《操作系统原理》实验报告

姓名	侯竣	学号	U202116003	专业班级	密码 2101 班	时间	2023.12.16

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。

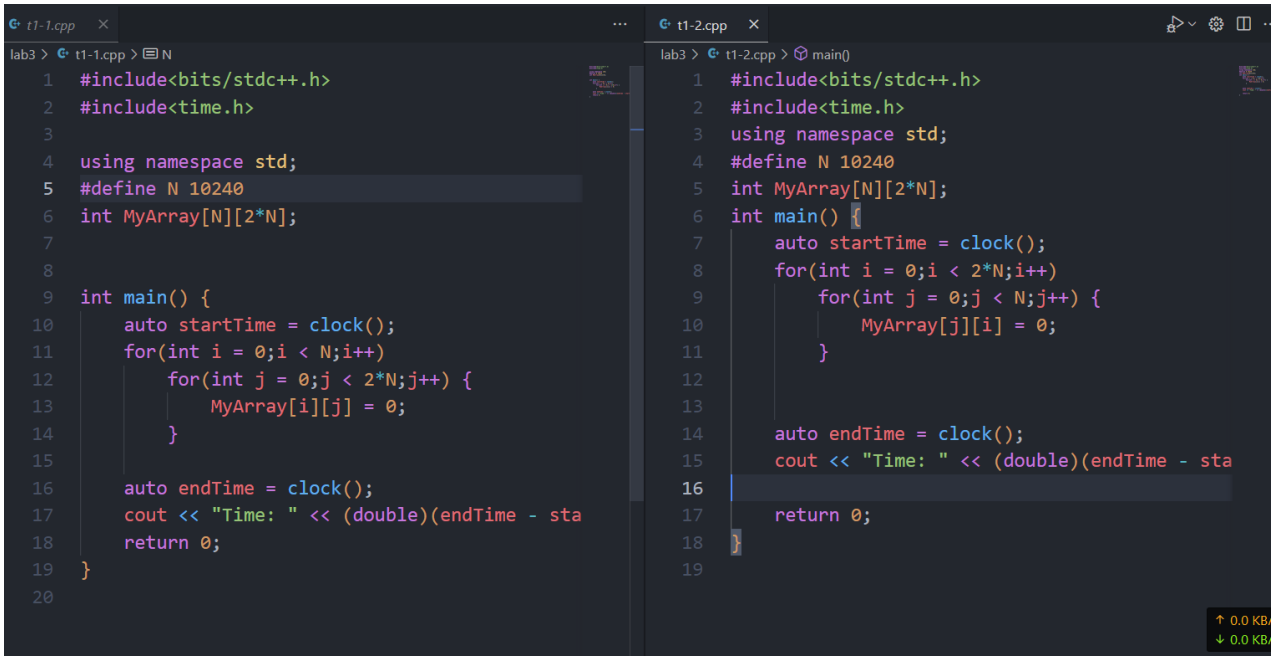
（3）研读并修改 Linux 内核的缺页处理函数 do_no_page 或页框分配函数 get_free_page，并用 printk 打印调试信息。注意：需要编译内核。建议优麒麟或麒麟系统。

（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

所有环境均为优麒麟 20.04，内核版本是 5.15，编译工具 gcc11

3.1 编写二维数组遍历程序，理解局部性的原理



```cpp
#include<bits/stdc++.h>
#include<time.h>

using namespace std;
#define N 10240
int MyArray[N][2*N];


int main() {
    auto startTime = clock();
    for(int i = 0;i < N;i++)
        for(int j = 0;j < 2*N;j++) {
            MyArray[i][j] = 0;
        }

    auto endTime = clock();
    cout << "Time: " << (double)(endTime - sta
    return 0;
}
```

