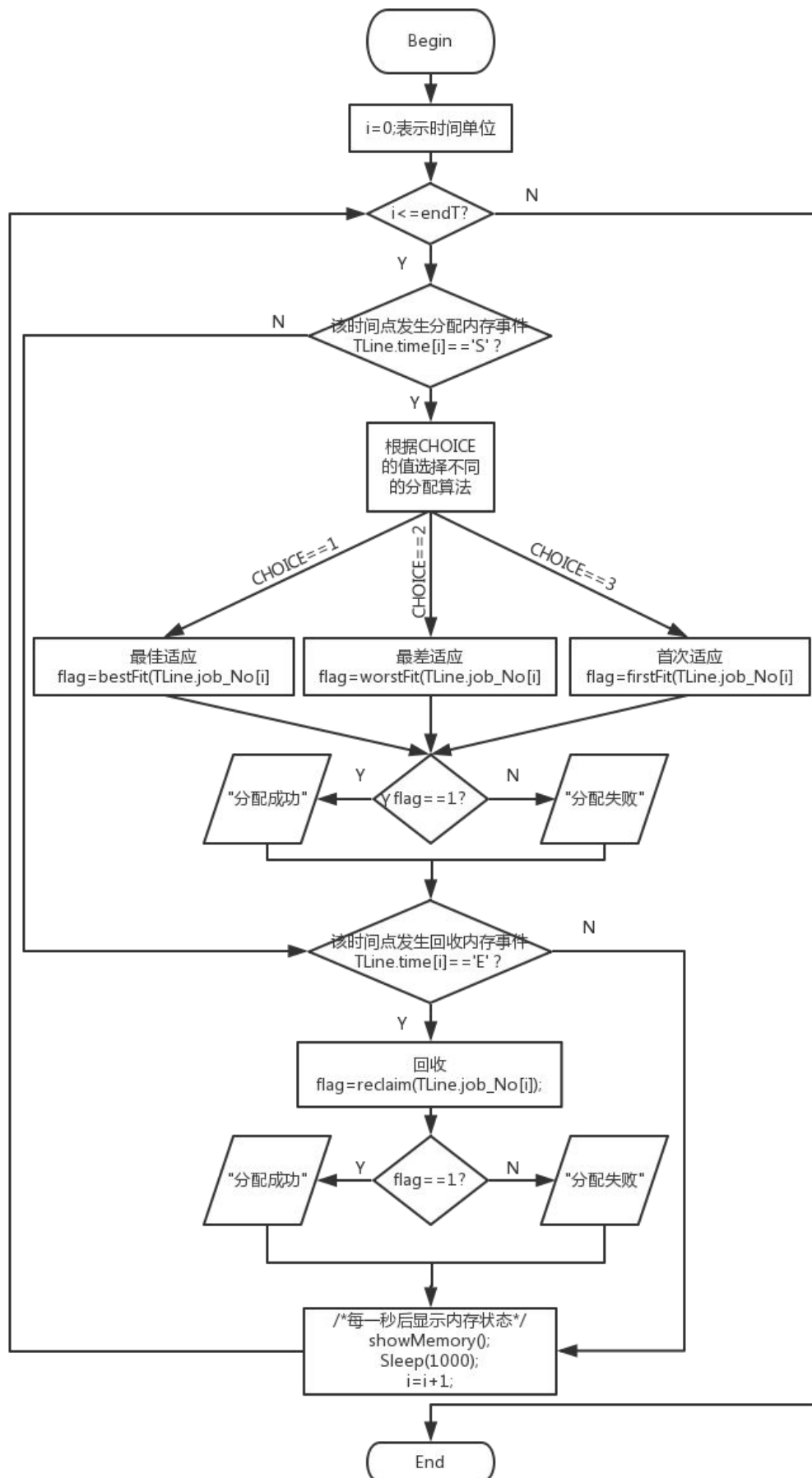
3. 制作时间轴 TLine: play()_1 (本实验采用的调度模式为先来先服务, 故需建立时间轴; 也可以采用其他的调度方式)



