



华中科技大学

操作系统原理课程设计报告

姓 名：罗南清

学 院：网络空间安全学院

专 业：信息安全

班 级：1701

学 号：U201714868

指导教师：羌卫中

分 数	
教师签名	

2020 年 2 月 21 日

目 录

实验一 进程并发实验	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	4
4.1 实验步骤	4
4.2 实验调试及心得	6
附录 实验代码	7
实验二 添加系统功能调用	14
1. 实验目的	14
2. 实验内容	14
3. 实验设计	14
3.1 开发环境	14
3.2 实验设计	14
4. 实验调试	14
4.1 实验步骤	14
4.2 实验调试及心得	16
附录 实验代码	16
实验三 添加系统驱动程序	18
1. 实验目的	18
2. 实验内容	18
3. 实验设计	18
3.1 开发环境	18
3.2 实验设计	18
4. 实验调试	21
4.1 实验步骤	21
4.2 实验调试及心得	24
附录 实验代码	25

实验四 /PROC 文件分析.....	35
1. 实验目的	35
2. 实验内容	35
3. 实验设计	35
3.1 开发环境.....	35
3.2 实验设计	35
4. 实验调试	36
4.1 实验步骤.....	36
4.2 实验调试及心得	37
附录 实验代码	39
实验五 模拟文件系统	82
1. 实验目的	82
2. 实验内容	82
3. 实验设计	82
3.1 开发环境.....	82
3.2 实验设计	82
4. 实验调试	88
4.1 实验步骤.....	88
4.2 实验调试及心得	91
附录 实验代码	94

实验一 进程并发实验

1. 实验目的

熟悉和理解 Linux 编程环境

2. 实验内容

本次实验的内容主要分为两个任务：

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 <源文件名> <目标文件名>`
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(一个窗口实时显示当前系统时间，一个窗口循环显示 0 到 9，一个窗口做 1 到 1000 的累加求和，刷新周期均为 1 秒)。

3. 实验设计

3.1 开发环境

Ubuntu 18.04 系统, gtk 3.0, gcc version 7.4.0

3.2 实验设计

任务 1：实验 1 的第一个阶段需要实现一个简单的 C 语言程序，首先需要有一个复制的源文件，同时需要创建一个存放复制内容的复制文件。文件复制传输的中间缓存站可以设置一个缓冲数组，此次实验设计了一个大小为 1024 字节的数组。打开源文件，返回一个 INT 标识，使用 `readbuf()` 函数，每次最多读取 1024 字节内容，将内容放入缓冲区数组，使用 `writebuf()` 函数将缓冲区的内容写入到复制文件，重复上述过程直至将整个复制过程完成。

任务 2：实验的第二个阶段是进行进程的同步，图形界面的工具是 gtk 3.0。主要实现 3 个进程的同步，3 个进程分别为实时显示系统时间、循环显示 0-9、显示 1-1000 的累加和。以上程序的刷新时间设置为 1 秒钟。为了实现 3 个进程，为三个进程分别创建一个独立的运行文件，每个文件能够独立表示它的功能。按照各自对应的需求实现 3 个独立的文件后，将其均编译成独立的执行文件。

再创建一个主文件，其中实现 3 个进程的并发执行。

4. 实验调试

4.1 实验步骤

4.1.1 任务一

1. 首先创建复制的源文件和目的文件，在代码中打开方式以只读和只写方式打开。

```
fd_src = open(argv[1], O_RDONLY);  
fd_dest = open(argv[2], O_WRONLY|O_CREAT, 0755); //S_IRUSR|S_IWUSR
```

2. 创建缓冲区，大小为 1024 字节的字符数组

```
#define BUFFER_SIZE 1024  
char buf[BUFFER_SIZE] = "";
```

3. 通过循环读写的方式实现文件的复制

```
do{  
    memset(buf,0,sizeof(buf));  
    ret = read(fd_src, buf, sizeof(buf));  
    if(ret > 0)  
        write(fd_dest, buf, ret);  
}while(ret > 0);
```

4. 关闭两个文件

```
close(fd_src);  
close(fd_dest);
```

5. 在 Linux 终端下使用变异命令: gcc exp1-1.c -o test1-1 产生可执行文件
6. 创建源文件 src.txt 和目的文件 des.txt
7. 编译源文件，执行命令 ./test1-1 src.txt des.txt
8. 使用命令 diff src.txt des.txt 比较二者文件是否有不同来验证文件内容是否成功拷贝

4.1.2 任务二

1. 使用 fork()函数创建三个进程，使用 exec 函数实现程序的调用。进程的并发使用 fork()系统调用，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 创建三个窗口，分别显示要求的内容。

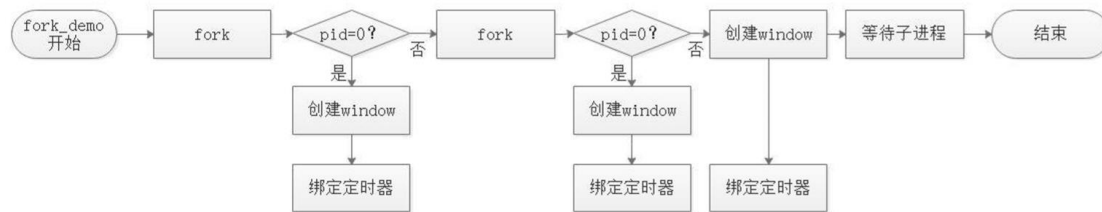


图 1-1 fork 流程图

2. 调用创建函数函数 `init_window()`,将进程中的信息在窗口中显示

```

int main( int argc, char *argv[]){
    pthread_t id;
    int i,ret;
    ret=pthread_create(&id,NULL,&thread,NULL);
    GtkWidget *vbox;          //定义一个组装箱;
    GtkWidget *window;
    /*初始化整个 GTK+程序，是每一个 GTK+程序必不可少的部分*/
    gtk_init(&argc, &argv);
    /*这里生成了一个窗口构件 GtkWidget，GTK_WINDOW_TOPLEVEL
    包含窗口的标题栏和边框，同意用窗口管理器来进行管理*/
    window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title(GTK_WINDOW(window), "sum");
    gtk_window_set_default_size(GTK_WINDOW(window), 300, 200);
    gtk_window_set_position(GTK_WINDOW(window),
GTK_WIN_POS_CENTER);
    label = gtk_label_new (s);
    gtk_container_add (GTK_CONTAINER (window), label);
    gtk_widget_show (label);
    /*开始显示窗口*/
    gtk_widget_show(window);
    gtk_main();
    return 0;
}
  
```

3. 分别创建三个程序实现显示系统时间、累加求和、循环显示 0-9 功能：

❖ 系统时间：

```

void gettime(){
    time_t timep;
    time (&timep);
  
```

```

        sprintf(t,"%s",ctime(&timep));
    }

    void *thread(void * argc){
        while(1){
            gettimeofday();
            gtk_label_set_text(GTK_LABEL(label),t);
            sleep(1);
        }
    }
}

```

❖ 累加求和:

```

void *thread(void * argc){
    int i,j,sum;
    for(i=1,sum=0,j=1;i<=100;i++){
        sleep(1);
        sum=sum+j;
        j+=1;
        sprintf(s,"%d",sum);
        gtk_label_set_text(GTK_LABEL(label),s);
    }
}

```

❖ 循环显示:

```

void *thread(void * argc){
    int i;
    for(i=0;i<=10000;i++){
        sleep(1);
        i = i % 10;
        sprintf(s,"%d",i);
        gtk_label_set_text(GTK_LABEL(label),s);
    }
}

```

4.2 实验调试及心得

4.2.1 实验调试

Src.txt 中的内容复制到空文件 des.txt 中，再比较二者文件是否有不同。

```
billy@billy-virtual-machine:~/桌面/Operating-System-Course-Project/exercise1$ ./
replication src.txt des.txt
billy@billy-virtual-machine:~/桌面/Operating-System-Course-Project/exercise1$ di
ff src.txt des.txt
```

图 1-2 文件内容比较图

直接运行编译后的可执行文件，如下图：

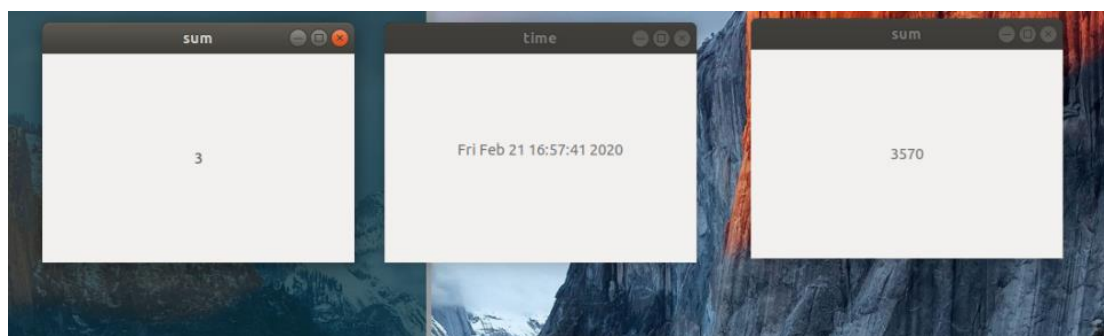


图 1-3 并行政程序结果图

4.2.2 心得

本次实验分为两个任务，第一任务是完成文件的拷贝。只是在应用函数 `write` 和 `read` 对文件进行读写操作。原理和编码都比较简单，其中需要注意的是有些操作需要错误处理，例如文件打开错误，输入文件错误等等。此任务注意的是加强程序的健壮性。

第二个任务是实现进程的并行执行以及结果的刷新显示。第一个问题是如何运用 `gtk` 与结果将输出显示在图形界面上，利用 `gtk 3.0` 的官方文档进行学习，可以很快将此问题解决。第二个问题是实现固定时间刷新结果显示。`Gtk 3.0` 通过函数 `gtk_label_set_text (GTK_LABEL (label), s)` 显示，`s` 为要输出的值，每一秒钟调用一次从而实现刷新。

附录 实验代码

❖ 任务一代码：

```
#include <stdio.h>
#include <fcntl.h>
#include <cstring>
#include <unistd.h>
#define BUFFER_SIZE 1024
int main(int argc, char *argv[]){
    if((argc == 3) && (strcmp(argv[1], argv[2]) != 0)){
```



```

int fd_src, fd_dest, ret;
fd_src = open(argv[1], O_RDONLY);
if(fd_src < 0){
    perror("open argv[1]");
    return -1;
}
fd_dest=open(argv[2],O_WRONLY|O_CREAT,0755);//S_IRUSR|S_IWUSR
if(fd_dest < 0){
    close(fd_src);
    perror("open argv[2]");
    return -1;
}
char buf[BUFFER_SIZE] = "";
do{
    memset(buf,0,sizeof(buf));
    ret = read(fd_src, buf, sizeof(buf));
    if(ret > 0)
        write(fd_dest, buf, ret);
}while(ret > 0);
close(fd_src);
close(fd_dest);
}
return 0;
}

```

❖ 任务二代码:

❖ Exp1-2.cpp

```

#include<sys/types.h>
#include <wait.h>
#include<stdio.h>
#include<unistd.h>
int main()
{
    pid_t time;
    pid_t cir;
    pid_t sum;

```

```

    if((time=fork())==-1){
        printf("fork error\n");
        return -1;
    }
    else if(time==0){
        execlp("./time",0);
    }else{
        if((cir=fork())==-1){
            printf("fork error\n");
            return -1;
        }
        if(cir==0){
            execlp("./cir",0);
        }else{
            if((sum=fork())==-1){
                printf("fork error\n");
                return -1;
            }
            if(sum==0){
                execlp("./sum",0);
            }else{//father process
                wait(&time);
                wait(&cir);
                wait(&sum);
            }
        }
    }
}

```

❖ Sum.cpp

```

#include<time.h>
#include<gtk/gtk.h>
#include <unistd.h>
#include<pthread.h>
#include <stdio>
char s[1000];

```

```

GtkWidget *label;
void *thread(void * argc){
    int i,j,sum;
    for(i=1,sum=0,j=1;i<=100;i++){
        sleep(1);
        sum=sum+j;
        j+=1;
        sprintf(s,"%d",sum);
        gtk_label_set_text(GTK_LABEL(label),s);
    }
}

int main( int argc, char *argv[])
{
    pthread_t id;
    int i,ret;
    ret=pthread_create(&id,NULL,&thread,NULL);
    GtkWidget *vbox;           //定义一个组装箱;
    GtkWidget *window;
    /*初始化整个 GTK+程序，是每一个 GTK+程序必不可少的部分*/
    gtk_init(&argc, &argv);
    /* 这里生成了一个窗口构件——GtkWindow，
GTK_WINDOW_TOPLEVEL 包含窗口的标题栏和边框，同意用窗口管理器来进行管理*/

    window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title(GTK_WINDOW(window), "sum");
    gtk_window_set_default_size(GTK_WINDOW(window), 300, 200);
    gtk_window_set_position(GTK_WINDOW(window),
GTK_WIN_POS_CENTER);
    label = gtk_label_new (s);
    gtk_container_add (GTK_CONTAINER (window), label);
    gtk_widget_show (label);
    /*开始显示窗口*/
    gtk_widget_show(window);
    gtk_main();
    return 0;
}

```

❖ Time.cpp

```
#include<ctime>
#include<gtk/gtk.h>
#include <unistd.h>
#include<pthread.h>
#include <stdio>
char t[50];
GtkWidget *label;
void gettime(){
    time_t timep;
    time (&timep);
    sprintf(t,"%s",ctime(&timep));
}
void *thread(void * argc){
    while(1){
        gettime();
        gtk_label_set_text(GTK_LABEL(label),t);
        sleep(1);
    }
}
int main( int argc, char *argv[]){
    pthread_t id;
    int i,ret;
    ret=pthread_create(&id,NULL,&thread,NULL);
    GtkWidget *vbox;           //定义一个组装箱;
    GtkWidget *window;
    /*初始化整个 GTK+程序，是每一个 GTK+程序必不可少的部分*/
    gtk_init(&argc, &argv);
    /* 这里生成了一个窗口构件——GtkWindow，
    GTK_WINDOW_TOPLEVEL 包含窗口的标题栏和边框，同意用窗口管理器来进行管理*/

    window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title(GTK_WINDOW(window), "time");
    gtk_window_set_default_size(GTK_WINDOW(window), 300, 200);
    gtk_window_set_position(GTK_WINDOW(window),
```

```

GTK_WIN_POS_CENTER);
    label = gtk_label_new (t);
    gtk_container_add (GTK_CONTAINER (window), label);
    gtk_widget_show (label);
    /*开始显示窗口*/
    gtk_widget_show(window);
    gtk_main();
    return 0;
}

