华中科技大学

操作系统原理课程设计报告

姓　　名：　木林

学　　院：　计算机学院

专　　业：　计算机电脑

班　　级：　CS1902

学　　号：　I201920024

指导教师：

分数	
教师签名	

2022年　3月　24日

目　录

1. 实验一 Linux 编程

1.1.实验目的

（1） 掌握 Linux 操作系统的使用方法。
（2） 熟悉和理解 Linux 编程环境。

1.2.实验内容

1. 编写一个 C 程序，用 read、write 等系统调用实现文件拷贝功能。
命令形式： copy <源文件> <目标文件>

（2）编写一个 C 程序，使用图形编程库 (QT/GTK)分窗口显示三个并发进 程的运行

1.3.实验设计

1.3.1.开发环境

本次课程设计使用的环境配置如下：
（1）操作系统版本：UBUNTU 20.04.3 LTS
（2）操作系统内核版本：5.4.0-105-generic
（3）编译器及其版本：GCC version 8.4.0
（4）图形库及其版本：GTK+ 3.0
（5）自动编译工具：CMake version 3.11.4

1.3.2. 实验设计

- 文件拷贝：功能是能复制source文件到target文件。实验要求要用到系统调用 open/read/write。

- 并发进程演示：
安装需要的工具
sudo apt-get install build-essential libgtk-3-dev
sudo apt-get install cmake
使用fork()。若fork返回0，则是子进程，否则是父进程。这样我们可以创建三个进程，并且利用GTK的功能来完成信息的显示

1.4. 实验调试

1.4.1. 实验步骤

- 文件拷贝：**copy.c**

在terminal中，写gcc copyfunc.c -o copyfunc
copyfunc用的方法：./copfunc <src> <target>
比如：下面要把test.txt的内容拷贝到testtarget.txt中如图1.1

```
loli@loli:~/os1$ gcc copyfunc.c -o copyfunc
loli@loli:~/os1$ ./copyfunc test.txt testtarget.txt
loli@loli:~/os1$
```

