操作系统课程实验报告

学生姓名：杨子曦

班 学 号：20161003538

指导教师：袁国斌

中国地质大学信息工程学院

2018年 6月 11日

实习题目：内存管理模型的设计与实现

【需求规格说明】

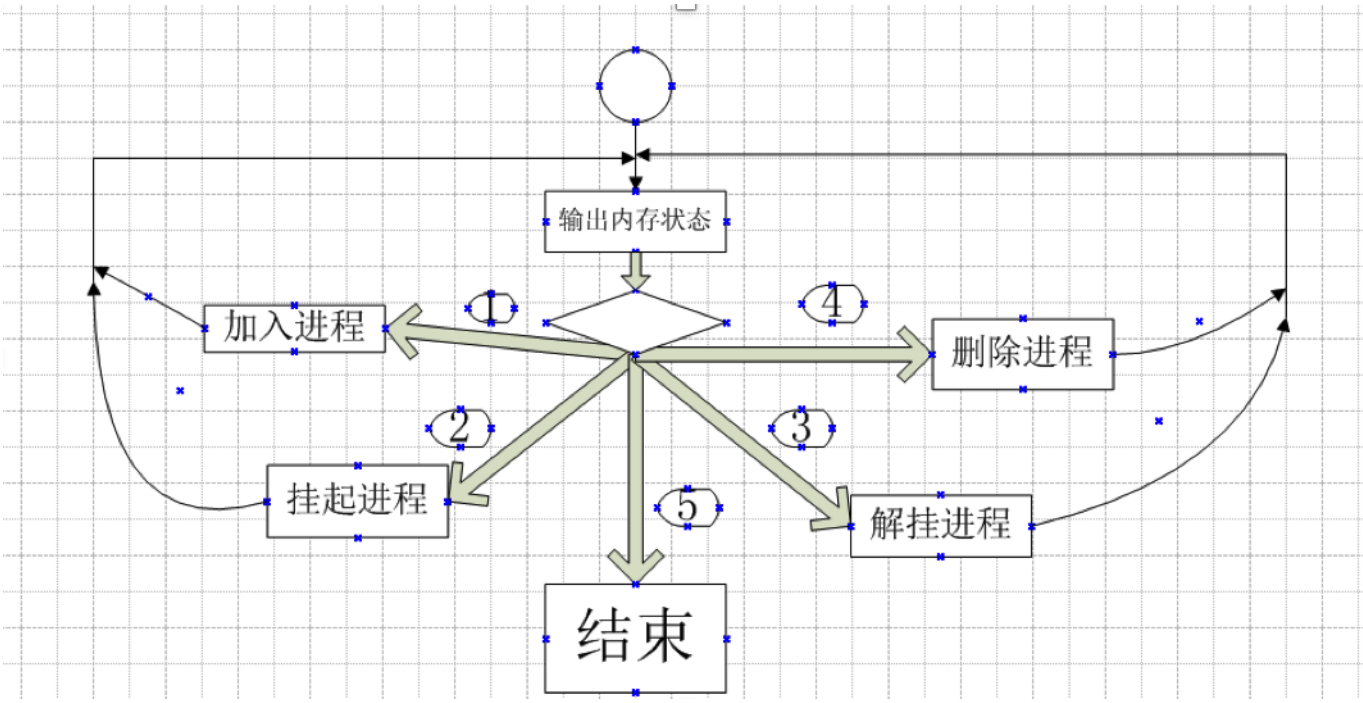
对内存的可变分区申请采用链表法管理进行模拟实现。要求如下：

1. 对于给定的一个存储空间自己设计数据结构进行管理，可以使用单个链表，也可以使用多个链表，自己负责存储空间的所有管理组织，要求采用分页方式（指定单元大小为页，如4K，2K，进程申请以页为单位）来组织基本内容；
2. 当进程对内存进行空间申请操作时，模型采用一定的策略（如：首先利用可用的内存进行分配，如果空间不够时，进行内存紧缩或其他方案进行处理）对进程给予分配；
3. 从系统开始启动到多个进程参与申请和运行时，进程最少要有3个以上，每个进程执行申请的时候都应能对系统当前的内存情况进行查看；
4. 对内存的申请进行内存分配，对使用过的空间进行回收，对给定的某种页面调度进行合理的页面分配。
5. 利用不同的颜色代表不同的进程对内存的占用情况，动态更新这些信息。