```

❖ Cir.cpp

```

#include<time.h>
#include<gtk/gtk.h>
#include <unistd.h>
#include<pthread.h>
#include <stdio>
char s[1000];
GtkWidget *label;
void *thread(void * argc){
    int i;
    for(i=0;i<=10000;i++){
        sleep(1);
        i = i % 10;
        sprintf(s,"%d",i);
        gtk_label_set_text(GTK_LABEL(label),s);
    }
}
int main( int argc, char *argv[])
{
    pthread_t id;
    int i,ret;
    ret=pthread_create(&id,NULL,&thread,NULL);
    GtkWidget *vbox;          //定义一个组装箱;
    GtkWidget *window;
    /*初始化整个 GTK+程序，是每一个 GTK+程序必不可少的部分*/
    gtk_init(&argc, &argv);
    /* 这里 生 成 了 一 个 窗 口 构 件 ——GtkWindow ,

```

GTK_WINDOW_TOPLEVEL 包含窗口的标题栏和边框，同意用窗口管理器来进行管理*/

```
    window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title(GTK_WINDOW(window), "cir");
    gtk_window_set_default_size(GTK_WINDOW(window), 300, 200);
    gtk_window_set_position(GTK_WINDOW(window),
GTK_WIN_POS_CENTER);
    label = gtk_label_new (s);
    gtk_container_add (GTK_CONTAINER (window), label);
    gtk_widget_show (label);
    /*开始显示窗口*/
    gtk_widget_show(window);
    gtk_main();
    return 0;
}
```

实验二 添加系统功能调用

1. 实验目的

掌握添加系统调用的方法

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。

3. 实验设计

3.1 开发环境

Ubuntu 18.04, 原内核 5.3.40, 新内核 5.5.5, gcc version 7.4.0

3.2 实验设计

本次实验需要添加的功能调用为文件的复制拷贝功能。首先编写文件复制的代码，将代码添加进 `kernel/sys.c` 文件中；将系统调用的声明添加到 `include/linux/syscall.h` 文件中；在 `arch/x86/syscalls/syscall_32.tbl` 新增为系统调用分配调用号和系统调用名。然后重建内核，生成内核配置文件，编译内核映像，生成并安装新的系统模块，最后安装新的系统。进入新的内核版本的系统后，编写新的测试程序检测是否成功。

4. 实验调试

4.1 实验步骤

1. 编写系统调用中实现文件拷贝的系统功能的代码

将调用函数名修改为以 `sys_` 开头的函数，命名为 `sys_mycall`。文件的打开函数以及输出结果函数也需要使用内核函数，避免内存检查出现错误。将代码编写

进入 kernel/sys.c 文件。

```
asm linkage int sys_mycall(char* sourceFile, char* destFile)
{
    int source=sys_open(sourceFile,O_RDONLY,0);
    int dest=sys_open(destFile,O_WRONLY|O_CREAT|O_TRUNC,0600);
    char buf[4096];
    mm_segment_t old_fs = get_fs();
    set_fs(get_ds());
    int i = 0;
    if(source>0 && dest>0){
        do{
            i=sys_read(source,buf,4096);
            sys_write(dest,buf,i);
        }while(i > 0);
    }
    else {
        printf("open file (src & des) failed!\n");
        return -1;
    }
    sys_close(source);
    sys_close(dest);
    set_fs(old_fs);
    return 1;
}
```

2. 链接新的系统功能调用

在 include/linux/syscall.h 文件中添加系统功能调用定义。接着在系统调用表中为该系统调用分配一个系统调用名和一个系统调用号，系统调用号为 436，即在 arch/x86/syscalls/syscall_32.tbl 文件进行修改

3. 重建内核

前两步已经在内核文件中做了修改，现在先生成内核配置文件，在对应的内核文件下执行命令 `make menuconfig`，由于不需要安装其他模块，进入该界面直接 `Exit`。之后直接进行执行命令 `make -j4`。完成此过程后输入操作指令 `make modules_install` 生成并安装模块，运行 `make install` 安装新的系统。

4. 更新 grub 和重启修改内核

运行 `sudo update-grub` 命令更新 grub 文件。重启后，在 grub 界面选择新安

装的内核版本，进入系统。

5. 编写程序测试

```
#include<stdio.h>
#include<linux/unistd.h>
#include<asm/unistd.h>
int main(int argc,char **argv){
    int i=syscall(436,argv[1],argv[2]);
    if (i==1)
        printf("succeed!\n");
    else
        printf("failed!\n");
    return 1;
}
```

4.2 实验调试及心得

本次实验的工程量相当大，步骤繁琐，并且每一步都必须正确，不然就会导致编译失败，而每一次编译耗时特别长，所以做起来就很麻烦。

内核版本的下载一定要和自己的版本很好的兼容，不然可能会出现未知的错误。添加的系统功能调用的代码一定要使用内核调用的函数进行编写，否则内存访问会收到限制，引起内存检测时出现错误。

在编译内核模块和安装内核模块时需要很长时间，需要确保在这段时间内电脑要有充足的内存与物理存储，虚拟机有时候会将系统卡死。

附录 实验代码

❖ 内核函数

```
asmlinkage int sys_mycall(char* sourceFile,char* destFile)
{
    int source=sys_open(sourceFile,O_RDONLY,0);
    int dest=sys_open(destFile,O_WRONLY|O_CREAT|O_TRUNC,0600);
    char buf[4096];
    mm_segment_t old_fs = get_fs();
    set_fs(get_ds());
    int i = 0;
```

```

if(source>0 && dest>0){
    do{
        i=sys_read(source,buf,4096);
        sys_write(dest,buf,i);
    }while(i > 0);
}
else {
    printf("open file (src & des) failed!\n");
    return -1;
}
sys_close(source);
sys_close(dest);
set_fs(old_fs);
return 1;
}

```

❖ 测试函数

```

#include<stdio.h>
#include<linux/unistd.h>
#include<asm/unistd.h>

int main(int argc,char **argv){
    int i=syscall(436,argv[1],argv[2]);
    if (i==1)
        printf("succeed!\n");
    else
        printf("failed!\n");
    return 1;
}

```

实验三 添加系统驱动程序

1. 实验目的

通过添加设备驱动程序掌握添加设备驱动程序的方法。

2. 实验内容

- 1) 采用模块方法，添加一个新的字符设备驱动程序，实现打开/关闭、读/写等基本操作。
- 2) 编写一个应用程序，测试添加的驱动程序。

3. 实验设计

3.1 开发环境

Ubuntu 18.04, 内核版本 5.3.40, gcc version 7.4.0

3.2 实验设计

首先是编写设备驱动程序，涉及的内容主要有

- 1) 完成函数关系绑定
- 2) 模块的初始化、需要的功能、模块的卸载以及释放内存。

其次是修改通用的 `Makefile` 文件完成编译操作。然后使用设备驱动模块的加载指令完成加载。再者，使用设备文件指令使其生成对应的设备文件名。最后再编写应用程序对新添加的设备进行测试。

3.2.1 功能设计

首先设计设备驱动功能，这里我将其设计为一个大小为 8KB 大小的字符设备，并且实现如下图几个功能：

其中，在加载和退出时实现自动创建和删除设备文件，无需再使用 `mknod` 等命令。由于该字符设备的大小限制为 8KB 了，`fseek` 命令只实现了 `SEEK_SET` 与 `SEEK_CUR`。此外，清空设备的 `clean` 命令使用通用接口 `ioctl` 实现。

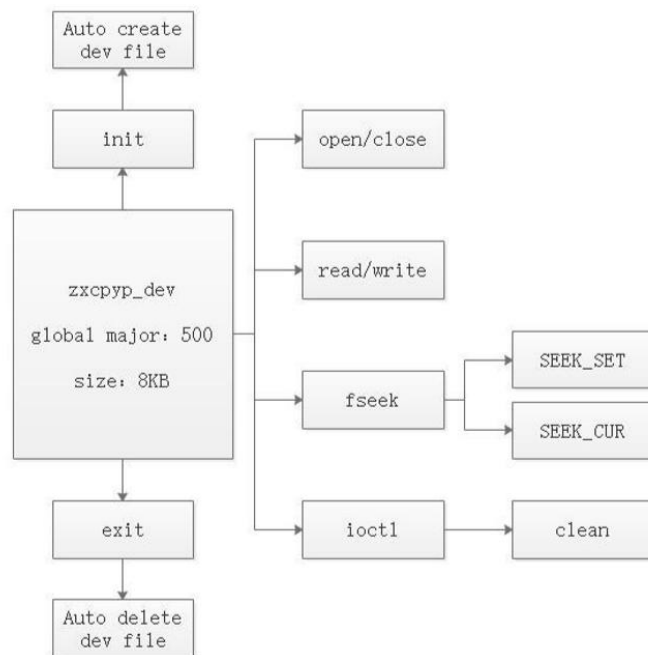


图 3-1 设备驱动设计

3.2.2 数据结构设计

将字符设备的 `cdev` 结构体和 `8KB` 的数据块整体作为一个结构体，并声明了一个全局变量指针，供其他函数使用。

```

struct lnq_dev {
    struct cdev cdev;
    unsigned char mem[LNQMEMP_SIZE];
};

static struct lnq_dev *lnq_devp;
  
```

3.2.3 需要实现的函数原型

在设计三中需要实现的各个子函数原型如下表：

入口点	函数原型	功能
<code>.open</code>	<code>static int zxcrypdriver_open(struct inode *inodep, struct file *filep);</code>	打开设备
<code>.release</code>	<code>static int zxcrypdriver_release(struct inode *inodep, struct file *filep)</code>	关闭设备
<code>.read</code>	<code>static ssize_t zxcrypdriver_read(struct file *filep, char __user *buf, size_t count, loff_t *offset)</code>	从设备中读数据
<code>.write</code>	<code>static ssize_t zxcrypdriver_write(struct file *filep, const char __user *buf, size_t count, loff_t *offset)</code>	向设备中写数据
<code>.llseek</code>	<code>static loff_t zxcrypdriver_llseek(struct file *filep, loff_t offset, int whence)</code>	更改设备当前的偏移量
<code>.unlocked_ioctl</code>	<code>static long zxcrypdriver_ioctl(struct file *filep, unsigned int cmd, unsigned long arg)</code>	通用接口，这里实现清空设备

3.2.4 接口注册

在 `file_operations` 结构体 `fops` 中存在着大量的函数入口点，在这里指定我们要实现的各个函数如下：

```
static const struct file_operations fops = {  
    .owner = THIS_MODULE,  
    .open = lnqdriver_open,  
    .release = lnqdriver_release,  
    .read = lnqdriver_read,  
    .write = lnqdriver_write,  
    .llseek = lnqdriver_llseek,  
    .unlocked_ioctl = lnqdriver_ioctl,  
};
```

3.2.5 打开/关闭函数设计

打开和关闭的实现是最简单的，打开设备只需要将 `file` 指针的私有数据 `private_data` 指向设备结构体 `lnq_dev` 即可。关闭设备可以直接关闭。

3.2.6 读/写函数设计

读写的实现稍微复杂一些，这里要分情况讨论。如果剩余空间足够的话，可以读/写指定的长度，如果空间不够的话，就只能读/写剩余的空间内的数据了。在完成操作后还要修改文件偏移量。

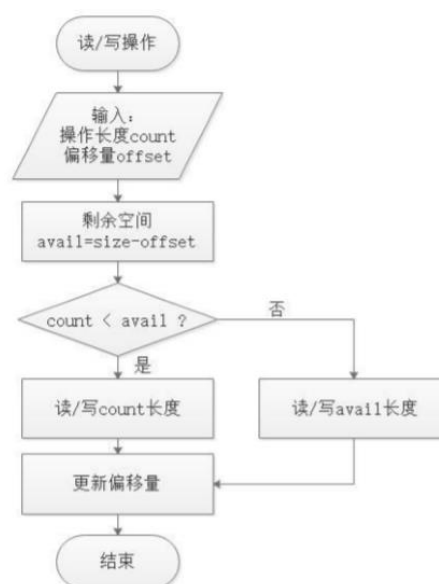


图 3-2 读写函数设计

3.2.7 fseek 函数设计

fseek 函数的核心就是更改偏移量，直接根据输入和当前的偏移量来计算新的偏移量即可，注意超过范围的操作应该不被允许。

标志	功能	实现
SEEK_SET	设置文件偏移量到从文件开头开始的某个位置	设置 filep->f_pos 为指定的位置
SEEK_CUR	设置文件偏移量从文件当前位置开始的某个位置	设置 filep->f_pos 为指定的位置+ offset
SEEK_END	设置文件偏移量到从文件结尾开始的某个位置	本设备中无法实现，因为该字符设备大小已经固定

3.2.8 clean 功能设计

clean 功能的实现采用了通用接口 ioctl，通过宏定义定义了一个代表 clean 操作的数值，在 ioctl 中通过 memset 函数实现清零。

4. 实验调试

4.1 实验步骤

1. 编写设备驱动程序，包括模块的初始化、设备的功能、以及模块的卸载

❖ 模块的初始化：

```
static int __init GlobalChar_init(void){
    int ret;
    ret = register_chrdev(char_major,DEV_NAME,&globalchar_fops);
    if(ret<0){
        printk(KERN_ALERT "GlobalChar Register Fail!\n");
        return -1;
    }
    else{
        printk(KERN_ALERT "GloblaChar Register Success!\n");
        char_major = ret;
        printk(KERN_ALERT "Major = %d\n",char_major);
    }
    return 0;
}
```

❖ 模块的卸载:

```
static void __exit GlobalChar_exit(void){
    unregister_chrdev(char_major,DEV_NAME);
    printk(KERN_ALERT "GlobalCharDev is dead now!\n");
    return;
}
```

❖ 设备功能:

```
static ssize_t GlobalRead(struct file *filp,char *buf,size_t len,loff_t *off){
    //GlobalData -= 1;
    if (copy_to_user(buf,&GlobalData,sizeof(int))){
        return -EFAULT;
    }
    return sizeof(int);
}

static ssize_t GlobalWrite(struct file *filp,const char *buf,size_t len,loff_t *off)
{
    if (copy_from_user(&GlobalData,buf,sizeof(int))){
        return -EFAULT;
    }
    return sizeof(int);
}
```

2. 编写 Makefile 文件完成编译操作

❖ 编写 Makefile 文件

```
ifneq ($(KERNELRELEASE),)
#kbuild syntax.
mymodule-objs :=test.o          //模块的文件组成
obj-m :=mydev.o                //生成的模块文件名
else
PWD :=$(shell pwd)
KVER :=$(shell uname -r)
KDIR :=/lib/modules/$(KVER)/build
all:
    $(MAKE) -C $(KDIR) M=$(PWD)
clean:
```

```
rm -f *.cmd *.o *.mod *.ko
```

```
endif
```