图1.1copyfunc

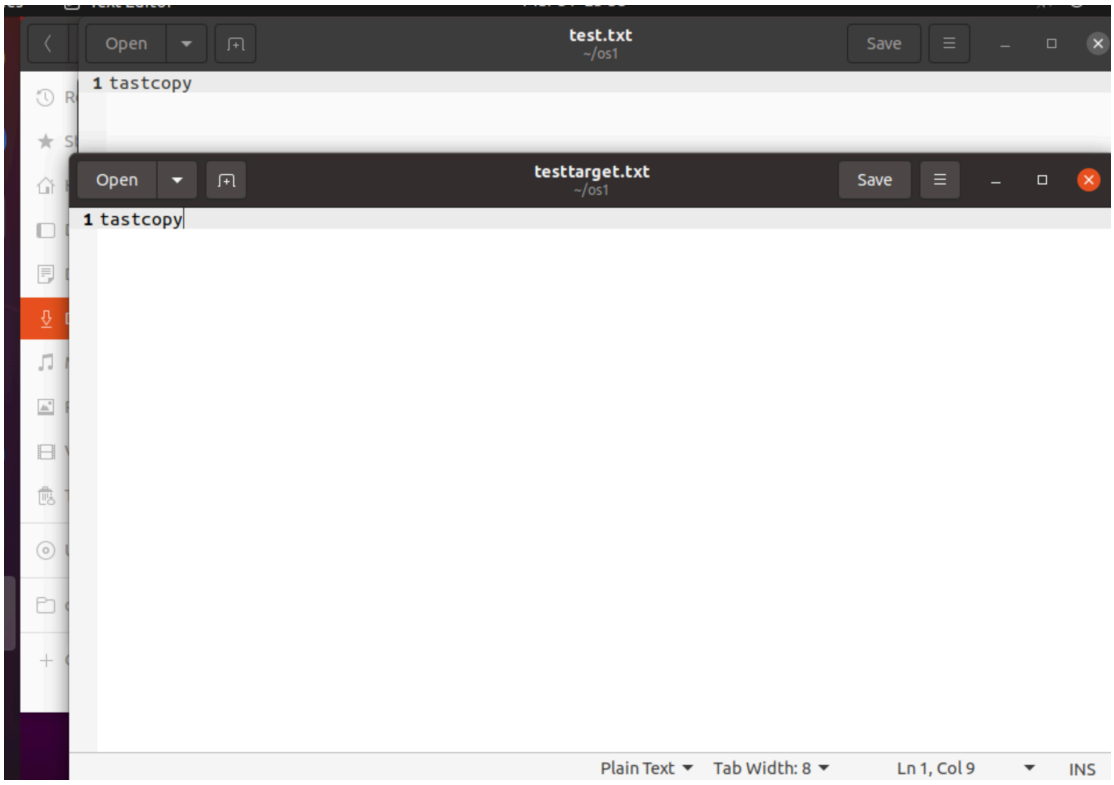
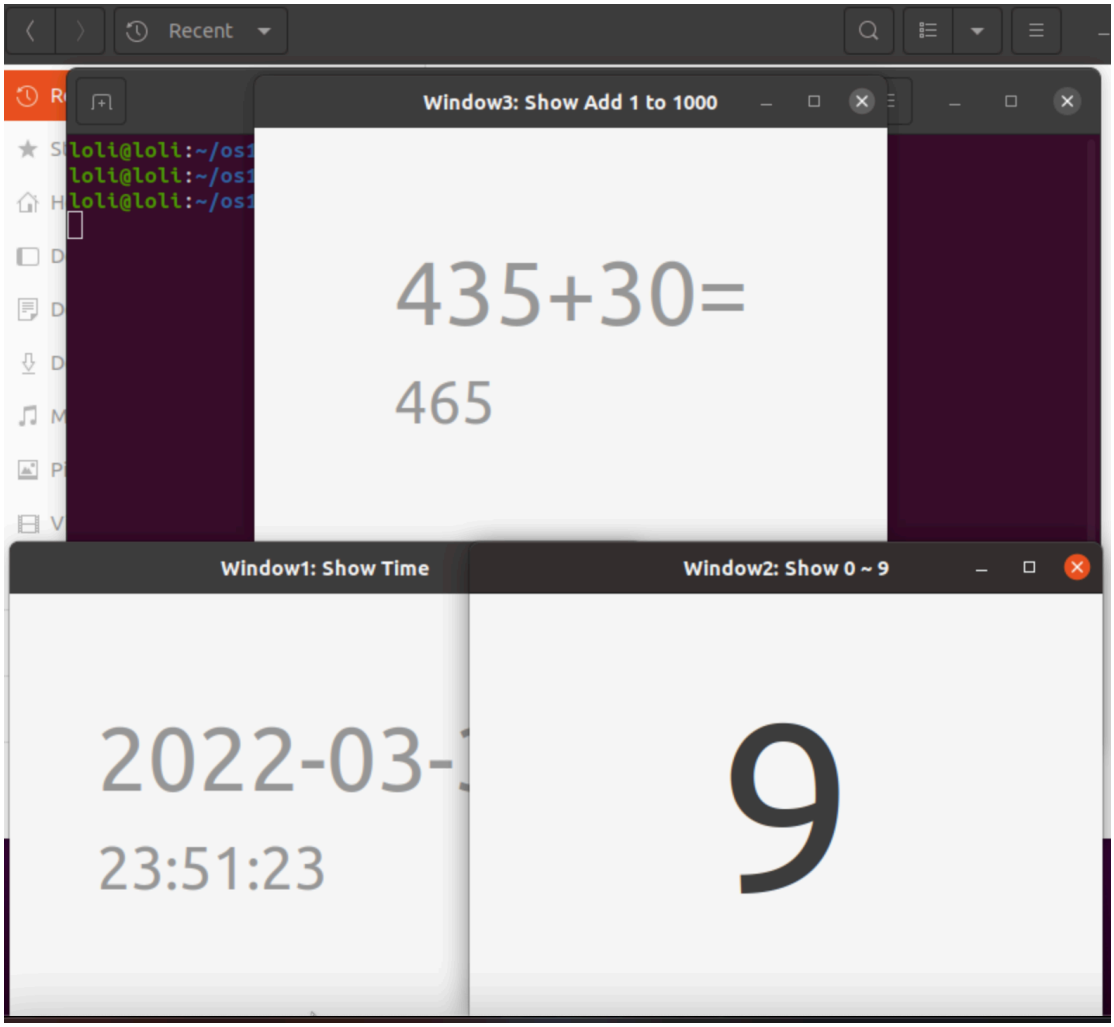