【算法设计】

1. 设计思想： 整个模拟系统采用最先适应法模拟主存中内存的分配和回收。对应用的数据结构分析：采用*双向*链表表示内存存储，每个节点包含的主要信息为：进程名、进程大小、存储起始地址、结束地址。头节点是操作系统本身占用的内存，依次往下是表示进程占用的内存，最后一个节点表示的进程的结束存储地址不能比主存空间大。每次加入内存即是从链表表头开始往下搜索合适位置插入表示进程的节点。每次进程结束删除相应节点即可。由此便完成了整个模拟内存管理。

2. 设计表示：



3. 详细设计表示：

由于考虑到程序的封装性，故在此将所有的分功能和小细节都采用类来进行构造。

- 存储数据结构的表示如下:分别是每个进程对象所对应的抽象，整个模拟系统的链表头，挂起队列的头

```
class Process
{
public:
    string NAME;
    LL SIZE;//一个进程占据的内存
    LL START;//开始地址
    LL END;//结束地址
    Process *NEXT;//尾指针
    Process *BEFORE;//头指针
};
//链表表头，存储进程数
class Head
{
public:
    LL count;
    Process *next;
};
//定义挂起队列
class Out
{
public:
    LL count;
    Process *next;
};
```

【调试报告】

如下图一是整个程序运行成功后的菜单栏: 主要功能为:

