



北京理工大学
Beijing Institute of Technology

本科实验报告

实验名称： 内存管理

课程名称：	操作系统原理	实验时间：	2018/4/11
任课教师：	王耀威	实验地点：	理学楼信抗实验中心
实验教师：	苏京霞	实验类型：	<input checked="" type="checkbox"/> 原理验证 <input type="checkbox"/> 综合设计 <input type="checkbox"/> 自主创新
学生姓名：	施念		
学号/班级：	1120161302/05011609	组 号：	53
学 院：	信息与电子学院	同组搭档：	
专 业：	电子信息工程	成 绩：	



信息与电子学院

SCHOOL OF INFORMATION AND ELECTRONICS

实验二：内存管理

一、实验目的

1. 通过编写和调试存储管理的模拟程序以加深对存储管理方案的理解；
2. 熟悉虚存管理的页面淘汰算法；
3. 通过编写和调试地址转换过程的模拟程序以加强对地址转换过程的了解。

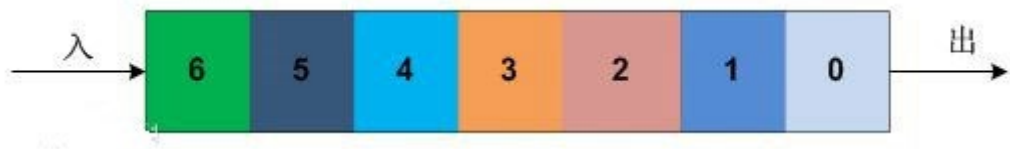
二、实验题目

1. 设计一个请求页式存储管理方案（自己指定页面大小），并予以程序实现。并产生一个需要访问的指令地址流。它是一系列需要访问的指令的地址。为不失一般性，你可以适当地（用人工指定地方法或用随机数产生器）生成这个序列。
2. 页面淘汰算法采用 FIFO 页面淘汰算法，并且在淘汰一页时，只将该页在页表中抹去。而不再判断它是否被改写过，也不将它写回到辅存。
3. 系统运行既可以在 Windows，也可以在 Linux。

三、实验基础知识

1. FIFO 算法

在计算机中，先入先出队列是一种传统的按序执行方法，先进入的指令先完成并引退，跟着才执行第二条指令（指令就是计算机在响应用户操作的程序代码，对用户而言是透明的）。如图 1 所示，当 CPU 在某一时段来不及响应所有的指令时，指令就会被安排在 FIFO 队列中，比如 0 号指令先进入队列，接着是 1 号指令、2 号指令……当 CPU 完成当前指令以后就会从队列中取出 0 号指令先行执行，此时 1 号指令就会接替 0 号指令的位置，同样，2 号指令、3 号指令……都会向前挪一个位置



2. 局部性原理

三种不同类型的局部性:

时间局部性 (Temporal Locality): 如果一个信息项正在被访问，那么在近期它很可能还会被再次访问。程序循环、堆栈等是产生时间局部性的原因。

空间局部性 (Spatial Locality): 在最近的将来将用到的信息很可能与现在正在使用的信息在空间地址上是临近的。

顺序局部性 (Order Locality)：在典型程序中，除转移类指令外，大部分指令是顺序进行的。顺序执行和非顺序执行的比例大致是 5:1。此外，对大型数组访问也是顺序的。指令的顺序执行、数组的连续存放等是产生顺序局部性的原因。