图 **1.2** 文件的内容

图 1.3 makefile

- 并发演示：**threetask.c**
  使用cmake工具构建Makefile，内容如图1.3：

  打开terminal，执行**make**，得到threetask.o 使用**./threetask.o** 就可以看到并发



进程的运行窗口。如图1.4：

图 1. 4 并发进程窗口

1.4.2. 实验调试及心得

这个实验教我如何使用 gtk 和 crate cmake 文件来用 gtk 编译代码

附录 实验代码

1. copyfunc.c

```
#define BUF_LEN 1024
```

```
mode_t get_file_mode(char *file) {
  struct stat s_buf;
  mode_t file_mode = 0x0;
  if (stat(file, &s_buf) != 0)
    err_exit("Stat");
  return s_buf.st_mode;
}

void do_copy(int read_fd, int write_fd) {
  int read_num;
  char buf[BUF_LEN];
  for (;;) {
    read_num = read(read_fd, buf, BUF_LEN);
    if (read_num == -1)
      err_exit("Copy");
    else if (read_num == 0)
      break;
    write(write_fd, buf, read_num);
  }
}

int main(int argc, char **argv) {
  int read_fd, write_fd;
  /* Check args */
  if (argc != 3)
    usage_err("./copy <src> <dst>");
  /* Open files */
  if ((read_fd = open(argv[1], O_RDONLY)) == -1)
    err_exit("Open source file");
  if ((write_fd = open(argv[2], O_WRONLY | O_CREAT | O_TRUNC,
get_file_mode(argv[1]))) == -1)
    err_exit("Open destnation file");
  do_copy(read_fd, write_fd);
  close(read_fd);
  close(write_fd);
  return 0;
}
```