- 加入进程
- 进程挂起
- 进程解挂
- Kill进程
- 退出

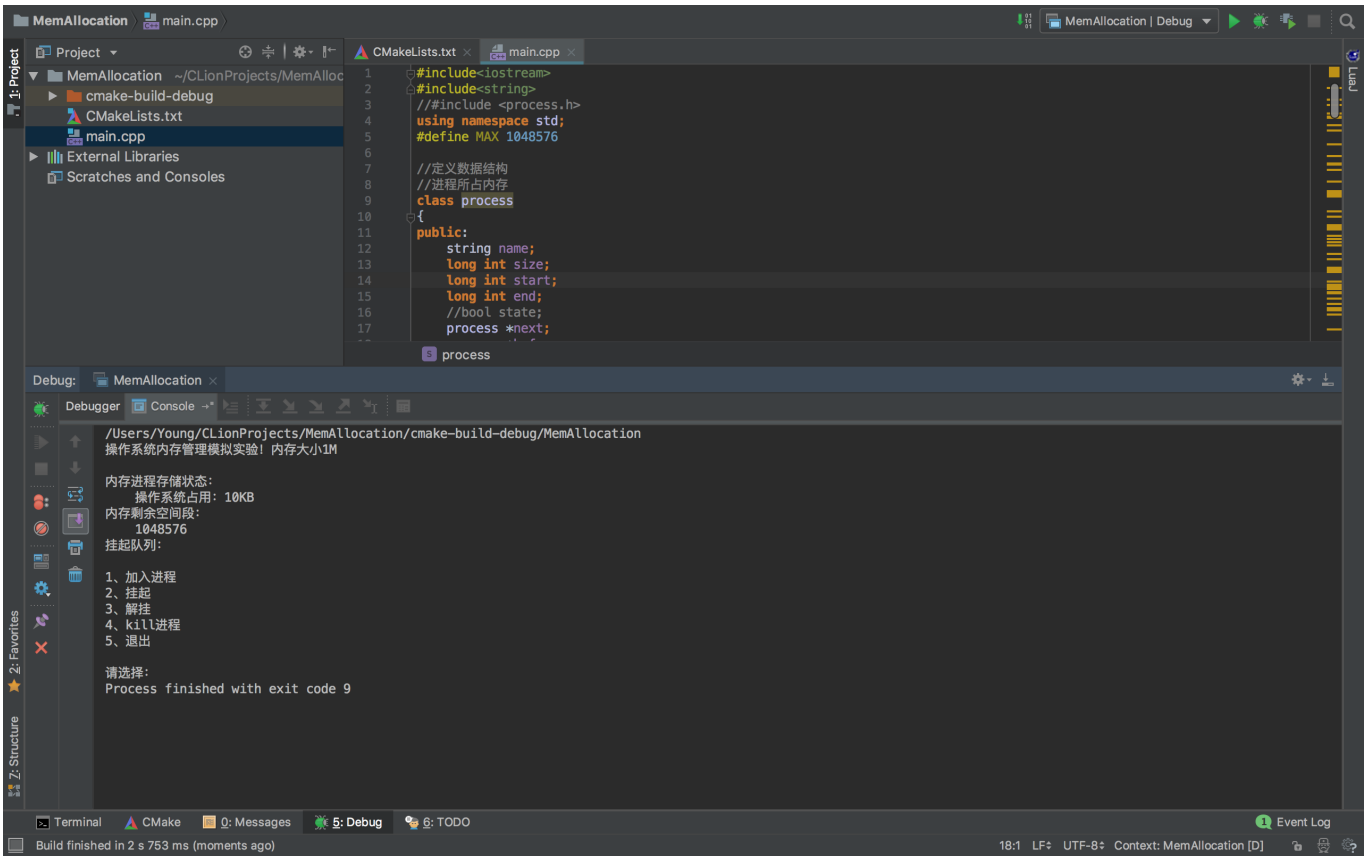


图 1

图二中是添加了 test1 和 test2 两个进程之后的结果：

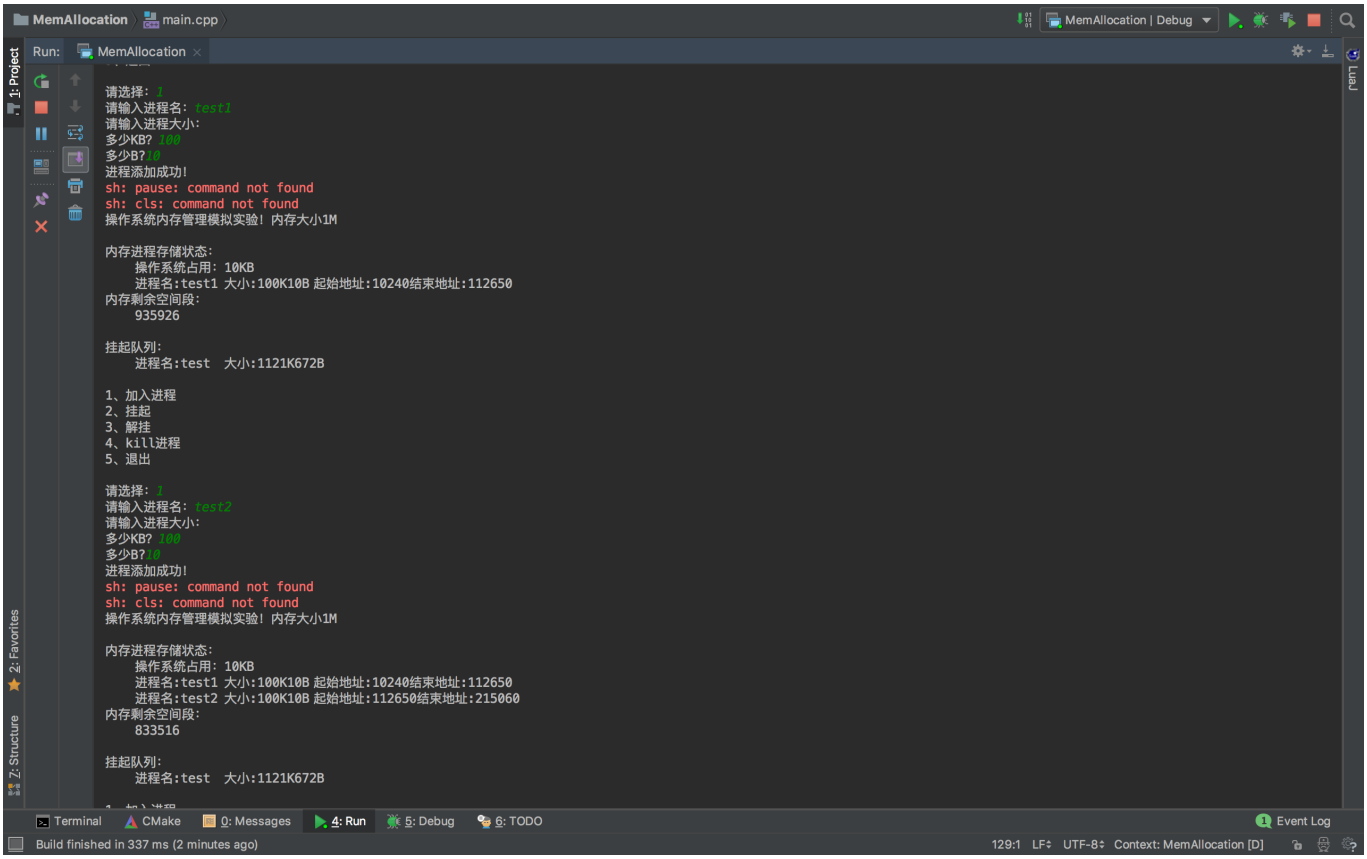


图 2

【附录】

运行环境如下：

```
CLion 2018.1.2
Build #CL-181.4668.70, built on April 25, 2018
JRE: 1.8.0_152-release-1136-b29 x86_64
JVM: OpenJDK 64-Bit Server VM by JetBrains s.r.o
macOS 10.13.4
```

