虚拟存储器

一、 问题描述

设：存储器 int[] Memory=new int[16K];系统以 zjie 寻址，虚拟存储每页 4Kzjie。(zjie=short)，编写函数：
int LW(int adr);和 void SW(int adr, int dat);
其中：adr 范围：0~1Gzjie。
做好分析，说明采用单页映射、反向页表、正向页表、多级页表哪种页表方式，每种方式的优缺点对比。
验证：设计一随机存储器访问序列，并计算其命中率。

二、 解决思路

由题可知，物理存储器的页数为 16K / 4K = 4，虚拟存储器的页数为 1G / 4K = 262144。采用正向页表的方式，则页表中共有 262144 项，每个页项包括一个有效位以及该页的物理地址（为简单起见，不考虑其他特殊位）。

LW 与 SW 操作实现的主要思路为：首先根据地址得到要访问的页号，根据页表该项的有效位确定该页是否在主存中。若在，则直接访问该页，根据页内偏移量取出（或存入）数据；若不在，则根据 LRU 原则，找出主存中最久没有访问过的那一页，将其与要访问的页替换，即将它存入磁盘，而将要访问的页存入物理存储器（主存）。

为简单起见，本程序使用五个数组来进行模拟：

1. Memory[16*1024]表示主存。
2. Disk[1024*1024*1024]表示磁盘。
3. PMPage[4]表示当前主存中的页。
4. recentUse[256*1024]表示每页被访问的时间，即第一次访问记为 1，第二次访问记为 2，以此类推。于是，主存中 recentUse 的值最小的那一页即为最久没有访问过的页。
5. pageTable[256*1024]表示页表。这是一个结构数组，包含两个元素：v 表示有效位，adr 表示该页在主存中的位置（若不在则置位-1）。

另外，由于有些数组太大，若数组元素为 int 类型会导致溢出，因此将其设为 char 类型。所以要写的两个函数实际上为 char LW(int adr)和 void SW(int adr, char dat)。

初始时，将序号为 0-3 的页存入主存，它们的有效位置为 1，其他页则是 0。主存（Memory）与磁盘（Disk）中的数据都预先设为'0'，以后每次 SW 操作都默认存入'A'。总次数记为 totalNum，缺页次数记为 missNum，则最后的 hit rate = (totalNum – missNum) / totalNum。

根据物理地址可以得到页号与偏移量：page = adr / pageSize; ofs = adr % pageSize。

LW 的实现细节是：若该页在主存中，则直接访问，返回 Memory[pageTable[page].adr * pageSize + ofs]；若不在，则首先根据 LRU 原则找到主存中最久没有访问的那一页（序号记为 p1），将其与要访问的页替换，即将这一页的所有数据复制到主存中，具体为：Memory[p1 * pageSize + i] =

Disk[PMPage[p1] * pageSize + i]，其中 i = 0, 1, …, pageSize。最后返回 Memory[p1 * pageSize + ofs]。

SW 的实现与 LW 类似，不同点在于是存入数据而非取出数据。

三、 页表方式分析

本次采用的是正向页表方式，即将页表建立在虚拟地址端。页表的项数为虚拟页，每个页项的宽度取决于物理页数，每个页项映射到物理地址。

正向页表的优点是能方便地由虚拟地址得到物理地址，实现较为简单。

反向页表建立在物理存储器端，标志每物理页所对应的虚拟页，项数由物理页决定。相当于正向页表的反面。

正向页表和反向页表由于页数较多，命中率相应提升，但效率有时不高。

单页映射的整个物理存储器为一页，页数太少，可能使命中率降低。

多级页表作为改进版，能够显著提升页表的访问效率与速度。

四、 结果验证

由于无法知道程序调用的内存地址究竟如何，还要满足时间局部性原理与空间局部性原理，模拟存储器的实际访问序列比较困难，因此只对小范围的地址进行了测试（即邻近地址周期性出现）。测试结果较好，命中率基本保持在 90%以上。但个人认为这种测试的实际意义并不大。

```
totalNum = 100
missNum = 7
hit rate = 93.0%

--------------------------------
Process exited with return value 0
Press any key to continue . . .
```

```
totalNum = 100
missNum = 9
hit rate = 91.0%

--------------------------------
Process exited with return value 0
Press any key to continue . . .
```

```
totalNum = 100
missNum = 11
hit rate = 89.0%

--------------------------------
Process exited with return value 0
Press any key to continue . . .
```