2. threetask.c

```
#include <wait.h>
#include <gtk/gtk.h>

#define WINDOW_WIDTH 450
#define WINDOW_HEIGHT 300
#define RE_FEQ 1000

/*
 * refresh_time - Refresh the time in window1
 */
```

```
gboolean refresh_time(gpointer label) {
  time_t times;
  struct tm *time_buf;
  time(&times);
  time_buf = localtime(&times);

  /* Get now time */
  gchar *text_day = g_strdup_printf("<span font_desc='48'>%04d-%02d-%02d</
span>", \
    1900 + time_buf->tm_year, 1 + time_buf->tm_mon, time_buf->tm_mday);
  gchar *text_time = g_strdup_printf("<span font_desc='32'>%02d:%02d:%02d</
span>", \
    time_buf->tm_hour, time_buf->tm_min, time_buf->tm_sec);
  gchar *text_data = g_strdup_printf("\n%s\n\n%s\n", text_day, text_time);

  gtk_label_set_markup(GTK_LABEL(label), text_data);
  return TRUE;
}

/*
 * refresh_num - Refresh the num in window2
 */
gboolean refresh_num(gpointer label) {
  static int num = 0;
  gchar *text_num = g_strdup_printf("<span font_desc='128'>%d</span>", num++);
  if (num == 10)
    num = 0;
  gtk_label_set_markup(GTK_LABEL(label), text_num);
  return TRUE;
}

/*
 * refresh_sum - Refresh the num in window3
 */
gboolean refresh_sum(gpointer label) {
  static int sum = 0;
  static int add = 1;

  gchar *text_old = g_strdup_printf("<span font_desc='48'>%d+%d=</span>", sum,
add);
  sum += add++;
  gchar *text_new = g_strdup_printf("<span font_desc='32'>%d</span>", sum);
  if (add == 1000) {
    sum = 0;
    add = 1;
  }

  gchar *text_data = g_strdup_printf("\n%s\n\n%s\n", text_old, text_new);
```

```
  gtk_label_set_markup(GTK_LABEL(label), text_data);
  return TRUE;
}

int main(int argc, char **argv) {
  int pid1, pid2;
  int wait_tmp;

  switch(pid1 = fork()) {

  case -1:
    err_exit("Fork child 1");

  /* Child pid 1 - Show time */
  case 0:
    gtk_init(&argc, &argv);
    GtkWidget *window1 = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    g_signal_connect(G_OBJECT(window1), "delete_event",
G_CALLBACK(gtk_main_quit), NULL);
    gtk_window_set_title(GTK_WINDOW(window1), "Window1: Show Time");

    /* Window1 inside */
    GtkWidget *label1 = gtk_label_new(NULL);
    gtk_container_add(GTK_CONTAINER(window1), label1);
    /*
     * g_timeout_add - Sets a function to be called at regular intervals,
     * with the default priority, G_PRIORITY_DEFAULT.
     *
     * 1000: the time between calls to the function, in milliseconds
     */
    gint s1 = g_timeout_add(RE_FEQ, refresh_time, (void *)label1);

    gtk_widget_set_size_request(window1, WINDOW_WIDTH, WINDOW_HEIGHT);
    gtk_widget_show_all(window1);
    gtk_main();
    printf("Window 1 Closed!\n");
    exit(0);

  default:
    switch(pid2 = fork()) {

    case -1:
      err_exit("Fork child 2");

    /* Child pid 2 - Show 0 to 9 */
    case 0:
      gtk_init(&argc, &argv);
      GtkWidget *window2 = gtk_window_new(GTK_WINDOW_TOPLEVEL);
```

```
    g_signal_connect(G_OBJECT(window2), "delete_event",
G_CALLBACK(gtk_main_quit), NULL);
    gtk_window_set_title(GTK_WINDOW(window2), "Window2: Show 0 ~ 9");

    /* Window2 inside */
    GtkWidget *label2 = gtk_label_new(NULL);
    gtk_container_add(GTK_CONTAINER(window2), label2);
    gint s2 = g_timeout_add(RE_FEQ, refresh_num, (void *)label2);

    gtk_widget_set_size_request(window2, WINDOW_WIDTH, WINDOW_HEIGHT);
    gtk_widget_show_all(window2);
    gtk_main();
    printf("Window 2 Closed!\n");
    exit(0);

  /* Parent pid - Add 1 to 1000 */
  default:
    gtk_init(&argc, &argv);
    GtkWidget *window3 = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    g_signal_connect(G_OBJECT(window3), "delete_event",
G_CALLBACK(gtk_main_quit), NULL);
    gtk_window_set_title(GTK_WINDOW(window3), "Window3: Show Add 1 to
1000");

    /* Window3 inside */
    GtkWidget *label3 = gtk_label_new(NULL);
    gtk_container_add(GTK_CONTAINER(window3), label3);
    gint s3 = g_timeout_add(RE_FEQ, refresh_sum, (void *)label3);

    gtk_widget_set_size_request(window3, WINDOW_WIDTH, WINDOW_HEIGHT);
    gtk_widget_show_all(window3);
    gtk_main();
    /* Wait for child pid */
    printf("Window 3 Closed!\n");
    waitpid(pid1, &wait_tmp, 0);
    waitpid(pid1, &wait_tmp, 0);
    exit(0);
  }
 }
}
```

2. 实验二 增加系统调用

2.1.实验目的

（1）了解 Linux 系统内核代码结构。
（2）掌握添加系统调用的方法。

2.2.实验内容

（1）采用编译内核的方法，添加一个新的系统调用，实现文件拷贝功能。
（2）编写一个应用程序，测试新加的系统调用。

2.3.实验设计

2.3.1. 开发环境

本次课程设计使用的环境配置如下：
（1）操作系统版本：UBUNTU 20.04.4 LTS
（2）操作系统内核版本：4.19.236
（3）编译器及其版本：GCC version 8.4.0
（5）自动编译工具：CMake version 3.11.4

