

《操作系统原理》实验报告

姓名	鄢湧棚	学号	U201911741	专业班级	网安 1902	时间	2022.1.6
----	-----	----	------------	------	---------	----	----------

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。

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 和 LRU 等淘汰算法。
- (3) 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。

三、实验过程

- (1) Windows 下二维数组遍历缺页问题

代码编写没什么难度，如下：

```
#include<iostream>
#include<ctime>
using namespace std;
const int times = 5;
const int row = 1024*times;
const int column = 1024*times;
int Data[row][column];
int main()
{
```

```

int i, j;
clock_t start, end;
int flag = 2;
if(flag == 1){
    cout<<"row first:";
    start = clock();
    for(i=0;i<row;i++){
        for(j=0;j<column;j++){
            Data[i][j] = 1;
        }
    }
    end = clock();
    cout<< end-start <<endl;
}
else{
    start = clock();
    cout<<endl<<"column first:";
    for(i=0;i<column;i++){
        for(j=0;j<row;j++){
            Data[j][i] = 1;
        }
    }
    end = clock();
    cout<<end - start <<endl;
}
system("pause");
return 0;
}

```

分别修改 flag 为 1 和 2，计算行优先遍历和列优先遍历耗费的时间。

(2) Windows 下模拟 OPT 和 FIFO 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。

通过地址生成函数,通过三种模式生成访问该模拟内存的地址序列(随机,顺序,重复)。

设置页面大小为 16, 页框数为 3, 一共访问 20 次。

OPT 算法:

该算法的思想是淘汰以后不再使用或者最远的将来才会使用的页面。一般来说在实际中无法实现,但是在该模拟中可以,建立一个表,逆序遍历要访问的内存地址,计算对应的页号,若不在表中,则加入,并将序号填入对应的一个栈中。

每次进行指令访问时,将对应页号的访问顺序出栈,当需要换页时,如果表中页号对应的栈为空,则表示之后不会再使用到它,则直接选择该页面淘汰,否则选择栈顶元素最大的栈进行淘汰。

代码如下:

```
void OPT() {
    int visit_page[max_visit_time]; //将要访问的所有页
    int i, j, remain;
    int priority;
    int lost_time = 0;
    for (i = 0; i < max_visit_time; i++)
        visit_page[i] = get_page_num(visit_order[i]);

    //待用信息表: 键名为待访问的页号, 键值为一个存有该页号访问次序的堆栈
    map<int, stack<int>> ms;
    //逆序变量访问序列
    for (int i = max_visit_time - 1; i >= 0; --i) {
        //若该页未被访问
        if (ms.count(visit_page[i]) == 0) {
            stack<int> tmp; //创建堆栈
            tmp.push(i); //在堆栈中添加访问次序
            ms.insert(pair<int, stack<int>>(visit_page[i], tmp));
        }
        else {
            ms.at(visit_page[i]).push(i); //在堆栈中添加访问次序
        }
    }

    //开始执行指令
    for(i=0;i<max_visit_time;i++){
        cout<<"当前访问序号: "<<visit_order[i]
        <<"\t 所在页: "<<visit_page[i]
        <<"\t 值: "<<proc[visit_order[i]]<<endl;

        if(is_in_page(visit_page[i])){
```

```

        cout<<"命中"<<endl;
        print_page();
    }
    else{
        cout<<"未命中"<<endl;
        lost_time++;
        remain = 0;
        for(j = 0;j<page_cnt;j++){ //若有空的页框则存入空页框中
            if(now_page[i] == -1){
                push_page(visit_page[i], j);
                remain = 1;
                break;
            }
        }
        if(remain) continue;
        //执行换页
        priority = now_page[0];
        for(j = 0;j<page_cnt;j++){
            //之后不再访问
            if(ms[now_page[j]].size() == 0){
                priority = j;
                break;
            }
            //若当前页框中的要更之后才访问，则提高优先级
            if(ms[now_page[j]].top() > ms[priority].top()){
                priority = now_page[j];
            }
        }
        push_page(visit_page[i], priority);
        print_page();
    }
}

cout<<"finish!"<<endl;
cout<<" 缺 页 次 数 : \t"<<lost_time<<"\t"<<" 缺 页 率 : \t"<<((lost_time*1.0)/(max_visit_time*1.0))<<endl;
}

```

FIFO 算法:

这个算法比较简单，按顺序填入页框，满了之后按顺序替换就可以。

代码如下:

```

void FIFO()
{
    int visit_page[max_visit_time]; //将要访问的所有页
    int i, j, remain;
}

