山西工程技术学院

实验报告

（ 2023 - 2024 学年第 学期）

课程名称：操作系统实验

专业班级：

学　　号：

学生姓名：

任课教师：

2024 年 6 月

实验报告

实验名称	内存的分配与回收			指导教师	
实验类型	验证型	实验学时	4	实验时间	2024.6

一、实验目的与要求

（1） 掌握内存分配与回收的概念

（2） 进一步加深内存分配与回收各种算法的理解

二、实验环境

Vscode

三、实验内容和步骤

模拟用可变分区的方式管理内存的实验。

本实验模拟的是用可变分区的方式管理内存的，使用两张表来管理内存，一张表是已经分配的内存块表，另一个张表是未分配的内存块表，两张表针对同一内存空间进行管理。

三种内存分配算法介绍

（1）首次适应算法 各个空闲地址块按照首地址从小到大的顺序排列，找到第一个能够满足某程序要求空间大小的空闲块分配给该程序。

（2）最佳适应算法 各个空闲块按照空间地址范围大小从小到大排列，找到第一个能够满足某程序要求空间大小的空闲块分配给该程序。

（3）最坏适应算法 各个空闲块按照空间地址范围大小从大到小排列，找到第一个能够满足某程序要求空间大小的空闲块分配给该程序。

内存块回收算法介绍 当某程序的内存空间被释放，需要把相应的内存空间回收，该内存块由已分配状态变为未分配状态。这里需要注意的是，该内存块和原有的空闲内存块有上邻、下邻、上下邻、上下都不邻的四种情况，如有相邻，要合并。

```
#include<stdio.h>
#include<iostream>
#include<cstdio>
#include<iomanip>
#include<malloc.h>
using namespace std;
typedef struct SNode{
    int id;
```

```
    int ad;
    int data;
    SNode *front;
    SNode *next;
}Snode;
void Insertlist(Snode *&l,int id,int ad,int data)
{
    Snode *s;
    s=(Snode *)malloc(sizeof(Snode));
    s->ad=ad;
    s->data=data;
    s->id=id;
    s->next=l->next;
    if(l->next!=NULL)
        l->next->front=s;
    s->front=l;
    l->next=s;
}
    int Changelist(Snode *&l,int data1)
{
    Snode *s;
    int ad=-1;
    s=l->next;
    while(s!=NULL)
    {
        if(s->data>=data1)
        {
            ad=s->ad;
            if(s->data==data1)
            {
                if(s->next!=NULL)
                {
                s->front->next=s->next;
                s->next->front=s->front;
```

```
                    free(s);
                    }
                    else
                    {
                         s->front->next=NULL;
                         free(s);
                    }
               }
               else
               {
                    s->data=s->data-data1;
                    s->ad=s->ad+data1;
               }
          return ad;
          }
          s=s->next;
      }
     return -1;
}
void InsertM(int M[],int ad,int data,int id)
{
     int p=0;
     for(int i=ad;i<ad+data;i++)
     {
          M[i]=id;
     }
}
int FindID(int M[],int record[],int id,int m)
{
     int c=-1;
     m++;
     record[0]=-1;record[1]=-1;
     for(int i=0;i<m;i++)
     {
```

```
        if(M[i]==id)
        {
            record[0]=i;
            for(int j=i;j<m;j++)
            {
                c++;
                if(M[j]!=id)
                {
                    record[1]=c;
                    return 1;
                }
                M[j]=-1;
            }
        }
    }
    return 0;
}
void Sortlist(Snode *&l,int j)
{
    Snode *f1;
    Snode *f2;
    Snode *f3;
    f1=l->next;
    f2=f1;
    f3=f1;
    while(f1->next!=NULL)
    {
            if(j==0)
             while(f3!=NULL)
            {

                if(f3->ad<f2->ad)
                f2=f3;
                f3=f3->next;
```

```
                }
                if(j==1)

while(f3!=NULL)
               {
                   if(f3->data<f2->data)
                   f2=f3;
                   f3=f3->next;
                }
                if(j==2)
                while(f3!=NULL)
               {

                   if(f3->data>f2->data)
                   f2=f3;
                   f3=f3->next;
                }
                if(f1!=f2)
                {
                   if(f2->next!=NULL)
                     {
                        f2->front->next=f2->next;
                        f2->next->front=f2->front;
                        f2->next=f1;
                        f1->front->next=f2;
                        f2->front=f1->front;
                        f1->front=f2;

                      }
                      else
                      {
                        f2->front->next=NULL;
                        f2->next=f1;
                        f1->front->next=f2;
```

```
                    f2->front=f1->front;
