

**学生实验报告**

实验课名称：操作系统

实验项目名称：存储管理

专业名称：计算机科学与技术（交通信息工程）

班级：2016240203

学号：2016902094

学生姓名：瞿强鑫

教师姓名：刘晓春

2019**年** 5 **月** 21**日**

1. 实验名称

存储管理

1. 实验内容

设计一个请求页式存储管理方案，并编写模拟程序实现之。淘汰算法采用两种不同的算法如：FIFO和LRU，并比较它们的不同之处。

1. 实验原理和设计思路

不同的置换算法，可使同一组进程发生的缺页率不同，如果采用的置换算法不当，会大大降低CPU的使用高效率。

FIFO算法优先置换最先进入内存的页。LRU每次选择离当前时间被访问最远的页置换。

在本实验设计的的数据结构实现过程中，差别就是LRU对应存储单元在置换时，如果内存中已经存了则将时间置为0（新存入），FIFO对应存储单元的存储值已经在内存中了则存储的时间不变（未置换）。

1. 源代码

#include<conio.h>

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#define bsize 4 //物理块大小

#define psize 16 //进程大小

typedef struct page

{

int num; /\*记录页面号\*/

int time; /\*记录调入内存时间\*/

}Page; /\* 页面逻辑结构，结构为方便算法实现设计\*/

Page b[bsize]; /\*内存单元数\*/

int c[bsize][psize]; /\*暂保存内存当前的状态：缓冲区\*/

int queue[100]; /\*记录调入队列\*/

int K; /\*调入队列计数变量\*/

int phb[bsize] = { 0 }; //物理块标号

int pro[psize] = { 0 }; //进程序列号

int flag[bsize] = { 0 }; //进程等待次数(存放最久未被使用的进程标志)

int i = 0, j = 0, k = 0; //i表示进程序列号,j表示物理块号

int m = -1, n = -1; //物理块空闲和进程是否相同判断标志

int max = -1, maxflag = 0; //标记替换物理块进程下标

int count = 0; //统计页面缺页次数

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//随机产生序列号函数

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int\* build()

{

printf("随机产生一个进程序列号为：\n");

int i = 0;

for (i = 0; i<psize; i++)

{

pro[i] = 10 \* rand() / (RAND\_MAX + 1) + 1;

printf("%d ", pro[i]);

}

printf("\n");

return(pro);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//查找空闲物理块

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int searchpb()

{

for (j = 0; j<bsize; j++)

{

if (phb[j] == 0)

{

m = j;

return m;

break;

}

}

return -1;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//查找相同进程

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int searchpro()

{

for (j = 0; j < bsize; j++)

{

if (phb[j] == pro[i])

{

n = j;

return j;

}

}

return -1;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//初始化内存

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void empty()

{

for (i = 0; i<bsize; i++)

phb[i] = 0;

count = 0; //计数器置零

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//先进先出页面置换算法

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*初始化内存单元、缓冲区\*/

void Init(Page \*b, int c[bsize][psize])

{

int i, j;

for (i = 0; i<psize; i++)

{

b[i].num = -1;

b[i].time = psize - i - 1;

}

for (i = 0; i<bsize; i++)

for (j = 0; j<psize; j++)

c[i][j] = -1;

}

/\*取得在内存中停留最久的页面,默认状态下为最早调入的页面\*/

int GetMax(Page \*b)

{

int i;

int max = -1;

int tag = 0;

for (i = 0; i<bsize; i++)

{

if (b[i].time>max)

{

max = b[i].time;

tag = i;

}

}

return tag;

}

/\*判断页面是否已在内存中\*/

int Equation(int fold, Page \*b)

{

int i;

for (i = 0; i<bsize; i++)

{

if (fold == b[i].num)

return i;

}

return -1;

}

void Lruu\_FIFO(int fold, Page \*b)

{

int i;

int val;

val = Equation(fold, b);//查看是否在内存中

if (val >= 0) //如果在

{

for (i = 0; i<bsize; i++)

b[i].time++; //所有页面++

}

else

{

queue[++K] = fold;/\*记录调入页面\*/

val = GetMax(b);

b[val].num = fold;

b[val].time = 0;

for (i = 0; i<bsize; i++)

if (i != val)

b[i].time++;

}

}

void FIFO()

{

int i, j;

K = -1;

Init(b, c);

for (i = 0; i<psize; i++)

{

Lruu\_FIFO(pro[i], b);

c[0][i] = pro[i];

/\*记录当前的内存单元中的页面\*/

for (j = 0; j<bsize; j++)

c[j][i] = b[j].num;