```

```

int last_page = 0;
int lost_time = 0;
for (i = 0; i < max_visit_time; i++)
    visit_page[i] = get_page_num(visit_order[i]);
for(i=0;i<max_visit_time;i++){
    cout<<"当前访问序号: "<<visit_order[i]
    <<"\t 所在页: "<<visit_page[i]
    <<"\t 值: "<<proc[visit_order[i]]<<endl;
    if(is_in_page(visit_page[i])){
        cout<<"命中"<<endl;
        print_page();
    }
    else{
        cout<<"未命中"<<endl;
        lost_time++;
        last_page++;
        last_page%=page_cnt;
        remain = 0;
        for(j = 0;j<page_cnt;j++){ //若有空的页框则存入空页框中
            if(now_page[i] == -1){
                push_page(visit_page[i], j);
                remain = 1;
                break;
            }
        }
        if(remain) continue;
        //换页
        push_page(visit_page[i], last_page);
        print_page();
        cout<<"finish!"<<endl;
        cout<<" 缺 页 次 数 : \t"<<lost_time<<"\t"<<" 缺 页 率 : \t"<<((lost_time*1.0)/(max_visit_time*1.0))<<endl;
    }
}
}
}

```

(3) 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在内存中，1 表示在，0 表示不在。0 到 54 位记录虚拟页的物理页号。

代码如下：

```
#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);

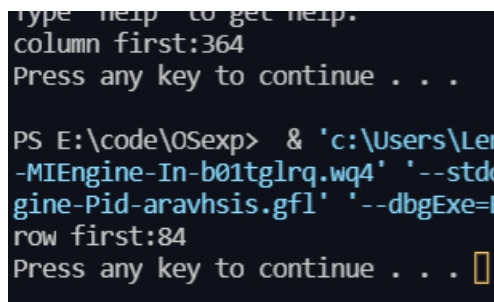
        printf("\nFunction add:\n");
        virtual_address = (void *)add;
        Get_physical_address((unsigned long)virtual_address);
        return 0;
    }

```

四、 实验结果

(1) Windows 下二维数组遍历缺页问题

以下为运行结果，可以看出，列优先遍历耗费的时间比行优先遍历耗费的时间多很多。



```

type help to get help.
column first:364
Press any key to continue . . .

PS E:\code\OSexp> & 'c:\Users\Len
-MIEngine-In-b01tqlrq.wq4' '--std
gine-Pid-aravhsis.gfl' '--dbgExe=
row first:84
Press any key to continue . . .

```

下图为列优先遍历的缺页次数

名称	PID	状态	用户名	CPU	内存(活动...	页面错误
exp3.1.exe	13508	正在运行	yyp	00	102,892 K	26,621

下图为行优先遍历的缺页次数

名称	PID	状态	用户名	CPU	内存(活动...	页面错误
exp3.1.exe	14612	正在运行	yyp	00	102,892 K	26,622

可以看到，相差不大。

(2) Windows 下模拟 OPT 和 FIFO 淘汰算法

OPT 算法随机访问的统计结果

```
finish!
缺页次数:      18      缺页率:      0.9
PS E:\code\OSexp> & 'c:\Users\Lenovo\.vscode\
```

FIFO 算法随机访问的统计结果

```
finish!
缺页次数:      19      缺页率:      0.95
PS E:\code\OSexp> █
```

可以看出，虽然页框少，不明显，但是 OPT 算法还是有优越性的。

(3) pagemap 计算物理地址

运行结果如下，可以得到各自的地址信息。

```
variable test:
virtual address:      0x000055d617164014
virtual page_num:      0x000000055d617164
virtual offset: 0x0000000000000014
item:      0x8180000000000000
Physical page_num:      0x0000000000000000
Physical address:      0x0000000000000014

Function add:
virtual address:      0x000055d617161229
virtual page_num:      0x000000055d617161
virtual offset: 0x0000000000000029
item:      0xa180000000000000
Physical page_num:      0x0000000000000000
Physical address:      0x0000000000000029
```

五、实验错误排查和解决方法

(1) Windows 下二维数组遍历缺页问题

不知道页面错误是不是缺页次数，后来老师说是。

(2) Windows 下模拟 OPT 和 LRU 淘汰算法

对于 OPT 算法的实现，不知道应该以怎样的形式来记录页面使用情况，百度参考了别人的实现方法。

(3) pagemap 计算物理地址

单看 PPT 对 pagemap 中的内容不是很明白，然后自己百度了解了更多

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