虚拟存储

应承峻 3170103456

1 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);
void SW(int adr, int dat);
其中：adr范围：0~1Gzjie。
```

2 算法分析

在本题中，我们采用单页映射的方式进行模拟，同时为了节省实际系统的内存空间，我们假定`adr`的范围在`8MB`。如果需要调整`adr`的范围为`1GB`，只需调整`PAGENUM=0x1000000`(2^{24})即可

变量定义如下：

```
const int PAGESIZE = 16384;  //16K
const int PAGENUM = 128;   //128page*64KB/page=8M
const int TESTNUM = 100;  //repeated 1000 times
const int MIN = 0;  //minimum address or data
const int MAX = PAGESIZE * PAGENUM; //maximum address or data
int curPage; //current page number
int cnt = 0; //hit number
```

初始化磁盘：

```
void init(int Memory[]) {
    ofstream out("Memory" , ios::out | ios::binary);
    out.seekp(0, ios::beg);
    for (int i = 0; i < PAGENUM; i++) {  //write zero to the file
        out.write((char*)Memory , 4 * PAGESIZE);
    }
}
```

将页读/写入主存：

```
void load(int page , int Memory[]) {
    ifstream in("Memory" , ios::in | ios::binary);
    in.seekg(4 * PAGESIZE * page , ios::beg);
    in.read((char*)Memory , 4 * PAGESIZE);
}

void save(int page , int Memory[]) {
    ofstream out("Memory" , ios::out | ios::binary);
    out.seekp(4 * PAGESIZE * page , ios::beg);
    out.write((char*)Memory , 4 * PAGESIZE);
}
```

数据读写操作：

```
int* getMemory(int adr , int Memory[]) {
    int pag = adr / PAGESIZE;  //pagenum
    int ofs = adr % PAGESIZE;  //offset
    if (curPage == pag) {  //hit the page
        cnt++;  //hit counter
        return &Memory[ofs];
    } else {  //miss the page
        save(curPage , Memory);  //store current page to disk
        load(pag , Memory);  //replace current page by target page
        return &Memory[ofs];
    }
}

int lw(int adr , int Memory[]) {
    int *p = getMemory(adr , Memory);
    return *p;
}

void sw(int adr , int val, int Memory[]) {
    int *p = getMemory(adr , Memory);
    *p = val;
}
```

3 程序测试

驱动程序：

```
init(Memory);
srand((unsigned)time(NULL)); //random number
for (int j = 0; j < 10; j++) { //repeated 10 times
    cnt = 0;
    for (int i = 0; i < TESTNUM; i++) {  //each iteration do 100 I/O operation
        adr = (rand() % (MAX - MIN + 1)) + MIN; //generate address from MIN to MAX
        dat = (rand() % (MAX - MIN + 1)) + MIN; //generate data from MIN to MAX
        if (adr & 1) dat = -dat;  //ramdom symbol
        sw(adr , dat , Memory);
    }
    cout << "accurate rate = " << 100.0 * cnt / (2 * TESTNUM) << " % " << endl;
}
```

测试结果：

4 结果分析

可以发现，使用单页映射的方式进行模拟命中率较低，基本保持在$20\% - 30\%$之间。

比较分析：

类型	优点	缺点
单页映射	易于实现	命中率较低；页表消耗大量物理内存
反向页表	减少页表占用主存空间；提高了命中率	存在链表存储空间的开销。为反向页表分配内存空间时，可能存在滥出或越界的问题。实现共享内存时存在困难
正向页表	节省了空间；提高了命中率	难于实现。
多级页表	使用多级页表可以使得页表在内存中离散存储。 使用多级页表可以节省页表内存。	使用一级页表时，读取内存中一页内容需要2次访问内存，第一次是访问页表项，第二次是访问要读取的一页数据。但如果是使用二级页表的话，就需要3次访问内存，第一次访问页目录项，第二次访问页表项，第三次访问要读取的一页数据。是以时间换空间的做法。