全部代码如下：

```
#include<iostream>
#include<string>
using namespace std;
#define MAX 1048576//限定系统的最大内存大小
#define LL long long


class Process
{
public:
    string NAME;
    LL SIZE;//一个进程占据的内存
    LL START;//开始地址
    LL END;//结束地址
    Process *NEXT;//尾指针
    Process *BEFORE;//头指针
};
//链表表头，存储进程数
class Head
{
public:
    LL count;
    Process *next;
};
//定义挂起队列
class Out
{
public:
    LL count;
    Process *next;
};

//实现向可用内存中加入进程：从内存空间由上往下搜索合适的空间并装入内存
LL ADD_Pro(Head *head, Process *p)//返回1,表示插入成功；返回0表示插入失败
{
    Process *p1 = head->next;
    //搜索插入位置
    if(0 == p1)//内存为空
    {
        head->next=p;
```

```
            p->START = 10*1024;
            p->END = p->SIZE + 10240;
            p->BEFORE = 0;
            p->NEXT = 0;
            return 1;
        }
        else//如果内存中有进程
        {
            LL temp = p1->START - 10240;
            if(p->SIZE <= temp)//如果第一个进程前面有足够空间，cut in before p1
            {
                p->NEXT=head->next;
                head->next->BEFORE = p;
                p->BEFORE = 0;
                head->next = p;
                p->START = 10240;
                p->END = p->SIZE+10240;
                return 1;
            }
            else//在两个进程间寻找合适位置
                while(p1->NEXT != 0)//当p1不是指向最后一个进程
                {
                    temp = p1->NEXT->START - p1->END;//以下有重复代码，可优化
                    if(p->SIZE <= temp)//搜索到了合适位置,cut in after p1
                    {
                        p->NEXT= p1->NEXT;
                        p1->NEXT->BEFORE = p;
                        p1->NEXT=p;
                        p->BEFORE = p1;
                        p->START = p1->END;
                        p->END = p->START + p->SIZE;
                        return 1;
                    }
                    p1 = p1->NEXT;
                }
            //都没找到可插入的位置
            if((p1->END + p->SIZE) <= MAX)//内存足够大，允许进程插入到内存末尾,此时p1
指向最后一个节点
            {
                p->NEXT= 0;
                p->BEFORE = p1;
                p1->NEXT=p;
                p->START = p1->END;
                p->END = p->START+p->SIZE;
                return 1;
            }
            else
                return 0;//返回0，表示插入失败
        }
    }
    //对ADD_Pro的调用
    void CALL_ADD_Pro(Head *head)
    {
        Process *p = new Process;//生成进程
```

```
    cout<<"请输入进程名：";
    cin>>p->NAME;
    cout<<"请输入进程大小：\n";
    LL kb,bb;
    cout<<"多少KB？";
    cin>>kb;
    cout<<"多少B?";
    cin>>bb;
    p->SIZE = 1024 * kb +bb;
    p->NEXT = 0;
    p->BEFORE = 0;
    kb = ADD_Pro(head, p);
    if(kb == 0)
        cout<<"进程添加失败！\n";
    else cout<<"进程添加成功！\n";
    return;
}

//杀掉进程
void KILL(Head *head)
{
    cout<<"请输入要杀掉的进程名：";
    string nam;
    cin>>nam;
    Process *p = head->next;
    while(p)
    {
        if(p->NAME == nam)
            break;
        p = p->NEXT;
    }
    if(0 == p)
    {
        cout<<"未找到进程："<<nam<<"！"<<endl;
        return;
    }

    //删除就绪队列中p节点
    if(head->next == p)//p是第一个节点
    {
        head->next = p->NEXT;
        if(0 != p->NEXT)
            p->NEXT->BEFORE = 0;
    }
    else if(p->NEXT == 0)//p是最后一个节点
    {
        p->BEFORE->NEXT = 0;
        p->BEFORE = 0;
    }
    else//中间节点
    {
        p->NEXT->BEFORE = p->BEFORE;
        p->BEFORE->NEXT = p->NEXT;
    }
```

```
    delete p;
    cout<<"成功删除进程:"<<nam<<'!'<<endl;
}
//挂起进程
void OUT_OF_Pro(Head *head, Out *out)
{
    cout<<"请输入要挂起的进程名: ";
    string nam;
    cin>>nam;
    Process *p = head->next;;//指向移动源head链表
    Process *p1 = out->next;;//指向移动目的out链表
    while(p != 0)//搜索要挂起的进程
    {
        if(p->NAME == nam)
            break;
        p = p->NEXT;
    }
    if(0 == p)//无相关进程
    {
        cout<<"找不到进程: "<<nam<<"! "<<endl;
