201311211914_赵帅帅_实验 7

1、 实验目的

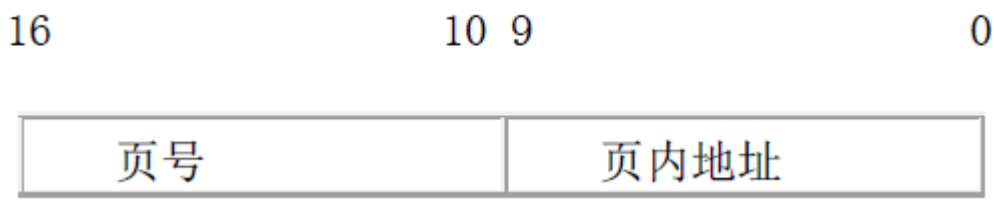
通过编写和调试请求页式存储管理的模拟程序以加深对请求页式存储管理方案的理解。

2、 实验任务

为了简单起见。页面淘汰算法采用 FIFO 页面淘汰算法，并且在淘汰一页时，判断它是否被改写过，如果被修改过，将它写回到辅存。

数据结构：

（1） 虚地址结构：



（2） 页表结构：

```
define N 64
struct {
        int lNumber; //页号
        int pNumber; //物理块号
        int dNumber; //在磁盘上的位置
        int write; //修改标志
        int flag; //存在标志
}page[N];
```

