```
        }
    else
        {
            while((pid2 = fork( )) = = -1);
            if(pid2 = = 0)
                {   lockf(fd[1],1,0);        //互斥
                    sprintf(outpipe,"child 2 process is sending message!");
                    write(fd[1],outpipe,50);
                    sleep(3);
                    lockf(fd[1],0,0);
                    exit(0);
                }
            else
                {   wait(0);                 //同步
                    read(fd[0],inpipe,50); //从管道中读长为 50 字节的串
                    printf("%s\n",inpipe);
                    wait(0);
                    read(fd[0],inpipe,50);
                    printf("%s\n",inpipe);
                    exit(0);
                }
        }
}
```

图 21　多进程的管道通信

**(四)编写程序**

(来自第三章习题)假定系统有三个并发进程 read,move 和 print 共享缓冲器 B1 和 B2。进程 read 负责从输入设备上读信息,每读出一个记录后把它存放到缓冲器 B1 中。进程 move 从缓冲器 B1 中取出一个记录,加工后存入缓冲器 B2。进程 print 将 B2 中的记录取出打印输出。缓冲器 B1 和 B2 每次只能存放一个记录。要求三个进程协调完成任务,使打印出来的与读入的记录的个数,次序完全一样。试创建三个进程,用 pipe( )打开两个管道,如图 22 所示,实现三个进程之间的同步。

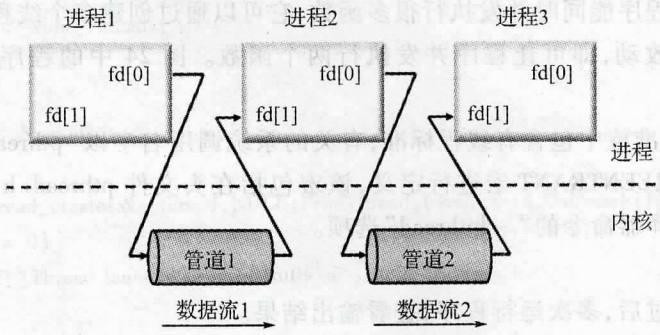


图 22　三个进程之间的同步

# 实验3 存储管理

## 实验目的

存储管理的主要功能之一是合理地分配空间。请求页式存储管理是一种常用的虚拟存储管理技术。

本实验的目的是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，掌握请求页式存储管理的页面置换算法。

## 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后地址部分。

具体的实施方法如下：

① 在[0,319]的指令地址之间随机选取一个起点 $m$。

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 重复步骤①～⑤，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1KB。

② 用户内存容量为4～32页。

③ 用户虚存容量为32KB。

在用户虚存中，按每页存放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0～9条指令为第0页(对应虚存地址为[0,9])；

第10～19条指令为第1页(对应虚存地址为[10,19])；

……

第310～319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) 计算并输出下述各种算法在不同内存容量下的命中率。

① 先进先出的算法(FIFO)；

② 最近最少使用算法(LRR)；

③ 最佳淘汰算法(OPT)：先淘汰最不常用的页地址；

④ 最少访问页面算法(LFR)；

⑤ 最近最不经常使用算法(NUR)。

其中③和④为选择内容。

命中率计算公式如下：

命中率＝1－页面失效次数/页地址流长度

在本实验中，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。

## 随机数产生办法

Linux或UNIX系统提供了函数srand()和rand()，分别进行初始化并产生随机数。例如，下面的语句可初始化一个随机数：

```
srand();
```

下面的语句可用来产生a[0]与a[1]中的随机数：

```
a[0]=319 * rand()/32767/32767/2+1;
a[1]=a[0] * rand()/32767/32767/2;
```

# 实验 3 指导

【任务】

设计一个虚拟存储区和内存工作区，并使用下述算法计算访问命中率。

(1) 先进先出的算法(FIFO)；

(2) 最近最少使用算法(LRU)；

(3) 最佳淘汰算法(OPT)；

(4) 最少访问页面算法(LFU)；

(5) 最近最不经常使用算法(NUR)。

命中率计算公式如下：

命中率＝(1－页面失效次数)/页地址流长度

【程序设计】

本实验的程序设计基本上按照实验内容进行，即首先用 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，并针对不同的算法计算出相应的命中率。相关定义如下。

(1) 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 为面号，counter 为一个周期内访问该页面的次数，time 为访问时间。