附录：源代码

```
#include<iostream>
#include<fstream>
#include<string>
#include<vector>

#define INFINITY 1e7

typedef short zijie;

using namespace std;

int PMSize = 16 * 1024;     //16KB
int VMSize = 1024 * 1024 * 1024;    //1GB
int pageSize = 4 * 1024;
int PMPageNum = 4;  //PMSize / pageSize
int VMPageNum = 256 * 1024; //VMSize / pageSize
char Memory[16 * 1024];
char Disk[1024 * 1024 * 1024];
int PMPage[4];  //which page it represents
int recentUse[256 * 1024];  //which time it recently used
int missNum = 0;
int totalNum = 0;
struct
{
    int v;  //value position
    int adr;    //which index of page in memory
}pageTable[256 * 1024];


int LRU()
{
    int min = INFINITY;
    int result;
    for (int i = 0; i < PMPageNum; i++)
    {
        if (recentUse[PMPage[i]] < min)
            result = i;
    }
    return result;
}

char LW(int adr)
```

```
{
    totalNum++;

    int page, ofs;
    page = adr / pageSize;
    ofs = adr % pageSize;

    recentUse[page] = totalNum;

    if (pageTable[page].v == 1)
    {
        return Memory[pageTable[page].adr * pageSize + ofs];
    }
    else
    {
        missNum++;

        int p1 = LRU();
        pageTable[PMPage[p1]].v = 0;
        pageTable[PMPage[p1]].adr = -1;
        pageTable[page].v = 1;
        pageTable[page].adr = p1;

        for (int i = 0; i < pageSize; i++)
        {
            Memory[p1 * pageSize + i] = Disk[PMPage[p1] * pageSize + i];
        }

        return Memory[p1 * pageSize + ofs];
    }

}

void SW(int adr, char dat)
{
    totalNum++;

    int page, ofs;
    page = adr / pageSize;
    ofs = adr % pageSize;

    recentUse[page] = totalNum;

    if (pageTable[page].v == 1)
```

```
87.     {
88.         Memory[pageTable[page].adr * pageSize + ofs] = dat;
89.         Disk[page * pageSize + ofs] = dat;
90.     }
91.     else
92.     {
93.         missNum++;
94.
95.         int p1 = LRU();
96.         pageTable[PMPage[p1]].v = 0;
97.         pageTable[PMPage[p1]].adr = -1;
98.         pageTable[page].v = 1;
99.         pageTable[page].adr = p1;
100.
101.          for (int i = 0; i < pageSize; i++)
102.          {
103.              Memory[p1 * pageSize + i] = Disk[PMPage[p1] * pageSize + i];
104.          }
105.
106.          Memory[p1 * pageSize + ofs] = dat;
107.          Disk[page * pageSize + ofs] = dat;
108.     }
109. }
110.
111. int main()
112. {
113.     //initialize
114.     for (int i = 0; i < VMPageNum; i++)
115.     {
116.         if (i < 4)   //at first, pages 0-3 are in memory
117.         {
118.             pageTable[i].v = 1;
119.             pageTable[i].adr = i;
120.         }
121.         else
122.         {
123.             pageTable[i].v = 0;
124.             pageTable[i].adr = -1;
125.         }
126.         recentUse[i] = 0;
127.     }
128.     for (int i = 0; i < PMPageNum; i++)
129.     {
130.         PMPage[i] = i;
```

```
131.     }
132.     for (int i = 0; i < PMSize; i++)
133.     {
134.         Memory[i] = '0';
135.     }
136.     for (int i = 0; i < VMSize; i++)
137.     {
138.         Disk[i] = '0';
139.     }
140.
141.     /*ifstream in("test.txt");
142.     int adr;
143.     char dat;
144.
145.     for (int i = 0; i < 13000; i++)
146.     {
147.         in >> adr;
148.         LW(adr);
149.     }
150.
151.     for (int i = 0; i < 50; i++)
152.     {
153.         in >> adr >> dat;
154.         SW(adr, dat);
155.     }
156.
157.     in.close();*/
158.
159.     double hitRate = (double)((totalNum - missNum) / totalNum);
160.
161.     cout << "totalNum = " << totalNum << endl;
162.     cout << "missNum = " << missNum << endl;
163.     cout << "hit rate = " << hitRate << endl;
164.
165.     system("pause");
166. }
```