《操作系统原理》实验报告

姓名	张旭	学号	U201817106	专业班级	1805	时间	2020.04.28

一、实验目的

（1）理解页面淘汰算法原理，编写程序演示页面淘汰算法。

（2）验证 Linux 虚拟地址转化为物理地址的机制

（3）理解和验证程序运行局部性的原理。

二、实验内容

（1）在 Windows 环境下编写一个程序，模拟实现 OPT,FIFO,LRU 等页面淘汰算法。

可以使用数组模拟内存，数组中的元素模拟为指令或数据。写不同方式的程序去访问数组来模拟 CPU 访问内存的情况。分析运算结果，在分配不同的物理块情况下，各算法的缺页情况有什么规律？可以 srand( )和 rand( )等函数定义和产生“指令”序列，然后将指令序列变换成相应的页地址流，并针对不同的算法计算出相应的命中率。例如，实验中可以产生 320 条“指令”，每个“虚拟页”存放 10 条指令。进程分配的页框是 4（可变，例如 32）。

（2）在 Linux 环境下，编写一个小程序，获取该程序中的某个变量的虚拟地址，虚拟页号，页内偏移地址，物理页框号，页内偏移地址，物理地址，并将它们打印出来。建议使用/proc/pid/pagemap 技术。

（3）在 Windows 环境下，编写一个函数（特点：比较耗时，比如大型的多维数组读写），用不同的方法测试其所花费的时间。在不同环境下比较其时间是否不同，并分析其含义。测量时间的函数请 baidu。

三、实验过程

（一）实验步骤

（1）页面淘汰算法

1. 定义一些有关的常量

```
const int MAXN = 320;   // 总共320条指令
const int VP_SIZE = 10; // 每个页面的大小为10
const int PF_NUMS = 8;  // 页框（内存）为4个页面
const int VP_NUMS = 32; // 32个页面
const int MAX_INF = 0x3f3f3f3f;
```

2. 实现相应的存储结构

使用 struct 创建指令和页框的结构

指令集

```
struct Command
{
    int id;
    int pageID;
    int offSet;
    void setData()
    {
        this->pageID = id / VP_SIZE;
        this->offSet = id % VP_SIZE;
    }
}command[MAXN];
```

页框

```
struct PageFrame
{
    int pageID;
    int lastUse;
    PageFrame(int x = -1, int lastUse = -1)
    {
        this->pageID = x;
        this->lastUse = lastUse;
    }
};
```

3. 随机生成指令

```
void creatCommandList()
{
    for (int i = 0; i < MAXN; i++)
    {
        command[i].id = rand() % MAXN;
        command[i].setData();
    }
}
```

随机生成指令，并且计算出页号和偏移

4. 实现 FIFO 算法

FIFO 算法比较简单，直接使用队列即可。由于 c++中 queue 不能够直接使用迭代器访问。所以使用 vector 模拟实现队列。

遍历指令集，如果需要的页面不在页框中，则出队元素，入队页号。

如果在则自增命中次数

```
double FIFO()
{
    vector<PageFrame> q;
    int front = 0;
    int existNum = 0;
    for (int i = 0; i < MAXN; i++)
    {
        bool flag = false;
        for (int j = front; j < q.size(); j++)
        {
            if (q[j].pageID == command[i].pageID)
            {
                flag = true;
                break;
            }
        }
        if (flag) existNum++;
        else
        {
            if (q.size() - front == PF_NUMS) front++;
            else q.push_back(PageFrame(command[i].pageID));
        }
    }
    return existNum * 1.0 / MAXN;
}
```

5. 实现 LRU 算法

遍历指令集，如果需要的页面不在页框中，则加入页框，如果页框满了则遍历页框淘汰 lastUse 最小的元素。

```
double LRU()
{
    PageFrame pf[PF_NUMS];
    int size = 0, existNum = 0;
    for (int i = 0; i < MAXN; i++)
    {
        // 是否存在，存在更新时间，不存在则插入
        bool flag = false;
        for (int j = 0; j < PF_NUMS; j++)
        {
            if (command[i].pageID == pf[j].pageID)
            {
                flag = true;
                pf[j].lastUse = i;
                break;
            }
        }
        if (flag)
        {
            existNum++;
            continue;
        }
        else
        {
            // 不存在，插入
            if (size < PF_NUMS) // 没有满
            {
                for (int j = 0; j < PF_NUMS; j++)
                {
                    if (pf[j].pageID == -1)
                    {
                        pf[j] = PageFrame(command[i].pageID, i);
                        size++;
                        break;
                    }
                }
            }
            else    // 满了，淘汰
            {
                int index = 0;
                for (int j = 1; j < PF_NUMS; j++)
                {
                    if (pf[j].lastUse < pf[index].lastUse)
                    {
                        index = j;
                    }
                }
                pf[index] = PageFrame(command[i].pageID, i);
            }
        }
    }
    return existNum * 1.0 / MAXN;
}
```