执行操作命令 `make` 完成驱动模块的编译，编译成功可得到 `.ko` 文件

3. 挂载内核中模块

设备驱动模块的加载指令为 `insmod mydev.o`，加载成功后会在文件 `/proc/devices` 中看到新增加的设备，包括设备名 `mydev` 和主设备号

4. 生成设备文件

执行命令 `mknod /dev/mydev C 248 0`

在此命令中，第一个参数是新建设备文件的地址和名字，第二个参数是指创建的是字符设备文件，第三个参数是主设备号，第四个参数是从设备号。执行成功会在 `/dev` 中看到一个新的设备文件 `sky_driver`。

5. 测试新的设备驱动

```
#include <sys/types.h>
#include <sys/stat.h>
#include <stdio.h>
#include <fcntl.h>
#define DEV_NAME "/dev/mydev"

int main()
{
    int fd,num = 9999;

    fd = open(DEV_NAME,O_RDWR,S_IRUSR | S_IWUSR);
    if (fd < 0)
    {
        printf("Open Device Failed!\n");
        return -1;
    }
    read(fd,&num,sizeof(int));
    printf("The Inqdriver is %d\n",num);
    printf("input a number written to Inqdriver: ");
    scanf("%d",&num);
    write(fd,&num,sizeof(int));
    read(fd,&num,sizeof(int));
    printf("The char you input is %d\n",num);
```



```
close(fd);
return 0;
}
```

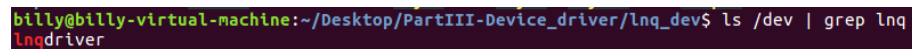
4.2 实验调试及心得

4.2.1 实验调试

在文件夹中运用 `make` 指令编译设备驱动程序后。在目录下执行以下命令，加载内核模块。同时可以自动生成设备文件。

`Sudo insmod lnq_dev.ko`

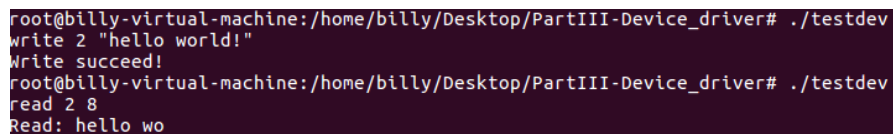
执行 `ls /dev`，可以看到生成的设备文件，加载完成。

A terminal window showing the command `ls /dev | grep lnq` being executed. The output is `lnqdriver`.

```
billy@billy-virtual-machine:~/Desktop/PartIII-Device_driver/lnq_dev$ ls /dev | grep lnq
lnqdriver
```

图 3-3 设备加载

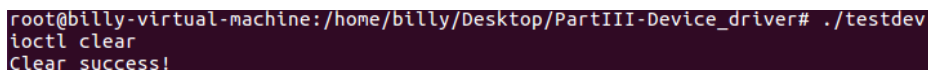
先在设备第 `m` 个位置开始写 `n` 长度的字符串，再从设备第 `m` 个位置开始读 `n` 长度的字符串。

A terminal window showing the execution of `./testdev`. It prompts for 'write' and 'read' operations. The user enters '2' for both. The output shows 'Write succeed!' and 'Read: hello wo' (truncated).

```
root@billy-virtual-machine:/home/billy/Desktop/PartIII-Device_driver# ./testdev
write 2 "hello world!"
Write succeed!
root@billy-virtual-machine:/home/billy/Desktop/PartIII-Device_driver# ./testdev
read 2 8
Read: hello wo
```

图 3-4 读写

最后将设备文件清理。

A terminal window showing the execution of `./testdev` with the `ioctl clear` command. The output is 'Clear success!'.

```
root@billy-virtual-machine:/home/billy/Desktop/PartIII-Device_driver# ./testdev
ioctl clear
Clear success!
```

图 3-5 清理

4.2.2 心得

本次实验主要是添加设备驱动程序的过程，其中包括驱动程序的编译、加载、测试等过程。在这期间遇到了很多问题，如新编写的驱动程序不知道放在哪个目录下进行编译，新编写的 `Makefile` 也不知道放在哪里对文件进行编译。在对应的内核文件路径下，任何修改都需要加上对应的访问权限，这样测试的时候才不会出现错误。

附录 实验代码

❖ 驱动程序模块:

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/cdev.h>
#include <linux/string.h>
#include <linux/slab.h>
#include <linux/fs.h>
#include <linux/uaccess.h>

#define InqMEM_SIZE 0x4000 /* 8KB for Inq driver */
#define GLOBAL_MAJOR 500 /* Set device major */
#define MEM_CLEAR 1 /* Use for ioctl to clear memory */

MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("A simple character driver module");

struct Inq_dev {
    struct cdev cdev;
    unsigned char mem[InqMEM_SIZE];
};

static struct Inq_dev *Inq_devp; /* Global dev pointer */
static struct device *device;
static struct class *class;

/*
 * Inqdriver_open - Open the driver
 */
static int Inqdriver_open(struct inode *inodep, struct file *filep) {
    printk(KERN_INFO "Inq Driver: open\n");
    filep->private_data = Inq_devp;
```

```

    return 0;
}

/*
 * Inqdriver_release - Release the driver
 */
static int Inqdriver_release(struct inode *inodep, struct file *filep) {
    printk(KERN_INFO "Inq Driver: release\n");
    return 0;
}

/*
 * Inqdriver_read - Read from the driver
 */
static ssize_t Inqdriver_read(struct file *filep, char __user *buf, size_t count,
loff_t *offset) {
    printk(KERN_INFO "Inq Driver: start read\n");

    int ret = 0;
    size_t avail = InqMEM_SIZE - *offset;
    struct Inq_dev *dev = filep->private_data;

    /* Available memory exists */
    if (count <= avail) {
        if (copy_to_user(buf, dev->mem + *offset, count) != 0)
            return -EFAULT;
        *offset += count;
        ret = count;
    }
    /* Available memory not enough */
    else {
        if (copy_to_user(buf, dev->mem + *offset, avail) != 0)
            return -EFAULT;
        *offset += avail;
        ret = avail;
    }
}

```

```

    }

    printk(KERN_INFO "lnq Driver: read %u bytes\n", ret);
    return ret;
}

/*
 * lnqdriver_write - Write from the driver
 */
static ssize_t lnqdriver_write(struct file *filep, const char __user *buf, size_t
count, loff_t *offset) {
    printk(KERN_INFO "lnq Driver: start write\n");

    int ret = 0;
    size_t avail = lnqMEM_SIZE - *offset;
    struct lnq_dev *dev = filep->private_data;
    memset(dev->mem + *offset, 0, avail);
    printk(KERN_INFO "lnq Driver: After write\n");

    /* Available memory exists */
    if (count > avail) {
        if (copy_from_user(dev->mem + *offset, buf, avail) != 0)
            return -EFAULT;
        *offset += avail;
        ret = avail;
    }
    /* Available memory not enough */
    else {
        if (copy_from_user(dev->mem + *offset, buf, count) != 0)
            return -EFAULT;
        *offset += count;
        ret = count;
    }

    printk(KERN_INFO "lnq Driver: written %u bytes\n", ret);
}

```

```

    return ret;
}

/*
 * lnqdriver_llseek - Set the current position of the file for reading and
writing
 */
static loff_t lnqdriver_llseek(struct file *filep, loff_t offset, int whence) {
    printk(KERN_INFO "lnq Driver: start llseek\n");

    loff_t ret = 0;
    switch (whence) {
        /* SEEK_SET */
        case 0:
            if (offset < 0) {
                ret = -EINVAL;
                break;
            }
            if (offset > lnqMEM_SIZE) {
                ret = -EINVAL;
                break;
            }
            ret = offset;
            break;
        /* SEEK_CUR */
        case 1:
            if ((filep->f_pos + offset) > lnqMEM_SIZE) {
                ret = -EINVAL;
                break;
            }
            if ((filep->f_pos + offset) < 0) {
                ret = -EINVAL;
                break;
            }
            ret = filep->f_pos + offset;
    }
}

```

```

        break;
/*
 * SEEK_END: Here we can't use SEEK_END,
 *           beacuse the memory is solid.
 *
case 2:
    if (offset < 0) {
        ret = -EINVAL;
        break;
    }
    ret = lnqMEM_SIZE + offset;
    break;*/
/* Else: return error */
default:
    ret = -EINVAL;
}

if (ret < 0)
    return ret;

printk(KERN_INFO "lnq Driver: set offset to %u\n", ret);
filep->f_pos = ret;
return ret;
}

/*
 * lnqdriver_ioctl - Control the lnq driver(memory clear)
 */
static long lnqdriver_ioctl(struct file *filep, unsigned int cmd, unsigned long
arg) {
    printk(KERN_INFO "lnq Driver: start memory clear\n");

    struct lnq_dev *dev = filep->private_data;
    switch (cmd) {
    case MEM_CLEAR:

```

```

        memset(dev->mem, 0, lnqMEM_SIZE);
        printk("lnq Driver: memory is set to zero\n");
        break;
default:
        return -EINVAL;
    }
    return 0;
}

/*
 * Set operation pointers
 */
static const struct file_operations fops = {
    .owner = THIS_MODULE,
    .open = lnqdriver_open,
    .release = lnqdriver_release,
    .read = lnqdriver_read,
    .write = lnqdriver_write,
    .llseek = lnqdriver_llseek,
    .unlocked_ioctl = lnqdriver_ioctl,
};

/*
 * lnqdriver_init - Initial function for lnqdriver
 */
static int __init lnqdriver_init(void) {
    printk(KERN_INFO "Load module: lnqdriver\n");

    int ret;
    dev_t devno = MKDEV(GLOBAL_MAJOR, 0);
    ret = register_chrdev_region(devno, 1, "lnqdriver");
    if (ret < 0) {
        printk(KERN_ALERT "Registering the character device failed
with %d\n", ret);
        return ret;
    }
}

```

```

    }

    /* Alloc memory for device */
    lnq_devp = kzalloc(sizeof(struct lnq_dev), GFP_KERNEL);
    if (lnq_devp == NULL) {
        printk(KERN_ALERT "Alloc memory for device failed\n");
        ret = -ENOMEM;
        goto failed;
    }
    memset(lnq_devp->mem, 0, lnqMEM_SIZE);

    /* Setup device */
    cdev_init(&lnq_devp->cdev, &fops);
    lnq_devp->cdev.owner = THIS_MODULE;
    cdev_add(&lnq_devp->cdev, devno, 1);

    /* Creat device file */
    class = class_create(THIS_MODULE, "lnqdriver");
    if (IS_ERR(class)) {
        ret = PTR_ERR(class);
        printk(KERN_ALERT "Creat class for device file failed with %d\n",
ret);
        goto failed;
    }
    device = device_create(class, NULL, devno, NULL, "lnqdriver");
    if (IS_ERR(device)) {
        class_destroy(class);
        ret = PTR_ERR(device);
        printk(KERN_ALERT "Creat device file failed with %d\n", ret);
        goto failed;
    }

    return 0;

failed:

```



```

    unregister_chrdev_region(devno, 1);
    return ret;
}

/*
 * Inqdriver_exit - Exit function for Inqdriver
 */
static void __exit Inqdriver_exit(void) {
    printk(KERN_INFO "Unload module: Inqdriver\n");

    device_destroy(class, MKDEV(GLOBAL_MAJOR, 0));
    class_unregister(class);
    class_destroy(class);

    cdev_del(&Inq_devp->cdev);
    kfree(Inq_devp);
    unregister_chrdev_region(MKDEV(GLOBAL_MAJOR, 0), 1);
}

module_init(Inqdriver_init);
module_exit(Inqdriver_exit);

```

❖ 测试程序：

```

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ioctl.h>
#include <string.h>
#include <fcntl.h>
#include <unistd.h>

/* Cmd for ioctl */
#define MEM_CLEAR 1

/*

```

```

* print_usage - Print usage of testdev
*/

void print_usage(void) {
    printf("Usage: \n");
    printf("\t./testdev read <startpos> <readnum>\n");
    printf("\t./testdev write <startpos> <string>\n");
    printf("\t./testdev ioctl clear\n");
}

int main(int argc, char **argv) {
    int fd, start, num;
    char buf[1024];
    fd = open("/dev/lnqdriver", O_RDWR);
    if (fd < 0) {
        printf("Open error!\n");
        return 0;
    }
    if (argc == 4 && strcmp(argv[1], "read", 4) == 0) {
        start = atoi(argv[2]);
        num = atoi(argv[3]);
        lseek(fd, start, SEEK_SET);
        read(fd, buf, num);
        printf("Read: %s\n", buf);
    }
    else if (argc == 4 && strcmp(argv[1], "write", 5) == 0) {
        start = atoi(argv[2]);
        lseek(fd, start, SEEK_CUR);
        write(fd, argv[3], strlen(argv[3]));
        printf("Write succeed!\n");
    }
    else if (argc == 3 && strcmp(argv[1], "ioctl", 5) == 0) {
        if (strcmp(argv[2], "clear", 5) == 0) {
            ioctl(fd, MEM_CLEAR, NULL);
            printf("Clear success!\n");
        }
    }
}

```

