《操作系统原理》实验报告

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 姓名 | MTX | 学号 | U2021 | 专业班级 | 信安 | 时间 | 2023.12.5 |

**一、实验目的**

（1）理解页面淘汰算法原理，编写程序演示页面淘汰算法。

（2）验证Linux虚拟地址转化为物理地址的机制

（3）理解和验证程序运行局部性的原理。

（4）理解和验证缺页处理的流程。

**二、实验内容**

（1）Win/Linux编写二维数组遍历程序，理解局部性的原理。

（2）Windows/Linux模拟实现OPT或FIFO或LRU淘汰算法。

（3）研读并修改Linux内核的缺页处理函数do\_no\_page 或页框分配函数 get\_free\_page，并用printk打印调试信息。注意：需要编译内核。建议优麒麟或麒麟系统。

（4）Linux下利用/proc/pid/pagemap技术计算某个变量或函数虚拟地址对应的物理地址等信息。建议优麒麟或麒麟系统。

**三、实验环境和核心代码**

**3.1 Windows下二维数组遍历缺页问题**

实验环境：windows 10.0.22000.2538

核心代码：

#include <stdio.h>#include <time.h>#include<windows.h>#define ROWS 2048#define COLS 2048#define TIMES 2int main() { int row = ROWS\*TIMES; int col = COLS\*TIMES; int array[row][col]; int i, j; clock\_t start\_time, end\_time; int time\_taken; int flag = 0; DWORD pid = GetCurrentProcessId(); printf("当前PID：%d\n",pid); while(flag!=3) { scanf("%d",&flag); // 根据 flag 设置遍历模式 if (flag == 1) { start\_time = clock(); // 遍历数组并访问元素 for (i = 0; i < row; i++) { for (j = 0; j < col; j++) { // 访问数组元素 array[i][j] = 1; } } end\_time = clock(); // 计算遍历时间 time\_taken = end\_time - start\_time; printf("行优先遍历时间：%d\n", time\_taken); } else if(flag==2) { start\_time = clock(); // 遍历数组并访问元素（不同的循环次序或数组大小等） for (j = 0; j < row; j++) { for (i = 0; i < col; i++) { // 访问数组元素 array[i][j] = 2; } } end\_time = clock(); // 计算遍历时间 time\_taken = end\_time - start\_time; printf("列优先遍历时间：%d\n", time\_taken); } } return 0;}