                    f1->front=f2;
                  }
              }
              else
              {
                  f1=f1->next;
              }
              f2=f1;
              f3=f2;
      }
  }
void Recyclelist(Snode *&l,int ad)
{
    Snode *s;
    Snode *s1;
    Snode *ad1;
    int j=0;
    ad1=l->next;
    while(ad1->ad!=ad)
    {
        ad1=ad1->next;
    }
    s=ad1;
        if(s->front!=l&&s->front->ad+s->front->data==s->ad)
        {
            if(s->next==NULL)
            {
                s->front->data=s->front->data+s->data;
                s->front->next=NULL;
                free(s);
                s=NULL;

            }
```

```
            else{
            s->front->data=s->front->data+s->data;
            s->front->next=s->next;
            s->next->front=s->front;
            s1=s->front;
            free(s);
            s=s1;

            }
        }
        if(s!=NULL&&s->next!=NULL&&s->ad+s->data==s->next->ad)
        {
            if(s->next->next==NULL)
            {
                s->data=s->data+s->next->data;
                free(s->next);
                s->next=NULL;
            }
else{
            s=s->next;
            s->front->data=s->front->data+s->data;
            s->front->next=s->next;
    s->next->front=s->front;
            free(s);
            }
        }
 }
int main()
{
    Snode *l;
    int record[2];
    int m;
    cout<<"请输入内存初始大小:";
    cin>>m;//存储内存
```

```
int M[m+1];
M[m]=-2;
cout<<"初始内存状态如下(-1 表示该单位空闲):\n";
for(int i=0;i<m;i++)
{
     M[i]=-1;
     cout<<M[i]<<" ";
}
l=(Snode *)malloc(sizeof(Snode));
l->next=NULL;
l->front=NULL;
Insertlist(l,-1,0,m);
int getselect=-1;
int getid=-1;
int getdata=-1;
int b;
int j;
while(1)
{
     getid=getdata=-1;
     getselect=-1;
     j=-1;
     cout<<"\n 请输入进行的步骤：（0 代表插入进程，1 代表回收进程）：";
     cin>>getselect;
     if(getselect==0)
     {
          cout<<"请输入进程 ID 及大小:\n";
          cin>>getid;
          cin>>getdata;
          cout<<"请输入选择的插入方法，0-FF，1-BF,2-WF:";
          cin>>j;
          Sortlist(l,j);

b=Changelist(l,getdata);
```

```
            if(b!=-1)
            {
                InsertM(M,b,getdata,getid);
            }
            else
            cout<<"程序过大，插入失败";
        }
        if(getselect==1)
        {
            cout<<"请输入查找 ID: ";
            cin>>getid;
            if (FindID(M,record,getid,m))
            {
                Insertlist(l,-1,record[0],record[1]);
                Sortlist(l,0);
                Recyclelist(l,record[0]);
            }
            else
            cout<<"查无进程，回收失败";
        }
        cout<<"\n 内存情况(-1 表示空闲,数字表示相应 ID 占用空间)：\n";
        for(int i=0;i<m;i++)
        {
        cout<<" "<<M[i];
        }
         cout<<"\n 空闲链表情况(表头地址-空闲大小):\n";
        Snode *s5;
        s5=l->next;
        while(s5 !=NULL)
        {
            cout<<"->"<<s5->ad<<"-"<<s5->data;
            s5=s5->next;
        }
    }
```

```
    return 0;

}
```

```
请输入内存初始大小:10
初始内存状态如下(-1表示该单位空闲):
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1
请输入进行的步骤：（0代表插入进程，1代表回收进程）：0
请输入进程ID及大小：
1 1
请输入选择的插入方法，0-FF，1-BF,2-WF：0

内存情况(-1表示空闲,数字表示相应ID占用空间):
 1 -1 -1 -1 -1 -1 -1 -1 -1 -1
空闲链表情况(表头地址-空闲大小):
->1-9
请输入进行的步骤：（0代表插入进程，1代表回收进程）：0
请输入进程ID及大小：
2 2
请输入选择的插入方法，0-FF，1-BF,2-WF：0

内存情况(-1表示空闲,数字表示相应ID占用空间):
 1 2 2 -1 -1 -1 -1 -1 -1 -1
空闲链表情况(表头地址-空闲大小):
->3-7
请输入进行的步骤：（0代表插入进程，1代表回收进程）：0
请输入进程ID及大小：
3 3
请输入选择的插入方法，0-FF，1-BF,2-WF：0

内存情况(-1表示空闲,数字表示相应ID占用空间):
 1 2 2 3 3 3 -1 -1 -1 -1
空闲链表情况(表头地址-空闲大小):
->6-4
请输入进行的步骤：（0代表插入进程，1代表回收进程）：1
请输入查找ID：2

内存情况(-1表示空闲,数字表示相应ID占用空间):
 1 -1 -1 3 3 3 -1 -1 -1 -1
空闲链表情况(表头地址-空闲大小):
->1-2->6-4
请输入进行的步骤：（0代表插入进程，1代表回收进程）：
```

四、实验小结和思考

在本次实验中，我们模拟实现了内存管理中的可变分区分配方案，并重点测试了三种内存分配算法（首次适应算法、最佳适应算法和最坏适应算法）以及内存回收算法。在本次实验中，我们模拟实现了内存管理中的可变分区分配方案，并重点测试了三种内存分配算法（首次适应算法、最佳适应算法和最坏适应算法）以及内存回收算法。

实验成绩		批阅日期		批阅人	