

实验四 存储管理

一. 实验目的

1. 通过请求页式存储管理中页面置换算法模拟设计, 了解虚拟存储技术的特点
2. 掌握请求页式存储管理的页面置换算法。

二. 实验工具与设备

装有 Linux 操作系统的计算机。

三. 实验内容

1. 通过随机数产生一个指令序列, 共 320 条指令。指令的地址按下述原则生成:

- (1) 50% 的指令是顺序执行的;
- (2) 25% 的指令是均匀分布在前地址部分;
- (3) 25% 的指令是均匀分布在前地址部分。

具体的实施方法是:

- (1) 在 $[0, 319]$ 的指令地址之间随机选取一起点 m ;
- (2) 顺序执行一条指令, 即执行地址为 $m+1$ 的指令;
- (3) 在前地址 $[0, m+1]$ 中随机选取一条指令并执行, 该指令的地址为 m' ;
- (4) 顺序执行一条指令, 其地址为 $m'+1$;
- (5) 在后地址 $[m'+2, 319]$ 中随机选取一条指令并执行;
- (6) 重复上述步骤, 直到执行 320 次指令。

2. 将指令序列变换成为页地址流

- 设: (1) 页面大小为 1K;
- (2) 用户内存容量为 4 页到 32 页;
 - (3) 用户虚存容量为 32K。

在用户虚存中, 按每 K 存放 10 条指令排列虚存地址, 即 320 条指令在虚存中的存放方式为:

第 0 条—第 9 条指令为第 0 页 (对应虚存地址为 $[0, 9]$);

第 10 条—第 19 条指令为第 1 页 (对应虚存地址为 $[10, 19]$);

⋮

第 310 条—第 319 条指令为第 31 页 (对应虚存地址为 $[310, 319]$);

按以上方式, 用户指令可组成 32 页。

3. 计算并输出下面各种算法在不同内存容量下的命中率:

- (1) FIFO (先进先出算法)
- (2) LRU (最近最少使用算法)
- (3) OPT (最优算法)
- (4) LFU (最少使用页面算法)
- (5) CLOCK (时钟算法)

命中率 = $1 - \text{页面失效次数} / \text{页地址流长度}$

其中: 页地址流长度为 320, 页面失效次数为每次访问相应指令时, 该指令所对应的页不在内存的次数。

随机数生成办法:

Linux 系统提供了函数 `srand()` 和 `rand()`，分别进行初始化和产生随机数。例如：`srand()` 语句可初始化一个随机数：`a[0]=10*rand()/32767*319+1; a[1]=10*rand()/32767*a[0];` 语句可用来产生 `a[0]` 与 `a[1]` 中的随机数。

4. 实验指导

为了设计上述算法，可先用 `srand()` 和 `rand()` 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，并针对不同算法计算出命中率。

程序中的数据结构:

(1) 页面类型

```
typedef struct {
    int pn, pfn, counter, time;
} pl_type;
```

其中 `pn` 为页号，`pfn` 为页面号，`counter` 为一个周期内访问该页面次数，`time` 为访问时间

(2) 页面控制结构

```
struct pfc_struct {
    int pn, pfn;
    struct pfc_struct *next;
};
```

```
typedef struct pfc_struct pfc_type;
pfc_type pfc[total_vp], *freepf_head, *busypf_head;
pfc_type *busypf_tail;
```

其中，`pfc[total_head]` 定义用户进程虚页控制结构，`*freepf_head` 为空页面头的指针，`*busypf_head` 为忙页面头的指针，`*busypf_tail` 为忙页面尾的指针

(3) 变量定义

```
int a[total_instruction]; 指令流数据组
int page[total_instruction]; 每条指令所属页号
int offset[total_instruction]; 每页装入10条指令后取模运算页号偏移值
int total_pf; 用户进程的内存页面数
int disaffect; 页面失效次数
```