输入flag=1时，记录行优先遍历的时间；输入flag=2时，记录列优先遍历的时间。

**3.2 windows下实现OPT、FIFO和LRU淘汰算法**

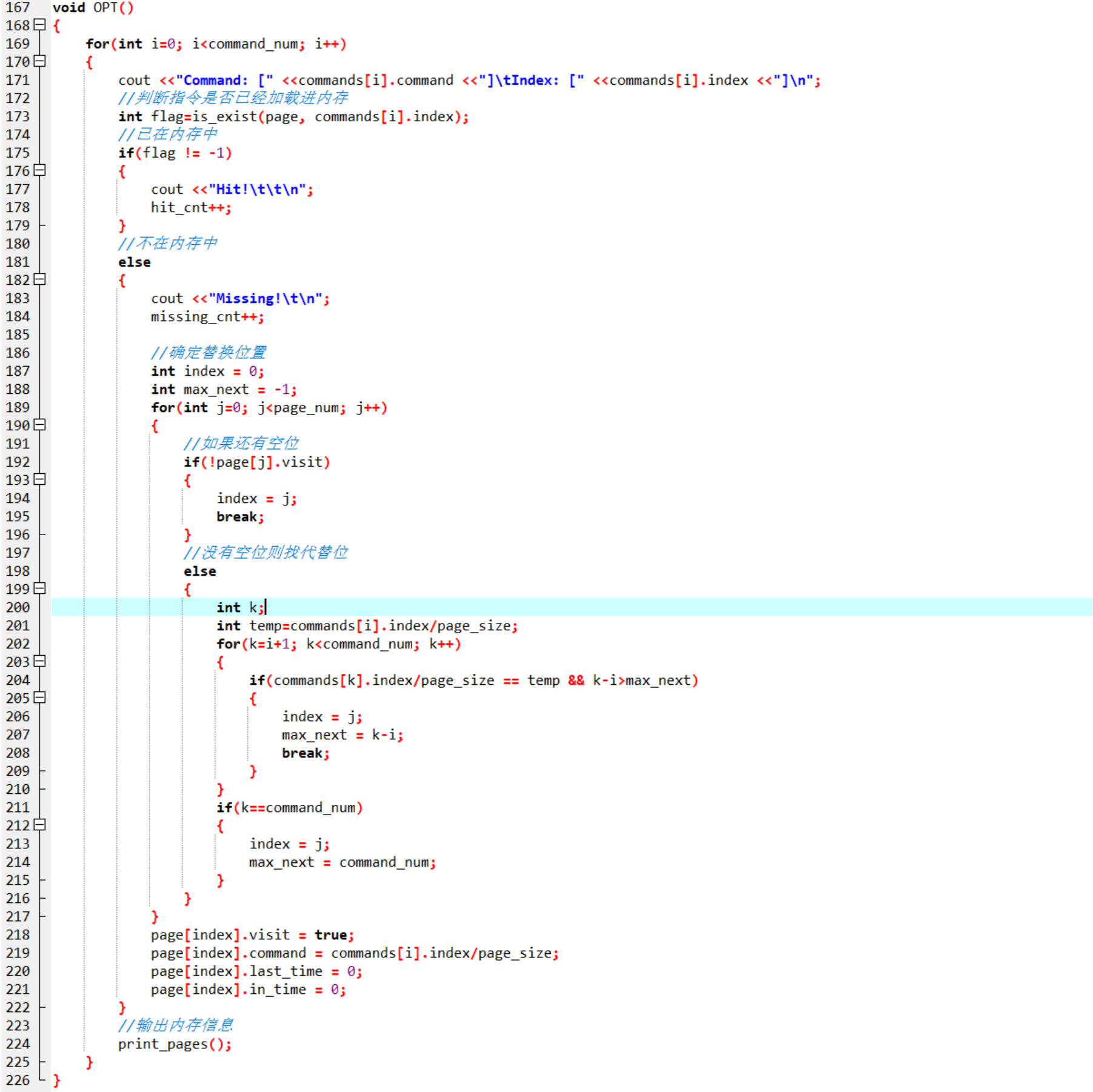
创建一个数组模拟内存，然后在内存中放置随机数模拟指令，指令是随机生成的。

实验环境：windows 10.0.22000.2538

核心代码：

1. OPT算法

判断指令是否已经加载进内存，通过调用 is\_exist() 函数进行判断。如果已在内存中，则命中计数 hit\_cnt 增加，并输出 "Hit!"。若不存在则首先检查是否存在未访问的空位，如果有，则将指令放入该空位；如果没有空位，则根据 OPT 算法的原理，在后续指令中搜索需要最长时间才会被访问的物理块，选择作为替换位置。更新被选择的替换位置的信息，包括标记为已访问、更新指令索引、重置最近访问时间和进入内存时间。



