# 暨南大学本科实验报告专用纸

课程名称 操作系统原理 成绩评定

实验项目名称 动态分区分配方式 指导教师 赵阔

实验项目编号 0806002905 实验项目类型 设计型 实验地点

学生姓名 陈旭天 学号 2021100733

学院 智能科学与工程学院 系 人工智能 专业

实验时间 2023 年 5 月 4 日 上 午～ 2023 月 5 日 4 午 温度 ℃湿度

# 暨南大学本科实验报告专用纸(附页)

## 一、实验目的和要求

了解动态分区分配方式中使用的数据结构和分配算法，进一步加深对动态分区存储管理方式及其实现过程的理解。提高学生设计实验、发现问题、分析问题和解决问题的能力，并学习撰写规范的科学研究报告。

1.用 C 或其他语言分别实现采用首次适应算法和最佳适应算法的动态分区分配过程和回收过程。

2．设置初始状态，每次分配和回收后显示出空闲内存分区链的情况。

## 二、实验原理和主要内容

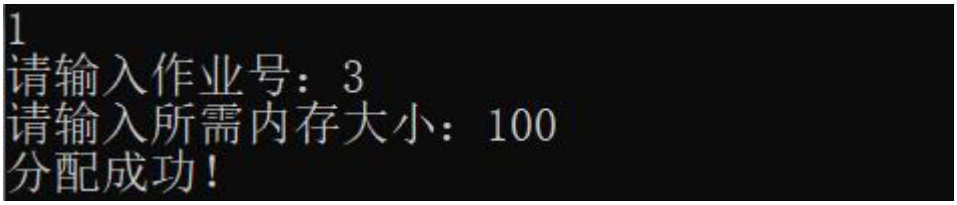
**最佳适应算法**：首先，定义一个 p 指针，让 p 指针遍历空闲分区链表，当找到第一个满足进程请求空间大小的空闲区时，记录此位置，并且保存请求大小与空闲分区实际大小的差值记为 a，然后让 p 指针继续遍历空闲分区链表，每当满足请求内存大小小于空闲区大小时，就记录两者的差值并且记录为 b，比较 a 与 b 的大小关系，当 a>b 时，将 b 的值赋予 a，并且修改记录位置为此空闲区的位置。若，a<=b,不做操作。继续遍历链表，重复上面的操作，直到 p->next 指向 null 为止。

**首次适应算法**：首次适应算法比较简单，只要找到满足条件的空闲区，就将此区的空间分配给进程。首先，用 P 指针遍历链表，找到第一个空间大于或者等于请求大小的位置，将此空间分配给进程，当此空闲区大小大于请求空间大小时，将空闲区分为两部分，一部分分配给进程，另一部分为空闲区，它的大小为之前空闲区大小减去分配给进程的空间大小。

**内存回收算法**：内存回收时，回收分区与空闲分区有四种关系。第一种情况为回收分区 r 上临一个空闲分区，此时应该合并为一个连续的空闲区，其始址为 r 上相邻的分区的首地址，而大小为两者大小之和。第二种情况为回收分区 r 与下相邻空闲分区，合并后仍然为空闲区，该空闲区的始址为回收分区 r 的地址。大小为两者之和，第三种情况为回收部分 r 与上下空闲区相邻，此时将这三个区域合并，始址为 r 上相邻区域的地址，大小为三个分区大小之和。当回收部分 r 上下区域都为非空闲区域，此时建立一个新的空闲分区，并且加入到空闲区队列中去。