5. 程序的结构框架

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

#define TRUE 1
#define FALSE 0
#define INVALID -1
#define NULL 0

#define total_instruction 320 /*指令流长*/
#define total_vp 32 /*虚页长*/
#define clear_period 50 /*清零周期*/

typedef struct { /*页面结构*/
    int pn, pfn, counter, time;
```

```

    }pl_type;
    pl_type pl[total_vp];          /*页面结构数组*/

    struct pfc_struct{              /*页面控制结构*/
        int pn, pfn;
        struct pfc_struct *next;
    };

    struct pfc_struct pfc[total_vp], *freepf_head, *busypf_head, *busypf_tail;

    int diseffect,  a[total_instruction];
    int page[total_instruction],  offset[total_instruction];
    void initialize();
    void FIFO( );
    void LRU( );
    void OPT( );
    void LFU( );
    void CLOCK( );

    int main()
    {
        int S,i;
        srand(10*getpid());
        /*由于每次运行时进程号不同, 故可用来作为初始化随机数队列的“种子”*/
        S=(int)(319.0*rand()/RAND_MAX)+1;
        for(i=0;i<total_instruction;i+=4) /*产生指令队列*/
        {
            a[i]=S;          /*任选一指令访问点*/
            a[i+1]=a[i]+1;    /*顺序执行下一条指令*/
            a[i+2]=(int)(1.0*a[i]*rand()/RAND_MAX);          /*执行前地址指令m'*/
            a[i+3]=a[i+2]+1;  /*执行后地址指令*/
            S=(int)(1.0*rand()*(318-a[i+2])/RAND_MAX)+a[i+2]+2;
        }

        for(i=0;i<total_instruction;i++) /*将指令序列变换成页地址流*/
        {
            page[i]=a[i]/10;
            offset[i]=a[i]%10;
        }
        for(i=4;i<=32;i++) /*用户内存工作区从4个页面到32个页面*/
        {
            printf("%2d page frames\t",i);
            FIFO(i);
            LRU(i);
            OPT(i);
            LFU(i);
            CLOCK(i);
        }
        return 0;
    }

    void initialize(int total_pf) /*初始化相关数据结构*/

```

```

{
    int i;
    diseffect=0;
    for(i=0;i<total_vp;i++)
    {
        pl[i].pn=i;
        pl[i].pfn=INVALID; /*置页面控制结构中的页号，页面为空*/
        pl[i].counter=0;
        pl[i].time=-1; /*置页面控制结构中的访问次数，时间为-1*/
    }
    for(i=1;i<total_pf;i++)
    {
        pfc[i-1].next=&pfc[i];
        pfc[i-1].pfn=i-1;
    } /*建立pfc[i-1]和pfc[i]之间的链接*/
    pfc[total_pf-1].next=NULL;
    pfc[total_pf-1].pfn=total_pf-1;
    freepf_head=&pfc[0];
    /*空页面队列的头指针为pfc[0]*/
}

void FIFO(int total_pf)
{
    int i;
    struct pfc_struct *p;
    initialize(total_pf); /*初始化相关页面控制用数据结构*/
    busypf_head=busypf_tail=NULL; /*忙页面队列，队列尾链接*/
    for(i=0;i<total_instruction;i++)
    {
        if(pl[page[i]].pfn==INVALID) /*页面失效*/
        {
            diseffect+=1; /*失效次数*/
            if(freepf_head==NULL) /*无空闲页面*/
            {
                p=busypf_head->next;
                pl[busypf_head->pn].pfn=INVALID;
                freepf_head=busypf_head; /*释放忙页面队列中的第一个页面*/
                freepf_head->next=NULL;
                busypf_head=p;
            }
            p=freepf_head->next; /*按FIFO方式调新页面入内存页面*/
            freepf_head->next=NULL;
            freepf_head->pn=page[i];
            pl[page[i]].pfn=freepf_head->pfn;
            if(busypf_tail==NULL) busypf_head=busypf_tail=freepf_head;
            else
            {
                busypf_tail->next=freepf_head;
                busypf_tail=freepf_head;
            }
        }
    }
}

```

```
    }
    freepf_head=p;
  }
}
printf("FIFO:%6.4f  ",1-(float)diseffect/320);
}

void LRU(int total_pf)
{ }

void OPT(int total_pf)
{ }

void LFU(int total_pf)
{ }

void CLOCK(int total_pf)
{ }
```

四. 思考题

1. 为什么要进行内存管理，虚拟存储器的特点是什么？
2. 几种内存管理算法有何区别与联系？