**图1 OPT核心代码**

1. FIFO算法

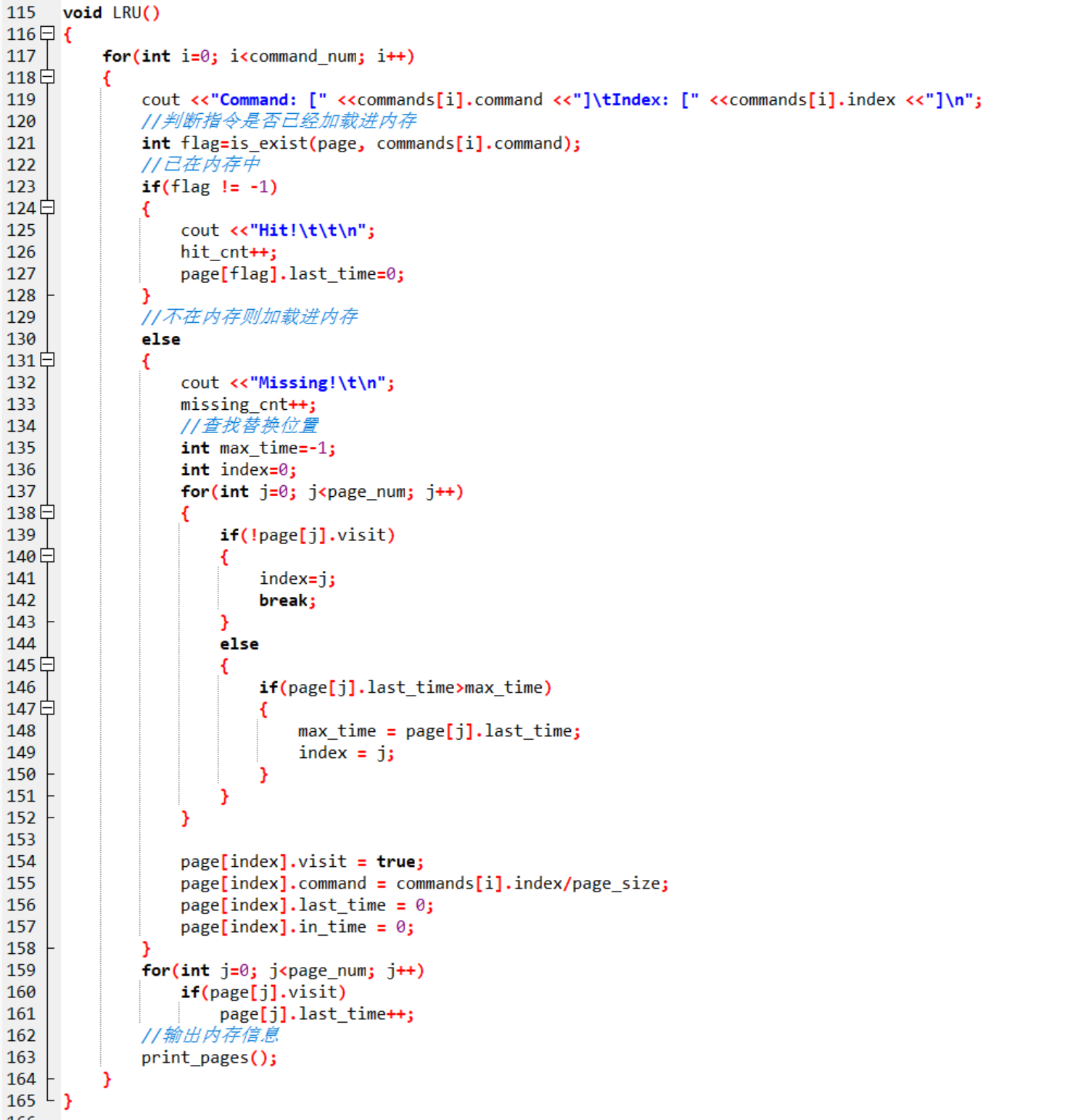
判断指令是否已经加载进内存，通过调用 is\_exist() 函数进行判断。如果已在内存中，则命中计数 hit\_cnt 增加，并输出 "Hit!"。如果指令不在内存中，则缺页计数 missing\_cnt 增加，并执行以下步骤：将指令加载进内存的下一个物理块，即 page[fifo\_index]。更新被选择的物理块的信息，包括标记为已访问、更新指令索引。更新 FIFO 索引 fifo\_index，使其指向下一个物理块（循环队列的形式）。



**图2 FIFO核心代码**

1. LRU算法

判断指令是否已经加载进内存，通过调用 is\_exist() 函数进行判断。如果已在内存中，则命中计数 hit\_cnt 增加，并输出 "Hit!"。同时将命中的物理块的最近访问时间 last\_time 设置为0，表示最近使用。如果指令不在内存中，则缺页计数 missing\_cnt 增加，并执行以下步骤：查找替换位置，即选择最久未被访问的物理块作为替换位置。遍历物理块数组，找到最大的 last\_time 值对应的物理块。更新被选择的物理块的信息，包括标记为已访问、更新指令索引、将最近访问时间 last\_time 设置为0。遍历物理块数组，对所有已访问的物理块的最近访问时间 last\_time 进行加一操作，表示经过一轮指令后的时间增加。



**图3 LRU核心代码**

**3.4 pagemap计算物理地址**

Linux中的/proc/self/pagemap文件可以查看当前进程的虚拟页的物理地址。每个记录8字节，最高位记录了当前虚拟页是否再内存中，1表示在，0表示不在。0到54位记录虚拟页的物理页号。代码输出了一个全局变量add和自定义函数add的物理地址信息（Get\_physical\_address函数打印出虚拟地址、虚拟页号、虚拟偏移、读取的记录、物理页号和物理地址）。