## 三、C 程序源码

1. 实验结果






2. 实验源码

```
#include <iostream>
#include<stdlib.h>
using namespace std;

#define FREE 0
#define    BUSY 1
#define    MAX_length 640

typedef struct freeArea//首先定义空闲区分表结构
  {
int flag;
int size;
int ID;
int address;
  }Elemtype;

typedef struct Free_Node
{
      Elemtype date;
      struct Free_Node *front;
      struct Free_Node *next;
```

```
}Free_Node,*FNodeList;

FNodeList block_first;
FNodeList block_last;
int alloc(int tag);//内存分配
int free(int ID);//内存回收
int first_fit(int ID,int size);//首次适应算法
int best_fit(int ID,int size);//最佳适应算法
void show();//查看分配
void init();//初始化
void Destroy(Free_Node *p);//销毁节点
void menu();
void init()//
{
    block_first=new Free_Node;
    block_last = new Free_Node;
    block_first->front=NULL;
    block_first->next=block_last;
    block_last->front=block_first;
    block_last->next=NULL;
    block_last->date.address=0;
    block_last->date.flag=FREE;
    block_last->date.ID=FREE;
    block_last->date.size=MAX_length;
}

//实现内存分配
int alloc(int tag)
{
    int ID,size1;
    cout<<"请输入作业号：";
    cin>>ID;
    cout<<"请输入所需内存大小：";
    cin>>size1;
    if (ID<=0 || size1<=0)
    {
        cout<<"输入错误！请输入正确的 ID 和请求大小："<<endl;
        return 0;
    }

    if (tag==1)//采用首次适应算法
    {
        if(first_fit(ID,size1))
        {
```

```
                cout<<"分配成功！"<<endl;
            }
            else cout<<"分配失败！"<<endl;
            return 1;
        }
        else
        {
            if (best_fit(ID,size1))
            {
                cout<<"分配成功！"<<endl;
            }
            else cout<<"分配失败！"<<endl;
            return 1;
        }

}

int first_fit(int ID,int size)//首次适应算法
{
        FNodeList temp=(FNodeList)malloc(sizeof(Free_Node));
        Free_Node *p=block_first->next;
        temp->date.ID=ID;
        temp->date.size=size;
        temp->date.flag=BUSY;
       while(p)
       {
            if (p->date.flag==FREE && p->date.size==size)//请求大小刚好满足
            {
                p->date.flag=BUSY;
                p->date.ID=ID;
                return 1;
                break;
            }
            if (p->date.flag==FREE && p->date.size>size)//说明还有其他的空闲区间
            {
                temp->next=p;
                temp->front=p->front;
                temp->date.address=p->date.address;
              p->front->next=temp;
              p->front=temp;
              p->date.address=temp->date.address+temp->date.size;
              p->date.size-=size;
              return 1;
              break;
```

```
            }
            p=p->next;
        }
        return 0;
}

int best_fit(int ID,int size)//最佳适应算法
{
        int surplus;//记录可用内存与需求内存的差值
        FNodeList temp=(FNodeList)malloc(sizeof(Free_Node));
        Free_Node *p=block_first->next;
        temp->date.ID=ID;
        temp->date.size=size;
        temp->date.flag=BUSY;
        Free_Node *q=NULL;//记录最佳位置
        while(p)//遍历链表，找到第一个可用的空闲区间将他给 q
        {
            if (p->date.flag==FREE&&p->date.size>=size)
            {
                q=p;
                surplus=p->date.size-size;
                break;
            }
            p=p->next;
        }
        while(p)//继续遍历，找到更加合适的位置
        {
            if (p->date.flag==FREE&&p->date.size==size)
            {
                p->date.flag=BUSY;
                p->date.ID=ID;
                return 1;
                break;
            }
            if (p->date.flag==FREE&&p->date.size>size)
            {
                if (surplus>p->date.size-size)
                {
                    surplus=p->date.size-size;
                    q=p;
                }

            }
            p=p->next;
```

```
        }
        if (q==NULL)//如果没有找到位置
        {
            return 0;
        }
        else//找到了最佳位置
        {
            temp->next=q;
            temp->front=q->front;