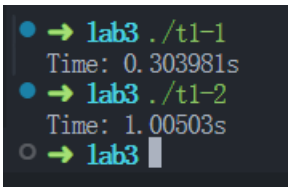
```cpp
#include<bits/stdc++.h>
#include<time.h>
using namespace std;
#define N 10240
int MyArray[N][2*N];
int main() {
    auto startTime = clock();
    for(int i = 0;i < 2*N;i++)
        for(int j = 0;j < N;j++) {
            MyArray[j][i] = 0;
        }


    auto endTime = clock();
    cout << "Time: " << (double)(endTime - sta

    return 0;
}
```

图 3-1 实验 1 代码

代码如图，通过 clock()计时可以输出程序运行时间，t1-1.cpp 是局部性好，t1-2.cpp 局部性差. 通过运行结果也可以证明这一点



```
➜ lab3 ./t1-1
Time: 0.303981s
➜ lab3 ./t1-2
Time: 1.00503s
➜ lab3
```

图 3-2 实验 1 结果

3.2 模拟实现 OPT 或 FIFO 或 LRU 淘汰算法。

我在这个实验主要实现了 OPT 和 LRU 算法，FIFO 算法比较简单就没有实现

OPT 算法的实现思路是根据已经给定的访问序列，通过便利将来可能访问的页面，找到将来不使用或者最远使用的那个页面，将其淘汰，并输出每次的页框情况

核心代码如图:

```
    }
    //若未命中
    if (j == cnt) {
        // cout << "缺页 ";
        cout << "cache miss ";
        ++pageMissCnt;  //缺页次数+1
        //若页框已全占满
        if (cnt == pageFrameCnt) {
            auto maxT = 0;
            //遍历页框根据待用信息寻找不在需要或最远的将来才用的页面
            for (int k = 0; k < pageFrameCnt; ++k) {
                if (ms.at(pageIdx[k]).size() == 0) {
                    maxT = k;
                    break;
                }
                else if (ms.at(pageIdx[k]).top() > ms.at(pageIdx[maxT]).top()) {
                    maxT = k;
                }
            }
            //淘汰页面复制新数据
            copyFromTo(maxT, pageNo);
            cout << pageFrame[maxT][offset] << endl;
        }
        //页框未全部占满则直接将页复制到空页框
        else {
            copyFromTo(cnt, pageNo);
            cout << pageFrame[cnt][offset] << endl;
            ++cnt;
        }
    }
    //输出每次的页框信息
```

图 3-3 OPT 核心代码

LRU 算法比较难一点，其思路是为每个页面维护一个未使用时间:

1. 当页面命中时，未使用时间置为 0

2. 其他未命中的页面则时间+1

当需要淘汰页面的时候，遍历所有的页面，找到未使用时间最长的页面进行淘汰

每次输出页框情况方便调试

核心代码如图:

```
        // 循环遍历页框若命中，则将该页的未使用时间置0
        for (j = 0; j < cnt; ++j) {
            if (pageIdx[j] == pageNo) {
                cout << "cache hit " << pageFrame[j][offset] << endl;
                timer[j] = 0;
                break;
            }
        }
        // 若未命中
        if (j == cnt) {
            cout << "cache miss ";
            ++pageMissCnt;  // 缺页次数+1

            // 若页框已全占满
            if (cnt == pageFrameCnt) {
                auto maxT = 0;

                //找到未使用时间最长的页框进行淘汰
                for (int k = 0; k < pageFrameCnt; ++k) {
                    if (timer[k] > timer[maxT]) maxT = k;
                }
                copyFromTo(maxT, pageNo);
                timer[maxT] = 0;
                cout << pageFrame[maxT][offset] << endl;
            } else {
                // 页框未全部占满则直接将页复制到空页框
                copyFromTo(cnt, pageNo);
                cout << pageFrame[cnt][offset] << endl;
                ++cnt;
            }
        }
        for (int j = 0; j < cnt; ++j) ++timer[j];
        showPageFrame();
    }
```