实验环境：Ubuntu 20.04.6 LTS，GCC 9.4.0

核心代码：

#include <stdio.h>

#include <unistd.h>

#include <stdint.h>

#include <stdlib.h>

#include <inttypes.h>

#include <string.h>

int test = 0; //用作输出的全局变量

int add(int a, int b) //用作输出的函数

{

return a+b;

}

int Get\_physical\_address(unsigned long va)

{

unsigned long page\_num;

unsigned long offset;

int page\_size;

page\_size = getpagesize();

page\_num = va/page\_size;

offset = va%page\_size;

printf("virtual address:\t 0x%016lx\n",va);

printf("virtual page\_num:\t 0x%016lx\n",page\_num);

printf("virtual offset:\t 0x%016lx\n",offset);

FILE\* fp;

if((fp=fopen("/proc/self/pagemap", "rb")) == NULL) {

printf("Open /proc/self/pagemap Error.\n");

return 1;

}

//每项记录为8字节，找到当前页号对应的记录

unsigned long fileOffset = page\_num \* sizeof(uint64\_t);

if(fseek(fp,fileOffset,SEEK\_SET)!=0) {

printf("fSeek Error.\n");

return 1;

}

uint64\_t it;

//读取记录

if(fread(&it,sizeof(uint64\_t),1,fp) != 1) {

printf("fread Error.\n");

return 1;

}

fclose(fp);

printf("item:\t\t 0x%016lx\n", it);

//记录的第63位记录当前页面位置：1为在物理内存中，0表示不在物理内存中

if((it >> 63) & 1 == 0) {

printf("Page Present is 0.\nThe present page isn't in the Physical Memory.\n");

return 1;

}

//记录的0-54位位物理页号

uint64\_t phyPageIdx = (((uint64\_t)1 << 55) - 1) & it;

printf("Physical page\_num:\t 0x%016lx\n",phyPageIdx);

//物理地址=物理页号\*页大小+页内偏移

unsigned long pa = phyPageIdx \* page\_size + offset;

printf("Physical address:\t 0x%016lx\n\n", pa);

return 0;

}

int main()

{

void \* virtual\_address;

printf("variable test:\n");

virtual\_address = &test;

Get\_physical\_address((unsigned long)virtual\_address);

printf("\nFunction add:\n");

virtual\_address = (void \*)add;

Get\_physical\_address((unsigned long)virtual\_address);

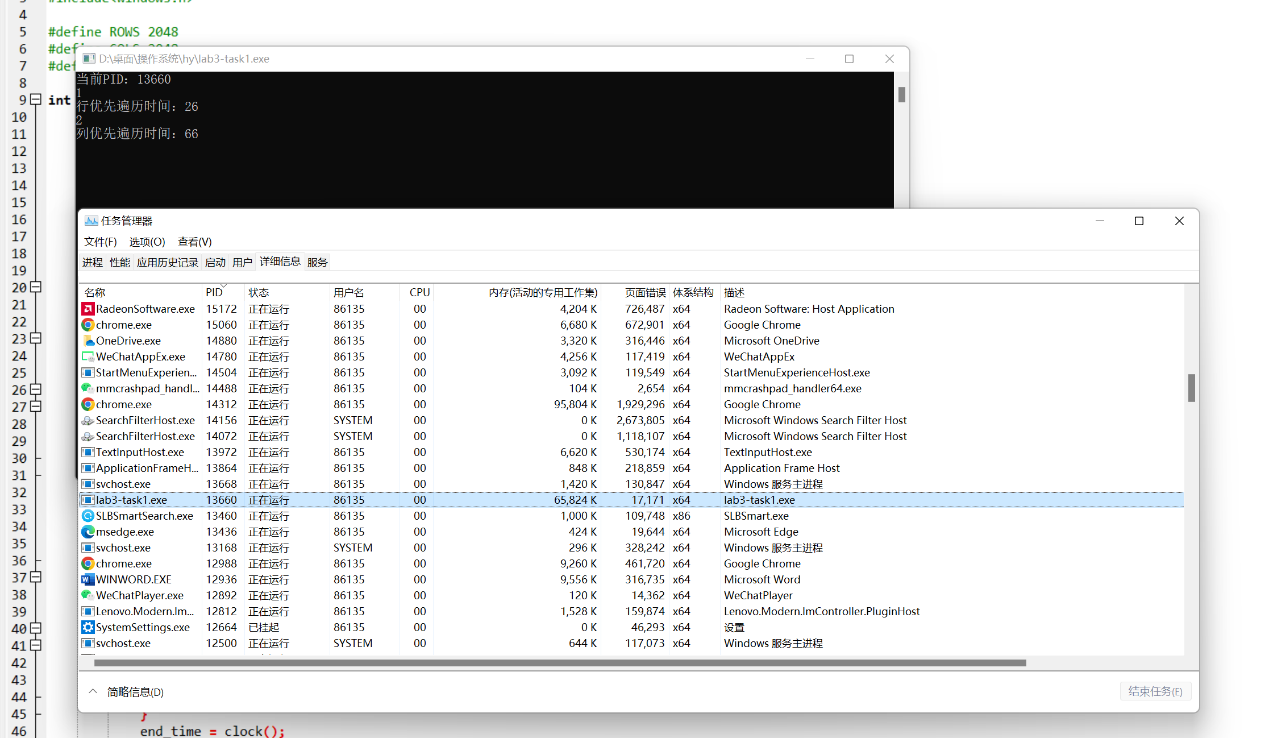
return 0;