如果满了，则遍历一下页框寻找 lastUse 最小的元素，淘汰他

6. 实现 OPT 算法

```
double OPT()
{
    int existNum = 0, size = 0;
    PageFrame pf[PF_NUMS];
    for (int i = 0; i < MAXN; i++)
    {
        // 是否存在，存在更新时间，不存在则插入
        bool flag = false;
        for (int j = 0; j < PF_NUMS; j++)
        {
            if (command[i].pageID == pf[j].pageID)
            {
                flag = true;
                break;
            }
        }
        if (flag)
        {
            existNum++;
            continue;
        }
        if (size < PF_NUMS)
        {
            for (int j = 0; j < PF_NUMS; j++)
            {
                if (pf[j].pageID == -1)
                {
                    pf[j] = PageFrame(command[i].pageID);
                    size++;
                    break;
                }
            }
        }
        else
        {
            int index = 0;
            bool flag = false;
            for (int j = MAXN - 1; j >= i; j--)
            {
                for (int k = 0; k < PF_NUMS; k++)
                {
                    if (command[j].pageID == pf[k].pageID)
                    {
                        index = k;
                        flag = true;
                        break;
                    }
                }
                if (flag) break;
            }
            pf[index] = PageFrame(command[i].pageID);
        }
    }
    return existNum * 1.0 / MAXN;
}
```

淘汰页面时，从指令集的最后往前遍历，每次访问一个指令看他的页面是否在页框中，如果在这淘汰该页面。

7. 实现主函数调用

```
int main()
{
    srand((unsigned int)time(NULL));
    creatCommandList();
    cout << "FIFO:" << FIFO() << endl;
    cout << "LRU:" << LRU() << endl;
    cout << "OPT:" << OPT() << endl;
    return 0;
}
```

（2）求物理地址

1. 定义一个变量，取地址，得到虚拟地址

调用函数得到页面大小，求出虚拟页号和偏移地址

```
int a = 100;
int pageSize = getpagesize();
unsigned long vAddr = (unsigned long)&a;
unsigned long vPageID = vAddr / pageSize; //  2
unsigned long PageOffset = vAddr % pageSize; //  3
unsigned long vOffSet = vPageID * sizeof(uint64_t);
```

得到偏移地址后可以求出偏移量，方便去找虚拟页和物理页的对应

```
unsigned long vOffSet = vPageID * sizeof(uint64_t);
```

2. 打开/proc/self/pagemap 文件

```
Bits 0-54: page frame number(PFN) if present
Bits 0-4:  swap type if swapped
Bits 5-54: swap offset if swapped
Bits 55-60:page shift
Bit 61:    reserved ofr future use
Bit 62:    page swapped
Bit 63:    page present
```

各个位的对应信息

```
int file = open("/proc/self/pagemap", O_RDONLY);
if(file == -1)
{
    cout << "打开失败" << endl;
    return -1;
}
```

跳过不需要的页面，

```
if(lseek(file, vOffSet, SEEK_SET) == -1)
{
    cout << "lseek error" << endl;
    return -1;
}
```

找到了虚拟页和物理页对应的地方

读取出对应的物理页

```
if(read(file, &item, sizeof(uint64_t)) != sizeof(uint64_t))
{
    cout << "read error" << endl;
    return -1;
}
```

查找对应的物理页面

```
uint64_t pPageID = ((uint64_t)1 << 55) & item;  // 4
unsigned long pAddr = pPageID * pageSize + PageOffset;
/*
获取该程序中的某个变量的虚拟地址
```

得到物理页，求出物理地址

（3）测试运行时间

由于数组的大小具有限制，所以不能开很大的数组。在 VS2019 中最大的数组为 0x7fffffff.所以直接在堆上开辟一个 1e8 的数组。然后初始化，记录时间。

```
using namespace std;
const int maxn = 1e8;
```

```
a = new int[maxn];
for (int i = 0; i < maxn; i++)
{
    *(a + i) = 1e9;
}
```