（3） 存放页号的队列 p[m]，m 为分配给进程的内存块数。当装入一个新的页面时，将其页号存入数组。在模拟试验中，我们采用页面预置的方法。

（4） 请求分页存储管理算法流程图如下：

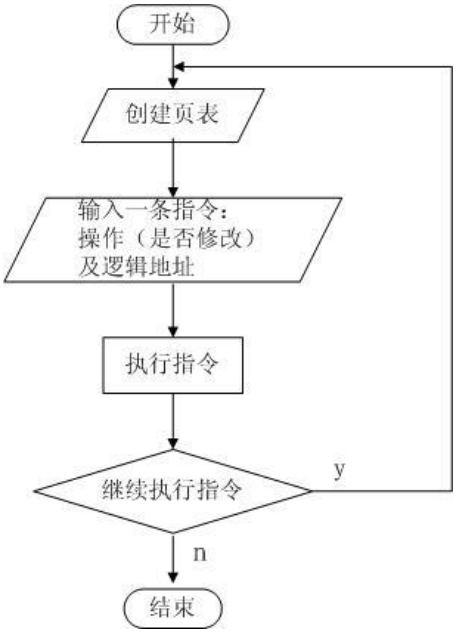


图 1：请求页式存储管理模拟程序的完整流程图

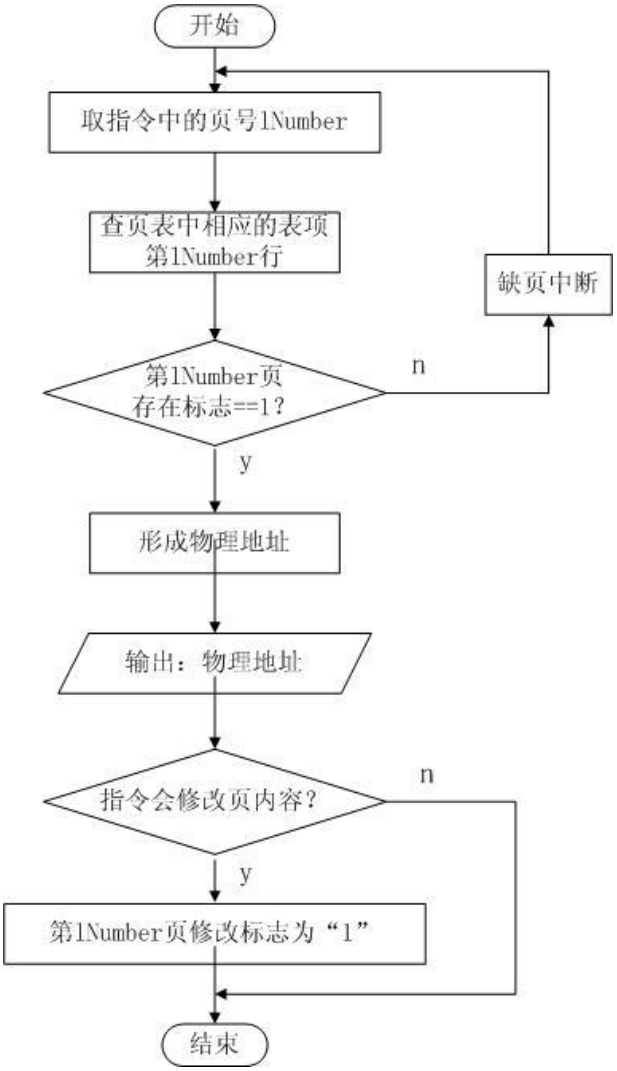


图 2：一条指令执行的模拟流程图

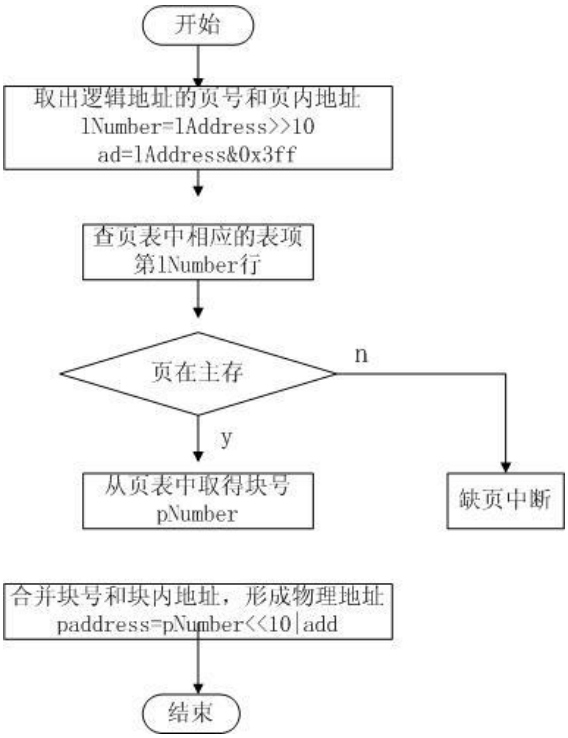


图 3：模拟地址转换的流程图

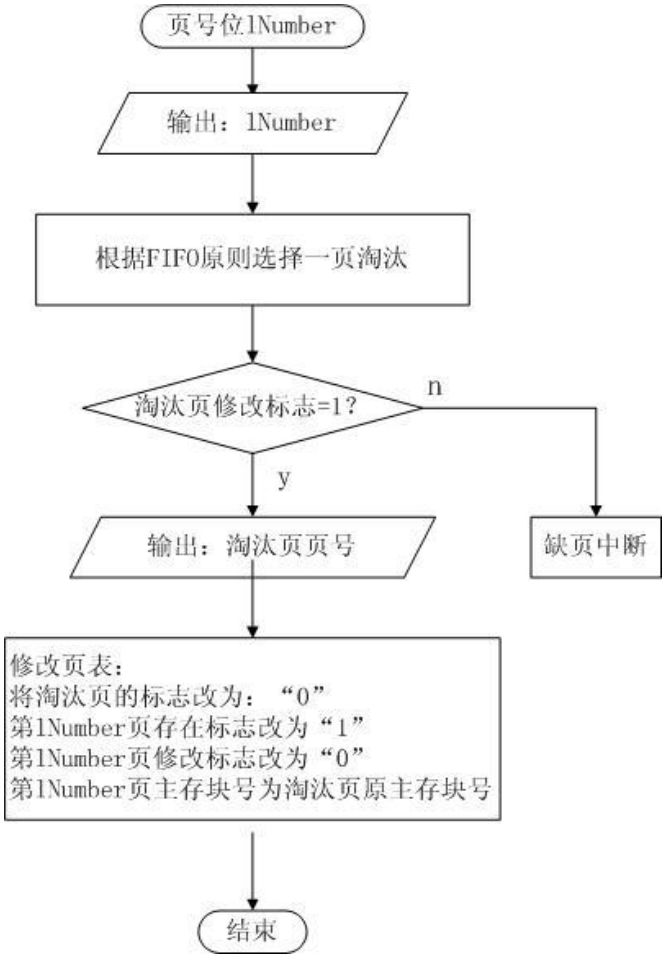


图 4：采用 FIFO 页面置换算法的缺页中断流程图

3、 代码

```
#include<stdio.h>
#include<stdlib.h>

const int N =64;
int page_count = 0;      //进程总页数
int block_count = 0;//进程在内存中的总块数
int p[N];                //块队列
struct Page
{
    int lNumber;//页号
    int pNumber;//物理块号
    int dNumber;//在磁盘上的位置
    int write;       //修改标志
    int flag;        //存在标志
}page[N];
void print()
{
 printf("page\tlNumber\tpNumber\twrite\tflag\n");
 for(int i = 0; i < page_count; i ++)
 {
     printf("%d\t", page[i].lNumber);
     printf("%d\t", page[i].pNumber);
     printf("%d\t",page[i].dNumber);
     printf("%d\t", page[i].write);
     printf("%d\n", page[i].flag);
 }
 printf("\n");
}
void exch_page(int virtual_page_number)
{
 int out_page = p[0];
 int i = 0;
 while(i < block_count -1)
 {
     p[i] = p[i + 1];
     i ++;
 }
 p[block_count - 1] = virtual_page_number;
 page[virtual_page_number].pNumber = page[out_page].pNumber;
 page[out_page].flag = 0;
 page[virtual_page_number].flag = 1;
printf("淘汰主存%d 中的页%d,从磁盘%d 调入页%d.\n",page[out_page].pNumber,
```

```
out_page, page[virtual_page_number].dNumber,virtual_page_number);
    page[out_page].pNumber = -1;
    if(page[out_page].write == 1)
    {
        printf("页面%d 被改写过，写回磁盘成功！\n", out_page);
    }
    else
    {
        printf("页面%d 未被改写过，不用写回磁盘！\n", out_page);
    }
   }

   void find(int choice, int address)
   {
    int virtual_page_number = address>>10;
    int block_inner_add = address & 0x03FF;
    if(virtual_page_number >= page_count)
    {
        printf("该页不存在!!! \n\n");
        return ;
    }
    if(page[virtual_page_number].flag == 0)
    {
        printf("发生缺页中断!!! \n\n");
        exch_page(virtual_page_number);
        find(choice, address);
    }
    else
    {
        int phy_page_number = page[virtual_page_number].pNumber;
        int phy_address = (phy_page_number<<10)|block_inner_add;
        printf("逻辑地址是： %d 对应的物理地址为： %d\n\n", address,
phy_address);
        if(choice == 1)