}

**四、实验结果**

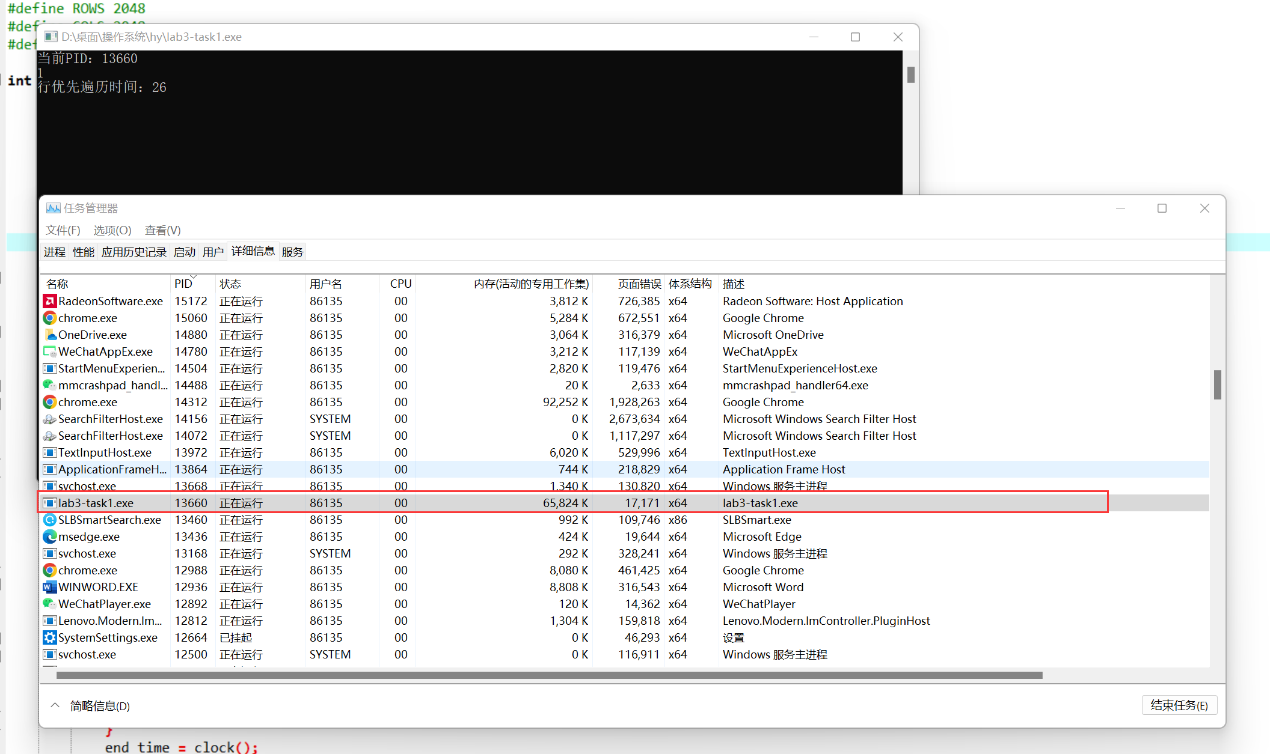
**4.1 Windows下二维数组遍历缺页问题**

Flag为1时，打印行优先遍历时间结果，单位为CLOCKS\_PER\_SE，flag为2时，打印列优先遍历结果，如下图4可以明显看出，列优先遍历耗费的时间比行优先遍历耗费的时间多很多。