```
        else
            print_usage();
    }
    else
        print_usage();

    close(fd);
    return 0;
}
```

实验四 /proc 文件分析

1. 实验目的

- (1) 掌握实例操作系统的实现方法。
- (2) 使用 GTK/QT 实现一个系统监控器

2. 实验内容

- (1) 了解/proc 文件的特点和使用方法。
- (2) 监控系统状态，显示系统部件的使用情况。
- (3) 用图形界面监控系统状态，包括 CPU 和内存利用率、进程信息等。

3. 实验设计

3.1 开发环境

Ubuntu 18.04, gtk 2.0, gcc version 7.4.0

3.2 实验设计

系统监视器利用 notebook 控件设计了六个页面

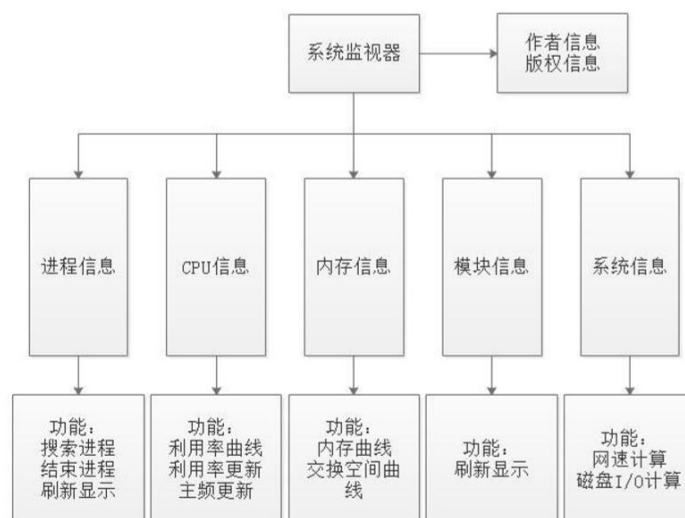


图 4-1 总体设计

4. 实验调试

4.1 实验步骤

1. 进程详细信息 /proc 文件系统中存在着很多以数字为文件名的文件夹，这些文件夹包含的就是对应进程号的详细信息，其中我们要获取进程的详细信息被存放在 status 和 stat 两个文件内。他们的区别是 status 比较直观，按行列出了进程的每个属性和其对应的值，而 stat 则是只有数字，每个一个空格一个。

这里为了方便读取，针对 stat 进行操作，我们需要的以下几位的数据：

- 1) 第 1 位：pid，进程号
- 2) 第 2 位：name，进程名
- 3) 第 3 位：status，进程状态
- 4) 第 4 位：ppid，父进程号
- 5) 第 18 位：priority，优先级
- 6) 第 23 位：memory，占用内存

2. CPU 详细信息 CPU 相关的信息被存放在/proc/cpuinfo 文件里，按行列出了每个 CPU 每个属性和其对应的值。

我们需要的是以下几行的信息：

- 1) 第 5 行：model name，CPU 名
- 2) 第 8 行：cpu MHz，CPU 主频
- 3) 第 9 行：cache size，缓存大小
- 4) 第 13 行：cpu cores，CPU 核数
- 5) 第 25 行：address sizes，地址空间

此外，CPU 的利用率还需要由计算获得，其数据在/proc/stat 文件内，该文件每行记录了一个 CPU 的数据，第 1 位为 CPU 名，第 2 到 5 位分别为 user, nice, system 和 idle，则有：

$$\text{cpu}(\text{total}) = \text{user} + \text{nice} + \text{system} + \text{idle}$$

$$\text{utilization} = 100\% * (\Delta \text{total} - \Delta \text{idle}) / \Delta \text{total}$$

3. 内存信息

内存相关的信息被存放在/proc/meminfo 文件里，也按行列出了内存的每个属性和其对应的值。

我们需要的是以下几行的信息：

- 1) 第 1 行：MemTotal，总内存大小
- 2) 第 2 行：MemFree，剩余可用内存大小
- 3) 第 15 行：SwapTotal，交换空间大小

4) 第 16 行: `cpu cores`, 剩余可用交换空间大小
内存利用率直接利用 $(1 - \text{free} / \text{total}) * 100\%$ 即可。

4. 模块信息

模块相关的信息被存放在 `/proc/modules` 文件里, 按行列出了每个加载的模块的属性。

每一行的信息含义如下:

- 1) 第 1 位: `modules name`, 模块名
- 2) 第 2 位: `used memory`, 占用内存
- 3) 第 3 位: `used times`, 使用次数

5. 系统信息

系统相关的信息分别可以在以下文件中获取:

主机名 `Hostname: /etc/hostname`.

操作系统名 `OS Name: /etc/issue`.

操作系统类型 (32 or 64) `OS Type: sizeof(char *) * 8;`

内核版本 `Kernel Version: /proc/sys/kernel/osrelease`.

GCC 版本 `GCC Version: /proc/version`.

开机时间 `Uptime: /proc/uptime`.

6. 网速与磁盘 I/O 速度

与网络相关的信息全部存放在 `/proc/net` 目录下, 其中关于发送和接收数据大小的信息在 `dev` 文件内:

该文件内每一行记录一个网卡的信息, 我们只需要获取 `Receive` 的 `bytes` 和 `Transmit` 的 `bytes` 信息即可, 记录每两秒前后的信息即可计算出上传和下载的速度。

与磁盘相关的信息存放在 `/proc/diskstats` 中, 我们需要第 6 位和第 10 位的信息, 分别代表读的扇区数 `rd_sectors` 和写的扇区数 `wr_sectors`, 记录每两秒前后的信息, 然后利用以下公式即可计算出速率。

$$\text{read speed} = (\Delta \text{rd_sectors} / \Delta t) * (\text{block_size} / 1024)$$
$$\text{write speed} = (\Delta \text{wr_sectors} / \Delta t) * (\text{block_size} / 1024)$$

这里 `block_size` 是 512

4.2 实验调试及心得

4.2.1 实验调试

运行结果的截图如下:

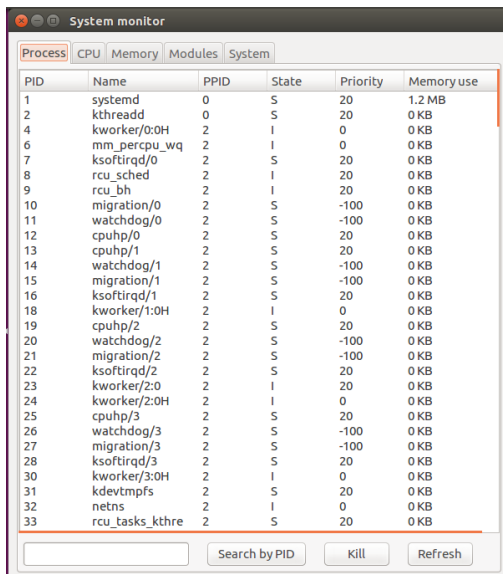


图 4-2 process

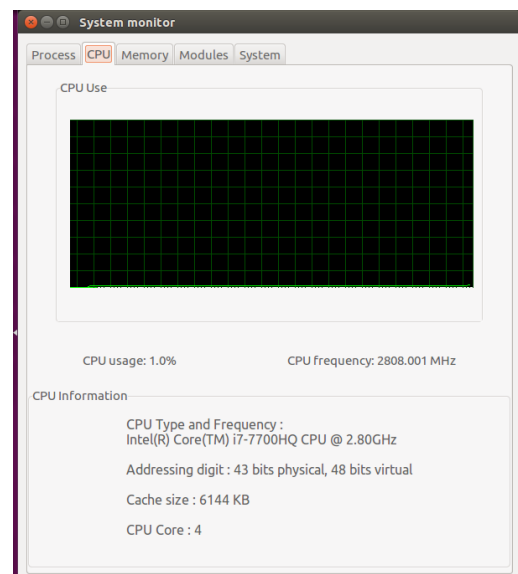


图 4-3 CPU

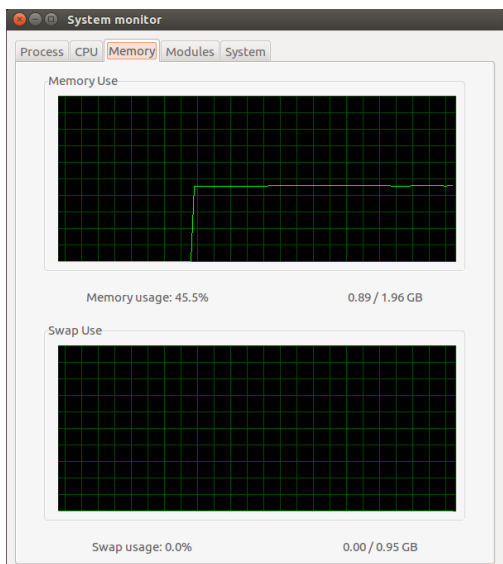


图 4-4 Memory

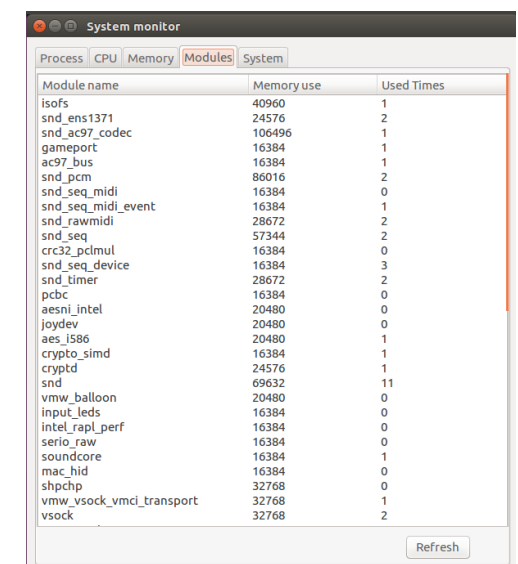


图 4-5 Modules

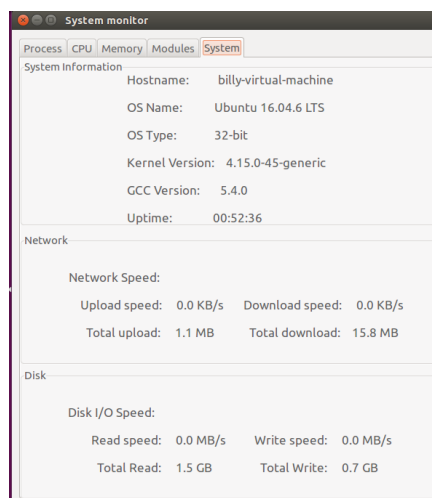


图 4-6 System

4.2.2 心得

本次实验其实基本都是将对应文件中的内容读入，再将他们放入缓冲区，再输出到对应的图形化界面控件。但是我的图形化控件对应输出的 CPU 使用率图像残缺不全。最后发现这个问题可能是图像控件的原因，并没有深究。

附录 实验代码

❖ Sysmonitor.c

```
#include "sysmonitor.h"

/*
 * read_stat - Read data from /proc/pid/stat
 */
void read_stat(char (*info)[1024], char *stat_file) {
    /*
     * stat file format:
     *
     * pid (name) status ppid .....(13 data) priority (5 data) memory
     */
    int pos;

    /* Get pid */
    for (pos = 0; pos < 1024; pos++) {
        if (stat_file[pos] == ' ')
            break;
    }
    stat_file[pos] = '\0';
    strcpy(info[0], stat_file);
    stat_file += pos;
    stat_file += 2;

    /* Get name */
    for (pos = 0; pos < 1024; pos++) {
        if (stat_file[pos] == ')')
            break;
    }
}
```



```

}
stat_file[pos] = '\0';
strcpy(info[1], stat_file);
stat_file += pos;
stat_file += 2;

/* Get status */
for (pos = 0; pos < 1024; pos++) {
    if (stat_file[pos] == ' ')
        break;
}
stat_file[pos] = '\0';
strcpy(info[3], stat_file);
stat_file += pos;
stat_file += 1;

/* Get ppid */
for (pos = 0; pos < 1024; pos++) {
    if (stat_file[pos] == ' ')
        break;
}
stat_file[pos] = '\0';
strcpy(info[2], stat_file);
stat_file += pos;
stat_file += 1;

/* Get priority */
int i;
for (i = 0, pos = 0; pos < 1024; pos++) {
    if (stat_file[pos] == ' ')
        i++;
    if (i == 13)
        break;
}
stat_file[pos] = '\0';

```

```

stat_file += pos;
stat_file += 1;
for (pos = 0; pos < 1024; pos++) {
    if (stat_file[pos] == ' ')
        break;
}
stat_file[pos] = '\0';
strcpy(info[4], stat_file);
stat_file += pos;
stat_file += 1;

/* Get memory use */
for (i = 0, pos = 0; pos < 1024; pos++) {
    if (stat_file[pos] == ' ')
        i++;
    if (i == 5)
        break;
}
stat_file[pos] = '\0';
stat_file += pos;
stat_file += 1;
for (pos = 0; pos < 1024; pos++) {
    if (stat_file[pos] == ' ')
        break;
}
stat_file[pos] = '\0';
char buf[1024];
if (atoi(stat_file) > 1024)
    sprintf(buf, "%.1f MB\0", atoi(stat_file) / 1024.0);
else
    sprintf(buf, "%d KB\0", atoi(stat_file));
strcpy(info[5], buf);
}

/*

```

```

/* get_process_info - Get process info in /proc and add into clist
*/

void get_process_info(void) {
    DIR *dir;
    struct dirent *dir_info;
    int fd;
    char pid_file[1024];
    char stat_file[1024];
    char *one_file = NULL;
    char info[6][1024];
    gchar *txt[6];

    /* Set clist column title */
    gtk_clist_set_column_title(GTK_CLIST(clist), 0, "PID");
    gtk_clist_set_column_title(GTK_CLIST(clist), 1, "Name");
    gtk_clist_set_column_title(GTK_CLIST(clist), 2, "PPID");
    gtk_clist_set_column_title(GTK_CLIST(clist), 3, "State");
    gtk_clist_set_column_title(GTK_CLIST(clist), 4, "Priority");
    gtk_clist_set_column_title(GTK_CLIST(clist), 5, "Memory use");

    /* Set clist column width */
    gtk_clist_set_column_width(GTK_CLIST(clist), 0, 75);
    gtk_clist_set_column_width(GTK_CLIST(clist), 1, 125);
    gtk_clist_set_column_width(GTK_CLIST(clist), 2, 75);
    gtk_clist_set_column_width(GTK_CLIST(clist), 3, 75);
    gtk_clist_set_column_width(GTK_CLIST(clist), 4, 75);
    gtk_clist_set_column_width(GTK_CLIST(clist), 5, 125);
    gtk_clist_column_titles_show(GTK_CLIST(clist));

    /* Read proc info */
    dir = opendir("/proc");
    process_num = 0;
    while (dir_info = readdir(dir)) {
        /*
         * If start with number, then read it

```

```

        */
    if ((dir_info->d_name)[0] >= '0' && ((dir_info->d_name)[0]) <= '9') {
        sprintf(pid_file, "/proc/%s/stat", dir_info->d_name);
        fd = open(pid_file, O_RDONLY);
        read(fd, stat_file, 1024);
        close(fd);
        one_file = stat_file;
        read_stat(info, one_file);
        for (int i = 0; i < 6; i++)
            txt[i] = utf8_fix(info[i]);
        gtk_clist_append(GTK_CLIST(clist), txt);
        process_num++;
    }
}
closedir(dir);
}

void get_modules_info(void) {
    FILE *fp;
    char modules_info[1024];
    char *line = NULL;
    char info[3][1024];
    gchar *txt[3];
    int pos = 0;

    /* Set clist column title */
    gtk_clist_set_column_title(GTK_CLIST(clist2), 0, "Module name");
    gtk_clist_set_column_title(GTK_CLIST(clist2), 1, "Memory use");
    gtk_clist_set_column_title(GTK_CLIST(clist2), 2, "Used Times");

