

实验 10 FIFO 页面置换算法

1. 实验目的

通过请求页式存储管理中的先进先出（FIFO）算法的模拟设计，理解 FIFO 算法的设计思想及 FIFO 算法的实现方法。

2. 相关知识

(1) FIFO 算法的基本思想

在请求页式管理中，当发生缺页中断且主存没有空闲页面时，总是淘汰最先进入主存的页面，即选择在主存中驻留时间最久的页面被淘汰。

(2) FIFO 算法实现原理

把一个进程已调入主存的页面按先后次序链接成一个队列，并设置一个指针即可。该算法是一种最直观但性能较差的页面置换算法，并可能会有 BELADY 异常现象，即分配的页面数增加时，缺页中断次数反而增加。

3. 实验要求

假如某进程 P 有 5 个页，进程访问页的顺序为：1，2，3，4，1，2，5，1，2，3，4，5；如果在内存中分配给该进程 3 个页面，给出缺页中断次数和页面的淘汰顺序，示例如下：

页面	1	2	3	4	1	2	5	1	2	3	4	5
页面 1	1	2	3	4	1	2	5	5	5	3	4	4
页面 2		1	2	3	4	1	2	2	2	5	3	3
页面 3			1	2	3	4	1	1	1	2	5	5
缺页	Y	Y	Y	Y	Y	Y	Y	N	N	Y	Y	N

4. 实验程序

```
【程序 4_1】FIFO 算法实验程序。
#include "stdafx.h"
void main()
{
    int i, j, k=0;
    char cc[13];        //Y 表示产生缺页中断，N 表示未产生缺页中断
    int a[12]={1,2,3,4,1,2,5,1,2,3,4,5};    //页面的调度顺序
    int b[3][13],c[13],p=0;
    b[0][0]=0;        //C 语言定义数组，其初值是不确定的
    b[1][0]=0;
    b[2][0]=0;
    printf("      ");
    for(i=0;i<12;i++)
        printf("%6d",a[i]);
    printf("\n      =====\n");
    for(i=0;i<12;i++)
    {
        if(a[i]==b[0][i]||a[i]==b[1][i]||a[i]==b[2][i])
```

```

        {
            b[0][i+1]=b[0][i];
            b[1][i+1]=b[1][i];
            b[2][i+1]=b[2][i];
            cc[i]='N';
        }
    else
    {
        if(i>2)
        {
            c[p]= b[2][i];
            p++;
        }
        b[0][i+1]=a[i];
        b[1][i+1]=b[0][i];
        b[2][i+1]=b[1][i];
        cc[i]='Y';
        k=k+1;
    }
}

for(j=0;j<13;j++)
printf("%6d",b[0][j]);
printf("\n      -----\\n");
for(j=0;j<13;j++)
printf("%6d",b[1][j]);
printf("\n      -----\\n");
for(j=0;j<13;j++)
    printf("%6d",b[2][j]);
printf("\n      -----\\n");
printf("      ");
for(j=0;j<13;j++)    //打印是否产生缺页中断
{
    printf("      ");
    putchar(cc[j]);
}
printf("\\n 缺页中断次数: %3d\\n",k);
printf("页面淘汰顺序: ");
for(j=0;j<p;j++)
    printf("%3d",c[j]);
printf("\\n\\n");
}

```

5. 【程序运行结果截图】

	1	2	3	4	1	2	5	1	2	3	4	5
=====												
0	1	2	3	4	1	2	5	5	5	3	4	4
=====												
0	0	1	2	3	4	1	2	2	2	5	3	3
=====												
0	0	0	1	2	3	4	1	1	1	2	5	5
=====												
	Y	Y	Y	Y	Y	Y	Y	N	N	Y	Y	N
缺页中断次数: 9												
页面淘汰顺序: 1 2 3 4 1 2												

6. 实验思考与扩充

将【程序 4_1】改为 4 个页面，运行后的结果与【程序 4_1】的运行结果进行对比。

实验 11 LRU 页面置换算法

1. 实验目的

通过请求页式存储管理中的最近最久未使用页面置换算法（LRU）的模拟设计，进一步理解 LRU 算法的基本思想，给出 LRU 算法的实现原理和方法，并具体编程实现该算法的模拟程序。

