实验报告

姓名：王迎旭
学号：16340226
班级：16级软件工程6班

一、实验目的

2.18　在2.3小节中，介绍了一个从一个文件向一个目标文件复制内容的程序。这个程序首先提示用户输入源文件和目标文件的名称。用Win32或POSIX的API写出这个C程序，并确信包括了所有必需的错误检测和文件存在的保证。一旦你正确地设计并测试了此程序，如果用一个系统来支持它，采用跟踪系统调用的工具来运行它。Linux系统提供了ptrace工具，而Solaris系统则采用truss或dtrace命令。在Mac OS X中，dtrace工具提供了类似的功能。

二、实验条件

系统：Fedora-Linux
编程语言：C

三、编程代码

```
#include <unistd.h>
#include <assert.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/ptrace.h>
#include <sys/wait.h>
#include <sys/reg.h>
#include <sys/syscall.h>
#include <sys/user.h>
void mmapcopy(int src_fd, size_t src_len, int dst_fd);
int main(int argc, char *argv[])
{
    if (argc != 3) {
    printf("Usage: %s<src_file><dst_file>\n", argv[0]);
    return -1;
}
//文件复制模块
```

```
    int src_fd ;
    int dst_fd;
    if ((src_fd = open(argv[1], O_RDONLY)) < 0){
            printf("file1 open failed! \n");
            return -1;
    }
    if ((dst_fd = open(argv[2], O_RDWR | O_CREAT | O_TRUNC, S_IRWXU)) < 0) {
            printf("file2 open failed! \n");
            return -1;
    }
    struct stat stat;
    fstat(src_fd, &stat); // 获取文件信息
    truncate(argv[2], stat.st_size); // 设置文件大小
    mmapcopy(src_fd, stat.st_size, dst_fd);
    Close(src_fd);
    close(dst_fd);
    return 0;
}
```

```
void mmapcopy(int src_fd, size_t src_len, int dst_fd)
{
    void *src_ptr, *dst_ptr;
    src_ptr = mmap(NULL, src_len, PROT_READ, MAP_PRIVATE, src_fd, 0);
    dst_ptr = mmap(NULL, src_len, PROT_WRITE | PROT_READ, MAP_SHARED, dst_fd, 0);
    if (dst_ptr == MAP_FAILED) {
        printf("mmap error: %s\n", strerror(errno));
        return ;
    }
    memcpy(dst_ptr, src_ptr, src_len); // 实现拷贝
    munmap(src_ptr, src_len);
    munmap(dst_ptr, src_len);
}
```

四、实验截图

（1）新建 a.text 与 b.text，然后随意向 a.text 文件中输入字符

（2）gcc 编译 test.c(源文件)

（3）运行并监控信息

（4）截图分析

```
[dell@localhost Desktop]$ gcc test.c
[dell@localhost Desktop]$ strace ./a.out a.text b.text
execve("./a.out", ["./a.out", "a.text", "b.text"], 0x7ffd9043eae0 /* 44 vars */)
 = 0
brk(NULL)                               = 0x14c2000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f6
320213000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=73512, ...}) = 0
mmap(NULL, 73512, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f6320201000
close(3)                                = 0
open("/lib64/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0 \5\2\0\0\0\0\0"..., 832)
 = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=2115824, ...}) = 0
mmap(NULL, 3955040, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f
631fc2a000
mprotect(0x7f631fde7000, 2093056, PROT_NONE) = 0
mmap(0x7f631ffe6000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENY
WRITE, 3, 0x1bc000) = 0x7f631ffe6000
mmap(0x7f631ffec000, 14688, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANON
YMOUS, -1, 0) = 0x7f631ffec000
close(3)                                = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f6
3201ff000
arch_prctl(ARCH_SET_FS, 0x7f63201ff700) = 0
mprotect(0x7f631ffe6000, 16384, PROT_READ) = 0
mprotect(0x600000, 4096, PROT_READ)     = 0
mprotect(0x7f6320215000, 4096, PROT_READ) = 0
munmap(0x7f6320201000, 73512)           = 0
open("a.text", O_RDONLY)                = 3
open("b.text", O_RDWR|O_CREAT|O_TRUNC, 0700) = 4
fstat(3, {st_mode=S_IFREG|0664, st_size=181, ...}) = 0
truncate("b.text", 181)                 = 0
mmap(NULL, 181, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f6320212000
```

```
open("a.text", O_RDONLY)                = 3
open("b.text", O_RDWR|O_CREAT|O_TRUNC, 0700) = 4
fstat(3, {st_mode=S_IFREG|0664, st_size=181, ...}) = 0
truncate("b.text", 181)                 = 0
mmap(NULL, 181, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f6320212000
mmap(NULL, 181, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x7f6320211000
munmap(0x7f6320212000, 181)             = 0
munmap(0x7f6320211000, 181)             = 0
close(3)                                = 0
close(4)                                = 0
exit_group(0)                           = ?
+++ exited with 0 +++
[dell@localhost Desktop]$
```

①每一行都是一次系统调用。等号左边是系统调用函数及其参数，右边是返回值。

②系统首先调用 execve 开始一个新的进程，最后调用 exit_group 退出进程，完成整个程序的执行过程。对于命令行执行的程序，execve 或者 exec 系列中的任何一个，均为 strace 输出系统调用中的第一个。strace 首先调用 fork 或者 clone 函数新建一个子进程，然后在子进程中调用 exec 载入需要执行的程序。

③ brk(0) = 0x8803000, 以 0 作为参数的调用 brk，返回值是内存管理的起始地址，如果程序调用调用 malloc，内存管理分配将从 0x8803000 地址开始。

④access 是检查文件是否存在

⑤使用 nmap2 函数进行匿名内存映射，以此来获取内存空间，其第二参数就是需要获取内存空间的长度，返回值为内存空间的起始地址。匿名内存映射就是为了不涉及具体的文件名,避免了文件的创建和打开,只能用于具有亲缘关系的进程间通信。真正能与源码对应上的只有 write 系统调用,其他系统调用基本用于系统初始化工作，装载被执行程序，载入 libc 函数库，设置内存映射等。

五、实验心得

①编写普通的 C 程序实现文件的拷贝比较容易，但是由于第一次使用追踪工具，不太懂原理，耗费了比较多的时间在尝试不同的代码嵌套上面，效率也低了很多。

②关于进程监控，此次小实验，在 Linux 环境下我只是简单的看出系统进行了那些简单的调用，而室友在 windows 下使用进程追踪工具可以更加清楚的进行进程监控，我觉得对于初学者来说可能更需要去了解一些深入性内在的东西。

③在做题的过程中，多多和周围的人进行交流，有时候会有额外的收获，比如本次的作业，我一开始的做法是在代码中加入追踪工具，后来舍友说代码可以简化，直接用 strace 工具进行追踪，果然减少了代码量，而且更加直观的反映了追踪过程。