    /* Set clist column width */
    gtk_clist_set_column_width(GTK_CLIST(clist2), 0, 250);
    gtk_clist_set_column_width(GTK_CLIST(clist2), 1, 150);
    gtk_clist_set_column_width(GTK_CLIST(clist2), 2, 150);
    gtk_clist_column_titles_show(GTK_CLIST(clist2));
}

```

```

fp = fopen("/proc/modules", "r");

while ((line = fgets(modules_info, sizeof(modules_info), fp)) != NULL) {
    /* Read modules name */
    for (pos = 0; pos < 1024; pos++) {
        if (line[pos] == ' ')
            break;
    }
    line[pos] = '\0';
    strcpy(info[0], line);
    pos++;
    line += pos;

    /* Read modules memory use */
    for (pos = 0; pos < 1024; pos++) {
        if (line[pos] == ' ')
            break;
    }
    line[pos] = '\0';
    strcpy(info[1], line);
    pos++;
    line += pos;

    /* Read modules name */
    for (pos = 0; pos < 1024; pos++) {
        if (line[pos] == ' ')
            break;
    }
    line[pos] = '\0';
    strcpy(info[2], line);

    for (int i = 0; i < 3; i++)
        txt[i] = utf8_fix(info[i]);
}

```

```

        gtk_clist_append(GTK_CLIST(clist2), txt);
    }
    fclose(fp);
}

/*
 * get_cpu_info - get cpu information from /proc/cpuinfo
 */
void get_cpu_info(char *cpu_name,
                  char *cpu_addr_digit,
                  char *cpu_cache_size,
                  char *cpu_core) {
    /*
     * cpuinfo file format:
     *
     * model name: line 5
     * cache size: line 9
     * cpu cores: line 13
     * address sizes: line 25
     */
    int fd;
    char info_buf[2048]; /* Here 1024 is too small */
    char info_str[1024];
    char *pos = NULL;
    int i;
    fd = open("/proc/cpuinfo", O_RDONLY);
    read(fd, info_buf, sizeof(info_buf));
    close(fd);

    /* Read CPU Name */
    i = 0;
    pos = strstr(info_buf, "model name");
    while (*pos != ':')
        pos++;
    pos += 2;

```

```

while (*pos != '\n') {
    info_str[i] = *pos;
    i++;
    pos++;
}
info_str[i] = '\0';
strcpy(cpu_name, info_str);

/* Read Cache size */
i = 0;
pos = strstr(info_buf, "cache size");
while (*pos != ':')
    pos++;
pos += 2;
while (*pos != '\n') {
    info_str[i] = *pos;
    i++;
    pos++;
}
info_str[i] = '\0';
strcpy(cpu_cache_size, info_str);

/* Read CPU Cores */
i = 0;
pos = strstr(info_buf, "cpu cores");
while (*pos != ':')
    pos++;
pos += 2;
while (*pos != '\n') {
    info_str[i] = *pos;
    i++;
    pos++;
}
info_str[i] = '\0';

```

```

strcpy(cpu_core, info_str);

/* Read Addressing digital */
i = 0;
pos = strstr(info_buf, "address sizes");
while (*pos != ':')
    pos++;
pos += 2;
while (*pos != '\n') {
    info_str[i] = *pos;
    i++;
    pos++;
}
info_str[i] = '\0';
strcpy(cpu_addr_digit, info_str);
}

```

```

int main(int argc, char **argv) {
    gtk_init(&argc, &argv);
    /* GTK widgets */
    GtkWidget *top_window;
    GtkWidget *notebook;
    GtkWidget *hbox;
    GtkWidget *vbox;
    GtkWidget *label;
    GtkWidget *label1;
    GtkWidget *label2;
    GtkWidget *label3;
    GtkWidget *scrolled_window;
    GtkWidget *frame;
    GtkWidget *cpu_use;
    GtkWidget *mem_use;
    GtkWidget *swap_use;
    GtkWidget *button1;

```



```

GtkWidget *button2;
GtkWidget *button3;
GtkWidget *fixed;
GtkWidget *image;

/* Save page title */
char title_buf[1024];

/* Buffer to use */
char buffer1[1024];
char cpu_name[1024];
char cpu_addr_digit[1024];
char cpu_cache_size[1024];
char cpu_core[1024];

/* Create top window */
top_window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
gtk_window_set_title(GTK_WINDOW(top_window), "System monitor");
gtk_widget_set_size_request(top_window, 600, 700);
g_signal_connect(G_OBJECT(top_window), "delete_event",
G_CALLBACK(gtk_main_quit), NULL);
gtk_container_set_border_width(GTK_CONTAINER(top_window), 10);

/* Create notebook */
notebook = gtk_notebook_new();
gtk_container_add(GTK_CONTAINER(top_window), notebook);
gtk_notebook_set_tab_pos(GTK_NOTEBOOK(notebook), GTK_POS_TOP);

/*
 * Page 1: Process manage
 */
sprintf(title_buf, "Process");
vbox = gtk_vbox_new(FALSE, 0);

```

```

/* Create scrolled window for process info */
scrolled_window = gtk_scrolled_window_new(NULL, NULL);
gtk_widget_set_size_request(scrolled_window, 550, 500);

gtk_scrolled_window_set_policy(GTK_SCROLLED_WINDOW(scrolled_window),
GTK_POLICY_AUTOMATIC, GTK_POLICY_AUTOMATIC);

/* Create list with 6 columns */
clist = gtk_clist_new(6);
get_process_info();
gtk_signal_connect(GTK_OBJECT(clist), "select_row",
GTK_SIGNAL_FUNC(select_row_callback), NULL);

gtk_scrolled_window_add_with_viewport(GTK_SCROLLED_WINDOW(scrolled_w
indow), clist);

gtk_box_pack_start(GTK_BOX(vbox), scrolled_window, TRUE, TRUE, 5);

/* Create buttons */
hbox = gtk_hbox_new(FALSE, 10);
entry = gtk_entry_new();
gtk_entry_set_max_length(GTK_ENTRY(entry), 0);
button1 = gtk_button_new_with_label("Search by PID");
button2 = gtk_button_new_with_label("Kill");
button3 = gtk_button_new_with_label("Refresh");
g_signal_connect(G_OBJECT(button1), "clicked",
G_CALLBACK(search_proc), scrolled_window);
g_signal_connect(G_OBJECT(button2), "clicked", G_CALLBACK(kill_proc),
NULL);
g_signal_connect(G_OBJECT(button3), "clicked",
G_CALLBACK(refresh_proc), NULL);

gtk_widget_set_size_request(entry, 200, 30);
gtk_widget_set_size_request(button1, 120, 30);
gtk_widget_set_size_request(button2, 80, 30);
gtk_widget_set_size_request(button3, 80, 30);
gtk_box_pack_start(GTK_BOX(hbox), entry, FALSE, FALSE, 5);

```

```

gtk_box_pack_start(GTK_BOX(hbox), button1, FALSE, FALSE, 5);
gtk_box_pack_start(GTK_BOX(hbox), button2, FALSE, FALSE, 5);
gtk_box_pack_start(GTK_BOX(hbox), button3, FALSE, FALSE, 5);
gtk_box_pack_start(GTK_BOX(vbox), hbox, FALSE, FALSE, 5);
label = gtk_label_new(title_buf);
gtk_notebook_append_page(GTK_NOTEBOOK(notebook), vbox, label);

/*
 * Page 2: CPU info
 */
sprintf(title_buf, "CPU");
vbox = gtk_vbox_new(FALSE, 0);

/* Creat frame to show curve of CPU use */
hbox = gtk_hbox_new(FALSE, 0);
gtk_box_pack_start(GTK_BOX(vbox), hbox, TRUE, FALSE, 5);
cpu_use = gtk_frame_new("CPU Use");
gtk_container_set_border_width(GTK_CONTAINER(cpu_use), 5);
gtk_widget_set_size_request(cpu_use, 520, 300);
gtk_box_pack_start(GTK_BOX(hbox), cpu_use, TRUE, FALSE, 5);

hbox = gtk_hbox_new(FALSE, 0);
label1 = gtk_label_new("");
label2 = gtk_label_new("");
gtk_box_pack_start(GTK_BOX(hbox), label1, TRUE, FALSE, 5);
gtk_box_pack_start(GTK_BOX(hbox), label2, TRUE, FALSE, 5);
g_timeout_add(1000, (GtkFunction)get_cpu_ratio, (gpointer)label1);
g_timeout_add(1000, (GtkFunction)get_cpu_mhz, (gpointer)label2);
gtk_widget_set_size_request(hbox, 550, 30);
gtk_box_pack_start(GTK_BOX(vbox), hbox, TRUE, FALSE, 5);

/* Draw CPU use curve */
cpu_curve = gtk_drawing_area_new();
gtk_widget_set_size_request(cpu_curve, 0, 0);

```

```

        g_signal_connect(G_OBJECT(cpu_curve), "expose_event",
G_CALLBACK(cpu_curve_callback), NULL);
        gtk_container_add(GTK_CONTAINER(cpu_use), cpu_curve);

/* Creat frame to show CPU info */
        frame = gtk_frame_new("CPU Information");
        gtk_widget_set_size_request(frame, 500, 200);
        gtk_box_pack_start(GTK_BOX(vbox), frame, TRUE, TRUE, 5);
        get_cpu_info(cpu_name, cpu_addr_digit, cpu_cache_size, cpu_core);
        sprintf(buffer1, "CPU Type and Frequency : \n%s\n\nAddressing
digit : %s\n\nCache size : %s\n\nCPU Core : %s\n",
                cpu_name, cpu_addr_digit, cpu_cache_size, cpu_core);
        label = gtk_label_new(buffer1);
        set_label_fontsize(label, "12");
        gtk_container_add(GTK_CONTAINER(frame), label);

        label = gtk_label_new(title_buf);
        gtk_notebook_append_page(GTK_NOTEBOOK(notebook), vbox, label);

/*
    * Page 3: Memory info
    */
        sprintf(title_buf, "Memory");
        vbox = gtk_vbox_new(FALSE, 0);

/* Create frame to show curve of memory use */
        hbox = gtk_hbox_new(FALSE, 0);
        gtk_box_pack_start(GTK_BOX(vbox), hbox, TRUE, FALSE, 5);
        mem_use = gtk_frame_new("Memory Use");
        gtk_container_set_border_width(GTK_CONTAINER(mem_use), 5);
        gtk_widget_set_size_request(mem_use, 520, 250);
        gtk_box_pack_start(GTK_BOX(hbox), mem_use, TRUE, FALSE, 5);

        hbox = gtk_hbox_new(FALSE, 0);

```

```

label1 = gtk_label_new("");
label2 = gtk_label_new("");
gtk_box_pack_start(GTK_BOX(hbox), label1, TRUE, FALSE, 5);
gtk_box_pack_start(GTK_BOX(hbox), label2, TRUE, FALSE, 5);
g_timeout_add(1000, (GtkFunction)get_memory_ratio, (gpointer)label1);
g_timeout_add(1000, (GtkFunction)get_memory_fraction, (gpointer)label2);
gtk_widget_set_size_request(hbox, 550, 20);
gtk_box_pack_start(GTK_BOX(vbox), hbox, TRUE, FALSE, 5);

/* Draw Memory use curve */
mem_curve = gtk_drawing_area_new();
gtk_widget_set_size_request(mem_curve, 0, 0);
g_signal_connect(G_OBJECT(mem_curve), "expose_event",
G_CALLBACK(mem_curve_callback), NULL);
gtk_container_add(GTK_CONTAINER(mem_use), mem_curve);

/* Create frame to show curve of swap use */
hbox = gtk_hbox_new(FALSE, 0);
gtk_box_pack_start(GTK_BOX(vbox), hbox, TRUE, FALSE, 5);
swap_use = gtk_frame_new("Swap Use");
gtk_container_set_border_width(GTK_CONTAINER(swap_use), 5);
gtk_widget_set_size_request(swap_use, 520, 250);
gtk_box_pack_start(GTK_BOX(hbox), swap_use, TRUE, FALSE, 5);

hbox = gtk_hbox_new(FALSE, 0);
label1 = gtk_label_new("");
label2 = gtk_label_new("");
gtk_box_pack_start(GTK_BOX(hbox), label1, TRUE, FALSE, 5);
gtk_box_pack_start(GTK_BOX(hbox), label2, TRUE, FALSE, 5);
g_timeout_add(1000, (GtkFunction)get_swap_ratio, (gpointer)label1);
g_timeout_add(1000, (GtkFunction)get_swap_fraction, (gpointer)label2);
gtk_widget_set_size_request(hbox, 550, 20);
gtk_box_pack_start(GTK_BOX(vbox), hbox, TRUE, FALSE, 5);

/* Draw Swap use curve */

```

```

swap_curve = gtk_drawing_area_new();
gtk_widget_set_size_request(swap_curve, 0, 0);
g_signal_connect(G_OBJECT(swap_curve), "expose_event",
G_CALLBACK(swap_curve_callback), NULL);
gtk_container_add(GTK_CONTAINER(swap_use), swap_curve);

label = gtk_label_new(title_buf);
gtk_notebook_append_page(GTK_NOTEBOOK(notebook), vbox, label);

/*
 * Page 4: Modules
 */
sprintf(title_buf, "Modules");
vbox = gtk_vbox_new(FALSE, 0);

/* Create scrolled window for modules info */
scrolled_window = gtk_scrolled_window_new(NULL, NULL);
gtk_widget_set_size_request(scrolled_window, 550, 500);

gtk_scrolled_window_set_policy(GTK_SCROLLED_WINDOW(scrolled_window),
GTK_POLICY_AUTOMATIC, GTK_POLICY_AUTOMATIC);

/* Create list with 3 columns */
clist2 = gtk_clist_new(3);
get_modules_info();

gtk_scrolled_window_add_with_viewport(GTK_SCROLLED_WINDOW(scrolled_w
indow), clist2);
gtk_box_pack_start(GTK_BOX(vbox), scrolled_window, TRUE, TRUE, 5);

/* Create buttons */
fixed = gtk_fixed_new();
button1 = gtk_button_new_with_label("Refresh");
g_signal_connect(G_OBJECT(button1), "clicked",

```

```

G_CALLBACK(refresh_modules), NULL);

gtk_widget_set_size_request(button1, 80, 30);
gtk_fixed_put(GTK_FIXED(fixed), button1, 450, 0);
gtk_box_pack_start(GTK_BOX(vbox), fixed, FALSE, FALSE, 5);

label = gtk_label_new(title_buf);
gtk_notebook_append_page(GTK_NOTEBOOK(notebook), vbox, label);