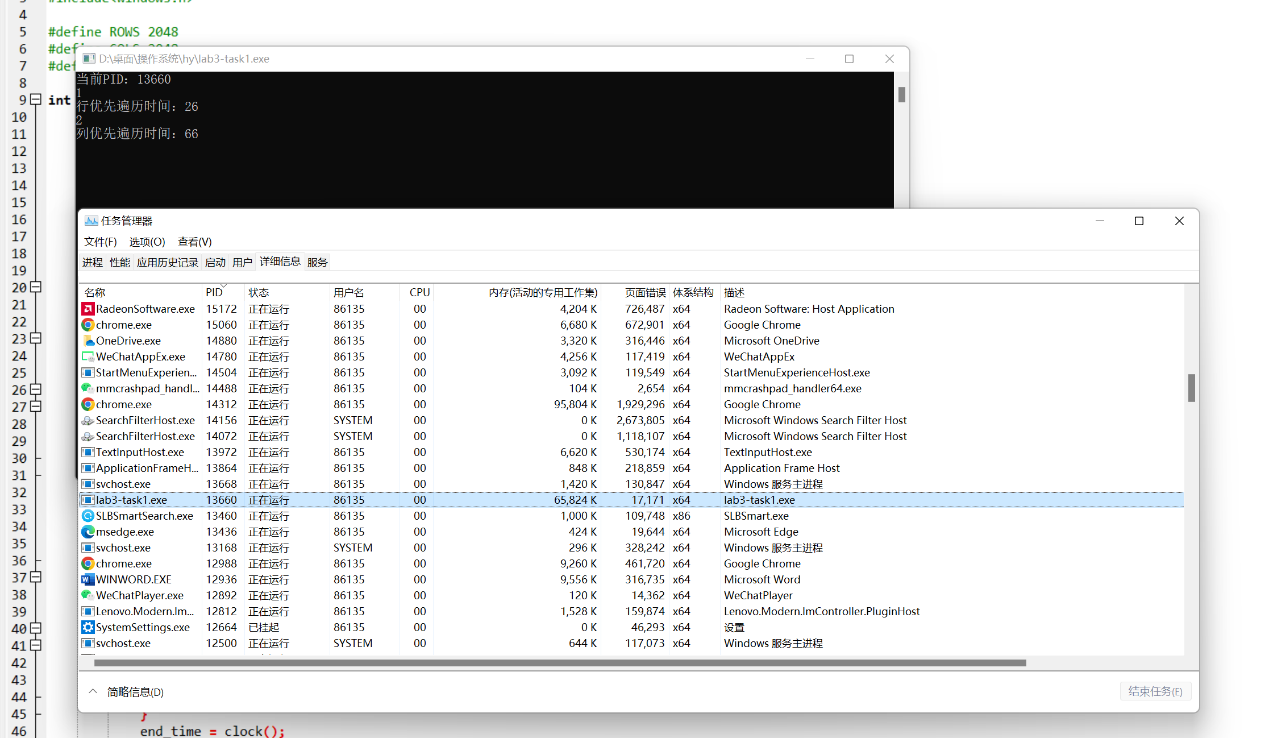


**图4 不同遍历方法时间示意图**

下图5、6分别为列优先遍历和行优先遍历的缺页次数，可以看到，结果相差不大。



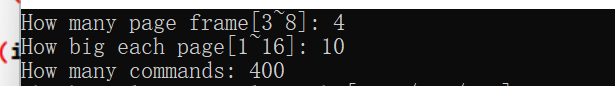
**图5 列行优先遍历缺页次数**



**图6 列优先遍历缺页次数**

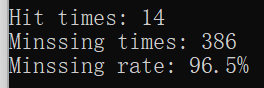
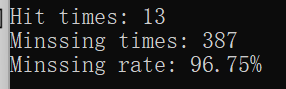
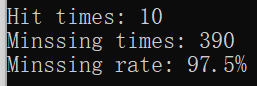
**4.2 windows下实现OPT、FIFO和LRU淘汰算法**