}

/\*结果输出\*/

printf("进程序列号输入流：\n");

for (j = 0; j<psize; j++)

printf("|%2d ", pro[j]);

printf("|\n");

printf("内存状态为：\n");

for (i = 0; i<bsize; i++)

{

for (j = 0; j<psize; j++)

{

if (c[i][j] == -1)

printf("|%2c ", 32);

else

printf("|%2d ", c[i][j]);

}

printf("|\n");

}

printf("\n调入队列为:");

for (i = 0; i<K + 1; i++)

printf("%3d", queue[i]);

printf("\n缺页次数为：%d\n缺页率：%.2f%%", K + 1, (float)(K + 1) / psize \* 100);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*LRU核心部分\*/

void Lruu(int fold, Page \*b)

{

int i;

int val;

val = Equation(fold, b);//查看是否在内存中

if (val >= 0) //如果在

{

b[val].time = 0; //与FIFO区别！！！！

for (i = 0; i<bsize; i++)

if (i != val)

b[i].time++; //所有其他页面++

}

else

{

queue[++K] = fold;/\*记录调入页面\*/

val = GetMax(b);

b[val].num = fold;

b[val].time = 0;

for (i = 0; i<bsize; i++)

if (i != val)

b[i].time++;

}

}

void LRU()

{

int i, j;

K = -1;

Init(b, c);

for (i = 0; i<psize; i++)

{

Lruu(pro[i], b);

c[0][i] = pro[i];

/\*记录当前的内存单元中的页面\*/

for (j = 0; j<bsize; j++)

c[j][i] = b[j].num;

}

/\*结果输出\*/

printf("进程序列号输入流：\n");

for (j = 0; j<psize; j++)

printf("|%2d ", pro[j]);

printf("|\n");

printf("内存状态为：\n");

for (i = 0; i<bsize; i++)

{

for (j = 0; j<psize; j++)

{

if (c[i][j] == -1)

printf("|%2c ", 32);

else

printf("|%2d ", c[i][j]);

}

printf("|\n");

}

printf("\n调入队列为:");

for (i = 0; i<K + 1; i++)

printf("%3d", queue[i]);

printf("\n缺页次数为：%d\n缺页率：%.2f%%", K + 1, (float)(K + 1) / psize \* 100);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//主函数

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void main()

{

int sel;

again:

build();

printf("先进先出算法(FIFO)\n"); FIFO(); empty(); printf("\n");

printf("最久未使用算法(LRU)\n"); LRU(); empty(); printf("\n");

printf("输入1继续生成随机进程序列号:");

scanf\_s("%d", &sel);

if (sel = 1)

goto again;

}

1. 结果截图

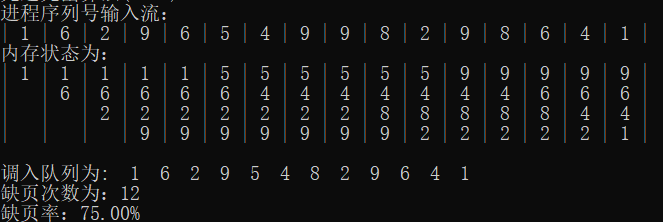


图1. FIFO算法

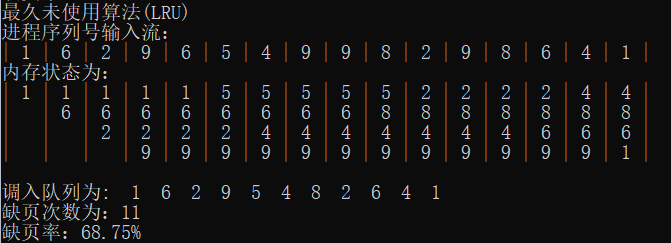


图2. LRU算法

1. 编程体会

本次实验做的是存储管理的模拟实验，在Visual Studio C语言环境下编写，使用的数据结构就是结构体数组，二维数组和队列，结构体数组用于存储单元的各项属性：值，存储时间，二维数组用于构建交互表格，将每个时间点存储单元的存储内容通过4\*16的二维数组将其展现出来，还有队列用于存储每次调入到内存的进程序列号（值）以展示每次参与置换的块。

在本实验设计的的数据结构实现过程中，差别就是LRU对应存储单元在置换时，如果内存中已经存了则将时间置为0（新存入），FIFO对应存储单元的存储值已经在内存中了则存储的时间不变（未置换）。

通过本次实验，对存储管理的LRU和FIFO算法有了更深入的了解和认识，学到了很多！