

《操作系统原理》实验报告

姓名	杜宇晗	学号	U202112151	专业班级	信安 2104	时间	2023.12.6
----	-----	----	------------	------	---------	----	-----------

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。
- 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 理解程序运行局部性的原理

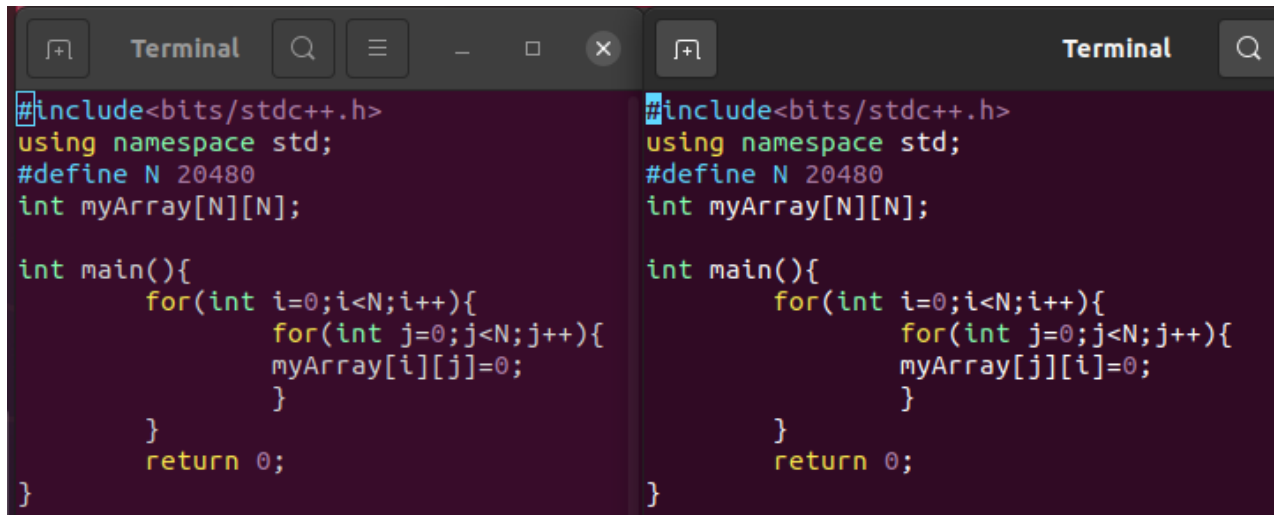
实验环境：VMware Workstation Pro 17

Ubuntu 20.04

内核版本：5.15.0-89-generic

编辑工具：gedit

先写出两个 `cpp` 文件



```
#include<bits/stdc++.h>
using namespace std;
#define N 20480
int myArray[N][N];

int main(){
    for(int i=0;i<N;i++){
        for(int j=0;j<N;j++){
            myArray[i][j]=0;
        }
    }
    return 0;
}
```

```
#include<bits/stdc++.h>
using namespace std;
#define N 20480
int myArray[N][N];

int main(){
    for(int i=0;i<N;i++){
        for(int j=0;j<N;j++){
            myArray[j][i]=0;
        }
    }
    return 0;
}
```

用 g++命令编译两个 cpp 文件，得到 1 和 2 两个文件

用 /usr/bin/time -v ./1 和 /usr/bin/time -v ./2 命令运行两个程序

3.2 Windows 模拟实现 OPT 和 LRU 淘汰算法

实验环境：Win11

编辑工具：vscode

编写如下代码

```
import java.util.HashMap;
```

```
import java.util.Map;
```

```
import java.util.Random;
```

```
import java.util.Scanner;
```

```
public class LruAndOpt {
```

```
    // 设置页面大小为 10
```

```
    private final int pageSize = 10;
```

```
    public static void main(String[] args) {
```

```
LruAndOpt lao = new LruAndOpt();

System.out.println("页面大小为 10,物理页框共 3 个");

System.out.print("请输入访问序列的个数:");

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

System.out.print("请输入随机数的限制:");

int limit = sc.nextInt();

final Random r = new Random();

int[] A1 = new int[n];

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

    A1[i] = r.nextInt(limit);

}

int[] A2 = new int[n];

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

    A2[i] = i;

}

int[] A3 = new int[n];

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

    A3[i] = i % 100;

}

System.out.println("=====OPT 算法=====");

System.out.println("-----随机序列-----");

lao.Opt(A1);

System.out.println("-----顺序序列-----");

lao.Opt(A2);
```

```

        System.out.println("-----循环序列-----");

        lao.Opt(A2);

        System.out.println("=====LRU 算法=====");

        System.out.println("-----随机序列-----");

        lao.Lru(A1);

        System.out.println("-----顺序序列-----");

        lao.Lru(A2);

        System.out.println("-----循环序列-----");

        lao.Lru(A2);

    }