2.3.2. 实验设计

- 安装一些依赖：
  sudo apt-get install libncurses5-dev openssl libssl-dev
  sudo apt-get install build-essential
  Sudo apt-get install dracut
  sudo apt-get install pkg-config
  sudo apt-get install bison
  sudo apt-get install flex
  sudo apt-get install libelf-dev

- 内核编译：
  下载内核源码，在terminate中执行以下命令即可：
  wget https://mirrors.edge.kernel.org/pub/linux/kernel/v4.x/linux-4.19.236.tar.xz
  或者在浏览器https://mirrors.edge.kernel.org/pub/linux/kernel 可下在别的kernel版本

  提取linux-4.19.236文件的内容
  在 linux-4.19.236/arch/x86/entry/syscalls/syscall_64.tbl 增加如下的三行，添加

```
346 333   common  calling                 __x64_sys_calling
347 334   common  string                  __x64_sys_string
348 335   common  copyfunc                __x64_sys_copyfunc
```

三个系统调用入口，如图2.1：。

图 2.1 syscall_64.tbl文件

在 linux-4.17.8/kernel/sys.c 文件中。最后行，添加系统调用实现的代码。如下

```
SYSCALL_DEFINE0(calling) {
 printk(KERN_INFO "Hello world!");
 return 0;
}

SYSCALL_DEFINE1(string, const char *, str) {
 char buf[256];
 long read_num;

 /* Set memory access range */
 mm_segment_t fs = get_fs();
 set_fs(get_ds());

 /* Copy str form user space to kernel space */
 read_num = strncpy_from_user(buf, str, sizeof(buf));
 if (read_num < 0 || read_num == sizeof(buf)) {
       set_fs(fs);
       return -EFAULT;
 }

 printk(KERN_INFO "System call %s", buf);
 set_fs(fs);
 return 0;
}

SYSCALL_DEFINE2(copyfunc, const char *, s_file, const char *, t_file) {
 struct kstat k_buf;
 char copy_buf[1024];
 char s_filename[256], t_filename[256];
 int read_fd, write_fd;
 long read_num;

 /* Set memory access range */
 mm_segment_t fs = get_fs();
 set_fs(get_ds());

 /* Get source and target file name */
 read_num = strncpy_from_user(s_filename, s_file, sizeof(s_filename));
 if (read_num < 0 || read_num == sizeof(s_filename)) {
       set_fs(fs);
       return -EFAULT;
 }
 read_num = strncpy_from_user(t_filename, t_file, sizeof(t_filename));
```

```
if (read_num < 0 || read_num == sizeof(t_filename)) {
    set_fs(fs);
    return -EFAULT;
}

/* Get source file mode */
if (vfs_stat(s_filename, &k_buf) != 0) {
    set_fs(fs);
    return -EFAULT;
}

/* Open files */
if ((read_fd = ksys_open(s_filename, O_RDONLY, S_IRUSR)) == -1) {
    set_fs(fs);
    return -EFAULT;
}
printk("system");
if ((write_fd = ksys_open(t_filename, O_WRONLY | O_CREAT | O_TRUNC, k_buf.mode))
== -1) {
    set_fs(fs);
    return -EFAULT;
}

/* Do copy */
for (;;) {
    read_num = ksys_read(read_fd, copy_buf, sizeof(copy_buf));
    if (read_num < 0) {
        set_fs(fs);
        return -EFAULT;
    } else if (read_num == 0)
        break;
    ksys_write(write_fd, copy_buf, read_num);
}

ksys_close(read_fd);
ksys_close(write_fd);

/* Restore previous memory access */
set_fs(fs);

return 0;
}
```

然后，查看自己的kernel版本，在 terminal 打 uname -r 然后在linux-4.19.236 文件中打开terminal 打 sudo cp /boot/config-5.4.0-105-generic .config，使用sudo make menuconfig 显示下面的窗口。在General setup 然后 Local version打自己的名称，然后Save & Exit，如图2.3：

