附件1:OPT算法实现代码

```
//OPT算法 最佳淘汰算法
class Optimal{
public:
    Optimal(const vector<short> &A)
    {
        a = vector<short>(A);
        //把类的a初始化为A
        pageno.resize(A.size());
    }

    void run()
    {
        getPageNo();
        ofstream out("opt.txt");
        for(int pageAssigned = 4;pageAssigned<=maxN;pageAssigned+=2)//实存容量从
4-32页
        {
            printf("PAGE NUMBER WITH SIZE %dk EACH ADDRESS IS:\n",
                   pagesize/1024);
            vector<unsigned char> memory;//内存
            int  miss= 0;//记录缺页次数
            for(int i = 0;i<pageno.size();i++)//每一个要装入的page都进入
            {
                int index=find(memory.begin(),memory.end(),pageno[i])-
memory.begin();
                //find函数如果在memory.end()表示没有找到
                //如果在memory.begin()和memory.end()之间表示找到了
                //find-memory.begin()表示在第几个
                if(index!= memory.end()-memory.begin())  //该页命中
                {
                    if(debug)//给自己debug时候看的
                        cout << "命中\n";
                }
                else//没命中的时候
                {
                    if(memory.size()< pageAssigned) //还有空余的内存
                    {
                        if(debug)
                            cout << "装入\n";
                        memory.push_back(pageno[i]);
                    }
                    else//没有空余内存 要置换
                    {
                        if(debug)
```

```
                            cout << "置换\n";

                        int maxindex = i;
                        int position = 0;
                        for(int j = 0;j<memory.size();j++)//用j来找出要被换走的页面-OPT算法是找到pageno中最晚出现的页
                        {
                            int k = find(pageno.begin()+i+1,pageno.end(),memory[j])-pageno.begin();
                            //k表示找到memory[j]所表示页面的位置
                            if(k == pageno.end()-(pageno.begin()+i+1))
                            {
                                //表示memory[j]表示的页是画物理内存最后一页了
                                maxindex = k;
                                position = j;
                                break;
                            }
                            else
                                //表示找到了memory[j]表示的页
                            {
                                if(k > maxindex)//ii比当前的maxindex大 所以更新maxindex
                                {
                                    maxindex = k;
                                    position = j;
                                } } }
                        memory[position] = pageno[i]; //置换完毕
                    }
                    miss++;//缺页次数+1
                }
                if(debug)//查看一下每个memory里面放的页
                {
                    for(int k = 0;k < memory.size();k++)
                        printf("%d ", memory[k]);
                    cout << endl;
                }
            }
            printf("%-10d %E\n",pageAssigned,miss*1.0/pageno.size());
            out << miss*1.0/pageno.size() << endl;
            pagesize *= 2;
            if(debug)
                break;
        }
        out.close();
    }


private:
    void getPageNo()
```

```
    {
        for(int i =0;i<a.size();i++)
        {
            pageno[i] = a[i]/pagesize+1;
            if(debug)
            {
                printf("pageno[%03d]=%-10d",i, pageno[i]);
                if((i+1)%4 == 0)
                    cout << endl;
            }

        }

    }
    vector<short> a;
    vector<unsigned char> pageno;
    int pagesize = 1024;
};
```

附件2:FIFO算法实现代码

```
//FIFO 先进先出算法
class FIFO{
public:
    FIFO(const vector<short> &A)
    {
        a = vector<short>(A);
        pageno.resize(A.size());
    }

    void run()
    {
        getPageNo();
        ofstream out("fifo.txt");
        for(int pageAssigned = 4;pageAssigned<=maxN;pageAssigned+=2)
        {
            printf("PAGE NUMBER WITH SIZE %dk EACH ADDRESS
IS:\n",pagesize/1024);
            vector<unsigned char> memory;
            int miss = 0;
            for(int i = 0;i<pageno.size();i++)
            {
                int index=find(memory.begin(),memory.end(),pageno[i])-
memory.begin();
```

```
                if(index != memory.end()-memory.begin())
                {
                    if(debug)
                        cout << "命中\n";
                }
                else
                {
//                    cout << "缺页\n";
                    if(memory.size() < pageAssigned)
                        //有空闲页
                    {
                        memory.push_back(pageno[i]);
                        if(debug)
                            cout << "装入\n";
                    }
                    else
                    {
                        //置换
                        if(debug)