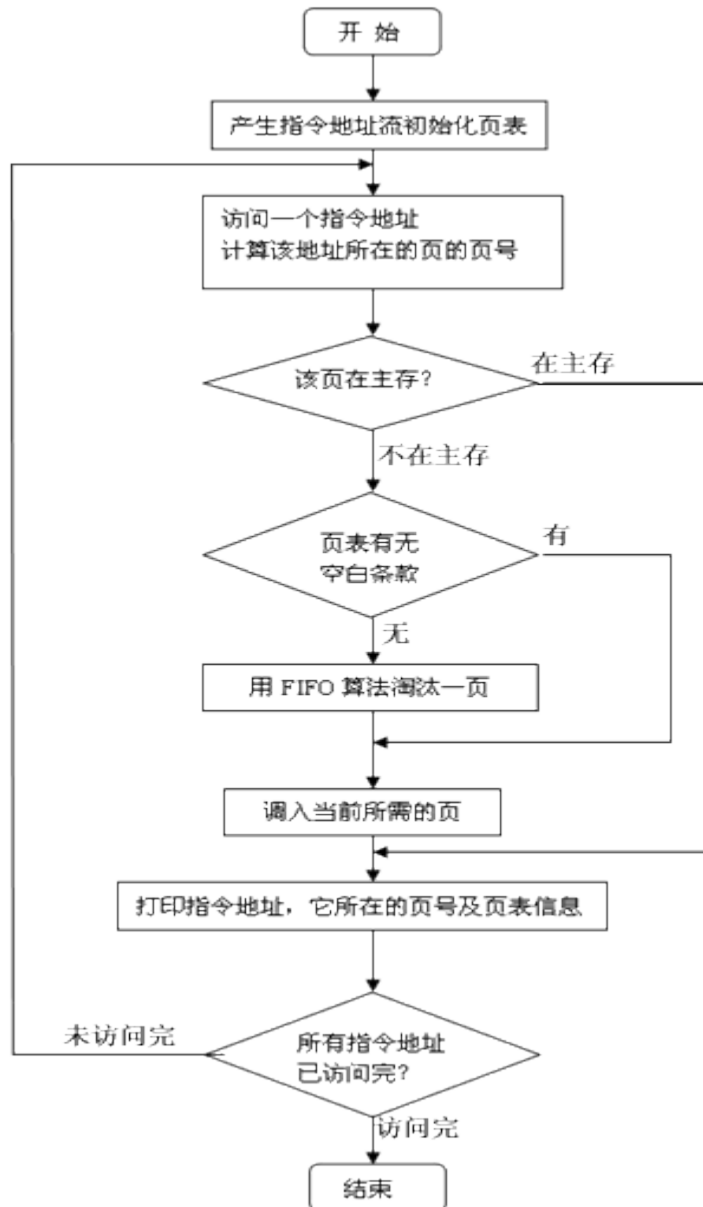
3. 缺页中断

缺页中断就是要访问的页不在主存，需要操作系统将其调入主存后再进行访问。在这个时候，被内存映射的文件实际上成了一个分页交换文件。

中断是指计算机在执行程序的过程中，当出现异常情况或特殊请求时，计算机停止现行程序的运行，转向对这些异常情况或特殊请求的处理，处理结束后再返回现行程序的间断处，继续执行原程序。

四、实验设计方法

1. 流程图



2. 具体方法

- 1) void ExecFile(const char *filename) & void EndFile()
执行打开文件(filename)和关闭文件的操作, Endfile()中, 因为 fp 是全局变量, 故可以直接访问并关闭。
- 2) void AllocMemory(UInt8 block)
先进行判断: if (Fifo[Fifo_ptr] != -1), 如果满, 则先废弃页面并记录, 如果未滿,

则直接对 freeMem 进行赋值操作，即直接分配。之后进行页框和页号的记录并读取文件中的字符。

3) void VisitMemory(UInt8 addr)

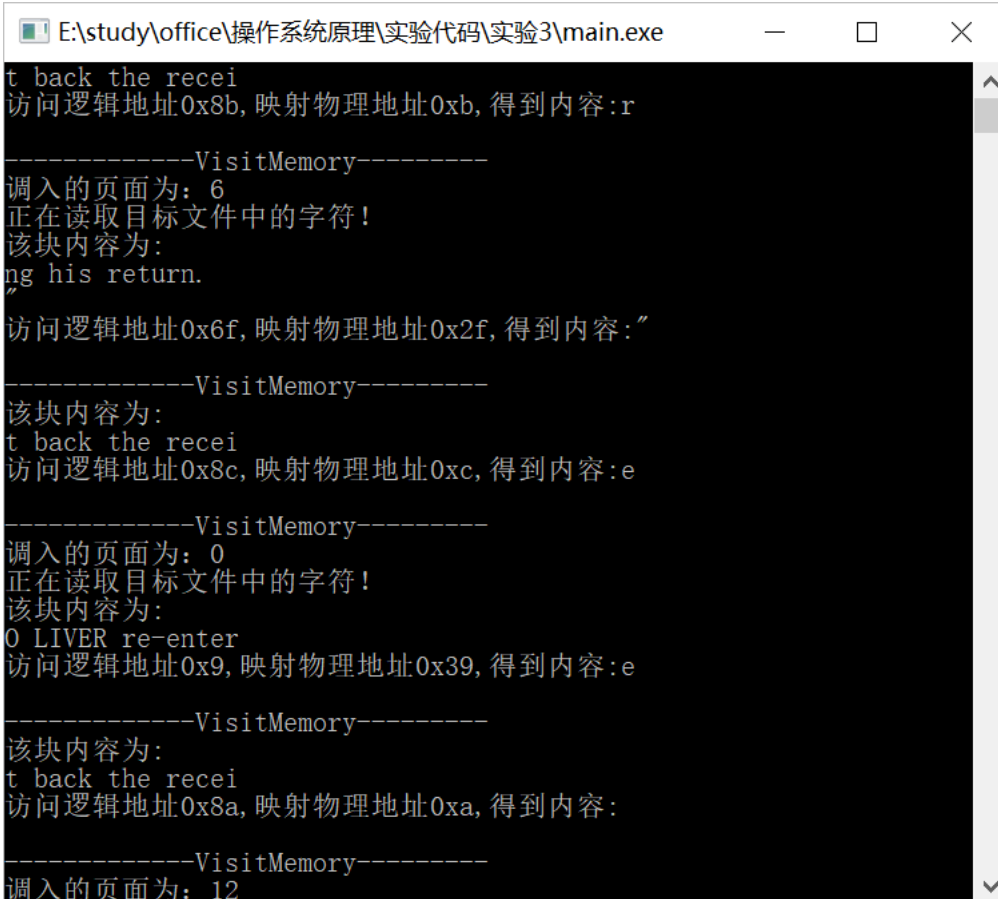
函数参数为逻辑地址 addr，通过逻辑地址 addr，映射物理地址 menAddr，读取得到内容 content。