```

```

public int getPageNumber(int address) {

    return address / pageSize;

}

```

```

public void Opt(int[] A) {

    int count = 0;

    int n = A.length;

    Map<Integer, Integer> pageMap = new HashMap<>();

    // 顺序访问

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

        int page = getPageNumber(A[i]);

        if (i == 0) {

            // 第一次访问一定缺页

            count++;

```

```

        pageMap.put(page, 1);
    } else {

        // 若不缺页则不作处理 下面处理缺页

        // 我们自己设定物理页框就 3 个

        if (!pageMap.containsKey(page)) {

            count++;

            if (pageMap.size() == 1) {

                pageMap.put(page, 2);

            } else if (pageMap.size() == 2) {

                pageMap.put(page, 3);

            } else {

                // 要淘汰的页号是最远的将来第一次出现的那一页

                // eliminate 要淘汰的页号

                int eliminate = 0, latest = 0;

                // 对于页表里的逻辑页 k

                for (int k : pageMap.keySet()) {

                    boolean found = false;

                    int time = 0;

                    for (int j = i + 1; j < n; j++) {

                        if (k == getPageNumber(A[j])) {

                            found = true;

                            time = j;

                            break;

                        }

                    }

                }

            }

        }

    }
}

```

```

        if (found) {
            if (latest < time) {
                latest = time;
                eliminate = k;
            }
        } else {
            //以后的元素都没有第 k 页的 直接淘汰
            eliminate = k;
            break;
        }
    }

    int lzw = pageMap.get(eliminate);
    pageMap.remove(eliminate);
    pageMap.put(page, lzw);
}

}

}

System.out.println("访问次数:" + n + ",缺页次数:" + count + ",缺页率:"
    + String.format("%.2f", ((double) count / n * 100)) + "%");
}

```

```

public void Lru(int A[]) {
    int n = A.length;
    int count = 0;

```

```

Map<Integer, Integer> pageMap = new HashMap<>();

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

    int page = getPageNumber(A[i]);

    if (i == 0) {

        // 第一次访问一定缺页

        count++;

        pageMap.put(page, 1);

    } else {

        // 若不缺页则不作处理 下面处理缺页

        if (!pageMap.containsKey(page)) {

            count++;

            if (pageMap.size() == 1) {

                pageMap.put(page, 2);

            } else if (pageMap.size() == 2) {

                pageMap.put(page, 3);

            } else {

                // 要淘汰的页号是从当前页前一页开始往前扫描最后一个
出现的

                // eliminate 要淘汰的页号

                int eliminate = 0, latest = Integer.MAX_VALUE;

                int k = 0;

                for (int j = i - 1; j >= 0; j--){

                    if (pageMap.containsKey(getPageNumber(A[j])) &&
latest > j){

                        latest = j;

                        k++;

```

```

        eliminate = getPageNumber(A[j]);

        if (k == 3){

            break;

        }

    }

}

int lzw = pageMap.get(eliminate);

pageMap.remove(eliminate);

pageMap.put(page, lzw);

}

}

}

}

System.out.println("访问次数:" + n + ",缺页次数:" + count + ",缺页率:"

    + String.format("%.2f", ((double) count / n * 100)) + "%");

}

```

}设置页面大小为 10，页框为 3 个

3.3 计算虚拟地址对应的物理地址等

编写如下代码

```

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <unistd.h>

#include <sys/stat.h>

#include <fcntl.h>

```



```
#include <stdint.h>
```

```
//传入虚拟地址 vaddr
```

```
void mem_addr(unsigned long vaddr)
```

```
{
```

```
    printf("虚拟地址: %lx\n",vaddr);
```

```
    //获取系统设定的页面大小
```

```
    int pageSize = getpagesize();
```

```
    printf("页面大小: %x\n",pageSize);
```

```
    //计算此虚拟地址相对于 0x0 的经过页面数
```

```
    unsigned long v_pageIndex = vaddr / pageSize;
```

```
    printf("虚拟页号: %lx\n",v_pageIndex);
```

```
    unsigned long v_offset = v_pageIndex * sizeof(uint64_t);
```

```
    //页内偏移地址
```

```
    unsigned long page_offset = vaddr % pageSize;
```

```
    printf("页内偏移地址: %lx\n",page_offset);
```

```
    uint64_t item = 0;//存储对应项的值
```

```
    int fd = open("/proc/self/pagemap",O_RDONLY);
```

```

if(fd == -1)//判断是否打开失败

{

    printf("open /proc/self/pagemap error\n");

    return;

}

//将游标移动到相应位置，即对应项的起始地址且判断是否移动失败

if(lseek(fd,v_offset,SEEK_SET) == -1)

{

    printf("sleek error\n");

    return;

}

//读取对应项的值，并存入 item 中，且判断读取数据位数是否正确

if(read(fd,&item,sizeof(uint64_t)) != sizeof(uint64_t))

{

    printf("read item error!\n");

    return;

}

//判断当前物理页是否在内存中，

if((((uint64_t)1 << 63) & item) == 0)

{

    printf("page present is 0\n");

    return;

}


//获得物理页号，即取 item 的 bit（0～54）