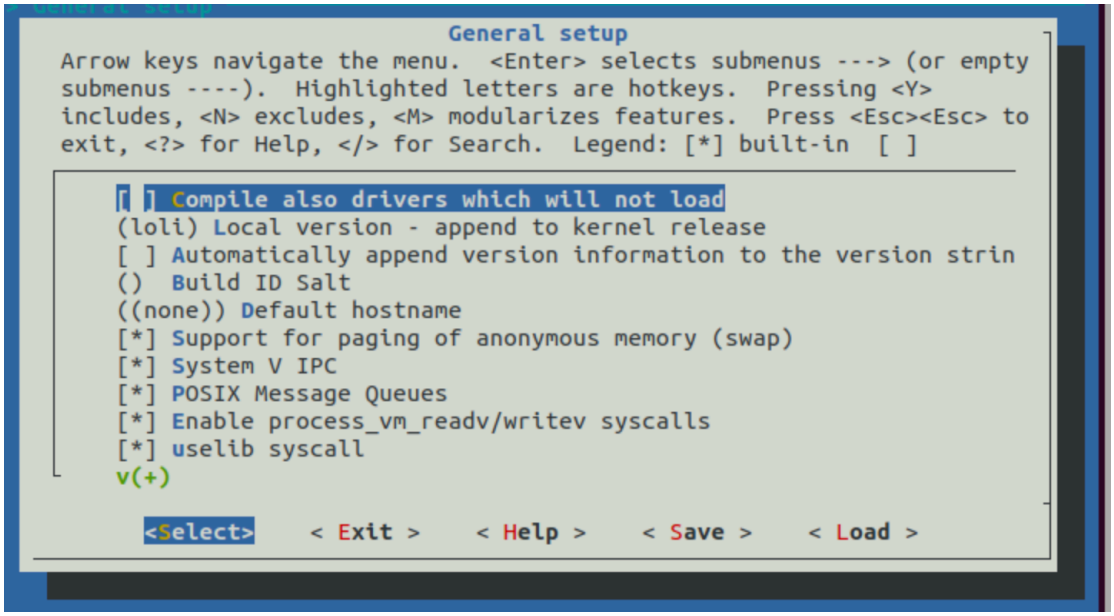


图 2.3 Kernel Configuration

开始编译内核使用sudo make -jn，n为虚拟机的内核两倍，sudo make 完以后进行大 sudo make install 和 sudo make modules_install

sudo make modules_install 成功以后，将新的kernel移动到/usr/src/如下面的指令:

sudo mv linux-4.19.236 /usr/src/, 然后打sudo mkinitramfs -o /boot/initrd.img-4.19.236loli

sudo update-initramfs -c -k 4.19.236loli 最后sudo update-grub2 和sudo update- grub

全部过程完成以后，重新打开虚拟机，在terminal中打reboot。

进行换用刚刚安装的kernel，在虚拟机booting中按住shift或者打esc打开GRUB BOOT MENU，选advance ubuntu option 然后选刚刚安装新的kernel名字。

2.4. 实验调试

2.4.1. 实验步骤

- 测试testcall 和 teststr:
  ./calling
  ./string dfdjsdjfh
  ./string sdjhjdfh
  dmesg或者sudo dmesg检查
  得到的结果如图2.5:

图 **2.5** 打印结果

- 测试 copyfunc，使用方法 ./copyfunc <src> <target> 执行dmesg 可看到的结果，如图2.6:

:

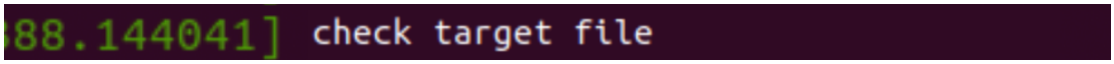


图2.6 copyfunc dimes 显示

2.4.2. 实验调试及心得

这个实验教会了我如何修改和定制我们自己的内核在这个过程中也遇到了很多问题和错误。 这些问题
内核版本而异，我们将用于为我定制案例错误是：sudo make 中

1. ./security/selinux/include/classmap.h:245:2: error: #error New address family defined, please update secclass_map.
#error New address family defined, please update secclass_map.
-git a/scripts/selinux/mdp/mdp.c b/scripts/selinux/mdp/mdp.c:
solution：找文件classmap.h 增加 #linclude <linux/socket.h>，找 mbp.c 和 gensheader.c 删除 #inlcude <sys/socket.h>