/*
 * Page 5: System info
 */
sprintf(title_buf, "System");
vbox = gtk_vbox_new(FALSE, 10);

frame = gtk_frame_new("System Information");
label1 = gtk_label_new("");
gtk_container_add(GTK_CONTAINER(frame), label1);
g_timeout_add(1000, (GtkFunction)get_sys_info, (gpointer)label1);

gtk_box_pack_start(GTK_BOX(vbox), frame, TRUE, FALSE, 5);

frame = gtk_frame_new("Network");
label2 = gtk_label_new("");
gtk_container_add(GTK_CONTAINER(frame), label2);
g_timeout_add(2000, (GtkFunction)get_network_info, (gpointer)label2);
gtk_widget_set_size_request(frame, 550, 180);
gtk_box_pack_start(GTK_BOX(vbox), frame, TRUE, FALSE, 0);

frame = gtk_frame_new("Disk");
label3 = gtk_label_new("");
gtk_container_add(GTK_CONTAINER(frame), label3);
g_timeout_add(2000, (GtkFunction)get_disk_info, (gpointer)label3);
gtk_widget_set_size_request(frame, 550, 180);
gtk_box_pack_start(GTK_BOX(vbox), frame, TRUE, FALSE, 0);

```

```

label = gtk_label_new(title_buf);
gtk_notebook_append_page(GTK_NOTEBOOK(notebook), vbox, label);


/* GTK show */
gtk_widget_show_all(top_window);
gtk_main();

return 0;
}

/*****CALLBACKS*****/

/*
 * select_row_callback - Callback for select_row
 */
void select_row_callback(GtkWidget *clist, gint row, gint column,
GdkEventButton *event, gpointer data) {
    gtk_clist_get_text(GTK_CLIST(clist), row, 0, &now_pid);
    gtk_entry_set_text(GTK_ENTRY(entry), (gchar *)now_pid);
    return;
}

/*
 * select_row_callback - Search process in GTK clist
 */
void search_proc(GtkButton *button, gpointer data) {
    const gchar *entry_txt;
    gchar *txt;
    gint ret, i = 0;
    entry_txt = gtk_entry_get_text(GTK_ENTRY(entry));
    while ((ret = gtk_clist_get_text(GTK_CLIST(clist), i, 0, &txt)) != 0) {

```



```

        if (!strcmp(entry_txt, txt))
            break;
        i++;
    }
    gtk_clist_select_row(GTK_CLIST(clist), i, 0);
    scroll_to_line(data, process_num, i);
    return;
}

/*
 * select_row_callback - Kill process clicked
 */
void kill_proc(void) {
    int ret;
    if (now_pid != NULL) {
        ret = kill(atoi(now_pid), SIGKILL);
        if (ret == -EPERM) {
            popup_window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
            popup_label = gtk_label_new("You need root privilege\n    to kill this
process!");
            set_label_fontsize(popup_label, "14");
            gtk_widget_set_size_request(popup_window, 300, 180);
            gtk_container_add(GTK_CONTAINER(popup_window), popup_label);
            gtk_window_set_title(GTK_WINDOW(popup_window), "ERROR!");
            gtk_widget_show_all(popup_window);
        }
    }
    return;
}

/*
 * refresh_proc - Refresh the process info in clist
 */
void refresh_proc(void) {
    gtk_clist_freeze(GTK_CLIST(clist));

```

```

    gtk_clist_clear(GTK_CLIST(clist));
    get_process_info();
    gtk_clist_thaw(GTK_CLIST(clist));
    gtk_clist_select_row(GTK_CLIST(clist), 0, 0);
    return;
}

/*
 * cpu_curve_callback - Refresh the CPU use curve once every second
 */

gboolean cpu_curve_callback(GtkWidget *widget, GdkEventExpose *event,
gpointer data) {
    static int flag = 0;
    draw_cpu_curve((gpointer)widget);
    if (flag == 0) {
        g_timeout_add(1000, (GtkFunction)draw_cpu_curve, (gpointer)widget);
        flag = 1;
    }
    return TRUE;
}

/*
 * mem_curve_callback - Refresh the Memory use curve once every second
 */

gboolean mem_curve_callback(GtkWidget *widget, GdkEventExpose *event,
gpointer data) {
    static int flag = 0;
    draw_mem_curve((gpointer)widget);
    if (flag == 0) {
        g_timeout_add(1000, (GtkFunction)draw_mem_curve, (gpointer)widget);
        flag = 1;
    }
    return TRUE;
}

```

```

/*
 * swap_curve_callback - Refresh the Swap use curve once every second
 */
gboolean swap_curve_callback(GtkWidget *widget, GdkEventExpose *event,
gpointer data) {
    static int flag = 0;
    draw_swap_curve((gpointer)widget);
    if (flag == 0) {
        g_timeout_add(1000, (GtkFunction)draw_swap_curve, (gpointer)widget);
        flag = 1;
    }
    return TRUE;
}

/*
 * refresh_modules - Refresh the modules info in clist
 */
void refresh_modules(void) {
    gtk_clist_freeze(GTK_CLIST(clist2));
    gtk_clist_clear(GTK_CLIST(clist2));
    get_modules_info();
    gtk_clist_thaw(GTK_CLIST(clist2));
    gtk_clist_select_row(GTK_CLIST(clist2), 0, 0);
    return;
}

/*****LOOPS*****/

/*
 * draw_cpu_curve - Draw CPU curve
 */
gboolean draw_cpu_curve(gpointer widget) {
    GtkWidget *cpu_curve = (GtkWidget *)widget;
    GdkColor color;
    GdkGC *gc = cpu_curve->style->fg_gc[GTK_WIDGET_STATE(widget)];

```

```

static int flag = 0;
static int now_pos = 0;
int draw_pos = 0;

/* Darw background */
color.red = 0;
color.green = 0;
color.blue = 0;
gdk_gc_set_rgb_fg_color(gc, &color);
gdk_draw_rectangle(cpu_curve->window, gc, TRUE, 15, 30, 480, 200);

/* Draw background lines */
color.red = 0;
color.green = 20000;
color.blue = 0;
gdk_gc_set_rgb_fg_color(gc, &color);
for (int i = 30; i <= 220; i += 20)
    gdk_draw_line(cpu_curve->window, gc, 15, i, 495, i);
for (int i = 15; i <= 480; i += 20)
    gdk_draw_line(cpu_curve->window, gc, i + cpu_curve_start, 30, i +
cpu_curve_start, 230);

/* Settle cpu curve start position to make it live */
cpu_curve_start -= 4;
if (cpu_curve_start == 0)
    cpu_curve_start = 20;

/* Initial data */
if (flag == 0) {
    for (int i = 0; i < 120; i++) {
        cpu_ratio_data[i] = 0;
        flag = 1;
    }
}

```

```

/* Add data */
cpu_ratio_data[now_pos] = cpu_ratio / 100;
now_pos++;
if (now_pos == 120)
    now_pos = 0;

/* Draw lines */
color.red = 0;
color.green = 65535;
color.blue = 0;
gdk_gc_set_rgb_fg_color(gc, &color);
draw_pos = now_pos;
for (int i = 0; i < 119; i++) {
    gdk_draw_line(cpu_curve->window, gc,
                  15 + i * 4, 230 - 200 * cpu_ratio_data[draw_pos % 120],
                  15 + (i + 1) * 4, 230 - 200 * cpu_ratio_data[(draw_pos +
1) % 120]);
    draw_pos++;
    if (draw_pos == 120)
        draw_pos = 0;
}

/* Reset the color */
color.red = 25000;
color.green = 25000;
color.blue = 25000;
gdk_gc_set_rgb_fg_color(gc, &color);

/* To loop this function, it must return TRUE */
return TRUE;
}

/*
* draw_mem_curve - Draw Memory use curve
*/

```

```

gboolean draw_mem_curve(gpointer widget) {
    GtkWidget *mem_curve = (GtkWidget *)widget;
    GdkColor color;
    GdkGC *gc = mem_curve->style->fg_gc[GTK_WIDGET_STATE(widget)];
    static int flag = 0;
    static int now_pos = 0;
    int draw_pos = 0;

    /* Darw background */
    color.red = 0;
    color.green = 0;
    color.blue = 0;
    gdk_gc_set_rgb_fg_color(gc, &color);
    gdk_draw_rectangle(mem_curve->window, gc, TRUE, 15, 10, 480, 200);

    /* Draw background lines */
    color.red = 0;
    color.green = 20000;
    color.blue = 0;
    gdk_gc_set_rgb_fg_color(gc, &color);
    for (int i = 10; i <= 230; i += 20)
        gdk_draw_line(mem_curve->window, gc, 15, i, 495, i);
    for (int i = 15; i <= 480; i += 20)
        gdk_draw_line(mem_curve->window, gc, i + mem_curve_start, 10, i +
mem_curve_start, 210);

    /* Settle memory curve start position to make it live */
    mem_curve_start -= 4;
    if (mem_curve_start == 0)
        mem_curve_start = 20;

    /* Initial data */
    if (flag == 0) {
        for (int i = 0; i < 120; i++) {
            mem_ratio_data[i] = 0;

```

```

        flag = 1;
    }
}

/* Add data */
mem_ratio_data[now_pos] = mem_ratio / 100;
now_pos++;
if (now_pos == 120)
    now_pos = 0;

/* Draw lines */
color.red = 0;
color.green = 65535;
color.blue = 0;
gdk_gc_set_rgb_fg_color(gc, &color);
draw_pos = now_pos;
for (int i = 0; i < 119; i++) {
    gdk_draw_line(mem_curve->window, gc,
                  15 + i * 4, 210 - 200 * mem_ratio_data[draw_pos % 120],
                  15 + (i + 1) * 4, 210 - 200 * mem_ratio_data[(draw_pos +
1) % 120]);
    draw_pos++;
    if (draw_pos == 120)
        draw_pos = 0;
}

/* Reset the color */
color.red = 25000;
color.green = 25000;
color.blue = 25000;
gdk_gc_set_rgb_fg_color(gc, &color);

/* To loop this function, it must return TRUE */
return TRUE;
}

```

```

/*
 * draw_swap_curve - Draw Swap use curve
 */
gboolean draw_swap_curve(gpointer widget) {
    GtkWidget *swap_curve = (GtkWidget *)widget;
    GdkColor color;
    GdkGC *gc = swap_curve->style->fg_gc[GTK_WIDGET_STATE(widget)];
    static int flag = 0;
    static int now_pos = 0;
    int draw_pos = 0;

    /* Darw background */
    color.red = 0;
    color.green = 0;
    color.blue = 0;
    gdk_gc_set_rgb_fg_color(gc, &color);
    gdk_draw_rectangle(swap_curve->window, gc, TRUE, 15, 10, 480, 200);

    /* Draw background lines */
    color.red = 0;
    color.green = 20000;
    color.blue = 0;
    gdk_gc_set_rgb_fg_color(gc, &color);
    for (int i = 10; i <= 230; i += 20)
        gdk_draw_line(swap_curve->window, gc, 15, i, 495, i);
    for (int i = 15; i <= 480; i += 20)
        gdk_draw_line(swap_curve->window, gc, i + mem_curve_start, 10, i +
mem_curve_start, 210);

    /* Settle memory curve start position to make it live */
    swap_curve_start -= 4;
    if (swap_curve_start == 0)
        swap_curve_start = 20;

```



```

/* Initial data */
if (flag == 0) {
    for (int i = 0; i < 120; i++) {
        swap_ratio_data[i] = 0;
        flag = 1;
    }
}

/* Add data */
swap_ratio_data[now_pos] = swap_ratio / 100;
now_pos++;
if (now_pos == 120)
    now_pos = 0;

/* Draw lines */
color.red = 0;
color.green = 65535;
color.blue = 0;
gdk_gc_set_rgb_fg_color(gc, &color);
draw_pos = now_pos;
for (int i = 0; i < 119; i++) {
    gdk_draw_line(swap_curve->window, gc,
                  15 + i * 4, 210 - 200 * swap_ratio_data[draw_pos % 120],
                  15 + (i + 1) * 4, 210 - 200 * swap_ratio_data[(draw_pos +
1) % 120]);
    draw_pos++;
    if (draw_pos == 120)
        draw_pos = 0;
}

/* Reset the color */
color.red = 0;
color.green = 0;
color.blue = 0;
gdk_gc_set_rgb_fg_color(gc, &color);

```

```

    /* To loop this function, it must return TRUE */
    return TRUE;
}

/*
 * get_cpu_ratio - Get CPU use ratio from /proc/stat
 */
gboolean get_cpu_ratio(gpointer label) {
    /*
     * stat file format
     *
     * cpu user nice system idle iowait
     *
     * t1, t2: Two near moments
     * cpu(total) = user+nice+system+idle+iowait+irq+softirq
     * pcpu = 100 *(total - idle) / total
     * total =total(t2) - total(t1)
     * idle =idle(t2) - idle(t1)
     */
    /* Data holder */
    static long old_idle, old_total;
    static int flag = 0;

    long user, nice, system, idle, iowait, irq, softirq, total;
    float total_diff, idle_diff;
    char cpu[10], buffer[256], cpu_ratio_char[256];
    int fd;
    fd = open("/proc/stat", O_RDONLY);
    read(fd, buffer, sizeof(buffer));
    close(fd);
    sscanf(buffer, "%s %ld %ld %ld %ld %ld %ld %ld", cpu, &user, &nice,
&system, &idle, &iowait, &irq, &softirq);

    /* First */

```

```

if (flag == 0) {
    flag = 1;
    old_idle = idle;
    old_total = user + nice + system + idle + iowait + irq + softirq;
    cpu_ratio = 0;
}
/* Others */
else {
    total = user + nice + system + idle + iowait + irq + softirq;
    total_diff = total - old_total;
    idle_diff = idle - old_idle;
    cpu_ratio = 100 * (total_diff - idle_diff) / total_diff;
    total = old_total;
    idle = old_idle;
}
sprintf(cpu_ratio_char, "CPU usage: %0.1f%%", cpu_ratio);
gtk_label_set_text(GTK_LABEL(label), cpu_ratio_char);
return TRUE;
}

/*
 * get_cpu_mhz - Get CPU frequency from /proc/cpuinfo line 8
 */
gboolean get_cpu_mhz(gpointer label) {
    int fd;
    char info_buf[1024];
    char info_str[1024];
    char cpu_freq_char[256];
    char *pos = NULL;
    int i;
    fd = open("/proc/cpuinfo", O_RDONLY);
    read(fd, info_buf, sizeof(info_buf));
    close(fd);

    /* Read CPU mhz */

```

```

i = 0;
pos = strstr(info_buf, "cpu MHz");
while (*pos != ':')
    pos++;
pos += 2;

while (*pos != '\n') {
    info_str[i] = *pos;
    i++;
    pos++;
}
info_str[i] = '\0';
strcpy(cpu_freq, info_str);
sprintf(cpu_freq_char, "CPU frequency: %s MHz", cpu_freq);
gtk_label_set_text(GTK_LABEL(label), cpu_freq_char);
return TRUE;
}