        return;
    }
    //处理就绪队列,p指向要解挂节点
    if(head->next == p)//要挂起的进程在第一个
    {
        head->next = p->NEXT;
        if(0 != p->NEXT)//有至少两个链表
            p->NEXT->BEFORE = 0;
    }
    else if(p->NEXT == 0)//要挂起的进程在最后一个
    {
        p->BEFORE->NEXT=0;
        p->BEFORE=0;
    }
    else //要挂起的进程在中间
    {
        p->BEFORE->NEXT = p->NEXT;
        p->NEXT->BEFORE = p->BEFORE;
    }//done

    //处理挂起序列, p指向要插入挂起队列的节点, p1指向挂起队列第一个节点
    if(0 == p1)//挂起链表为空
    {
        out->next = p;
        p->BEFORE = 0;
    }
    else
    {
        while(0 != p1->NEXT){
            p1 = p1->NEXT;
        }//p1指向挂起进程链表的末尾
        p1->NEXT = p;
        p->BEFORE = p1;
    }
```

```
    p->NEXT = 0;//done
}
//解挂进程
void rein_pro(Head *head,Out *out)
{
    cout<<"请输入要解挂进程的名字: ";
    string nam;
    cin>>nam;
    Process *p = out->next;;//指向移动源out链表
    while(0 != p)
    {
        if(p->NAME == nam)
            break;
        p = p->NEXT;
    }
    if(0 == p)//无相关进程
    {
        cout<<"无相关进程: "<<nam<<"! "<<endl;
        return;
    }

    //删除挂起队列中的p
    if(out->next == p)//要删除的进程在第一个
    {
        out->next = p->NEXT;
        if(0 != p->NEXT)//如果p后面还有节点
            p->NEXT->BEFORE = 0;
    }
    else if(p->NEXT == 0)//要删除的进程在最后一个
    {
        p->BEFORE->NEXT=0;
    }
    else//要挂起的进程在中间
    {
        p->BEFORE->NEXT = p->NEXT;
        p->NEXT->BEFORE = p->BEFORE;
    }
    ADD_Pro(head, p);
    cout<<"成功解挂进程: "<<nam<<endl;
    return;
}

int main()
{
    Head *head = new Head;//定义内存进程头节点
    head->next = 0;
    head->count =0;

    Out *out = new Out;//定义挂起进程头节点
    out->next = 0;

    LL temp = 0;

    Process *p = head->next;
```

```
    Process *p1 = out->next;
    while(1)
    {
        cout<<"操作系统内存管理模拟实验！限定内存大小1M\n"<<endl;
        cout<<"内存进程存储状态："<<endl;
        cout<<"\t操作系统占用：10KB"<<endl;
        p = head->next;
        while(0 != p)
        {
            cout<<"\t进程名:"<<p->NAME<<"\t大小:"<<(p->SIZE)/1024<<"K"<<(p-
>SIZE)%1024<<"B\t起始地址:"<<p->START<<"结束地址:"<<p->END<<endl;
            p = p->NEXT;
        }
        p = head->next;
        cout<<"内存剩余空间段：\n";
        if(0 == p)//无进程
            cout<<'\t'<<MAX;
        else//有进程
        {
            if((p->START-10240)!= 0)
                cout<<'\t'<<(p->START-10240)<<endl;
            if(0 == p->NEXT)//仅有一个进程时
                cout<<'\t'<<MAX-(p->END)<<endl;
            else while(1)//至少有两个进程，p->next != 0
                {
                    temp = p->NEXT->START - p->END;
                    if(temp)
                        cout<<'\t'<<temp<<endl;
                    p = p->NEXT;
                    if(0 == p->NEXT)//p指向最后一个节点时
                    {
                        cout<<'\t'<<MAX-(p->END)<<endl;
                        break;
                    }
                }
        }

        cout<<"\n挂起队列："<<endl;
        p1 = out->next;
        while(p1)
        {
            cout<<"\t进程名:"<<p1->NAME<<"\t大小:"<<(p1->SIZE)/1024<<"K"
<<(p1->SIZE)%1024<<'B'<<endl;
            p1 = p1->NEXT;
        }
        cout<<"\n1、加入进程\n2、挂起进程\n3、解挂进程\n4、杀掉进程\n5、退出系统"
<<endl;
        cout<<"\n请选择需要进行的操作：";
        LL chose;
        cin>>chose;
        switch(chose)
        {
            case 1:
                CALL_ADD_Pro(head);
```

```
                    break;
                case 2:
                    OUT_OF_Pro(head, out);
                    break;
                case 3:
                    rein_pro(head,out);
                    break;
                case 4:
                    KILL(head);
                    break;
                case 5:
                    return 0;
                default:
                    return 0;
            }
        }
    }
```