图 3-4 LRU 核心代码

3.3 计算某个变量或函数虚拟地址对应的物理地址等信息。

这个实验主要是对/proc/pid/pagemap 文件的内容的解析，设计了一个通用函数:

void getAddressInfo(char* str, UL pid, UL viraddress, UL* phyaddress)

str 主要是用来标识打印的时候的名字, pid 是进程号，剩下的是虚拟地址和物理地址

核心代码并不长，如图:

```
int fd = open(buf, O_RDONLY); // 只读打开
lseek(fd, vir_offset, SEEK_SET); // 游标移动
read(fd, &temp, sizeof(U64)); // 读取对应项的值

U64 phy_pageIndex = (((U64)1 << 55) - 1) & temp; // 物理页号
*phyaddress = (phy_pageIndex * pageSize) + page_offset; // 加页内偏移量得物理地址
```

图 3-5 实验 4 核心代码

通过打开 pagemap 文件，读取相应的地址赋给变量即可

四、实验结果

3.1 编写二维数组遍历程序，理解局部性的原理

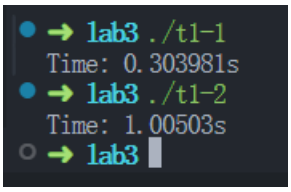


图 3-6 局部性结果

如图可见，局部性差的代码需要 1s 执行，而局部性好的代码只需要 0.3s

3.2 模拟实现 OPT 或 FIFO 或 LRU 淘汰算法。

OPT 算法是取未来最晚使用时间，如图，举例说明

定义的访问序列是{

456,499,45,2002,2001,2002,119,112,113,455,2001,600,601,

89,119,489,490,499,489,1000,1101,89,119,489,114,115,

1100,1101,1102,2189,2002,210 }

当到如图位置的时候，也就是需要 784 值，通过访问序列得到最晚使用的页面是

page=37 的页面，因此淘汰这个页面

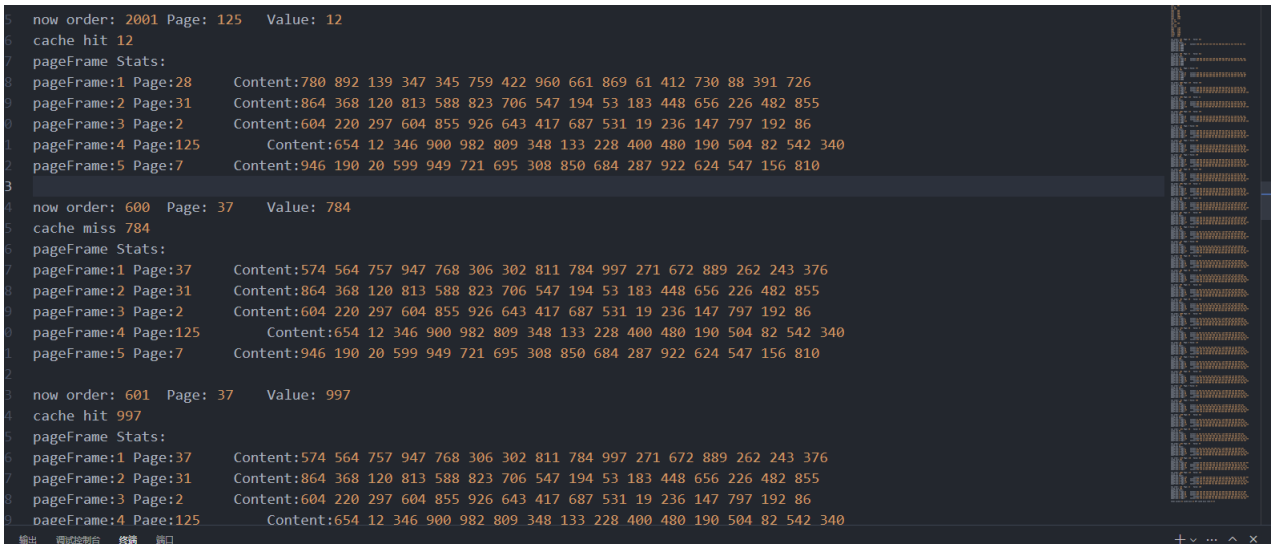


图 3-7 OPT 结果

LRU 运行结果如图，举一例分析：访问 page=30 的 857 时发生了 cache miss 缺页，由右边的 Timer 可知 Page Frame1 未使用时间最久，因此淘汰 page=28 的页面，符合 LRU 算法的规则，实验成功!