/*
 * get_memory_ratio - Get memory use ratio in /proc/meminfo
 */
gboolean get_memory_ratio(gpointer label) {
    int fd;
    char mem_buf[1024];
    char mem_total_char[1024];
    char mem_free_char[1024];
    char mem_ratio_char[1024];
    char *pos = NULL;
    int i;
    fd = open("/proc/meminfo", O_RDONLY);
    read(fd, mem_buf, sizeof(mem_buf));
    close(fd);

    /* Read memory total */
    i = 0;

```

```

pos = strstr(mem_buf, "MemTotal");
while (*pos != ':')
    pos++;
pos += 1;
while (*pos == ' ')
    pos++;

while (*pos != ' ') {
    mem_total_char[i] = *pos;
    i++;
    pos++;
}
mem_total_char[i] = '\0';
mem_total = atof(mem_total_char) / (1024 * 1024);

/* Read memory free */
i = 0;
pos = strstr(mem_buf, "MemAvailable");
while (*pos != ':')
    pos++;
pos += 1;
while (*pos == ' ')
    pos++;

while (*pos != ' ') {
    mem_free_char[i] = *pos;
    i++;
    pos++;
}
mem_free_char[i] = '\0';
mem_free = atof(mem_free_char) / (1024 * 1024);

/* Get memory use ratio */
mem_ratio = 100 - (mem_free / mem_total) * 100;
sprintf(mem_ratio_char, "Memory usage: %0.1f%%", mem_ratio);

```

```

    gtk_label_set_text(GTK_LABEL(label), mem_ratio_char);
    return TRUE;
}

/*
 * get_memory_fraction - Set memory use fraction
 */
gboolean get_memory_fraction(gpointer label) {
    char mem_fraction[1024];
    sprintf(mem_fraction, "%0.2f / %0.2f GB", mem_total - mem_free,
mem_total);
    gtk_label_set_text(GTK_LABEL(label), mem_fraction);
    return TRUE;
}

/*
 * get_swap_ratio - Get sawp use ratio in /proc/meminfo
 */
gboolean get_swap_ratio(gpointer label) {
    int fd;
    char swap_buf[1024];
    char swap_total_char[1024];
    char swap_free_char[1024];
    char swap_ratio_char[1024];
    char *pos = NULL;
    int i;
    fd = open("/proc/meminfo", O_RDONLY);
    read(fd, swap_buf, sizeof(swap_buf));
    close(fd);

    /* Read swap total */
    i = 0;
    pos = strstr(swap_buf, "SwapTotal");
    while (*pos != ':')
        pos++;

```

```

pos += 1;
while (*pos == ' ')
    pos++;

while (*pos != ' ') {
    swap_total_char[i] = *pos;
    i++;
    pos++;
}
swap_total_char[i] = '\0';
swap_total = atof(swap_total_char) / (1024 * 1024);

/* Read swap free */
i = 0;
pos = strstr(swap_buf, "SwapFree");
while (*pos != ':')
    pos++;
pos += 1;
while (*pos == ' ')
    pos++;

while (*pos != ' ') {
    swap_free_char[i] = *pos;
    i++;
    pos++;
}
swap_free_char[i] = '\0';
swap_free = atof(swap_free_char) / (1024 * 1024);

/* Get swap use ratio */
swap_ratio = 100 - (swap_free / swap_total) * 100;
sprintf(swap_ratio_char, "Swap usage: %0.1f%%", swap_ratio);
gtk_label_set_text(GTK_LABEL(label), swap_ratio_char);
return TRUE;
}

```

```

/*
 * get_swap_fraction - Set swap use fraction
 */

gboolean get_swap_fraction(gpointer label) {
    char swap_fraction[1024];
    sprintf(swap_fraction, "%0.2f / %0.2f GB", swap_total - swap_free,
swap_total);
    gtk_label_set_text(GTK_LABEL(label), swap_fraction);
    return TRUE;
}

/*
 * get_sys_info - Set system information
 */

gboolean get_sys_info(gpointer label) {
    int fd;
    FILE *fp;
    int i, j;
    char buffer[1024];
    char *pos = NULL;
    char host_name[128];
    char os_name[128];
    int os_type;
    char kernel_version[128];
    char gcc_version[128];
    int setup_time;
    int uphour, upminute, upsecond;

    /* Get host name */
    fp = fopen("/etc/hostname", "r");
    fgets(host_name, sizeof(host_name), fp);
    fclose(fp);

    /* Get os name */

```



```

memset(buffer, 0, sizeof(buffer));
fd = open("/etc/issue", O_RDONLY);
read(fd, buffer, sizeof(buffer));
close(fd);
for (i = 0; i < 1024; i++) {
    if (buffer[i] == '\\')
        break;
}
buffer[i - 1] = '\0';
strcpy(os_name, buffer);

/* Get os type */
os_type = sizeof(char *) * 8;

/* Get kernel version */
fp = fopen("/proc/sys/kernel/osrelease", "r");
fgets(kernel_version, sizeof(kernel_version), fp);
fclose(fp);

/* Get gcc version */
memset(buffer, 0, sizeof(buffer));
fd = open("/proc/version", O_RDONLY);
read(fd, buffer, sizeof(buffer));
close(fd);
pos = strstr(buffer, "gcc version");
for (i = 0, j = 0; i < 1024; i++) {
    if (*pos == ' ')
        j++;
    if (j == 2)
        break;
    pos++;
}
pos++;
for (i = 0; i < 1024; i++) {
    if (pos[i] == ' ')

```

```

        break;
        gcc_version[i] = pos[i];
    }
    gcc_version[i] = '\0';

    /* Get setup time */
    memset(buffer, 0, sizeof(buffer));
    fd = open("/proc/uptime", O_RDONLY);
    read(fd, buffer, sizeof(buffer));
    close(fd);
    for (i = 0; i < 1024; i++) {
        if (buffer[i] == ' ')
            break;
    }
    buffer[i] = '\0';
    setup_time = atoi(buffer);
    upsecond = setup_time % 60;
    upminute = (setup_time / 60) % 60;
    uphour = setup_time / 3600;

    sprintf(buffer, "Hostname:          %s\n\
OS Name:          %s\n\n\
OS Type:          %d-bit\n\n\
Kernel Version:   %s\n\
GCC Version:      %s\n\n\
Uptime:           %02d:%02d:%02d",
            host_name, os_name, os_type, kernel_version, gcc_version, uphour,
            upminute, upsecond);

    gtk_label_set_text(GTK_LABEL(label), buffer);
    set_label_fontsize((GtkWidget *)label, "13");

    return TRUE;
}

```

```

/*
 * get_network_info - Get network information in /proc/net/dev
 */
gboolean get_network_info(gpointer label) {
    /*
     * Network dev file format:
     *
     *          Inter-|          Receive
| Transmit
     *  face |bytes   packets errs drop fifo frame compressed multicast|bytes
packets errs drop fifo colls carrier compressed
     *   dev1: ...
     *   dev2: ...
     *   ...
     */
    FILE *fp;
    long r_byte, s_byte;
    long receive_byte, send_byte;
    float receive_diff, send_diff;
    static long old_receive, old_send;
    static int flag = 0;
    char buffer[256];
    char *pos;

    fp = fopen("/proc/net/dev", "r");
    fgets(buffer, sizeof(buffer), fp);
    fgets(buffer, sizeof(buffer), fp);

    receive_byte = 0;
    send_byte = 0;

    while ((pos = fgets(buffer, sizeof(buffer), fp)) != NULL) {
        while (*pos != ':')
            pos++;
        pos++;
    }

```

```

        sscanf(pos, "%ld %*d %*d %*d %*d %*d %*d %*d %ld", &r_byte,
&s_byte);
        receive_byte += r_byte;
        send_byte += s_byte;
    }
    fclose(fp);

    if (flag == 0) {
        old_receive = receive_byte;
        old_send = send_byte;
        receive_diff = 0;
        send_diff = 0;
        flag = 1;
    }
    else {
        receive_diff = receive_byte - old_receive;
        send_diff = send_byte - old_send;
        old_receive = receive_byte;
        old_send = send_byte;
    }

    /* Count speed (KB) */
    receive_speed = receive_diff / (1024 * 2);
    send_speed = send_diff / (1024 * 2);

    sprintf(buffer, "      Network Speed:\n\n\
\tUpload speed: %7.1f KB/s\tDownload speed: %7.1f KB/s\n\n\
\t  Total upload: %7.1f MB\t\t  Total download: %7.1f MB",
        send_speed, receive_speed, send_byte / (1024.0 * 1024.0),
receive_byte / (1024.0 * 1024.0));
    gtk_label_set_text(GTK_LABEL(label), buffer);
    set_label_fontsize((GtkWidget *)label, "13");

    return TRUE;
}

```

```

/*
 * get_disk_info - Get network information in /proc/diskstat
 */

gboolean get_disk_info(gpointer label) {
    /*
     * Disk status file format:
     *
     * 8 num sdx rd_ios rd_merges rd_sectors rd_ticks wr_ios wr_merges
wr_sectors \
     * wr_ticks in_flight io_ticks time_in_queue
     *
     * read speed = ( $\Delta$ rd_sectors/ $\Delta$ t) * (block_size / 1024)
     * write speed = ( $\Delta$ wr_sectors/ $\Delta$ t) * (block_size / 1024)
     * Here block_size is 512
     */
    FILE *fp;
    long rd_sector, wr_sector;
    long rd_sectors, wr_sectors;
    float rd_diff, wr_diff;
    static long old_rd_sectors, old_wr_sectors;
    static int flag = 0;
    char buffer[256];
    char *pos;

    fp = fopen("/proc/diskstats", "r");

    rd_sectors = 0;
    wr_sectors = 0;

    while((pos = fgets(buffer, sizeof(buffer), fp)) != NULL) {
        sscanf(buffer, "%*d %*d %*s %*d %*d %d %*d %*d %*d %d",
&rd_sector, &wr_sector);
        rd_sectors += rd_sector;
        wr_sectors += wr_sector;
    }
}

```

```

    }
    fclose(fp);

    if (flag == 0) {
        old_rd_sectors = rd_sectors;
        old_wr_sectors = wr_sectors;
        rd_diff = 0;
        wr_diff = 0;
        flag = 1;
    }
    else {
        rd_diff = rd_sectors - old_rd_sectors;
        wr_diff = wr_sectors - old_wr_sectors;
        old_rd_sectors = rd_sectors;
        old_wr_sectors = wr_sectors;
    }

    /* Count speed (MB) */
    read_speed = (rd_diff * 512) / (2 * 1024 * 1024);
    write_speed = (wr_diff * 512) / (2 * 1024 * 1024);

    sprintf(buffer, "Disk I/O Speed:\n\n\
\tRead speed: %7.1f MB/s\t\tWrite speed: %7.1f MB/s\n\n\
\t Total Read: %7.1f GB\t\t Total Write: %7.1f GB",
            read_speed, write_speed, (rd_sectors * 512) / (1024.0 * 1024.0 *
1024.0), (wr_sectors * 512) / (1024.0 * 1024.0 * 1024.0));
    gtk_label_set_text(GTK_LABEL(label), buffer);
    set_label_fontsize((GtkWidget *)label, "13");

    return TRUE;
}

/*****ASSISTS*****/

```

```

/*
 * utf8_fix - To settle warning: Invalid UTF-8 string passed to
pango_layout_set_text()
 *
 *
 * Referencing from:
https://stackoverflow.com/questions/43753260/pango-warning-invalid-utf-8-string-passed-to-pango-layout-set-text-in-gtk
 */
char* utf8_fix(char *c) {
    return g_locale_to_utf8(c, -1, NULL, NULL, NULL);
}

/*
 * scroll_to_line - To set the scroll window position
 *
 * Modified from: https://my.oschina.net/plumsoft/blog/79950
 */
void scroll_to_line(gpointer scrolled_window, gint line_num, gint to_line_index)
{
    GtkAdjustment *adj;
    gdouble lower_value, upper_value, page_size, max_value, line_height,
to_value;
    adj =
gtk_scrolled_window_get_vadjustment(GTK_SCROLLED_WINDOW(scrolled_win
dow));
    lower_value = gtk_adjustment_get_lower(adj);
    upper_value = gtk_adjustment_get_upper(adj);
    page_size = gtk_adjustment_get_page_size(adj);
    max_value = upper_value - page_size;
    line_height = upper_value / line_num;
    to_value = line_height * to_line_index;
    if (to_value < lower_value)
        to_value = lower_value;
    if (to_value > max_value)
        to_value = max_value;
}

```

```

        gtk_adjustment_set_value(adj, to_value);
    return;
}

/*
 * set_label_fontsize - To set font size in a label
 *
 * Referencing from: https://blog.csdn.net/gl\_ding/article/details/4939355
 */
void set_label_fontsize(GtkWidget *label, char *fontsize) {
    PangoFontDescription *desc_info =
pango_font_description_from_string(fontsize);
    gtk_widget_modify_font(label, desc_info);
    pango_font_description_free(desc_info);
}

```

❖ Sysmonitor.h

```

#include "../lib/sys.h"
#include <dirent.h>
#include <gtk/gtk.h>

/* Global variables */
GtkWidget *popup_window;
GtkWidget *popup_label;
char *now_pid = NULL; /* Hold the pid clicked by mouse */
gint process_num = 0; /* Hold the Process num */
GtkWidget *clist; /* Hold the Process info */
GtkWidget *clist2; /* Hold the Modules info */
GtkWidget *entry; /* Hold the input */
float cpu_ratio = 0; /* Hold the CPU use ratio */
float cpu_ratio_data[120]; /* Hold the CPU use ratio histories */
char cpu_freq[1024]; /* Hold the CPU frequency */
float mem_total = 0; /* Hold the Memory total size */
float mem_free = 0; /* Hold the Memory free size */
float mem_ratio = 0; /* Hold the Memory use ratio */

```



```

float mem_ratio_data[120]; /* Hold the Memory use ratio histories */
float swap_total = 0;      /* Hold the Swap total size */
float swap_free = 0;       /* Hold the Swap free size */
float swap_ratio = 0;      /* Hold the Swap use ratio */
float swap_ratio_data[120]; /* Hold the Swap use ratio histories */
GtkWidget *cpu_curve;      /* Hold the CPU use curve */
GtkWidget *mem_curve;      /* Hold the Memory use curve */
GtkWidget *swap_curve;     /* Hold the Swap use curve */
int cpu_curve_start = 20;   /* Hold the CPU use curve start position */
int mem_curve_start = 20;   /* Hold the CPU use curve start position */
int swap_curve_start = 20;  /* Hold the CPU use curve start position */
float receive_speed = 0;    /* Hold the Network receive speed */
float send_speed = 0;       /* Hold the Network send speed */
float read_speed = 0;       /* Hold the Disk read speed */
float write_speed = 0;      /* Hold the Disk write speed */