4) int main()

读取文件“Acess.txt”并进行初始化，开始循环访问内存。每按一个回车，进行一次循环。

五、实验结果及数据分析

1. 图像：



```
E:\study\office\操作系统原理\实验代码\实验3\main.exe
t back the recei
访问逻辑地址0x8b, 映射物理地址0xb, 得到内容:r

-----VisitMemory-----
调入的页面为: 6
正在读取目标文件中的字符!
该块内容为:
ng his return.
访问逻辑地址0x6f, 映射物理地址0x2f, 得到内容:"

-----VisitMemory-----
该块内容为:
t back the recei
访问逻辑地址0x8c, 映射物理地址0xc, 得到内容:e

-----VisitMemory-----
调入的页面为: 0
正在读取目标文件中的字符!
该块内容为:
0 LIVER re-enter
访问逻辑地址0x9, 映射物理地址0x39, 得到内容:e

-----VisitMemory-----
该块内容为:
t back the recei
访问逻辑地址0x8a, 映射物理地址0xa, 得到内容:

-----VisitMemory-----
调入的页面为: 12
```

(图一 初始)

```
E:\study\office\操作系统原理\实验代码\实验3\main.exe
-----VisitMemory-----
该块内容为:
sir.
"Why not?"
访问逻辑地址0xdb, 映射物理地址0x6b, 得到内容:n

-----VisitMemory-----
该块内容为:
sir.
"Why not?"
访问逻辑地址0xdb, 映射物理地址0x6b, 得到内容:n

-----VisitMemory-----
该块内容为:
sir.
"Why not?"
访问逻辑地址0xd3, 映射物理地址0x63, 得到内容:.

-----VisitMemory-----
调入的页面为: 11
注意: 废弃了第1个页框中的页面!
正在读取目标文件中的字符!
该块内容为:
he had a chance
访问逻辑地址0xb5, 映射物理地址0x15, 得到内容:d

-----VisitMemory-----
该块内容为:
memployer, before
访问逻辑地址0xa1, 映射物理地址0x1, 得到内容:p
```

(图二 开始出现出栈)

```
选择E:\study\office\操作系统原理\实验代码\实验3\main.exe
-----VisitMemory-----
调入的页面为: 10
注意: 废弃了第7个页框中的页面!
正在读取目标文件中的字符!
该块内容为:
mployer, before
访问逻辑地址0xaf, 映射物理地址0x7f, 得到内容:

-----VisitMemory-----
调入的页面为: 9
注意: 废弃了第0个页框中的页面!
正在读取目标文件中的字符!
该块内容为:
pt?" asked his e
访问逻辑地址0x97, 映射物理地址0x7, 得到内容:k

-----VisitMemory-----
该块内容为:
mployer, before
访问逻辑地址0xa1, 映射物理地址0x71, 得到内容:p

-----End-----

Process exited after 979.8 seconds with return value 0
请按任意键继续. . .
```