```

```
uint64_t phy_pageIndex = (((uint64_t)1 << 55) - 1) & item;

printf("物理页框号: %lx\n",phy_pageIndex);

//获取物理地址

unsigned long paddr = (phy_pageIndex * pageSize) + page_offset;

printf("物理地址: %lx\n",paddr);

}
```

```
const int a = 100;//全局变量 a

int main()

{

    int b =100;//局部变量 b

    static int c =100;//局部静态变量 c

    const int d =100;//局部常量 d


    printf("全局常量 a:\n");

    mem_addr((unsigned long)&a);


    printf("\n 局部变量 b:\n");

    mem_addr((unsigned long)&b);


    printf("\n 全局静态变量 c:\n");

    mem_addr((unsigned long)&c);

}
```

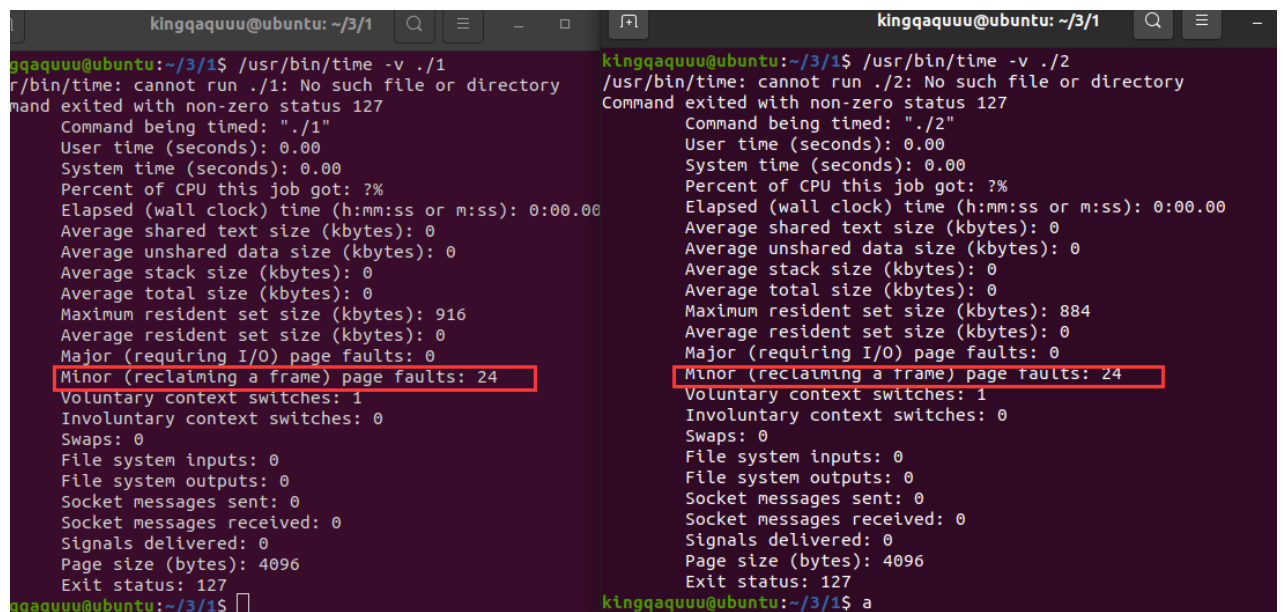
```
printf("\n 局部常量 d:\n");

mem_addr((unsigned long)&d);

}
```