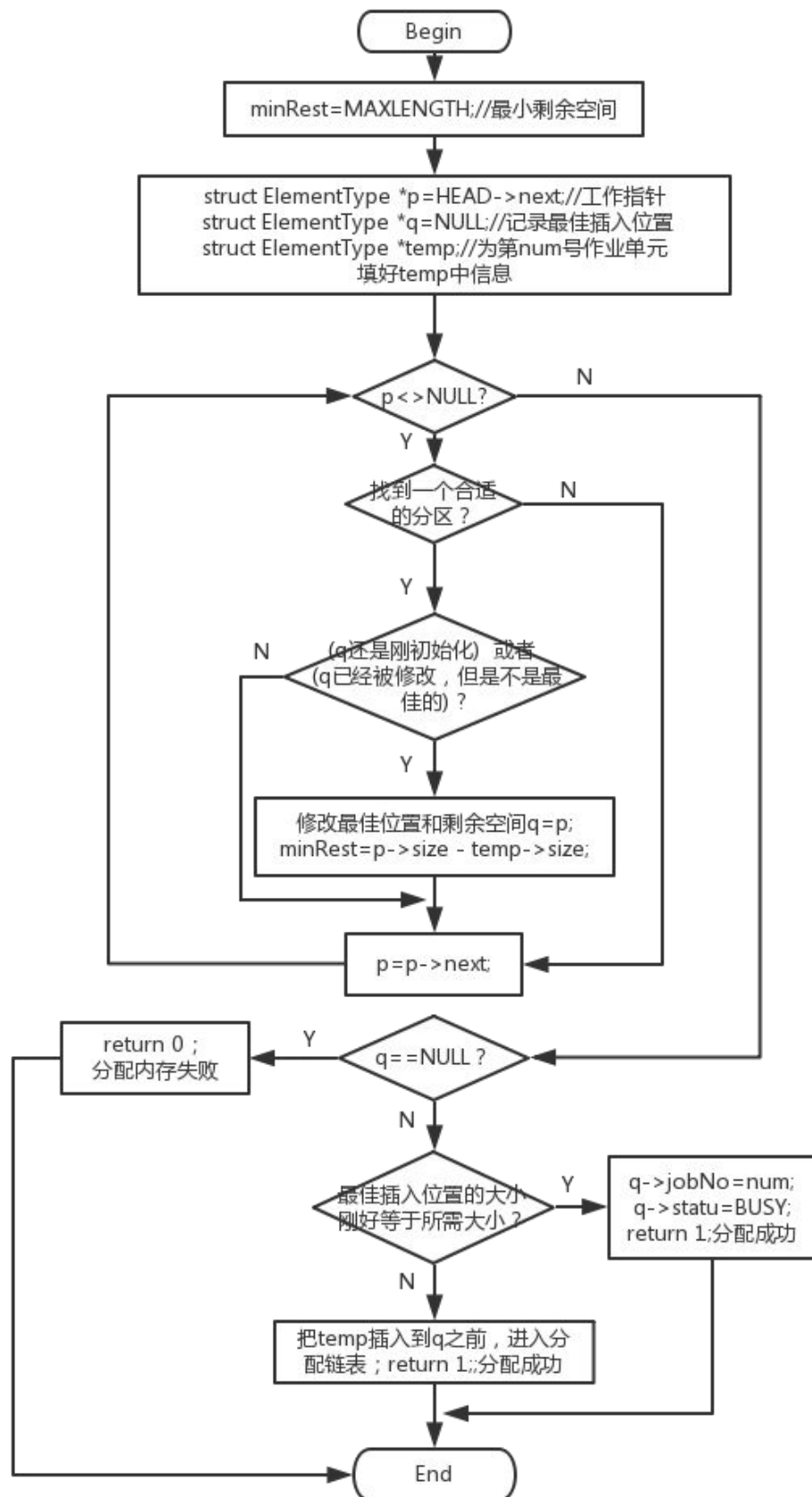
4. 开辟一个带有头结点的分配链表 `newLink()`



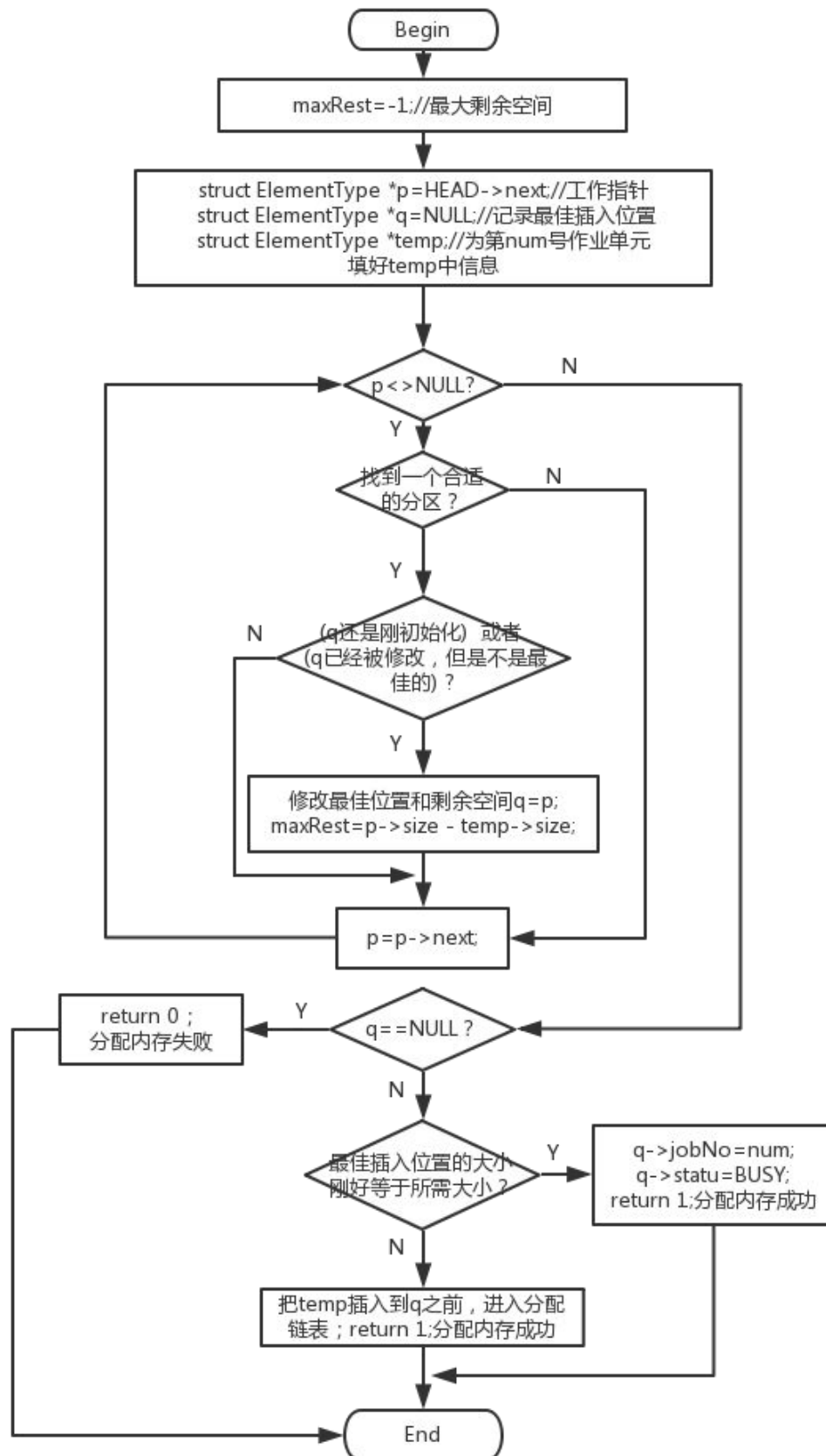
5. 遍历时间轴 play()_2



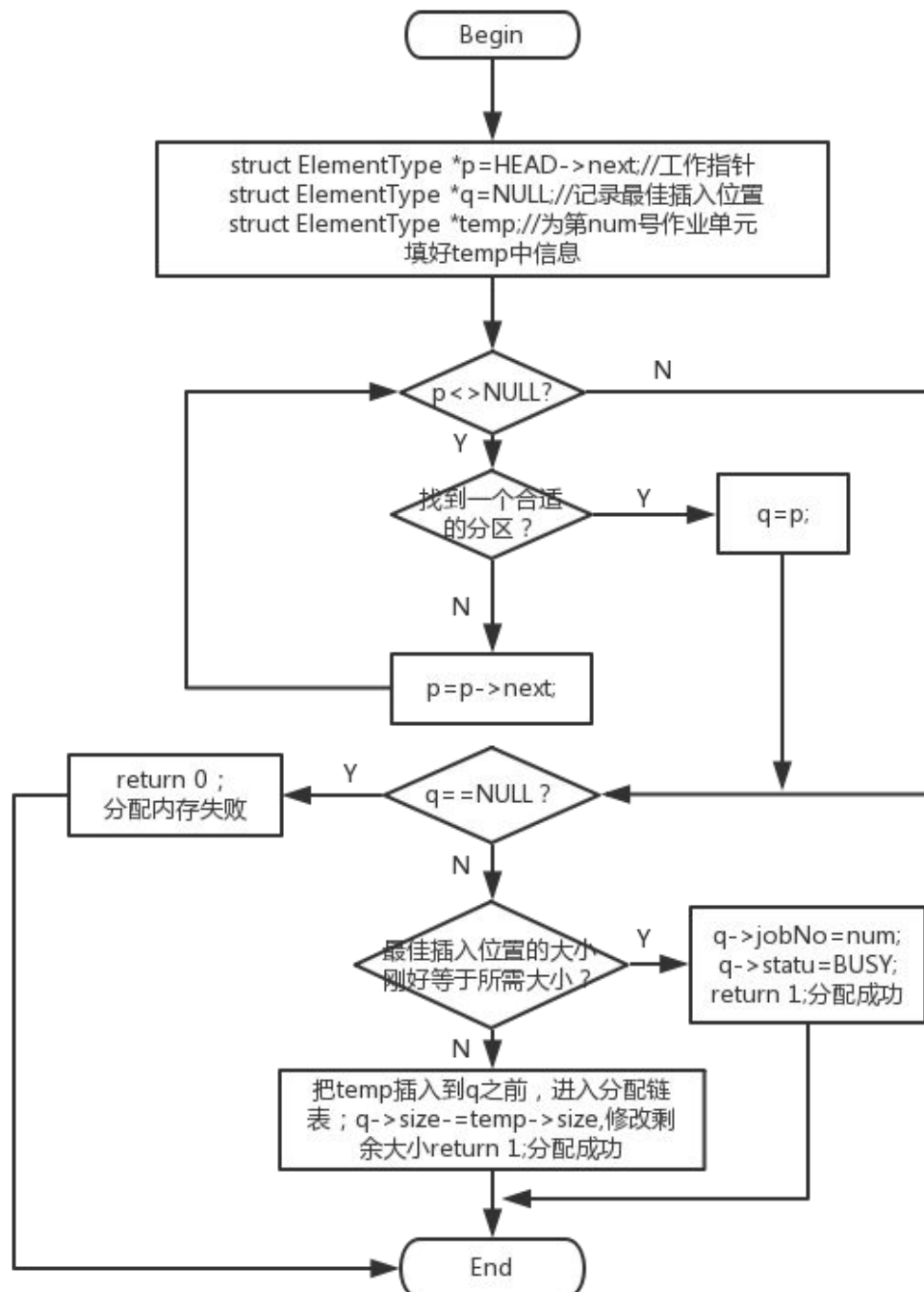
6. 最佳适应算法 bestFit(int num)