            temp->date.address=q->date.address;
            q->front->next=temp;
            q->front=temp;
            q->date.size=surplus;
            q->date.address+=size;
            return 1;
        }
}

int free(int ID)//主存回收
{
        Free_Node *p=block_first->next;
        while(p)
        {
            if (p->date.ID==ID)//找到要回收的 ID 区域
            {
                p->date.flag=FREE;
                p->date.ID=FREE;
                //判断 P 与前后区域关系
                if (p->front->date.flag==FREE&&p->next->date.flag!=FREE)
                {
                    p->front->date.size+=p->date.size;
                    p->front->next=p->next;
                    p->next->front=p->front;
                }
                if (p->front->date.flag!=FREE&&p->next->date.flag==FREE)
                {
                    p->date.size+=p->next->date.size;
                    if(p->next->next)
                    {
                        p->next->next->front=p;
                    p->next = p->next->next;
                    }
                    else p->next=p->next->next;
                }
```

```
                if(p->front->date.flag==FREE&&p->next->date.flag==FREE)
                {
                    p->front->date.size+=p->date.size+p->next->date.size;
                    if(p->next->next)
                    {
                    p->next->next->front=p->front;
                    p->front->next=p->next->next;
                    }
                    else p->front->next=p->next->next;
                }
                if(p->front->date.flag!=FREE&&p->next->date.flag!=FREE)
                {//

                }
                break;
            }
            p=p->next;
        }
        cout<<"回收成功！"<<endl;
        return 1;
}
void Destroy(Free_Node *p)
{

}
void show()
{
        cout<<"*******内存分配情况*******"<<endl;
        Free_Node *p=block_first->next;
        while(p)
        {
            cout<<"分区号：";
            if (p->date.ID==FREE)
            cout<<"FREE"<<endl;
            else cout<<p->date.ID<<endl;
            cout<<"起始地址："<<p->date.address<<endl;
            cout<<"内存大小："<<p->date.size<<endl;
            cout<<"分区状态：";
            if (p->date.flag==FREE)
                cout<<"空闲"<<endl;
            else
                 cout<<"已分配"<<endl;
            cout<<"**************************"<<endl;
            p=p->next;
```

```
        }
    }
    void menu()//菜单
    {
        int tag=0;
        int ID;
        init();
        cout<<"\t\t\t\t\t 动态分区分配方式的模拟"<<endl;
        while(tag!=5)
        {
            cout<<"\t\t************************* 请 选 择 要 进 行 的 操 作 *************************"<<endl;
            cout<<"\t\t\t\t\t1:首次适应算法\n\t\t\t\t\t2:最佳适应算法\n\t\t\t\t\t3:内存回收\n\t\t\t\t\t4:显示内存状况\n\t\t\t\t\t5:退出"<<endl;
                cin>>tag;
            switch(tag)
                {
                case 1:
                    alloc(tag);
                        break;
                case 2:
                        alloc(tag);
                        break;
                case 3:
                    cout<<"请输入需要回收的作业号：";
                    cin>>ID;
                    free(ID);
                        break;
                case 4:
                    show();
                    break;
                }
        }

    }
    int main()
    {
        menu();
        return 0;
    }
```

# 四、实验总结

存储管理可以有效地对外部存储资源和内存进行管理，可以完成存储分配，存储共享，存储保护，存储扩充，地址映射等重要功能，对操作系统的性能有很重要的影响。首次适应算法和最佳适应算法是存储管理中两个十分重要的页面置换算法。

首次适应算法从空闲分区链首开始查找，直至找到一个能满足其大小要求的空闲分区为止。然后再按照作业的大小，从该分区中划出一块内存分配给请求者，余下的空闲分区仍留在空闲分区链中。该算法倾向于使用内存中低地址部分的空闲区，在高地址部分的空闲区很少被利用，从而保留了高地址部分的大空闲区，为以后到达的大作业分配大的内存空间创造了条件。但是低地址部分不断被划分，留下许多难以利用、很小的空闲区，而每次查找又都从低地址部分开始，会增加查找的开销。

最佳适应算法总是把既能满足要求，又是最小的空闲分区分配给作业。为了加速查找，该算法将所有的空闲区按大小排序后，以递增顺序形成一个空白链。这样每次找到的第一个满足要求的空闲区，必然是最优的。这样，每次分配给文件的都是最合适该文件大小的分区，但是内存中留下许多难以利用的小的空闲区。