四、实验结果

4.1 理解程序运行局部性的原理



```
kingqaquuu@ubuntu: ~/3/1
kingqaquuu@ubuntu:~/3/1$ /usr/bin/time -v ./1
/usr/bin/time: cannot run ./1: No such file or directory
Command exited with non-zero status 127
Command being timed: "./1"
User time (seconds): 0.00
System time (seconds): 0.00
Percent of CPU this job got: 7%
Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.00
Average shared text size (kbytes): 0
Average unshared data size (kbytes): 0
Average stack size (kbytes): 0
Average total size (kbytes): 0
Maximum resident set size (kbytes): 916
Average resident set size (kbytes): 0
Major (requiring I/O) page faults: 0
Minor (reclaiming a frame) page faults: 24
Voluntary context switches: 1
Involuntary context switches: 0
Swaps: 0
File system inputs: 0
File system outputs: 0
Socket messages sent: 0
Socket messages received: 0
Signals delivered: 0
Page size (bytes): 4096
Exit status: 127
kingqaquuu@ubuntu:~/3/1$

kingqaquuu@ubuntu:~/3/1$ /usr/bin/time -v ./2
/usr/bin/time: cannot run ./2: No such file or directory
Command exited with non-zero status 127
Command being timed: "./2"
User time (seconds): 0.00
System time (seconds): 0.00
Percent of CPU this job got: 7%
Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.00
Average shared text size (kbytes): 0
Average unshared data size (kbytes): 0
Average stack size (kbytes): 0
Average total size (kbytes): 0
Maximum resident set size (kbytes): 884
Average resident set size (kbytes): 0
Major (requiring I/O) page faults: 0
Minor (reclaiming a frame) page faults: 24
Voluntary context switches: 1
Involuntary context switches: 0
Swaps: 0
File system inputs: 0
File system outputs: 0
Socket messages sent: 0
Socket messages received: 0
Signals delivered: 0
Page size (bytes): 4096
Exit status: 127
kingqaquuu@ubuntu:~/3/1$ a
```

可以发现缺页率相同

4.2 Windows 模拟实现 OPT 和 LRU 淘汰算法

```

页面大小为 10,物理页框共 3个
请输入访问序列的个数:2400
请输入随机数的限制:2400
=====OPT算法=====
-----随机序列-----
访问次数:2400,缺页次数:2182,缺页率:90.92%
-----顺序序列-----
访问次数:2400,缺页次数:240,缺页率:10.00%
-----循环序列-----
访问次数:2400,缺页次数:240,缺页率:10.00%
=====LRU算法=====
-----随机序列-----
访问次数:2400,缺页次数:2373,缺页率:98.88%
-----顺序序列-----
访问次数:2400,缺页次数:240,缺页率:10.00%
-----循环序列-----
访问次数:2400,缺页次数:240,缺页率:10.00%

```

```

页面大小为 10,物理页框共 3个
请输入访问序列的个数:2400
请输入随机数的限制:2400
=====OPT算法=====
-----随机序列-----
访问次数:2400,缺页次数:2177,缺页率:90.71%
-----顺序序列-----
访问次数:2400,缺页次数:240,缺页率:10.00%
-----循环序列-----
访问次数:2400,缺页次数:240,缺页率:10.00%
=====LRU算法=====
-----随机序列-----
访问次数:2400,缺页次数:2362,缺页率:98.42%
-----顺序序列-----
访问次数:2400,缺页次数:240,缺页率:10.00%
-----循环序列-----
访问次数:2400,缺页次数:240,缺页率:10.00%

```

多次运行后发现 LRU 会比

4.3 计算虚拟地址对应的物理地址等

```
kingqaquuu@ubuntu:~/3/4$ ./address
```

全局常量a:

虚拟地址：5569aaca50e4

页面大小：1000

虚拟页号：5569aaca5

页内偏移地址：e4

物理页框号：0

物理地址：e4

局部变量b:

虚拟地址：7ffffeb8b0a60

页面大小：1000

虚拟页号：7ffffeb8b0

页内偏移地址：a60

物理页框号：0

物理地址：a60

全局静态变量c:

虚拟地址：5569aaca7010

页面大小：1000

虚拟页号：5569aaca7

页内偏移地址：10

物理页框号：0

物理地址：10

局部常量d:

虚拟地址：7ffffeb8b0a64

页面大小：1000

虚拟页号：7ffffeb8b0

页内偏移地址：a64

物理页框号：0

物理地址：a64

五、实验错误排查和解决方法

5.1 理解程序运行局部性的原理

暂无

5.2 Windows 模拟实现 OPT 和 LRU 淘汰算法

最开始使用 cpp 编写，但写出来的序列是固定的，不能实时更改，后借鉴后改用 java

5.3 计算虚拟地址对应的物理地址等

暂无

5.4 任务 4 的简短名称（不超过 15 个字）

六、实验参考资料和网址

(1) 教学课件

(2) <https://www.cnblogs.com/pengdonglin137/p/6802108.html>