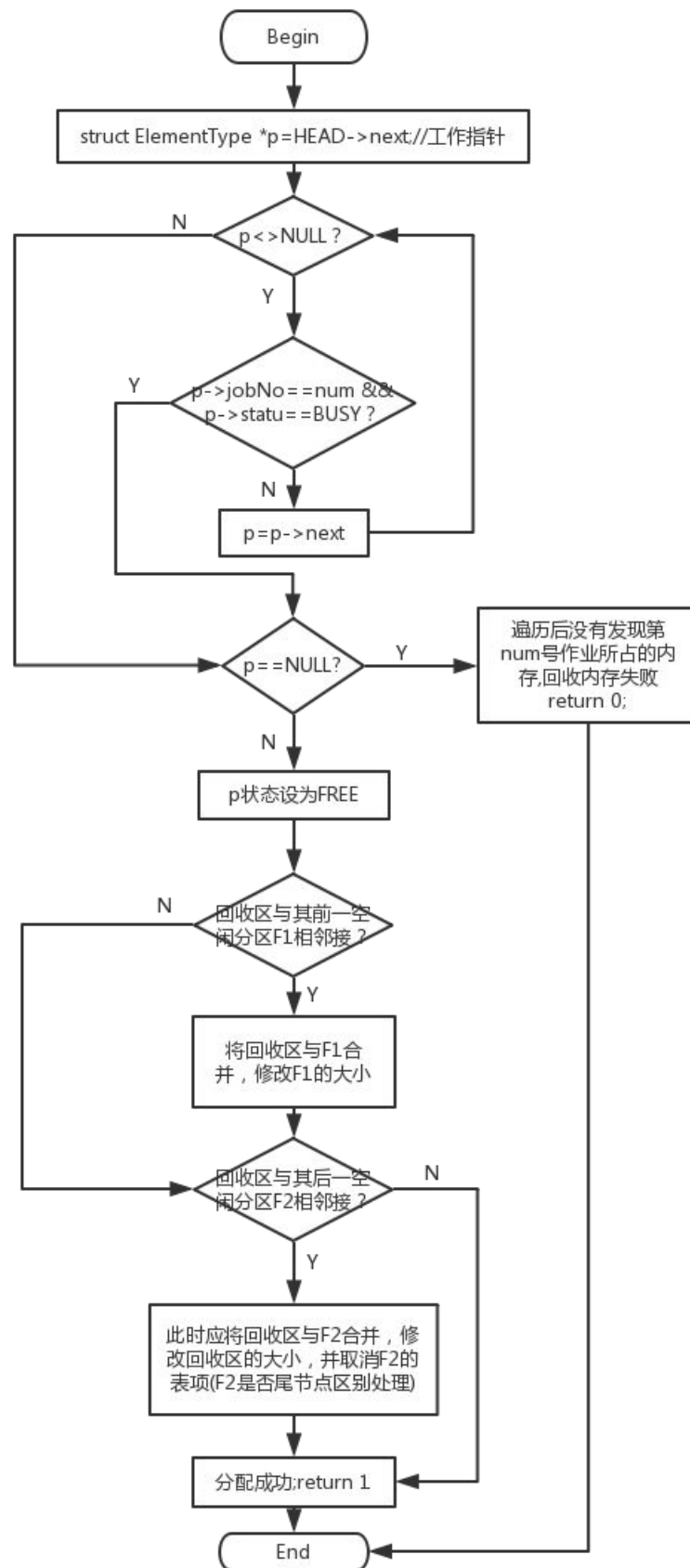
7. 最差适应算法 worstFit(int num)



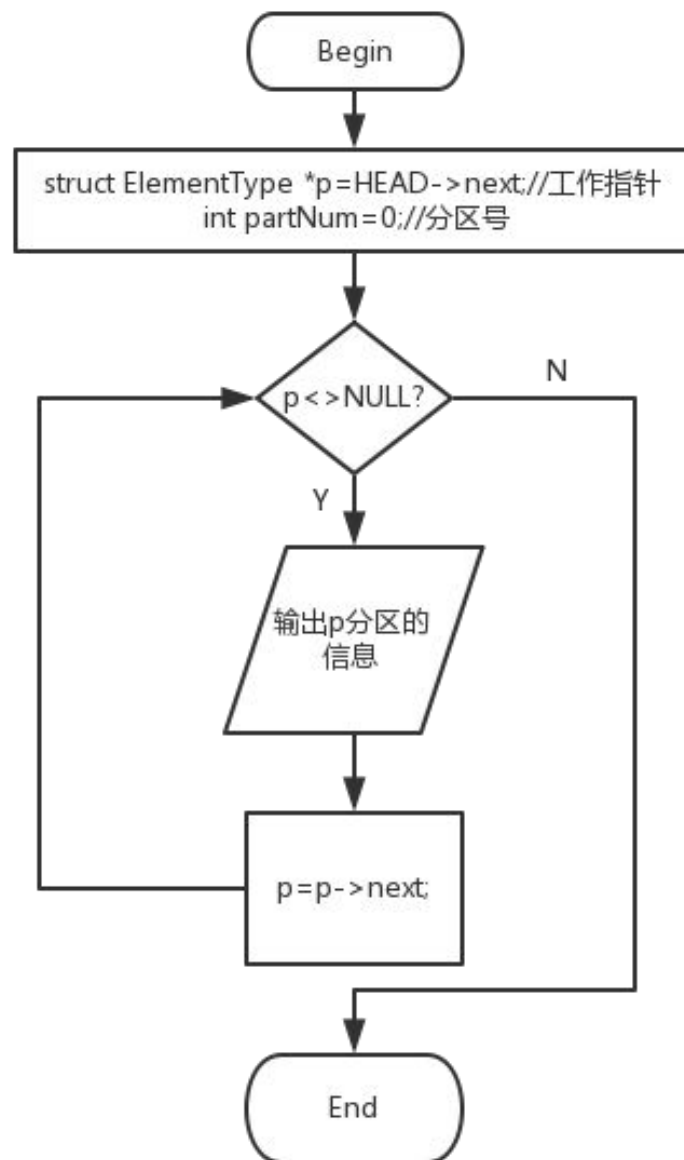
8. 首次适应算法 firstFit(int num)



9. 回收内存 reclaim(int num)



10. 显示内存状态 showMemory()



五、调试过程

1. 程序代码

运行时先由 MATLAB 产生满足特定概率分布的数据，再将产生的数据作为输入数据一起保存入输入文件 Store.in

产生满足特定概率分布的数据的程序代码：

Random.m

```
JOBNUM=10;  
JOBIF(:,1)=poissrnd(5,JOBNUM,1); %作业开始时间 此函数生成服从泊松(Poisson)  
分布的 JOBNUM 行 1 列数据。泊松分布的参数是 6  
JOBIF(:,2)=ceil(exprnd(5,JOBNUM,1)) %作业执行时间  
JOBIF(:,3)=ceil(exprnd(10000,JOBNUM,1)) %作业所需内存，生成服从参数为 5 的指  
数分布的数据矩阵, ceil 向正无穷取整
```

并发程序的存储管理程序：

Store.c

```
#include <stdio.h>  
#include <windows.h>  
  
#define FREE 0//空闲 /*表示每个物理块的状态*/  
#define BUSY 1//占用  
#define MAXLENGTH 100000//最大内存空间 100000 Byte  
#define MAXJOBNUM 1000//最大作业项数  
#define MAXTIME 10000//最大时间  
#define LEN sizeof(struct ElementType)  
  
int JOBNUM;//作业个数  
int CHOICE;//所选择的内存分配算法  
int JOBIF[3][MAXJOBNUM];  
/*
```

```

JOBIF[0][i]:第 i 项作业开始时间
JOBIF[1][i]:第 i 项作业执行时间
JOBIF[2][i]:第 i 项作业所需内存
**/

/*时间轴类型*/
struct TimeLine
{
    char time[MAXTIME];//记录 0-9999 这段时间内发生的分配内存'D'或回收内存'R'事件;
    int job_No[MAXTIME];//记录每个 D 或 R 对应的作业编号
};
struct TimeLine TLine;//时间轴

/*节点类型*/
struct ElementType
{
    int jobNo;//作业编号, 作业名
    int statu;//显示状态
    int startAddress;//起始地址
    int size;//大小

    int startTime;//起始时间
    int runTime;//运行时间

    struct ElementType *pre;
    struct ElementType *next;
};
struct ElementType *HEAD,*TAIL;//分配链表的头结点

void input()
{
    int i;