2. LRU 算法的基本思想

该算法的基本思想是：当需要淘汰某一页时，选择离当前时间最近的一段时间内最久没有使用过的页被淘汰。该算法的出发点是，如果某页被访问了，则它可能马上还要被访问；或者反过来说，如果某页很长时间未被访问，则它在最近一段时间也不会被访问。

3. LRU 算法实现原理

当某进程发生缺页中断且主存没有空闲页面时，为了选择离当前时间最近的一段时间内最久没有被访问的页被淘汰，可设计一种数据结构 $c[5][2]$ ，第一列表示页号，共有 5 页（页号 1~5），第二列表示每个页的计数值，其初值均为 0。

当访问某页时，如果该页在内存，则取计数值最大的且加 1，作为该页的计数值；如果该页不在内存且内存有空闲页面，则将该页调入内存，且取计数值最大的且加 1，作为该页的计数值；如果该页不在内存且内存没有空闲页面时，淘汰计数值最小的页（计数值最大的是刚刚被访问的页面，计数值最小的就是最近最久未被访问的页面），并将被淘汰页的计数值清零，然后将该页调入内存，并取计数值最大的且加 1，作为该页的计数值。

4. 实验要求

假如某进程 P 有 5 个页，进程访问页的顺序为：4, 3, 2, 1, 4, 3, 5, 4, 3, 2, 1, 5；如果在内存中分配给该进程 3 个页面，示例如下：

页面	4	3	2	1	4	3	5	4	3	2	1	5
页面 1	4	3	2	1	4	3	5	4	3	2	1	5
页面 2		4	3	2	1	4	3	5	4	3	2	1
页面 3			4	3	2	1	4	3	5	4	3	2
缺页	Y	Y	Y	Y	Y	Y	Y	N	N	Y	Y	Y

5. 实验程序

【程序 4_2】LRU 算法实验程序。

```

#include <stdafx.h>
void max_value(int x, int cc[][2]);    //函数声明，页表处理
int r_algorithm(int cc[][2]);    //函数声明，选择页面淘汰算法
char cc[13];    //Y 表示产生缺页中断，N 表示未产生缺页中断
void page_table(int page1, int c[5][2]);    //打印页表
void main( )
{
    int i, j, page, row=0, col=1;    //b[row][col], 行/列指针
    int k=0;    //记录缺页中断次数
    int a[12]={4,3,2,1,4,3,5,4,3,2,1,5};    //存放页的调度顺序
    int b[3][12];    //模拟内存（分配三个页面）
    int c[5][2]={ {1,0},{2,0},{3,0},{4,0},{5,0}};    //定义页表并赋初值
    int d[13],p=0;    //存放页面淘汰顺序，p 页面淘汰数组 d 的指针
    b[0][0]=0;    //数组的初值不确定，0 表示页面为空
    b[1][0]=0;
    b[2][0]=0;
    //*****页面调度处理*****
    for(i=0;i<12;i++)
    {
        if(a[i]==b[0][i]||a[i]==b[1][i]||a[i]==b[2][i])
        {
            b[0][i+1]=b[0][i];    //将前一列数据复制到下一列
            b[1][i+1]=b[1][i];
            b[2][i+1]=b[2][i];
            max_value(a[i],c);    //处理页表，a[i]页面是刚被访问的页面
            page_table(a[i],c);    //打印页表
            cc[i]='F';
            col++;    //col 指向下一列
        }
        else    //页面不在内存
        {
            if(row>2)    //row>2 表示内存已没有空闲页面
            {
                page = r_algorithm(c);    //返回淘汰的页面 page
                d[p] = page;    //d[]存放被淘汰的页面
                p++;
                k++;    //缺页中断次数
                b[0][i+1]=b[0][i];    //将前一列数据复制到下一列
                b[1][i+1]=b[1][i];
                b[2][i+1]=b[2][i];
                cc[i]='Y';
                if(b[0][i+1]==page)
                    b[0][i+1]=a[i];
                if(b[1][i+1]==page)