**为控制变量，更客观查看三种算法的运行结果，测试时page frame=4，page size=10，commands=400，设为定值，如图7。**



**图7 控制变量的参数**

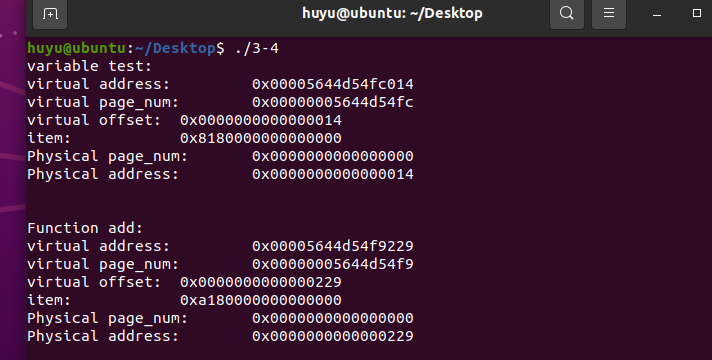
**FIFO，LRU，OPT测试结果分别如下图8、9、10所示，可以看出，即使数据量较小，页框数不多，OPT算法还是略胜一筹。**



**图8 FIFO 图 9 LRU 图10 OPT**

**4.4 pagemap计算物理地址**

如图11，Get\_physical\_address函数打印出全局变量test和自定义函数add的虚拟地址、虚拟页号、虚拟偏移、读取的记录、物理页号和物理地址。测试结果良好！