```

```

    freopen("Store.in", "r", stdin);
    printf("这是金洋(Sno:20131910023)的并发进程存储管理模拟实
验.\n");
    printf("1.最佳适应算法\n2.最差适应算法\n3.首次适应算法\n");
    printf("请输入内存分配算法:\n");
    scanf("%d",&CHOICE);

    printf("请输入作业个数:\n");
    scanf("%d",&JOBNUM);

    /*根据 MATLAB 产生的随机分布数填写每项进程信息*/
    printf("请依次输入 n 项作业的开始时间, 执行时间, 所需内存:\n");
    for (i=0;i<JOBNUM;i++)
    {
        scanf("%d %d %d",&JOBIF[0][i],&JOBIF[1][i],&JOBIF[2][i]);
    }
}

```

```

/*开辟一个带有头结点的分配链表*/
void newLink()
{
    HEAD=(struct ElementType*) malloc(LEN);
    TAIL=(struct ElementType*) malloc(LEN);

    TAIL->jobNo=-1;
    TAIL->statu=FREE;
    TAIL->startAddress=0;
    TAIL->size=MAXLENGTH;
    TAIL->pre=HEAD;
    TAIL->next=NULL;
    HEAD->next=TAIL;
}

```

```

    HEAD->pre=NULL;

}

/*最佳适应算法，为第 num 号作业分配内存*/
int bestFit(int num)
{
    int minRest=MAXLENGTH;//最小剩余空间
    struct ElementType *p=HEAD->next;//工作指针
    struct ElementType *q=NULL;//记录最佳插入位置

    struct ElementType *temp;//为第 num 号作业单元
    temp=(struct ElementType*) malloc(LEN);
    temp->jobNo=num;
    temp->statu=BUSY;
    temp->size=JOBIF[2][num];

    /*遍历整个分配链表*/
    while (p)
    {
        /*找到一个合适的分区*/
        if (p->statu==FREE && p->size>=temp->size)
        {
            /* q 还是刚初始化 */ /* q 已经被修改，但是不是最佳的*/
            if ((q==NULL) || (p->size < q->size))
            {
                q=p;
                minRest=p->size - temp->size;
            }
        }
        p=p->next;
    }
}

```

```
/*没有找到合适的物理块，分配失败*/
if (q==NULL) return 0;

/*最佳插入位置的大小刚好等于所需大小*/
if (q->size==temp->size)
{
    q->jobNo=num;
    q->statu=BUSY;
    return 1;
}
/*最佳插入位置的大小 大于 所需大小*/
else
{
    temp->startAddress=q->startAddress;
    temp->pre=q->pre;
    temp->next=q;

    temp->pre->next=temp;

    q->pre=temp;
    q->startAddress+=temp->size;
    q->size=minRest;//修改剩余大小
    return 1;
}
}

/*最差适应算法，为第 num 号作业分配内存*/
int worstFit(int num)
{
    int maxRest=-1;//最大剩余空间
    struct ElementType *p=HEAD->next;//工作指针
    struct ElementType *q=NULL;//记录最佳插入位置
```

```
struct ElementType *temp;//为第 num 号作业单元
temp=(struct ElementType*) malloc(LEN);
temp->jobNo=num;
temp->statu=BUSY;
temp->size=JOBIF[2][num];

/*遍历整个分配链表*/
while (p)
{
    /*找到一个合适的分区*/
    if (p->statu==FREE && p->size>=temp->size)
    {
        /* q 还是刚初始化 */ /* q 已经被修改，但是不是最佳的*/
        if ((q==NULL) || (p->size > q->size))
        {
            q=p;
            maxRest=p->size - temp->size;
        }
    }
    p=p->next;
}

/*没有找到合适的物理块，分配失败*/
if (q==NULL) return 0;

/*最佳插入位置的大小刚好等于所需大小*/
if (q->size==temp->size)
{
    q->jobNo=num;
    q->statu=BUSY;
    return 1;
}
/*最佳插入位置的大小 大于 所需大小*/
else
```

```
{
    temp->startAddress=q->startAddress;
    temp->pre=q->pre;
    temp->next=q;

    temp->pre->next=temp;

    q->pre=temp;
    q->startAddress+=temp->size;
    q->size=maxRest;//修改剩余大小
    return 1;
}
}
```

/*首次适应算法，为第 num 号作业分配内存*/

int firstFit(int num)

```
{
    struct ElementType *p=HEAD->next;//工作指针
    struct ElementType *q=NULL;//记录最佳插入位置

    struct ElementType *temp;//为第 num 号作业单元
    temp=(struct ElementType*) malloc(LEN);
    temp->jobNo=num;
    temp->statu=BUSY;
    temp->size=JOBIF[2][num];

    /*遍历整个分配链表*/
    while (p)
    {
        /*找到一个合适的分区*/
        if (p->statu==FREE && p->size>=temp->size)
            break;
        p=p->next;
    }
}
```

```
q=p;

/*没有找到合适的物理块，分配失败*/
if (q==NULL) return 0;

/*插入位置的大小刚好等于所需大小*/
if (q->size==temp->size)
{
    q->jobNo=num;
    q->statu=BUSY;
    return 1;
}
/*最佳插入位置的大小 大于 所需大小*/
else
{
    temp->startAddress=q->startAddress;
    temp->pre=q->pre;
    temp->next=q;

    temp->pre->next=temp;

    q->pre=temp;
    q->startAddress+=temp->size;
    q->size-=temp->size;/**修改剩余大小
    return 1;
}
}

/*回收内存，回收第 num 号作业所占的内存*/
int reclaim(int num)
{
    struct ElementType *p=HEAD->next;//工作指针
    if (num<0 || num>=JOBNUM) return 0;
```

```

/*遍历分配链表*/
while (p)
{
    if (p->jobNo==num && p->statu==BUSY) break;
    p=p->next;
}

/*遍历后没有发现第 num 号作业所占的内存,回收内存失败*/
if (p==NULL) return 0;

p->statu=FREE;
/*回收区与其前一空闲分区 F1 相邻接,此时应将回收区与 F1 合并,修改
F1 的大小*/
if (p->pre!=HEAD && p->pre->statu==FREE)
{
    p->pre->size+=p->size;
    p->pre->next=p->next;
    p->next->pre=p->pre;

    p=p->pre;
}

/*回收区与其后一空闲分区 F2 相邻接,此时应将回收区与 F2 合并,修改回
收区的大小,并取消 F2 的表项*/
/*后一分区不是尾节点*/
if (p->next!=TAIL && p->next->statu==FREE)
{
    p->size+=p->next->size;
    p->next=p->next->next;
    free(p->next->pre); //free(F2)相当于取消 F2 的表项
    p->next->pre=p;
}

/*后一分区是尾节点*/
if (p->next==TAIL && p->next->statu==FREE)