2. make[1]: *** No rule to make target 'debian/canonical-certs.pem', needed by certs/x509_certificate_list'. Stop.
make: *** [Makefile:1809: certs] Error 2
Solution:在terminal中，打指令sudo scripts/config —set-set SYSTEM_TRUSTED_KEYS”” 或者 sudo sudo scripts/config —disable SYSTEM_TRUSTED_KEYS

3. 使用错的gcc版本会出现多错误和warning。就我而言，我需要使用 gcc 8 与我使用的内核最兼容。安装新的gcc版本过程：
sudo apt-get gcc-8 (这里可以用数字表示gcc版本)
sudo update-alternatives —install /usr/bin/gcc gcc /usr/bin/gcc-8 8
Sudo update-alternatives —config gcc
然后选合适gcc版本

附录 实验代码

```
SYSCALL_DEFINE0(calling) {
 printk(KERN_INFO "Hello world!");
 return 0;
}
```

```
SYSCALL_DEFINE1(string, const char *, str) {
 char buf[256];
 long read_num;

 /* Set memory access range */
 mm_segment_t fs = get_fs();
 set_fs(get_ds());

 /* Copy str form user space to kernel space */
 read_num = strncpy_from_user(buf, str, sizeof(buf));
 if (read_num < 0 || read_num == sizeof(buf)) {
       set_fs(fs);
       return -EFAULT;
 }

 printk(KERN_INFO "System call %s", buf);
 set_fs(fs);
 return 0;
}

SYSCALL_DEFINE2(copyfunc, const char *, s_file, const char *, t_file) {
 struct kstat k_buf;
 char copy_buf[1024];
 char s_filename[256], t_filename[256];
 int read_fd, write_fd;
 long read_num;

 /* Set memory access range */
 mm_segment_t fs = get_fs();
 set_fs(get_ds());

 /* Get source and target file name */
 read_num = strncpy_from_user(s_filename, s_file, sizeof(s_filename));
 if (read_num < 0 || read_num == sizeof(s_filename)) {
       set_fs(fs);
       return -EFAULT;
 }
 read_num = strncpy_from_user(t_filename, t_file, sizeof(t_filename));
 if (read_num < 0 || read_num == sizeof(t_filename)) {
       set_fs(fs);
       return -EFAULT;
 }

 /* Get source file mode */
 if (vfs_stat(s_filename, &k_buf) != 0) {
       set_fs(fs);
       return -EFAULT;
 }
```

```
/* Open files */
if ((read_fd = ksys_open(s_filename, O_RDONLY, S_IRUSR)) == -1) {
      set_fs(fs);
      return -EFAULT;
}
printk("system");
if ((write_fd = ksys_open(t_filename, O_WRONLY | O_CREAT | O_TRUNC, k_buf.mode))
== -1) {
      set_fs(fs);
      return -EFAULT;
}

/* Do copy */
for (;;) {
      read_num = ksys_read(read_fd, copy_buf, sizeof(copy_buf));
      if (read_num < 0) {
            set_fs(fs);
            return -EFAULT;
      } else if (read_num == 0)
            break;
      ksys_write(write_fd, copy_buf, read_num);
}

ksys_close(read_fd);
ksys_close(write_fd);

/* Restore previous memory access */
set_fs(fs);

return 0;
}
```

3. 实验三 实验名

3.1.实验目的

3.2.实验内容

3.3.实验设计

3.3.1. 开发环境

3.3.2. 实验设计

3.4. 实验调试

3.4.1. 实验步骤

3.4.2. 实验调试及心得

附录 实验代码

4. 实验四 实验名

4.1.实验目的

4.2.实验内容

4.3.实验设计

4.3.1. 开发环境

4.3.2. 实验设计

4.4. 实验调试

4.4.1. 实验步骤

4.4.2. 实验调试及心得

附录 实验代码

5. 实验四 实验名

5.1.实验目的

5.2.实验内容

5.3.实验设计

5.3.1. 开发环境

5.3.2. 实验设计

5.4. 实验调试

5.4.1. 实验步骤

5.4.2. 实验调试及心得

附录 实验代码