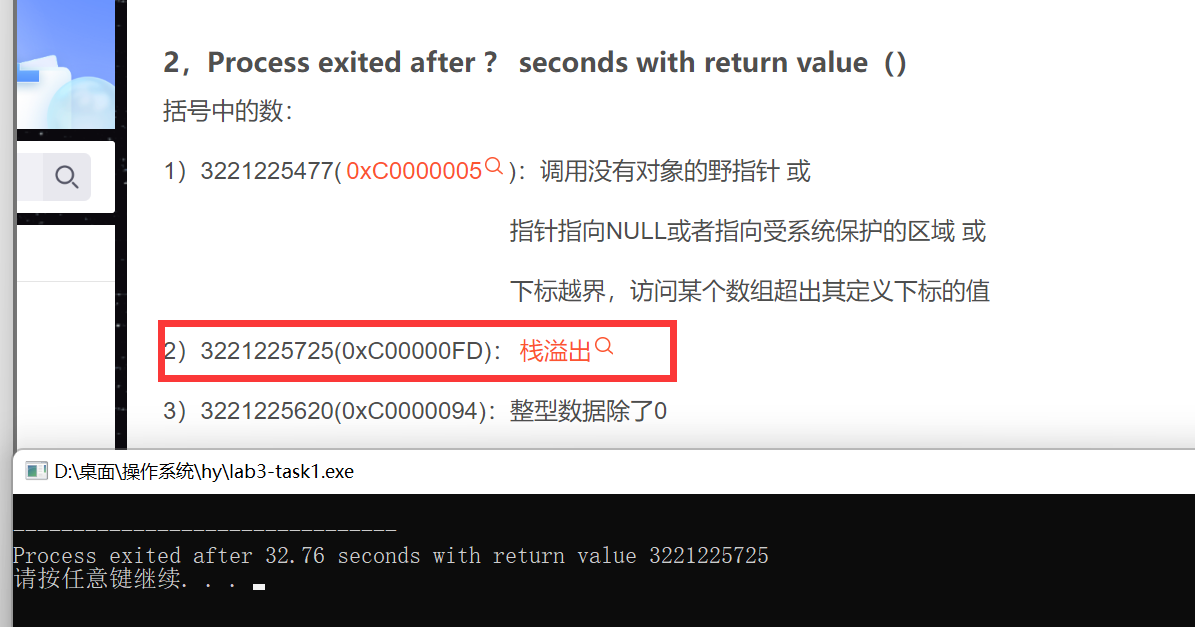


**图11 测试结果示意图**

# 五、实验错误排查和解决方法

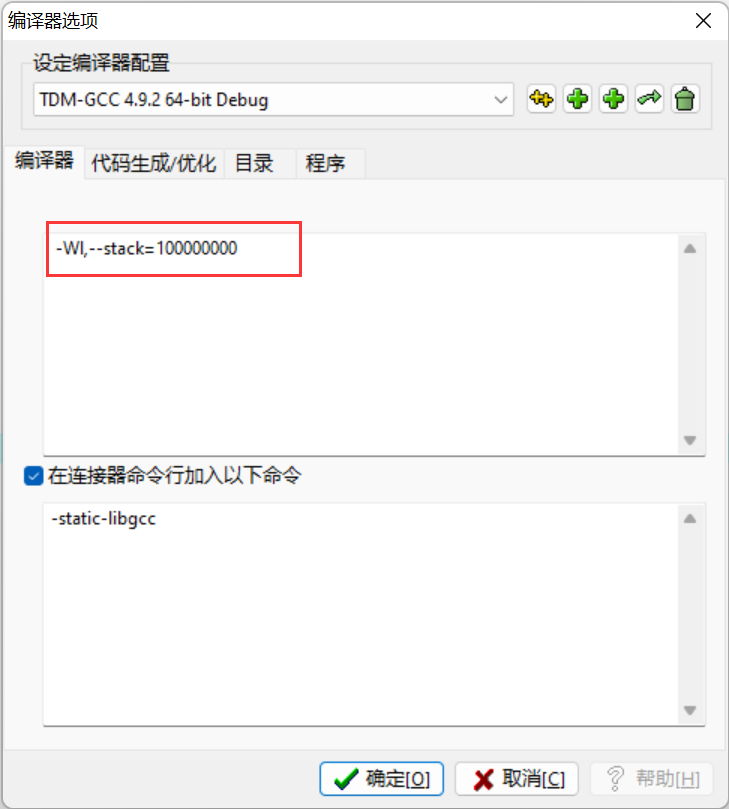
**5.1 Windows下二维数组遍历缺页问题**

在编译运行的时候，编译会报错，如下图12，网上查找资料后得知，可能是数组开的太大了，默认的栈大小不够。



**图12 栈溢出报错示意图**

于是在编译时加上命令-Wl,--stack=100000000，手动规定栈大小，即可，如下图13。



**图13 编译命令添加示意图**

**5.2 windows下实现OPT、FIFO和LRU淘汰算法**

对于OPT算法的实现，不知道如何记录页面访问的情况，在网上查找资料，借用了别人的思路。其他就没有什么难点了，毕竟上课老师算法讲的很清楚了，而且在计算机组成原理中也学过，代码实现不难。

总结一下：

OPT算法假设我们能够预测未来的页面访问情况，并选择在未来最长时间内不再使用的页面进行置换。LRU算法维护一个页面访问历史记录，当需要置换页面时，选择最久未被访问的页面进行置换。FIFO算法维护一个页面队列，当需要置换页面时，选择最早进入队列的页面进行置换。

**5.4 pagemap计算物理地址**

PPT中对pagemap的描述很少，通过百度的方式分析出具体64位的信息

每个虚拟页的64位值是/proc/pid/pagemap文件中的一个条目，它包含有关该虚拟页的信息。具体而言，每个64位条目的位表示如下：位0-54：物理页框号（Physical Page Frame Number，PPFN），指示虚拟页所映射的物理页的编号。位55：保留位，当前未使用。位56：软脏位（Soft Dirty Bit），指示虚拟页的内容是否已被修改。位57：页迁移位（Page Migration Bit），指示虚拟页是否已被迁移。位58：页脏位（Page Dirty Bit），指示虚拟页的内容是否已被修改。位59：保留位，当前未使用。位60：保留位，当前未使用。位61：保留位，当前未使用。位62：保留位，当前未使用。位63：存在位（Page Present Bit），指示虚拟页是否存在于物理内存中。而我们实际要用到的只有第63位和0-54位，当判断虚拟页存在于物理内存之后，只需要根据公式：物理地址=物理页号\*页大小+页内偏移。即可算出物理地址。

# 六、实验参考资料和网址

<https://blog.csdn.net/weixin_43482279/article/details/105979591>

https://blog.csdn.net/weixin\_34191734/article/details/86122595