```

```
{
    p->size+=p->next->size;
    p->next=NULL;
    TAIL=p;
}

return 1;
}

void showMemory()
{
    struct ElementType *p=HEAD->next;//工作指针
    int partNum=0;//分区号

    printf("内存分配情况:\n");
    printf("分区号\t起始地址\t分区大小\t状态\t作业号\n");

    while (p)
    {
        printf("  %d\t",partNum++);
        printf("  %d\t\t",p->startAddress);
        printf(" %d\t\t",p->size);

        if (p->statu==FREE)
            printf("空闲\t  -\n");
        else
            printf("占用\t  %d\n",p->jobNo);
        p=p->next;
    }
    printf("\n\n\n");
}
```

```
/*可以在此处决定调度模式，此处沿时间轴遍历，即先来先服务调度*/
void play()
{
    int flag;//分配、回收是否成功

    /*创建时间轴*/
    int i,startT=9999999,endT=-1;//最早开始时间和最迟结束时间

    int s,e;//记录每一项作业的开始时间和结束时间;
    for (i=0;i<JOBNUM;i++)
    {
        s=JOBIF[0][i];
        /*在时间轴上标记第 i 项作业开始时间*/
        while (TLine.time[s]=='S' || TLine.time[s]=='E')
        {
            s++;//此时已经有事件，则推后本事件
        }
        TLine.time[s]='S';
        TLine.job_No[s]=i;

        /*在时间轴上标记第 i 项作业结束时间*/
        e=JOBIF[1][i]+s;
        while (TLine.time[e]=='S' || TLine.time[e]=='E')
        {
            e++;
        }
        TLine.time[e]='E';
        TLine.job_No[e]=i;

        if (s<startT) startT=s;
        if (e>endT) endT=e;
    }

    /*开辟一个带有头结点的分配链表*/
    newLink();
}
```

```
/*沿时间轴遍历*/
/*即先来先服务调度*/
printf("\n");
for (i=0;i<=endT;i++)
{

    printf("第%d 时间单位:\n",i);
    if (TLine.time[i]=='S')
    {
        switch(CHOICE)
        {
            case 1:flag=bestFit(TLine.job_No[i]);break;
            case 2:flag=worstFit(TLine.job_No[i]);break;
            case 3:flag=firstFit(TLine.job_No[i]);break;
        }

        printf("为第%d 项作业",TLine.job_No[i]);
        if (flag==1) printf("分配内存成功\n");
        else printf("分配内存失败\n");
    }

    else if (TLine.time[i]=='E')
    {
        flag=reclaim(TLine.job_No[i]);
        if (flag==1) printf("回收第%d 项作业内存成功\n",TLine.job_No[i]);
        else printf("回收第%d 项作业内存失败\n",TLine.job_No[i]);
    }

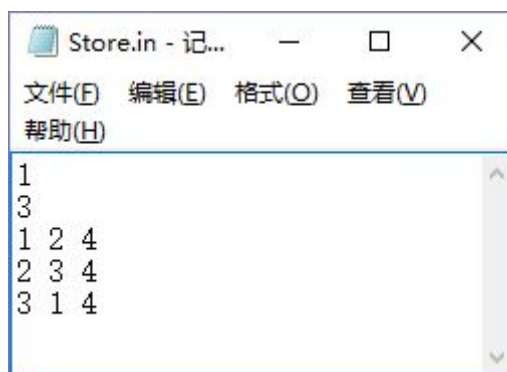
    /*每一秒后显示内存状态*/
    showMemory();
    Sleep(1000);
}
```

```
}
```

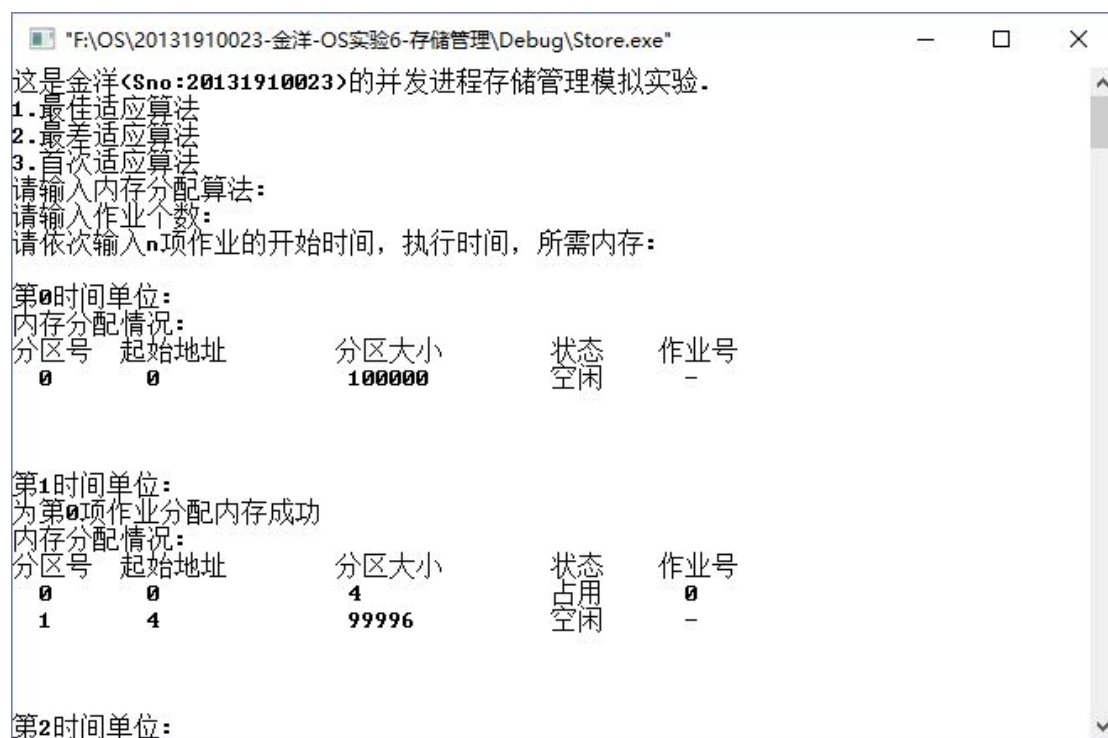
```
void main()
{
    input();
    play();
}
```

2. 运行结果

先尝试简单的数据，第一行表示所选择的存储算法，第二行为作业个数，第三行开始每一行表示对应作业的起始时间、执行时间、所需内存；



①最佳适应算法：



```
"F:\OS\20131910023-金洋-OS实验6-存储管理\Debug\Store.exe"
为第1项作业分配内存成功
内存分配情况:
分区号 起始地址      分区大小      状态      作业号
0       0              4            占用      0
1       4              4            占用      1
2       8            99992          空闲      -

第3时间单位:
回收第0项作业内存成功
内存分配情况:
分区号 起始地址      分区大小      状态      作业号
0       0              4            空闲      -
1       4              4            占用      1
2       8            99992          空闲      -

第4时间单位:
为第2项作业分配内存成功
内存分配情况:
分区号 起始地址      分区大小      状态      作业号
0       0              4            占用      2
1       4              4            占用      1
2       8            99992          空闲      -

第5时间单位:
回收第1项作业内存成功
内存分配情况:
分区号 起始地址      分区大小      状态      作业号
0       0              4            占用      2
1       4            99996          空闲      -

第6时间单位:
回收第2项作业内存成功
内存分配情况:
分区号 起始地址      分区大小      状态      作业号
0       0            100000          空闲      -

Press any key to continue
```

②最差适应算法:

```

"F:\OS\20131910023-金洋-OS实验6-存储管理\Debug\Store.exe"
这是金洋(Sno:20131910023)的并发进程存储管理模拟实验.
1.最佳适应算法
2.最差适应算法
3.首次适应算法
请输入内存分配算法:
请输入作业个数:
请依次输入n项作业的开始时间, 执行时间, 所需内存:

第0时间单位:
内存分配情况:
分区号 起始地址      分区大小      状态      作业号
0      0      100000      空闲      -

第1时间单位:
为第0项作业分配内存成功
内存分配情况:
分区号 起始地址      分区大小      状态      作业号
0      0      4      占用      0
1      4      99996      空闲      -

第2时间单位:

```

```

"F:\OS\20131910023-金洋-OS实验6-存储管理\Debug\Store.exe"
为第1项作业分配内存成功
内存分配情况:
分区号 起始地址      分区大小      状态      作业号
0      0      4      占用      0
1      4      4      占用      1
2      8      99992      空闲      -

第3时间单位:
回收第0项作业内存成功
内存分配情况:
分区号 起始地址      分区大小      状态      作业号
0      0      4      空闲      -
1      4      4      占用      1
2      8      99992      空闲      -

第4时间单位:
为第2项作业分配内存成功
内存分配情况:
分区号 起始地址      分区大小      状态      作业号
0      0      4      空闲      -
1      4      4      占用      1

```

```

F:\OS\20131910023-金洋-OS实验6-存储管理\Debug\Store.exe
2      8      4      占用      2
3      12     99988   空闲      -

第5时间单位:
回收第1项作业内存成功
内存分配情况:
分区号  起始地址      分区大小      状态      作业号
0        0           8      空闲      -
1        8           4      占用      2
2       12          99988   空闲      -

第6时间单位:
回收第2项作业内存成功
内存分配情况:
分区号  起始地址      分区大小      状态      作业号
0        0          100000   空闲      -

Press any key to continue

```

③首次适应算法

```

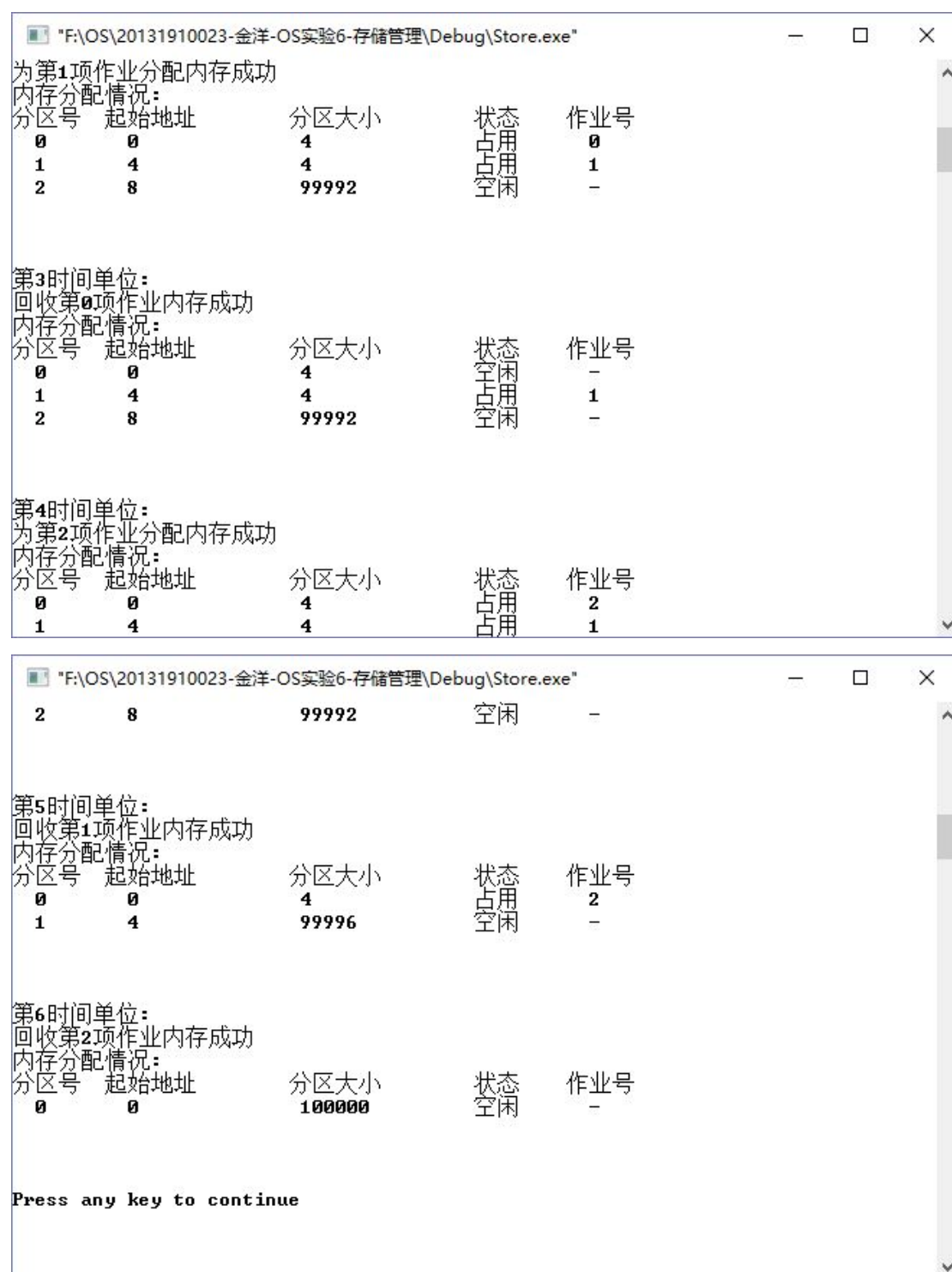
F:\OS\20131910023-金洋-OS实验6-存储管理\Debug\Store.exe
这是金洋<Sno:20131910023>的并发进程存储管理模拟实验.
1.最佳适应算法
2.最差适应算法
3.首次适应算法
请输入内存分配算法:
请输入作业个数:
请依次输入n项作业的开始时间, 执行时间, 所需内存:

第0时间单位:
内存分配情况:
分区号  起始地址      分区大小      状态      作业号
0        0          100000   空闲      -

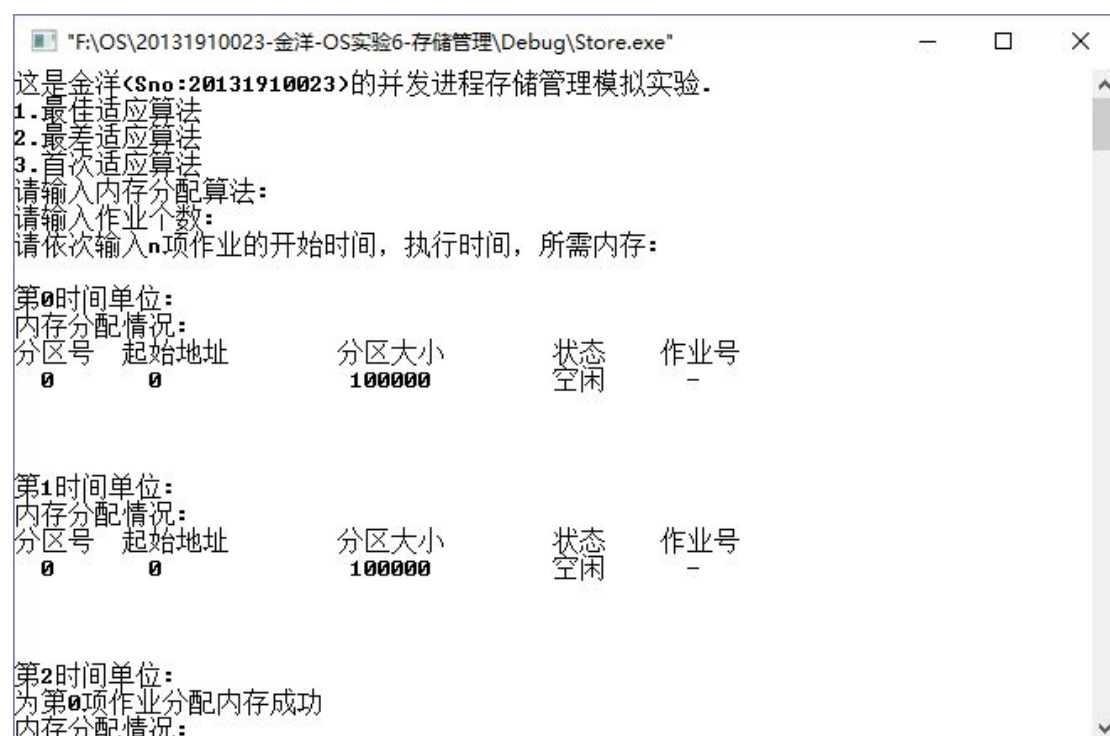
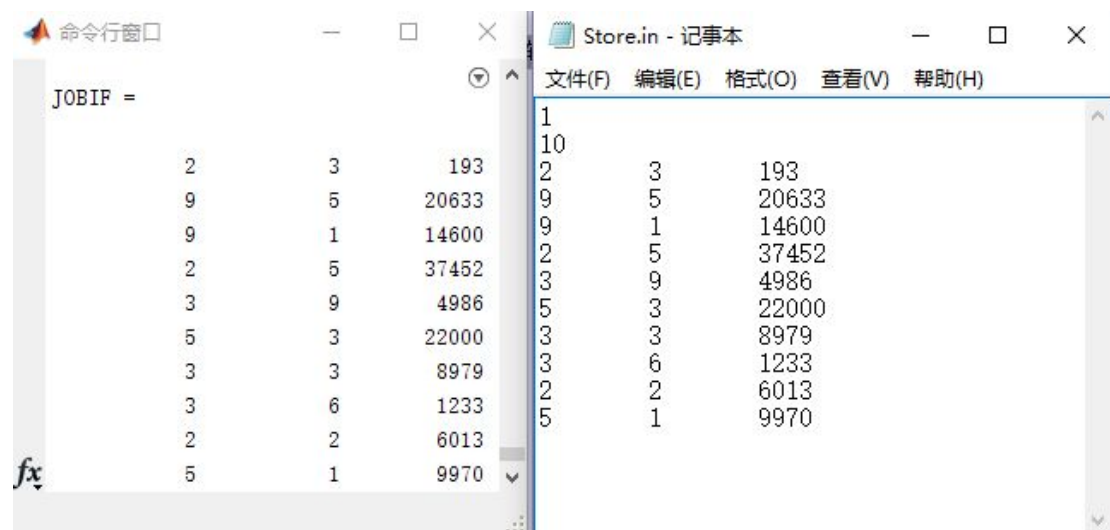
第1时间单位:
为第0项作业分配内存成功
内存分配情况:
分区号  起始地址      分区大小      状态      作业号
0        0           4      占用      0
1        4          99996   空闲      -

第2时间单位:

```



④由 MATLAB 产生满足泊松分布或指数分布的数据，并用最佳适应法运行程序



"F:\OS\20131910023-金洋-OS实验6-存储管理\Debug\Store.exe"				
分区号	起始地址	分区大小	状态	作业号
0	0	193	占用	0
1	193	99807	空闲	-
第3时间单位: 为第3项作业分配内存成功 内存分配情况:				
分区号	起始地址	分区大小	状态	作业号
0	0	193	占用	0
1	193	37452	占用	3
2	37645	62355	空闲	-
第4时间单位: 为第4项作业分配内存成功 内存分配情况:				
分区号	起始地址	分区大小	状态	作业号
0	0	193	占用	0
1	193	37452	占用	3
2	37645	4986	占用	4
3	42631	57369	空闲	-

"F:\OS\20131910023-金洋-OS实验6-存储管理\Debug\Store.exe"				
第5时间单位: 回收第0项作业内存成功 内存分配情况:				
分区号	起始地址	分区大小	状态	作业号
0	0	193	空闲	-
1	193	37452	占用	3
2	37645	4986	占用	4
3	42631	57369	空闲	-
第6时间单位: 为第5项作业分配内存成功 内存分配情况:				
分区号	起始地址	分区大小	状态	作业号
0	0	193	空闲	-
1	193	37452	占用	3
2	37645	4986	占用	4
3	42631	22000	占用	5
4	64631	35369	空闲	-
第7时间单位: 为第6项作业分配内存成功				

"F:\OS\20131910023-金洋-OS实验6-存储管理\Debug\Store.exe"

内存分配情况:

分区号	起始地址	分区大小	状态	作业号
0	0	193	空闲	-
1	193	37452	占用	3
2	37645	4986	占用	4
3	42631	22000	占用	5
4	64631	8979	占用	6
5	73610	26390	空闲	-

第8时间单位:
回收第3项作业内存成功

内存分配情况:

分区号	起始地址	分区大小	状态	作业号
0	0	37645	空闲	-
1	37645	4986	占用	4
2	42631	22000	占用	5
3	64631	8979	占用	6
4	73610	26390	空闲	-

第9时间单位:
为第1项作业分配内存成功

"F:\OS\20131910023-金洋-OS实验6-存储管理\Debug\Store.exe"

内存分配情况:

分区号	起始地址	分区大小	状态	作业号
0	0	37645	空闲	-
1	37645	4986	占用	4
2	42631	22000	占用	5
3	64631	8979	占用	6
4	73610	20633	占用	1
5	94243	5757	空闲	-

第10时间单位:
为第2项作业分配内存成功

内存分配情况:

分区号	起始地址	分区大小	状态	作业号
0	0	14600	占用	2
1	14600	23045	空闲	-
2	37645	4986	占用	4
3	42631	22000	占用	5
4	64631	8979	占用	6
5	73610	20633	占用	1
6	94243	5757	空闲	-