```

```

        b[1][i+1]=a[i];
        if(b[2][i+1]==page)
            b[2][i+1]=a[i];
        max_value(a[i],c);    //访问 a[i]页面， i 页面是刚被访问的页面
        page_table(a[i],c);    //打印页表
    }
    else
    {
        b[0][i+1]=b[0][i];    //将前一列数据复制到下一列
        b[1][i+1]=b[1][i];
        b[2][i+1]=b[2][i];
        cc[i]='Y';
        b[row][col]=a[i];    //a[i]页面进入内存
        max_value(a[i],c);    //访问 a[i]页面， i 页面是刚被访问的页面
        col++;
        k++;    //缺页中断次数
        row++;
        page_table(a[i],c);    //打印页表
    }
}

//=====显示处理结果=====
printf("\n      ");
for(i=0;i<12;i++)
    printf("%6d",a[i]);    //显示页面调度顺序
printf("\n      =====\n");
for(j=0;j<13;j++)
    printf("%6d",b[0][j]);
printf("\n      -----\n");
for(j=0;j<13;j++)
    printf("%6d",b[1][j]);
printf("\n      -----\n");
for(j=0;j<13;j++)
    printf("%6d",b[2][j]);
printf("\n      -----\n");
printf("      ");
for(j=0;j<13;j++)    //打印是否产生缺页中断
{
    printf("      ");
    putchar(cc[j]);
}
printf("\n 缺页中断次数: %4d\n",k);
printf("\n 页面淘汰顺序: ");
for(j=0;j<p;j++)

```

```

        printf("%3d",d[j]);    //显示页面淘汰顺序
    printf("\n\n");
}
//=====访问的页面在内存的处理(页表处理)=====
void max_value(int x,int cc[][2])    //x-页号：求页表中计数的最大值，并将该页面置为最新
访问的页面
{
    int i, max;
    max=cc[0][1];
    for(i=0; i<5; i++)
        if(max < cc[i][1])
            max=cc[i][1];
    for(i=0; i<5; i++)
        if(cc[i][0]==x)
            cc[i][1]=max+1;
}
//=====选择被淘汰的页面(页表处理)=====
int r_algorithm(int cc[5][2])
{
    int i, min, row, p;
    for(i=0;i<5;i++)    //查询第一个计数为非 0 的页面的计数值
        if(cc[i][1]!=0 )
        {
            min=cc[i][1];
            p=cc[i][0];
            row=i;
            break;
        }
    for(i=0; i<5; i++)    //寻找计数值最小的数页面
    {
        if(min>cc[i][1] && cc[i][1]!=0)
        {
            min=cc[i][1];
            p=cc[i][0];    //最小数所对应的页号被淘汰
            row=i;    //记录最小数所在的行
        }
    }
    cc[row][1]=0;    //在页表中被淘汰的页面计数清零
    return(p);    //返回被淘汰的页面--P
}
//=====
void page_table(int page1, int c[5][2])    //打印页表
{
    int i;

```

```

printf("页面调度顺序 page= %d\n",page1);
for(i=0;i<5;i++)
    printf("%5d%5d\n",c[i][0],c[i][1]);
}

```

6. 【程序运行结果截图】

```

      4    3    2    1    4    3    5    4    3    2    1    5
=====
0    4    4    4    1    1    1    5    5    5    2    2
-----
0    0    3    3    3    4    4    4    4    4    4    1
-----
0    0    0    2    2    2    3    3    3    3    3    5
-----
      Y    Y    Y    Y    Y    Y    Y    N    N    Y    Y
缺页中断次数:  10
页面淘汰顺序:  4  3  2  1  5  4  3

```

7. 实验思考与扩充

(1) 修改【程序 4_2】，将分配给该进程 3 个页面改为 4 个页面，并对运行结果进行分析与对比。

(2) 如果 LRU 算法实现原理应用于 FIFO 算法，则 FIFO 页面置换算法的实验程序应该如何修改？

(3) 如何实现 LRU 两个近似算法：最不经常使用的页面淘汰算法（LFU）和最近没使用的页面淘汰算法（NUR）？