        {
            page[virtual_page_number].write = 1;
        }
        print();
    }
   }

   int main(void)
   {
       for(int i = 0; i < N; i ++)
```

```
    {
        page[i].lNumber = 0;
        page[i].pNumber = -1;
        page[i].dNumber = 0;
        page[i].write = 0;
        page[i].flag = 0;
    }
    printf("输入页表信息，创建页表(以-1 结束)：\n\n");

    while(true)
    {
        printf("请输入第 %d 页辅存地址:", page_count);
        int disk_add = -1;
        scanf("%d", &disk_add);
        if(disk_add == -1)
        {
            break;
        }
        page[page_count].lNumber = page_count;
        page[page_count].dNumber = disk_add;
        page_count ++;
    }
print();

printf("请依次输入每一页的主块号（以-1 结束）：");

int i = 0;
for(i = 0; i < page_count; i ++)
{
    int pNumber = 0;
    scanf("%d", &pNumber);
    if(pNumber == -1)
    {
        break;
    }
    else
    {
        page[i].pNumber = pNumber;
        page[i].flag = 1;
        p[i] = i;
    }
}
block_count = i;
print();
```

```
printf("输入指令(1-修改，0-不修改，其他-程序结束)和逻辑地址：");
int choice = 0 ,address = 0;
while(scanf("%d%d", &choice, &address) && (choice == 0 || choice == 1))
{
    find(choice, address);
    printf("输入指令(1-修改，0-不修改，其他-程序结束)和逻辑地址：");
}
   return 0;
}
```

4、 运行结果

输入页表信息，创建页表(以-1 结束)：

请输入第 0 页辅存地址:0
请输入第 1 页辅存地址:1
请输入第 2 页辅存地址:2
请输入第 3 页辅存地址:3
请输入第 4 页辅存地址:4
请输入第 5 页辅存地址:-1

page	lNumber	pNumber	write	flag
0	-1	0	0	0
1	-1	1	0	0
2	-1	2	0	0
3	-1	3	0	0
4	-1	4	0	0

请依次输入每一页的主块号（以-1 结束)：10 11 -1

page	lNumber	pNumber	write	flag
0	10	0	0	1
1	11	1	0	1
2	-1	2	0	0
3	-1	3	0	0
4	-1	4	0	0

输入指令(1-修改，0-不修改，其他-程序结束)和逻辑地址：0 1024
逻辑地址是：1024 对应的物理地址为：11264

page	lNumber	pNumber	write	flag
0	10	0	0	1
1	11	1	0	1
2	-1	2	0	0

```
3       -1      3       0       0
4       -1      4       0       0

输入指令(1-修改，0-不修改，其他-程序结束)和逻辑地址：1 2048
发生缺页中断!!!

淘汰主存 10 中的页 0，从磁盘 2 调入页 2.
页面 0 未被改写过，不用写回磁盘！
逻辑地址是：2048 对应的物理地址为：10240

page    lNumber pNumber write   flag
0       -1      0       0       0
1       11      1       0       1
2       10      2       1       1
3       -1      3       0       0
4       -1      4       0       0

输入指令(1-修改，0-不修改，其他-程序结束)和逻辑地址：0 5120
该页不存在!!!

输入指令(1-修改，0-不修改，其他-程序结束)和逻辑地址：
```

5、 实验感想

1、 本实验加强了我对请求分页理念的理解。

2、 实验编程难度不大，注意求地址、页号、页内偏移时的技巧。