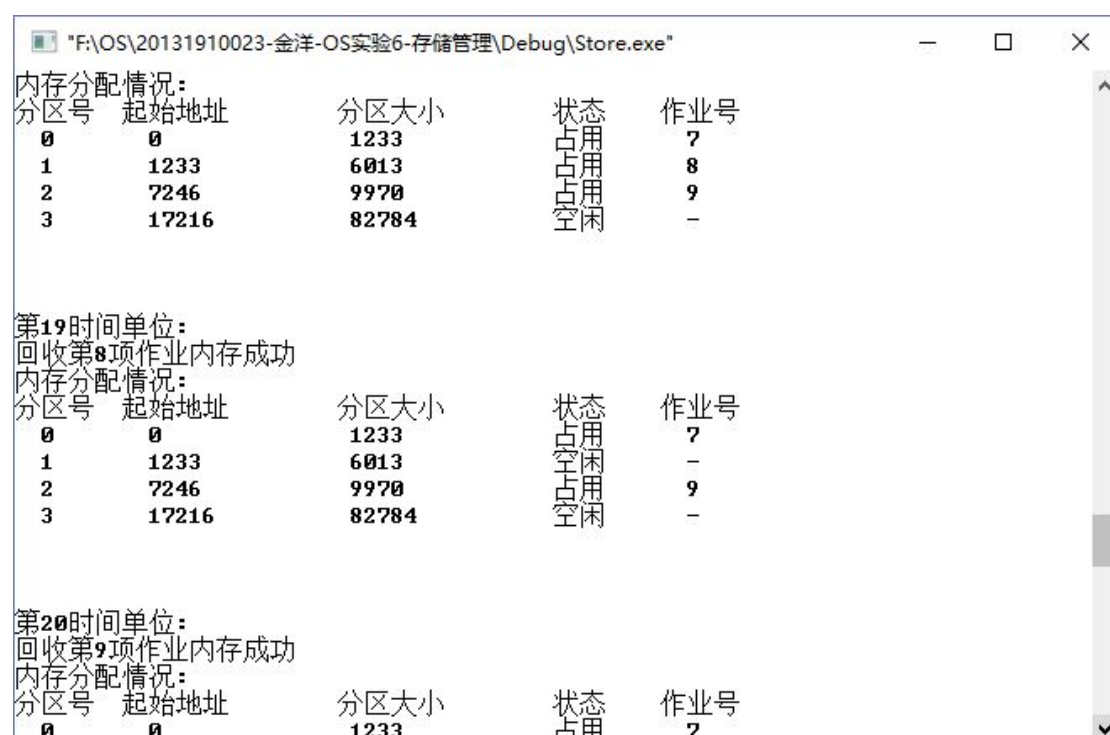
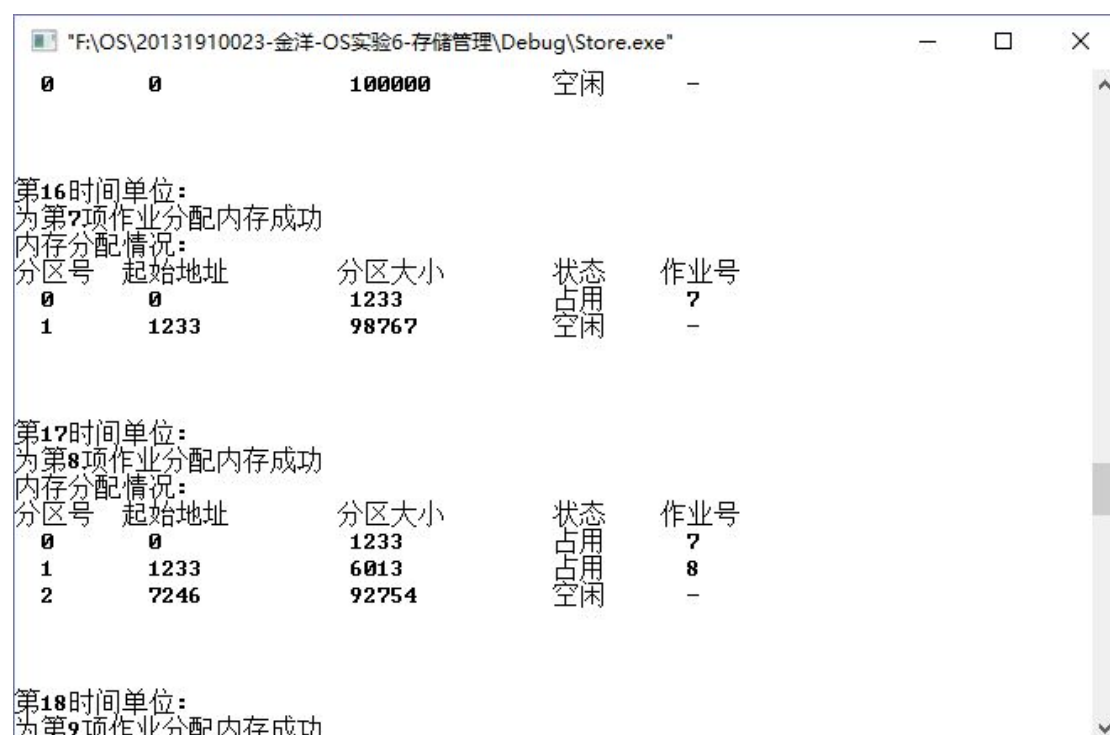
```
"F:\OS\20131910023-金洋-OS实验6-存储管理\Debug\Store.exe"
第11时间单位:
回收第2项作业内存成功
内存分配情况:
分区号 起始地址      分区大小      状态      作业号
0      0      37645      空闲      -
1      37645      4986      占用      4
2      42631      22000      占用      5
3      64631      8979      占用      6
4      73610      20633      占用      1
5      94243      5757      空闲      -

第12时间单位:
回收第5项作业内存成功
内存分配情况:
分区号 起始地址      分区大小      状态      作业号
0      0      37645      空闲      -
1      37645      4986      占用      4
2      42631      22000      空闲      -
3      64631      8979      占用      6
4      73610      20633      占用      1
5      94243      5757      空闲      -
```

```
"F:\OS\20131910023-金洋-OS实验6-存储管理\Debug\Store.exe"
第13时间单位:
回收第4项作业内存成功
内存分配情况:
分区号 起始地址      分区大小      状态      作业号
0      0      64631      空闲      -
1      64631      8979      占用      6
2      73610      20633      占用      1
3      94243      5757      空闲      -

第14时间单位:
回收第1项作业内存成功
内存分配情况:
分区号 起始地址      分区大小      状态      作业号
0      0      64631      空闲      -
1      64631      8979      占用      6
2      73610      26390      空闲      -

第15时间单位:
回收第6项作业内存成功
内存分配情况:
分区号 起始地址      分区大小      状态      作业号
```



```
"F:\OS\20131910023-金洋-OS实验6-存储管理\Debug\Store.exe"
1      1233      98767      空闲      -

第21时间单位:
内存分配情况:
分区号 起始地址      分区大小      状态      作业号
0      0      1233      占用      7
1      1233      98767      空闲      -

第22时间单位:
回收第7项作业内存成功
内存分配情况:
分区号 起始地址      分区大小      状态      作业号
0      0      100000      空闲      -

Press any key to continue_
```

六、总结

1. 内存价格昂贵，是一种宝贵而又紧俏的资源。应对其进行有效管理，来提高存储器的利用率，增加系统性能。

2. 动态分区分配有三种算法，最佳适应算法、最差适应算法、首次适应算法。在程序实现上首次适应算法最容易实现，最佳适应算法、最差适应算法实现难度上相近。

最佳适应算法因为每次分配后所切割下来的剩余部分总是最小，而且使得碎片是到处都有，总体来说它的效果比其他两者都要差。

最差适应算法每次都从最大的开始切割，使得碎片大大减少，缺点是使存储器中缺乏大的空闲分区。

首次适应算法由于是低地址部分不断地被划分，会留下许多难以利用的、很小的空闲分区，而每次查找又都是从低地址部分开始，这无疑会增加查找可用空闲分区时的开销。

3. 在内存管理中，内部碎片是已经被分配出去的内存空间大于请求所需的内存空间；

外部碎片是指还没有分配出去，但是由于大小太小而无法分配给申请空间的新进程的内存空间空闲块。

固定分区存在内部碎片，可变式分区分配会存在外部碎片；

页式虚拟存储系统存在内部碎片；段式虚拟存储系统，存在外部碎片；

为了有效的利用内存，使内存产生更少的碎片，要对内存分页，内存以页为单位来使用，最后一页往往装不满，于是形成了内部碎片。

为了共享要分段，在段的换入换出时形成外部碎片，

4. 实际中往往是许多进程并发执行，故在模拟存储管理时，不能以流程式即串行方式模拟，本实验利用 MATLAB 生成满足特定概率分布的数据来模拟真实作业的信息，可以通过调节参数来生成更加符合实际的数据。

5. 进程在执行上的特征是宏观并行、微观串行、随机执行。

6. 将为作业分配内存、回收内存的时间标记到时间轴上，通过遍历时间轴来对相应的事件作出处理，以此实现并发程序的内存管理模拟。

7. 在作业调度方式的选择上，本实验通过建立时间轴处理，故使用的是先到先服务调度方式。也可以通过建立“优先级轴”来实现优先级调度。

8. 通过本实验，将作业调度、进程并发、存储管理三块内容都相应地完成实现，很好的综合了本学期所学。而且本实验的模拟相比之前的实验更符合实际情况。

9. 本学期操作系统实验到此告一段落，相比开学初对实验的手足无措，在学期后阶段对实验明显有了一些把握。而这其中的原因可能便在于后阶段做到了仔细思考老师对问题的讲解、积极询问每一个实验的目的与要求、结合理论所学、尝试将理论课上的一些方法加以实现。学有所获，我会将后阶段这些良好习惯保持并应用于其他课程中。

七、参考文献

- [1]汤小丹，梁红兵，哲凤屏，汤子瀛. 计算机操作系统[M]（第三版）. 西安：西安电子科技大学出版社，2007 年 5 月；
- [2]谭浩强著. c 程序设计[M]（第三版）. 北京：清华大学出版社. 2005. 7；

八、教师评语