

操作系统原理

课程设计

**班 级： 18计科卓越**

**组 长： 周 超**

**成 员： 郭喜彬**

**2020 - 2021学年 第 二 学期**

1. 设计题目

模拟页面置换算法——FIFO算法

1. 设计目标
2. FIFO算法核心

先进先出（First In First Out，FIFO）算法的核心是更换最早进入内存的页面。先进先出是任何人都能直观想到的办法。 

最简单的分页替换算法就是先进先出算法，当每次有新的[分页](https://baike.baidu.com/item/%E5%88%86%E9%A1%B5" \t "https://baike.baidu.com/item/%E5%85%88%E8%BF%9B%E5%85%88%E5%87%BA%E7%AE%97%E6%B3%95/_blank)需要调入时，会选择调入内存时间最久的分页换出。有两种实现的方法：第一种是记录每个分页被调入到页框的时间，当每次需要换出分页时，会找到调入时间最早的一页，也就是在主存储器中存在最久的分页。另外一种方式就是利用[FIFO](https://baike.baidu.com/item/FIFO/64838" \t "https://baike.baidu.com/item/%E5%85%88%E8%BF%9B%E5%85%88%E5%87%BA%E7%AE%97%E6%B3%95/_blank)队列来实现，当要进行分页替换时，就把队列最前端的分页换出，再把要调入的分页放到队列的末端。

# FIFO算法原理

# FIFO（First In First Out）先进先出，也可以是FCFS（First Come First Serve 先来先服务）

1. FIFO算法流程

假设有3个进程A、B和C，首先强调，进程只能顺序执行。放宽任务同时到达的条件，假设A比B先到一点点，B比C先到一点点，根据FIFO的原理，执行顺序应该是：

A先执行；

A执行完，B再执行；

B执行完，C再执行；

1. FIFO算法特点

先到达的任务，先执行。

1. FIFO算法缺点

这种绝对的公平方式容易导致效率的降低。例如，如果最先加载进来的页面是经常被访问的页面，这样做很可能造成常被访问的页面替换到磁盘上，导致很快就需要再次发生缺页中断，从而降低效率。

（六）FIFO算法思想

当进程访问一个页面时，会有三种情况：

（a）有空闲的内存页,且内存中没有该页面

（b）没有空闲的内存页，但是内存中已有该页面

（c）没有空闲的内存页，内存中也没有该页面，要进行页面置换

举例如下：

测试用例：页面队列为：7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1；分配的内存物理块数为3。

FIFO就是寻找在内存中存在时间最久的那一个页面，将之替换出去，我们可以设置一个countOldPoint变量，初始值为0，来指示呆在内存中最久的页面;

设置一个变量PRO\_MEMORY来指示分配的内存物理块数。比如说，

此时内存可用的空间已满，那么countOldPoint初始值为0,也就是开始时指示7这个页面，当内存空间满时，且要调入内存空间的页面不在此时的内存中时，则要发生替换，则将7替换出去，然后，令countOldPoint++，此时countOldPoint指示的是页面0.所以实现时，即当发生页面替换时，就会令countOldPoint++，并且令countOldPoint % PRO\_MEMORY（当countOldPoint指向内存中最后一个物理块时，下一次）。这样便实现了之前所说的“替换指针”的作用。下面解释这段话：

从（1）状态到（2）状态，是内存调入了页面2，并把页面7替换出去，接着替换指针便指向了第二个内存物理块中存的页面；

从（2）状态到（3）状态，是因为内存需要页面0，但此页面已在内存中，所以不发生页面置换，所以替换指针也不发生变化；

从（3）状态到（4）状态，是内存调入了页面3，并把当前替换指针指向的页面0替换出去了，接着替换指针指向了第三个内存物理块中存的页面；

从（4）状态到（5）状态，是内存调入了页面0，并把当前替换指针指向的页面1替换出去了，因为总共分配了三块物理块，所以现在替换指针便又开始指向第一个内存物理块中的页面；

依次循环，执行完后面所有的页面。

总结：替换指针总是在发生页面置换的时候改变指向的内存物理块中，也就是改变了指向的页面。

1. 设计内容和步骤
2. 概要设计

1、数据设计

常变量:

物理页框数(pageFrameNum)；页面走向个数(pageFrontNum)

页面数组:

页面走向：pages[pageFrontNum]；物理页框：pageFrame[pageFrameNum]

变量:

缺页次数（pageMissNum），调用次数（count）

辅助栈：

stack<int> helpstack，实现栈的逆置；stack<int> pageDesert，淘汰页面

1. 详细设计

第一步:输入分配给该作业的物理页框数，输入该作业的页面走向个数。

第二步:判断物理页框中是否存在本次页面走向，若本次页面走向,物理页框中已存在，则直接进入下次页面走向。

第三步:判断物理页框有无空位。本次页面走向，物理页框中不存在，且有空位，按顺序放入。

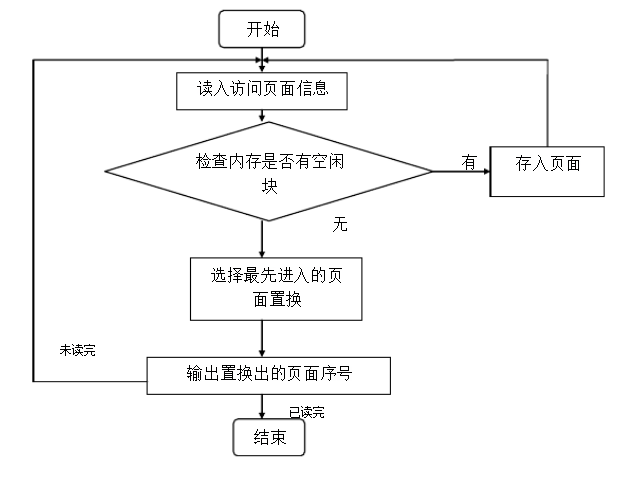
第四步:本次页面走向，物理页框中不存在，且物理页框中没有空位，调用FIFO 算法。

第五步：淘汰页面入栈。

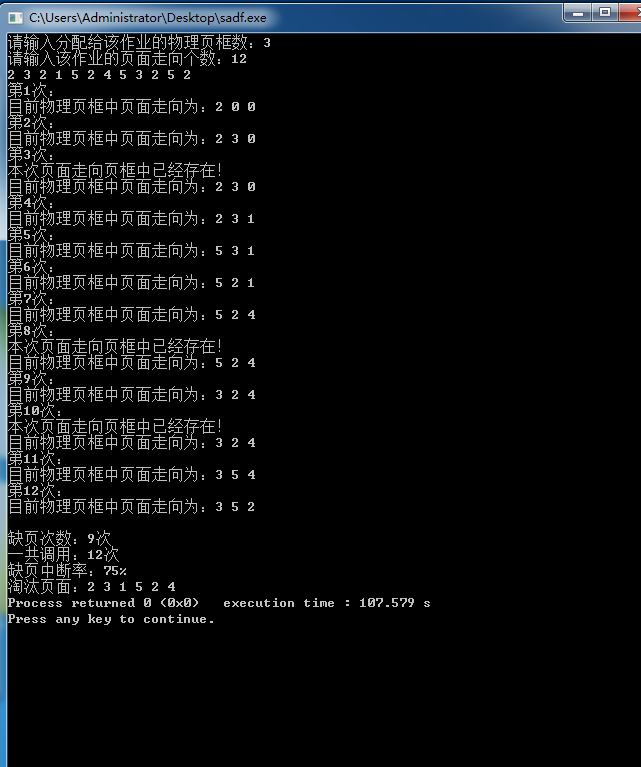
第六步：计算缺页次数、计算缺页中断率以及淘汰页面。

第七步：输出。

1. 流程图



1. 运行效果截图



1. 设计总结

这次实验让我们深刻理解了FIFO算法。在进程运行过程中，如果访问的页面在内存中不存在而需要把它们调入内存，但内存已无空闲时，为了保证该进程能够正常运行，系统必须从内存中调出一页程序或数据送到磁盘的对换区中。但应调出哪个页面，需要根据一定的算法来确定，算法的好坏会直接影响到系统的性能。FIFO算法总是淘汰最先进入内存的页面，即选择在内存中驻留时间最久的页面予以淘汰。该算法实现简单，只需要把一个进程已调入内存的页面，按照先后次序连接成一个队列，并设置一个替换指针使它总指向最老的页面。由于FIFO所依据的条件是各个页面存入的时间，而页面调入的先后并不能反映页面的使用情况，所以FIFO算法的性能较差。同时通过这个实验我们也体会到了编程的思路流程，结构流程图的作用，一个程序如果一开始计划的好，结构设计完善，才能够顺利进行。

最重要的是在这次课程设计中不仅检验了我们所学习的知识，也培养我们如何去把握一件事情，如何去做一件事情，又如何去完成一件事情。设计的过程中也体现出自己单独设计模具的能力以及综合运用知识的能力，体会到了学以致用、突出自己劳动成果的喜悦心情，从中也发现了自己平时学习的不足和薄弱环节，能够很好的弥补自己的不足。