(图三循环结束)

2. 分析

由运行结果可知:

- 1) 主存中有 128 个字节, 按 16 字节进行划分, 形成 8 个页框。
- 2) 第一次循环(图一)并未出现页框已满的操作, 所以无需进行出栈, 即废弃页面的操作, 但是在第接下来的循环中(图二), 如图中红框所示, 开始出现出栈操作, 此时按照程序设定的 FIFO 算法进行写入读取。
- 3) 循环结束(图三), 只是按照设定循环了 100 次。

六、总结

此次实验我加深了对 FIFO 的理解, 同时实践操作也使我对局部性原理等一些实际的函数操作有了更加深刻的了解。

从虚拟内存到物理地址再到实际内容的读取, 我对虚拟内存的印象逐渐加深, 理解了虚拟内存的重要性。

其次, 关于 Access.txt 中如果出现中文字符会有什么结果也进行了判断, 对代码的调试能力也逐渐提高。

七、附录

代码清单：

```
#include <iostream>
#include <time.h>
#include <assert.h>
using namespace std;
#ifdef _UNICODE
#undef _UNICODE
#endif
#define MEM_TOTAL    0x80
#define BLOCK_UNIT    0x10
#define PAGE_TOTAL    0x10
#define GetBlock(x)(x>>4)
#define GetMem(x,y)    ((x<<4)|(y&0x0F))
#define MEM_BLOCK    GetBlock(MEM_TOTAL)
typedef unsigned char Uint8;
typedef unsigned int    Uint32;
char Memory[MEM_TOTAL];
FILE *fp;
typedef struct Page
{
    Page():Exist(0){}
    Uint8 MemPage;
    Uint8 Exist;
}Page;
Page TotalPage[PAGE_TOTAL];
Uint8    Fifo_ptr=0;
char Fifo[MEM_BLOCK];
void InitFifo()
{
    for(int i=0;i<MEM_BLOCK;++i)
        Fifo[i]=-1; // 用 -1 进行初始化
}
void ExecFile(const char *filename)
{
    fp=fopen(filename,"r");
    if(!fp)
        exit(0);
    InitFifo();    //初始化
    srand((unsigned int)time(NULL));
}
void EndFile()
```



```

{
    fclose(fp);
}
void AllocMemory(UInt8 block)
{
    UInt8 freeMem;
    if(Fifo[Fifo_ptr]!=-1) // 判断
    {
        printf("注意： 废弃了第%d 个页框中的页面！ \n",Fifo_ptr); //废弃一个页面
        UInt8 freeblock=Fifo[Fifo_ptr];
        TotalPage[freeblock].Exist=0;
        freeMem=TotalPage[freeblock].MemPage;
    }
    else
        freeMem=Fifo_ptr;
    Fifo[Fifo_ptr]=block;
    TotalPage[block].MemPage=freeMem;
    TotalPage[block].Exist=1;

    printf("正在读取目标文件中的字符！ \n"); //读取文件中的字符
    assert(fp);
    fseek(fp,block<<4,SEEK_SET);
    fread(&Memory[freeMem<<4],sizeof(char),BLOCK_UNIT,fp);
    Fifo_ptr=(++Fifo_ptr)%MEM_BLOCK;
}
void VisitMemory(UInt8 addr)
{
    UInt8 memAddr;
    char content;
    UInt8 block=GetBlock(addr);
    if(!TotalPage[block].Exist)
    {
        printf("调入的页面为： %d\n",block); //调入页面
        AllocMemory(block);
        TotalPage[block].Exist=1;
    }
    memAddr=GetMem(TotalPage[block].MemPage,addr); // 物理地址
    content=Memory[memAddr]; // 内容
    //printf("-----\n");
    printf("该块内容为:\n");
    for(int i=0;i<0x10;++i)
    {
        printf("%c",Memory[GetMem(TotalPage[block].MemPage,i)]);
    }
}

```

```
    printf("\n");
    printf("访问逻辑地址 0x%x,映射物理地址 0x%x,得到内容:%c\n\n",addr,memAddr,content);

}
int main()
{
    ExecFile("Access.txt");
    for(int j=0;j<100;++j)
    {
        for(int i=0;i<10;++i)
        {    printf("-----VisitMemory-----\n");
            VisitMemory(rand()%0x100);
        }
        printf("-----End-----\n");
        getchar();
    }
    EndFile();
    return 0;
}
```