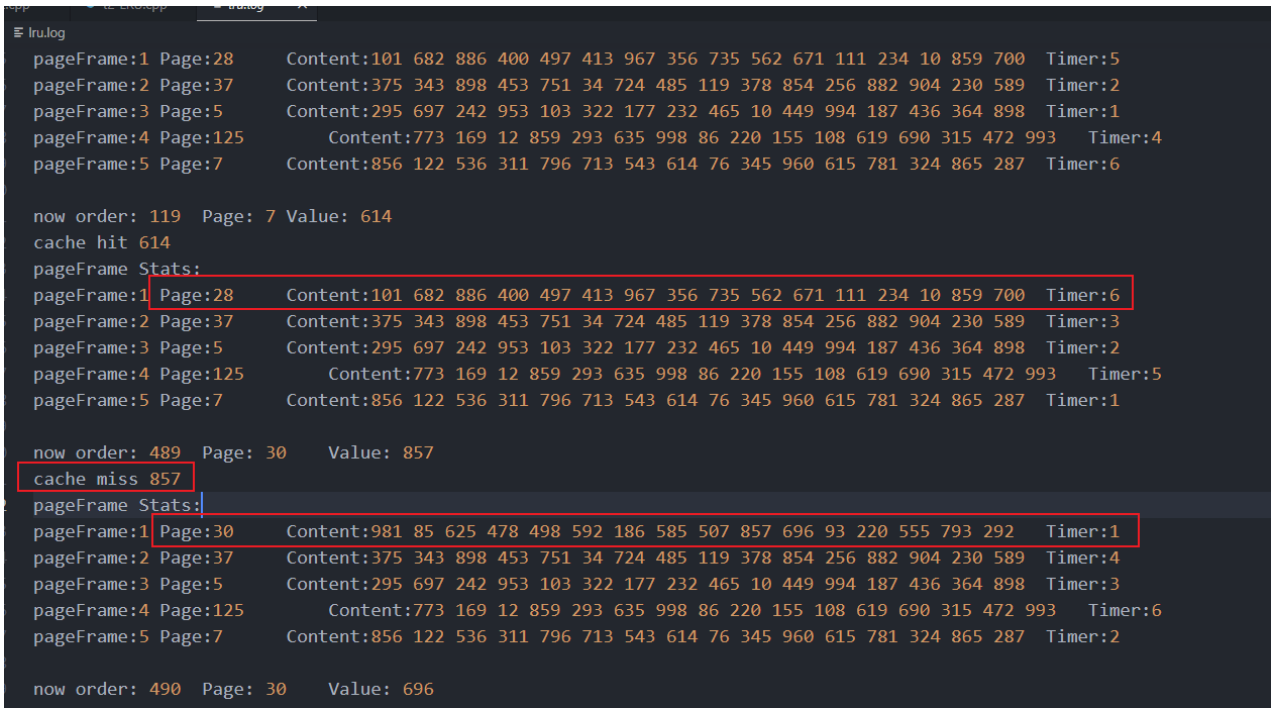


图 3-8 LRU 结果

最后，通过输出的缺页率，可知 OPT 的确优于 LRU 缺页率更少，这也是符合理论的

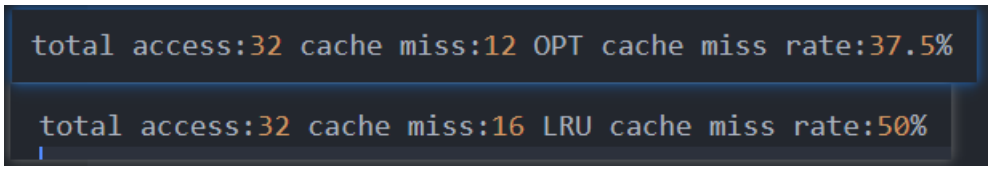


图 3-9 缺页率结果

3.3 计算某个变量或函数虚拟地址对应的物理地址等信息。

在这个实验我获取了几种不同的变量，分别是局部变量 b，常量 d，全局变量 phy

```
UL phy = 4;
int main()
{
    int b = 1;
    const int d = 3;
    int pid = fork();

    getAddressInfo("int b", getpid(), (UL)&b, &phy);

    getAddressInfo("const int d", getpid(), (UL)&d, &phy);

    getAddressInfo("Global UL phy = 4", getpid(), (UL)&a, &phy);

    return 0;
}
```

图 3-10 变量

结果如图:

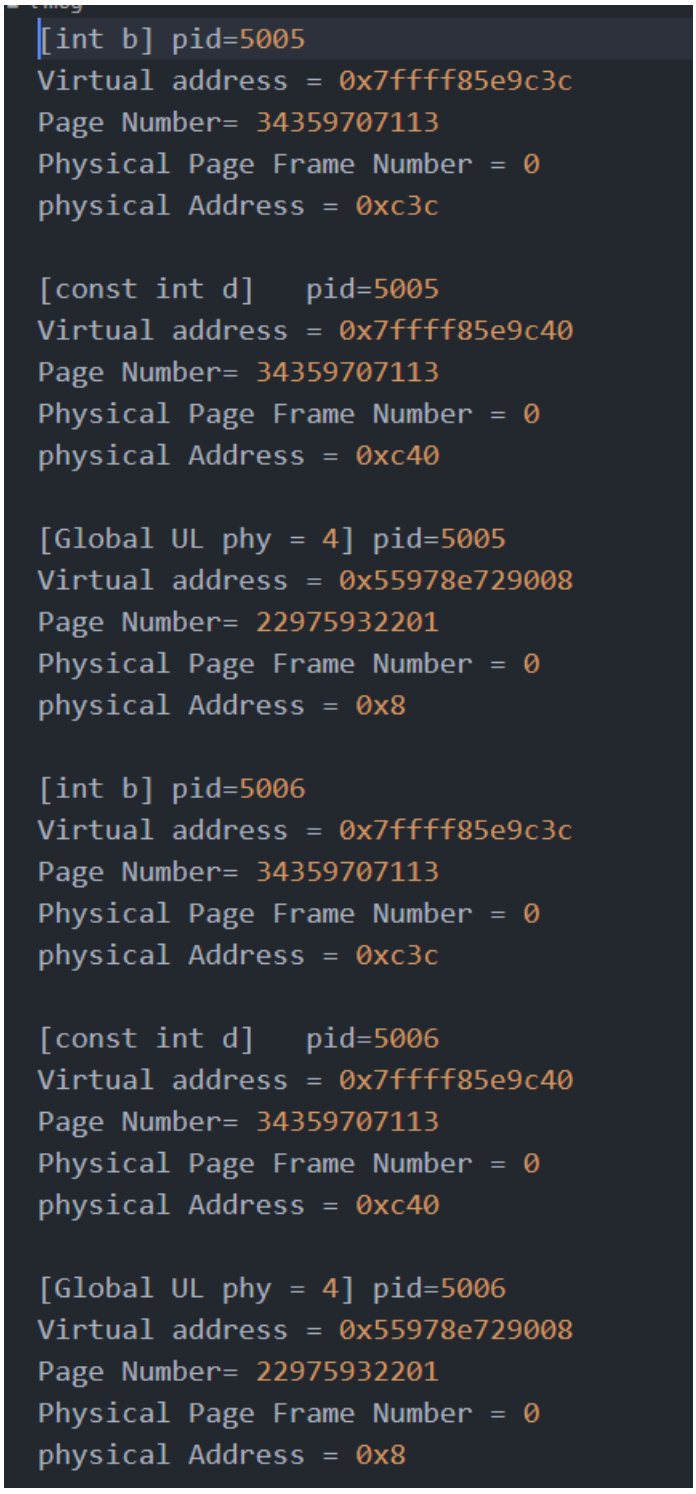


图 3-11 地址

通过结果可以看出几个有意思的地方，局部变量 b 和常量 d 的虚拟地址地址是相邻的，说明这两个都在栈上，而全局变量 phy 并不相邻，说明是在堆上，这也和 c 语言的内存布局的理论知识相符合

五、实验错误排查和解决方法

3.1 编写二维数组遍历程序，理解局部性的原理

这个实验比较简单，主要是体会一下局部性原理，在 10240 的情况下局部性快了大约 3 倍，非常可观，不过当我再调大的时候程序编译无法通过了，可能是分配了太大的内存

3.2 模拟实现 OPT 或 FIFO 或 LRU 淘汰算法。

OPT 的实现其实并不难，查看最远的不使用的页面即可，实际上是一个贪心算法. 实验完成后我还用算法设计课学到的”保持领先”策略尝试证明了算法是最优的，比较有意思.

在实现完课本上的 LRU 算法后，通过查阅资料发现大体上有两种实现思路:

1. 第一种是为每一个页面维护一个计时器，每次不用则+1，淘汰时选择最大的那个即可，实现比较容易，也容易理解，但是在页框数比较大的时候去遍历速度是比较慢的，如果页框数为 n，则每次淘汰是一个 O(n)的时间复杂度，当然也可以维护一个堆，把复杂度降到(lgn)
2. 第二种更快一些，是通过双向链表+哈希表的形式实现. 实现非常巧妙.

   因为计时器本身的值不重要，我们实际上只需要只要页面之间的大小关系即可，而位置可以用来代表大小关系. 也就是说，我们可以维护这样的一个序列: 每次把最新访问的页面提到最前面，这样显而易见的是，每次淘汰的排在末尾的页面就可以了. 由于涉及到移动，所以使用链表，但是另一方面需要支持随机访问，所以使用双向链表存页面指针以支持随机访问. 算法思路如下:

   每次访问时，通过哈希表查对应的页面，出现两种情况:

   a) 命中，则只需要把这个页面的前后节点相连(双向链表)，然后把这个节点移到最前面

   b) 缺页，这个时候的淘汰策略上面已经说明，淘汰最后一个元素即可，然后把这个元素移到最前面

   第二种算法时间复杂度是 O(1)，非常低，只不过需要占用比较多的空间(哈希表和链表)，而且硬件上应该不容易实现，此时回想到课件上的近似的硬件实现方法，再一次体会到了工程实践的魅力!

3.3 计算某个变量或函数虚拟地址对应的物理地址等信息。

实验代码不长，不过涉及到知识比较核心．一开始对 pagemap 无从下手，后面看了一个人的博客，模仿着写出来了，还扩展学习了一下写时拷贝技术，不同类型的变量可能符合写时拷贝也可能不符合．

通过对比不同变量的虚拟地址，还印证了以前学习 C 语言的时候学到的内存布局知识

六、实验参考资料和网址

（1）教学课件

（2）https://www.runoob.com/linux/linux-tutorial.html 菜鸟教程-linux 教程

（3）https://blog.csdn.net/qq_42257666/article/details/105125865 一分钟学会页面置换算法【OPT、FIFO、LRU、NUR】

（3）https://zhuanlan.zhihu.com/p/635447326 虚拟内存管理算法实现——LRU 和 OPT

（4）https://blog.csdn.net/kongkongkkk/article/details/74366200 Linux 下如何在进程中获取虚拟地址对应的物理地址

（5） https://kongkongk.github.io/2020/06/30/address-translation/ Linux 下如何在进程中获取虚拟地址对应的物理地址