1. 使用 clock 函数测试

2. 使用高精度时控函数 QueryPerformanceFrequency（），QueryPerformanceCounter（）

```
int main()
{
    double time = 0;
    double counts = 0;
    LARGE_INTEGER nFreq;
    LARGE_INTEGER nBeginTime;
    LARGE_INTEGER nEndTime;
    QueryPerformanceFrequency(&nFreq);
    QueryPerformanceCounter(&nBeginTime);//开始计时

    a = new int[maxn];
    for (int i = 0; i < maxn; i++)
    {
        *(a + i) = 1e9;
    }

    QueryPerformanceCounter(&nEndTime);//停止计时
    time = (double)(nEndTime.QuadPart - nBeginTime.QuadPart) / (double)nFreq.QuadPart;
    cout << time * 1000 << "ms" << endl;
    return 0;
```

（二）解决错误和优化

（1）页面淘汰算法

1.【特殊的语法错误】

忘记 queue 不能够迭代。所以使用 vector 实现队列

2.【运行错误】

现象：增加页框数，发现 OPT 算法的命中率远低于其他两种。

解决：检查代码，发现有一处没有使用变量替换，使用的是个常量。修改即可

（2）求物理地址

没有遇到问题

（3）测试运行时间

1.【特殊的语法错误】

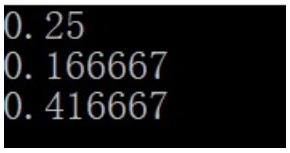
现象：报错，不能开过大的数组

解决：修改数组大小解决

四、实验结果

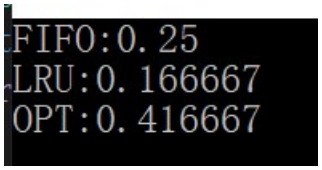
（1）页面淘汰算法

先使用课件中的例子



对比发现结果正确

再次随机生成（页框 4）

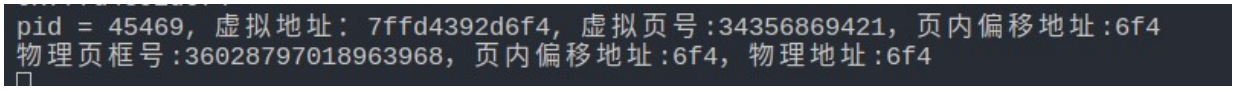


改变页框大小

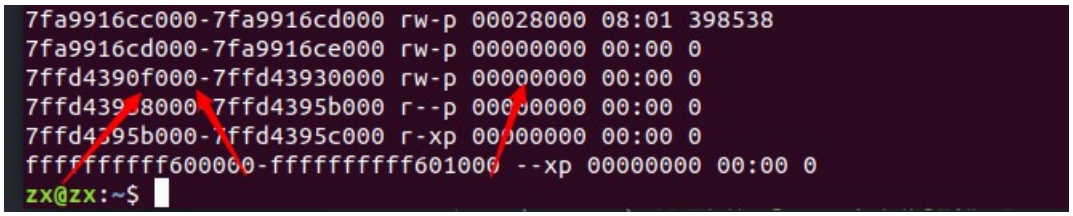
页框数增加命中率显著提高。

缺页数降低

（2）求物理地址

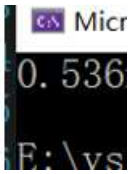


打开相应的文件查看虚拟页与物理页的对应关系，可以看到结果正确



（3）测试运行时间

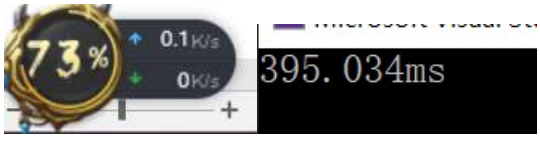
内存占用 70%左右时

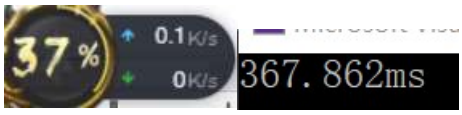


536 毫秒

内存占用 39%

373 毫秒





可以看出内存占用比较高时，程序运行时间较长。内存占用较高时，页框数比较少，命中率较低，缺页率较高，抖动剧烈，因此时间较长。

五、体会

通过本次实验，复习了页面淘汰算法，通过自己亲自实现，对页面淘汰算法的理解更加深刻。通过求解物理地址，也更加深刻的理解了虚拟地址和物理地址的关系。通过在不同的环境中测试运行时间，对缺页情况有了更多的了解。