② 页面控制结构

```
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_vp]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。



(2) 函数定义

void initialize()

void FIFO()：计

void LRU()：计

void OPT()：计

void LFU()：计

void NUR()：计

(3) 变量定义

int a[tatal_instr

int page[total_i

int offset[total_

int total_pf：用

int diseffect：页

(4) 程序流程图略。

【程序】

```
#define TRUE 1
#define FALSE 0
#define INVALI
#define NULL 0

#define total_
#define total_
#define clear_

typedef struct
{
    int pn,pfn
}pl_type;
pl_type pl[32]
typedef struct
{
    int pn,pfn
    struct pfc
}pfc_type;
pfc_type pfc[

int diseffect
int page[tota

void initiali
void FIFO();
```

(2) 函数定义

void initialize()：初始化函数，给每个相关的页面赋值。

void FIFO()：计算使用 FIFO 算法时的命中率。

void LRU()：计算使用 LRU 算法时的命中率。

void OPT()：计算使用 OPT 算法时的命中率。

void LFU()：计算使用 LFU 算法时的命中率。

void NUR()：计算使用 NUR 算法时的命中率。

(3) 变量定义

int a[tatal_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 diseffect：页面失效次数。

(4) 程序流程图

略。

【程序】

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

#define total_instruction 320
#define total_vp 32
#define clear_period 50

typedef struct
{
    int pn,pfn,counter,time;
}pl_type;
pl_type pl[32];
typedef struct pfc_struct
{
    int pn,pfn;
    struct pfc_struct * next;
}pfc_type;
pfc_type pfc[32], * 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 NUR();

void main()
{
    int s,i,j;
    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=(float)319*rand()/32767/32767/2+1;
    for(i=0;i<total_instruction;i+=4)            /*产生指令队列*/
    {
        if(s<0||s>319)
        {
            printf("When i==%d,Error,s==%d\n",i,s);
            exit(0);
        }
        a[i]=s;                                  /*任选一个指令访问点 m*/
        a[i+1]=a[i]+1;                           /*顺序执行一条指令*/
        a[i+2]=(float)a[i]*rand()/32767/32767/2;     /*执行前地址指令 m'*/
        a[i+3]=a[i+2]+1;                         /*顺序执行一条指令*/
        s=(float)(318-a[i+2])*rand()/32767/32767/2+a[i+2]+2;
        if((a[i+2]>318)||(s>319))
            printf("a[%d+2],a number which is:%d and s==%d\n",i,a[i+2],s);
    }
    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",i);
        FIFO(i);
        LRU(i);
        OPT(i);
        LFU(i);
        NUR(i);
        printf("\n");
    }
}

void initialize(total_pf)                        /*初始化相关数据结构*/
int total_pf;                                    /*用户进程的内存页面数*/
```

```
{
    int i;
    diseffect=0;

    for(i=0;i<total_vp;i++)
    {
        pl[i].pn=i;                          /*置页面控制结构中的页号,页面为空*/
        pl[i].pfn=INVALID;
        pl[i].counter=0;                     /*页面控制结构中的访问次数为0,时间为-
    1*/
    }
    for(i=0;i<total_pf-1;i++)
    {
        pfc[i].next=&pfc[i+1];
        pfc[i].pfn=i;                        /*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]*/
}
```

要求打印类似如下结果，并分析不同算法缺页率随内存页面数目变化情况

**【结果】**

```
4 page frames FIFO:0.5312 LRU:0.5281 OPT:0.5687 LFU:0.5344 NUR:0.5531
5 page frames FIFO:0.5344 LRU:0.5406 OPT:0.6000 LFU:0.5469 NUR:0.5594
6 page frames FIFO:0.5594 LRU:0.5563 OPT:0.6188 LFU:0.5719 NUR:0.5813
7 page frames FIFO:0.5719 LRU:0.5719 OPT:0.6438 LFU:0.5875 NUR:0.5906
8 page frames FIFO:0.5938 LRU:0.5875 OPT:0.6594 LFU:0.5969 NUR:0.5781
9 page frames FIFO:0.6062 LRU:0.6062 OPT:0.6656 LFU:0.6031 NUR:0.6219
10 page frames FIFO:0.6125 LRU:0.6188 OPT:0.7000 LFU:0.6250 NUR:0.6188
11 page frames FIFO:0.6469 LRU:0.6281 OPT:0.7063 LFU:0.6375 NUR:0.6469
12 page frames FIFO:0.6594 LRU:0.6531 OPT:0.7188 LFU:0.6406 NUR:0.6656
```