                            cout << "置换\n";
                        for(int k = 1;k<memory.size();k++)
                        {
                            //所有的页往前移动一个 空出最后一个位置给新页
                            memory[k-1] = memory[k];
                        }
                        memory[memory.size()-1] = pageno[i];
                    }
                    miss++;
                }

                if(debug)
                {
                    for(int k = 0;k < memory.size();k++)
                        printf("%d ", memory[k]);
                    cout << endl;
                }
            }
            printf("%-10d %E\n",pageAssigned,miss*1.0/pageno.size());
            out << miss*1.0/pageno.size() << endl;
            pagesize *= 2;
            if(debug)
                break;
        }
        out.close();
    }
private:
    void getPageNo()
    {
        for(int i =0;i<a.size();i++)
```

```
        {
            pageno[i] = a[i]/pagesize+1;
            if(debug)
            {
                printf("pageno[%03d]=%-10d",i, pageno[i]);
                if((i+1)%4 == 0)
                    cout << endl;
            }
        }
    }
    vector<short> a;
    vector<unsigned char> pageno;
    int pagesize = 1024;
    int clock = 0;
};
```

附件3:LRU算法实现代码

```
//LRU 最近最少使用
class LRU{
public:
    LRU(const vector<short> &A)//构造
    {
        a = vector<short>(A);
        pageno.resize(A.size());
    }

    void run()
    {
        getPageNo();
        ofstream out("lru.txt");
        for(int pageAssigned = 4;pageAssigned<=maxN;pageAssigned+=2)
        {
            printf("PAGE NUMBER WITH SIZE %dk EACH ADDRESS
IS:\n",pagesize/1024);
            vector<unsigned char> memory;
            vector<int> clocks;//记录最近的使用时间
            vector<int> clocks_stack;//用栈的方式实现LRU
            int miss= 0;
            int miss_stack=0;
            vector<string> registers;//用寄存器的方式实现LRU
            for(int i = 0;i<pageno.size();i++)
            {
                int index=find(memory.begin(),memory.end(),pageno[i])-
memory.begin();
```

```
                //求index的过程就是在物理内存中寻找对应页是否存在的过程
                if(index != memory.end()-memory.begin())//没有缺页
                {
                    vector<int>::iterator
pos=find(clocks_stack.begin(),clocks_stack.end(),pageno[i]);//在栈中的位置pos
                    clocks_stack.erase(pos);//删除在栈里的index
                    clocks_stack.push_back(pageno[i]);//pageno[i]放到栈顶
                    for(int r=0;r<pagesize/1024;r++)
                    {
                        if(r!=pageno[i])
                            registers[i]='0'+registers[i];
                        else
                            registers[i]='1'+registers[i];
                    }
                    if(debug){
                        cout << "命中\n";
                    }
                }
                else
                {
                    if(debug)
                        cout << "缺页\n";
                    if(memory.size() < pageAssigned)//有空闲页 可以直接装入
                    {
                        memory.push_back(pageno[i]);//把此页装到最后空闲位置
                        clocks_stack.push_back(pageno[i]);
                        if(debug)
                            cout << "装入\n";
                    }
                    else//置换页面
                    {
                        if(debug)
                            cout << "置换\n";
                        int minclocks_stack=clocks_stack.front(); //栈底是最近未
使用的页面
                        clocks_stack.erase(clocks_stack.begin());
                        vector<unsigned char>::iterator
pos2=find(memory.begin(),memory.end(),minclocks_stack);
                        pos2=memory.erase(pos2);
                        pos2=memory.insert(pos2,pageno[i]);
                        clocks_stack.push_back(pageno[i]);
                    }
                    miss++;
                }
                if(debug)
                {
                    for(int k = 0;k < memory.size();k++)
                        printf("%d ", memory[k]);
                    cout << endl;
```

```
                } }
            printf("%-10d %E\n",pageAssigned,miss*1.0/pageno.size());
            // miss/pageno.size()就是缺页率
            out << miss*1.0/pageno.size() << endl;
            pagesize *= 2;
        }
        out.close();
    }


private:
    void getPageNo()
    {
        for(int i =0;i<a.size();i++)
        {
            pageno[i] = a[i]/pagesize+1;
            if(debug)
            {
                printf("pageno[%03d]=%-10d",i, pageno[i]);
                if((i+1)%4 == 0)
                    cout << endl;
            }
        }
    }
    vector<short> a;
    vector<unsigned char> pageno;
    int pagesize = 1024;
    int clock = 0;
};
```