五、参考文献

[fifo算法\_FIFO深度的解释\_weixin\_39929096的博客-CSDN博客](https://blog.csdn.net/weixin_39929096/article/details/110654949?utm_medium=distribute.pc_relevant.none-task-blog-2~default~BlogCommendFromMachineLearnPai2~default-3.control&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2~default~BlogCommendFromMachineLearnPai2~default-3.control)

[深入理解FIFO（包含有FIFO深度的解释）\_后起的博客-CSDN博客\_fifo](https://blog.csdn.net/HouQi02/article/details/51683635?utm_medium=distribute.pc_relevant.none-task-blog-baidujs_title-1&spm=1001.2101.3001.4242)

邹恒明著．计算机的心智：操作系统之哲学原理．北京：机械工业出版社

薛智文编著．操作系统．北京：中国铁道出版社

六、代码

#include<iostream>

#include<stack>

using namespace std;

int main(){

int pageFrameNum; // 物理页框数

int pageFrontNum; // 页面走向个数

stack<int> pageDesert; // 淘汰页面

cout << "请输入分配给该作业的物理页框数：";

cin >> pageFrameNum;

cout << "请输入该作业的页面走向个数：";

cin >> pageFrontNum;

int pages[pageFrontNum]; // 页面走向

// c++中数组必须赋初值

for(int i = 0 ; i < pageFrontNum ; i++){

pages[i] = 0;

}

for(int i = 0 ; i < pageFrontNum ; i++){

cin >> pages[i]; // 获取页面走向数组

}

int pageFrame[pageFrameNum]; // 物理页框

// c++中数组必须赋初值

for(int i = 0 ; i < pageFrameNum ; i++){

pageFrame[i] = 0;

}

int pageMissNum = 0; // 缺页次数

int count = 0;

int helpNum = 0; // 实现FIFO算法

while (count < pageFrontNum)

{

cout << "第" << count+1 << "次：" << endl;

bool isMiss = true; // 判断本次是否缺页

bool isEmpty = true; // 判断物理页框是否有空位

bool isExist = false; // 判断物理页框中是否存在本次页面走向

// 判断物理页框中是否已经存在本次页面走向

for(int i = 0 ; i < pageFrameNum ; i++){

if(pages[count] == pageFrame[i]){

isExist = true;

break;

}

}

// 若本次页面走向,物理页框中已存在，则直接进入下次页面走向

if(isExist){

cout << "本次页面走向页框中已经存在！" << endl;

cout << "目前物理页框中页面走向为：";

for(int i = 0 ; i < pageFrameNum ; i++){

cout << pageFrame[i] << " ";

}

cout << endl;

count++;

continue;

}

// 判断物理页框有无空位

for(int i = 0 ; i < pageFrameNum ; i++){

if(pageFrame[i] == 0){

isEmpty = true;

break;

}else{

isEmpty = false;

}

}

// 本次页面走向，物理页框中不存在，且有空位，按顺序放入

if(!isExist && isEmpty){

for (int i = 0 ; i < pageFrameNum; i++){

if(pageFrame[i] == 0){

pageFrame[i] = pages[count];

break;

}

}

}

// 本次页面走向，物理页框中不存在，且物理页框中没有空位了

// 实现 FIFO 算法

if(!isExist && !isEmpty){

pageDesert.push(pageFrame[helpNum%pageFrameNum]); // 淘汰页面入栈

pageFrame[helpNum%pageFrameNum] = pages[count];

helpNum++;

}

// 计算缺页次数

if(isMiss == true){

pageMissNum++;

}

cout << "目前物理页框中页面走向为：";

for (int i = 0 ; i < pageFrameNum ; i++){

cout << pageFrame[i] << " ";

}

cout << endl;

count++;

}

cout << endl;

cout << "缺页次数：" << pageMissNum << "次" << endl;

cout << "一共调用：" << pageFrontNum << "次" << endl;

cout << "缺页中断率：" << pageMissNum\*1.0/pageFrontNum\*100 << "%" << endl;

cout << "淘汰页面：";

stack<int> helpstack; // 辅助栈，实现栈的逆置

while (!pageDesert.empty()){

helpstack.push(pageDesert.top());

pageDesert.pop();

}

while (!helpstack.empty()){

cout << helpstack.top() << " ";

helpstack.pop();

}

return 0;

}