/* Callback functions */
void select_row_callback(GtkWidget *clist, gint row, gint column, GdkEventButton
*event, gpointer data);
void search_proc(GtkButton *button,gpointer data);
void kill_proc(void);
void refresh_proc(void);
gboolean cpu_curve_callback(GtkWidget *widget, GdkEventExpose *event, gpointer
data);
gboolean mem_curve_callback(GtkWidget *widget, GdkEventExpose *event,
gpointer data);
gboolean swap_curve_callback(GtkWidget *widget, GdkEventExpose *event,
gpointer data);
void refresh_modules(void);

/* Loop functions */
gboolean draw_cpu_curve(gpointer widget);
gboolean draw_mem_curve(gpointer widget);
gboolean draw_swap_curve(gpointer widget);
gboolean get_cpu_ratio(gpointer label);

```

```
gboolean get_cpu_mhz(gpointer label);
gboolean get_memory_ratio(gpointer label);
gboolean get_memory_fraction(gpointer label);
gboolean get_swap_ratio(gpointer label);
gboolean get_swap_fraction(gpointer label);
gboolean get_sys_info(gpointer label);
gboolean get_network_info(gpointer label);
gboolean get_disk_info(gpointer label);

/* Assist functions */
char *utf8_fix(char *c);
void scroll_to_line(gpointer scrolled_window, gint line_num, gint to_line_index);
void set_label_fontsize(GtkWidget *label, char *fontsize);
```

实验五 模拟文件系统

1. 实验目的

- (1) 掌握实例操作系统的实现方法。
- (2) 设计并实现一个模拟的文件系统

2. 实验内容

- (1) 基于一个大文件(如 100M)，模拟磁盘。
- (2) 格式化，建立文件系统管理数据结构。
- (3) 实现文件/目录创建/删除，目录显示等基本功能(可自行扩充文件读/写、用户登录、权限控制、读写保护等其他功能)。

3. 实验设计

3.1 开发环境

Ubuntu 18.04, gcc version 7.4.0

3.2 实验设计

本课程设计分两部分，一是模拟磁盘格式化，建立数据结构，同时要维护所有的结构和变量。二是设计一个框架来对格式化的磁盘进行操作，这包括实现操作命令以及调用框架。本次课程设计模拟一块 64MB 大小的磁盘，磁盘每一块的大小为 1KB。第 0 块存放用户，第一块存放超级块，接着划分 1024 个 inode 的空间，最大允许 i 结点数 1024，后面跟着其他的 1KB 普通块。从第 0 块到 1026 块大小即为数据结构的大小，数据紧密排列。从 1027 开始的普通块按每块 1KB 的大小排列。设计三个全局变量维护超级块，i 结点，空闲块开始的位置，利用 `fseek` 函数改变文件偏移量的方法去读/写对应的块。

整个文件卷的设计如下图:

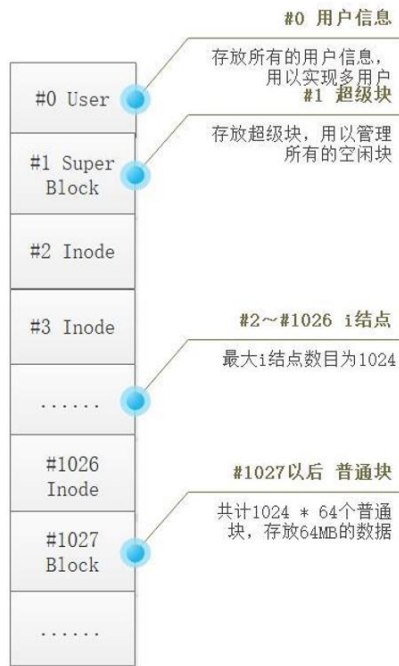


图 5-1 文件卷设计

3.2.1 数据结构设计

(1) 超级块 - super_block

超级块的结构如下：

```
typedef struct super_block {
    int inode_map[INODENUM];
    int block_map[BLOCKNUM];
    int inode_free_num;
    int block_free_num;
} super_block;
```

采用了位示图的方式来管理 i 结点和空闲块。位示图是利用二进制的一位来表示磁盘中的一个盘块的使用情况。当其值为“0”时，表示对应的盘块空闲；为“1”时，表示已经分配。磁盘上的所有盘块都有一个二进制位与之对应，这样，由所有盘块所对应的位构成一个集合，称为位示图。

此外超级块维护两个变量，表示空间 i 结点和空闲块数目。

(2) i 结点 - inode

i 结点的结构如下：

```
typedef struct inode {
    int block_used[FILEBLKMAX];
    int block_used_num;
```

```

int size;
int mode;
time_t creat_time;
time_t modify_time;
int user_id;
} inode;

```

i 结点的设计仿照了标准 Linux 的设计，存放了该文件占用的块号，占用块数目，文件大小，文件权限，创建时间，最后修改时间，所属用户 id。其中判断一个文件是普通文件还是目录要利用权限 mode 的最高位。

(3) 目录项 - directory

目录项的结构如下：

```

typedef struct directory {
    char name[FILENAMEMAX];
    int inode_id;
} directory;

```

目录项即目录文件内存放的数据，其要么指向下一级目录，要么指向一个普通文件，这个“指向”利用 i 结点编号实现。整个磁盘的数据结构大致如下图：



图 5-2 磁盘数据结构设计

3.2.2 核心函数设计

核心的函数即块管理的四个基本函数：inode_alloc, inode_free, block_alloc, block_free，以及两种 i 结点初始化函数 init_dir_inode 和 init_file_inode。有了这六个函数，我们就可以实现其他所有的功能。

- ❖ alloc 函数设计 两个 alloc 函数的算法是一致的，即：①检查是否还有剩余块，若没有，返回报错没有空闲块；②扫描位示图，寻找第一个空闲标志位为 0 的块；③将该块空闲标记置为 1，空闲块数目减 1，返回该块块号。
- ❖ block_free 函数设计 空闲块的 free 很简单，只需要回收块号即可，即：①将该块空闲标记置为 0，空闲块数目加 1。

- ❖ `inode_free` 函数设计 `i` 结点的 `free` 不仅要回收 `i` 结点本身占用的块，还要回收其管理的所有空闲块，即：
 - ①根据 `fseek` 函数读取 `i` 结点的信息；
 - ②对该 `i` 结点管理的所有块执行 `block_free` 函数；
 - ③将该块空闲标记置为 0，空闲块数目加 1. 这里的 `alloc` 和 `free` 均不涉及磁盘上的数据更改，仅仅是块号的分配和回收，这样便于我们操作，而且空闲块上有数据也不怕，因为它会在下次分配时被清除掉。
- ❖ 结点初始化函数设计 这个函数的操作很简单，设置好 `i` 结点结构体的初始值即可，目录和文件的区别在于对于目录，在创建之时就应该含有初始的两个目录项，分别是“.”和“..”，即指向自身和上一级目录的目录项。

3.2.3 多用户和权限设计

- ❖ 数据结构 多用户的信息存储在磁盘第 0 块上，利用如下两个结构体：

①用户信息

```
typedef struct user {
    char user_name[USERNAMEMAX];
    char user_pwd[USERPWDMAX];
} user;
```

②全体用户

```
typedef struct sys_users {
    int user_map[USERNUMMAX];
    int user_num;
    user users[USERNUMMAX];
} sys_users;
```

在登录用户时，在这些数据中逐一匹配用户名和密码，在登录之后登记全局变量当前用户 `id` 即可标识用户。

- ❖ 权限管理

权限设计也仿照标准 Linux 设计了 `drwxrwxrwx` 的权限，第 1 位 `d` 代表文件，后面的 `r` 代表可读，`w` 代表可写，`x` 代表可执行。第一组代表自己对该文件的权限，第二组代表同一用户组中其他用户对该文件的权限，第三组代表其他用户对 该文件的权限。实际上，由于是简化的文件系统，相比 OS，这里只用到了第一位的文件属性判断和第一组，第三组的 `rw` 位进行读写权限判断。权限的存储采用十进制数字，但为了便于权限的判断，设计了一个 8 进制转 10 进制的函数，就可

以方便的利用 8 进制数管理权限。

❖ 权限保护

➤ 本次课程设计中的权限设计如下：

磁盘中，以下几个操作需要读权限支持：

进入目录，列出当前目录下的文件及其详细信息，读取文件内的信息。

➤ 以下几个操作需要写权限支持：

创建文件（包括创建目录，普通文件，复制和移动），删除文件，保存文件数据。

➤ 以下几个操作需要所属用户权限支持：

修改权限，修改密码。

➤ 以下几个操作需要 root 权限支持：

添加、删除用户，格式化磁盘。

3.2.4 操作命令设计

❖ mkdir、touch - 创建目录、创建空文件

这两个函数的算法类似，而且在编程时使用了同一个接口。

①可行性检查，包括权限，重名，剩余空间，对于 touch，若文件存在则更新最后修改时间，退出函数；

②判断是否需要增加一块（对目录文件来说），若是，则调用 block_alloc 分配一块；

③分配新的 i 结点块；

④根据类型是目录还是普通文件，调用初始化函数初始化 i 结点；

⑤在目录包含的目录项中注册该目录项。

❖ rmdir、rm - 删除目录、删除文件

这两个函数的算法也类似，同样适用了同一个接口

① 可行性检查，包括权限，重名，剩余空间，类型，删除目录的话，还不能删除“.”和“..”以及非空文件；

②回收 i 结点块；

③如果当前目录下出现一块完整的空块，将其回收

❖ cd - 进入目录

①可行性检查，包括是否存在，是否是目录，权限；

②打开新的目录并更新全局变量；

③修改路径显示。

❖ ls 与 ls-l - 列出详细信息

①可行性检查，包括权限；

②循环获取每个目录项 *i* 结点的信息并打印。

❖ **vim** - 编辑文件

vim 的实现调用了外部接口，并采用在 `/tmp` 目录下生成一个缓冲文件的方法，打开文件即是从磁盘块上读数据到缓冲文件，接着 **vim** 打开缓冲文件并编辑。保存文件即是把缓冲文件上的数据写回磁盘。

- ①可行性检查，包括权限，是否存在，是否是普通文件，是否只读；
- ②将磁盘块上数据读到缓冲文件
- ③使用 **vim** 打开缓冲文件，并编辑
- ④检查是否可写，不可写则将文件还原并提升，否则将新的数据写回磁盘。

❖ **cat** - 输出文件内容

- ①可行性检查，包括权限，是否存在，是否是普通文件；
- ②输出文件内容。

❖ **cp** - 文件的复制

- ①可行性检查，包括权限，是否存在，是否是普通文件（这里没有实现递归复制目录的功能）；
- ②将文件内容读到缓冲文件中；
- ③判断是复制到其他目录内还是当前目内，若是其他目录，跳至⑥；
- ④若是当前目录，则执行创建新文件操作；
- ⑤将缓冲文件中的内容写到新文件里。
- ⑥若是其他目录，则进入其他目录；
- ⑦可行性检查，包括权限，是否重名；
- ⑧执行创建新文件操作；
- ⑨将缓冲文件中的内容写到新文件里。

❖ **mv** - 文件的移动 **mv** 相比 **cp** 就要简单很多，不需要复制文件和 *i* 结点，只需要在目录项层次上操作即可。

- ①可行性检查，包括权限，是否存在，是否是普通文件；
- ②判断是移动到其他目录内还是当前目内，若是其他目录，跳至⑤；
- ③若是当前目录，则创建新的目录项指向源文件的 *i* 结点；
- ④删除原目录项；
- ⑤若是其他目录，则进入其他目录；
- ⑥可行性检查，包括权限，是否重名；
- ⑦创建新的目录项指向源文件的 *i* 结点；
- ⑧删除原目录项。目录操作单纯的是对第 0 块上结构体数据的操作，比较简单，这里不再赘述，代码详见附录。

3.2.5 此外，还有两个十分重要的操作：

- **fmt** - 磁盘格式化 磁盘格式化是建立磁盘的第一部，是一切命令的基础。
 - ①将超级块设置为初始状态，并写回磁盘；
 - ②将 **root** 目录的 **i** 结点设置为初始状态，并写回磁盘；
 - ③在 **root** 目录下创建 “.” 和 “..” 两个目录项；
 - ④将路径设置为初始状态 “/”； 格式化不需要去清空原有的各种数据，只要把管理全局超级块处理掉，就实现了功能。
- **reset** 完成上面所有的功能后，我们并不能进入磁盘，因为没有任何一个用户，于是在此设计了一个开发者调试操作，用来把磁盘初始化，设置初始 **root** 用户以及初始密码。
 - ①清空用户结构；
 - ②添加第 0 个用户 **root**，设置初始密码为 123456 ③格式化磁盘。

4. 实验调试

4.1 实验步骤

```
(1) i 结点分配 - inode_alloc
int inode_alloc(void) {
    int ino;
    if (super.inode_free_num <= 0)
        return FS_NO_INODE;
    super.inode_free_num--;
    for (ino = 0; ino < INODENUM; ino++) {
        if (super.inode_map[ino] == 0) {
            super.inode_map[ino] = 1;
            break;
        }
    }
    return ino;
}

(2) i 结点释放 - inode_free
int inode_free(int ino) {
```

```

inode node;
fseek(disk, INODEPOS + ino * INODESIZE, SEEK_SET);
fread(&node, sizeof(inode), 1, disk);
for (int i = 0; i < node.block_used_num; i++)
    block_free(node.block_used[i]);
super.inode_map[ino] = 0;
super.inode_free_num++;
return FS_OK;
}

```

(3) 普通块分配 - block_alloc

```

int block_alloc(void) {
    int bno;
    if (super.block_free_num <= 0)
        return FS_NO_BLOCK;
    super.block_free_num--;
    for (bno = 0; bno < BLOCKNUM; bno++) {
        if (super.block_map[bno] == 0) {
            super.block_map[bno] = 1;
            break;
        }
    }
    return bno;
}

```

(4) 普通块释放 - block_free

```

int block_free(int bno) {
    super.block_free_num++;
    super.block_map[bno] = 0;
    return FS_OK;
}

```

(5) 目录结点初始化 - init_dir_inode

```

int init_dir_inode(int new_ino, int ino) {
    int bno;
    inode node;
    directory basic_link[2];

```

```

fseek(disk, INODEPOS + new_ino * INODESIZE, SEEK_SET);
fread(&node, sizeof(inode), 1, disk);
bno = block_alloc();

/* Set new inode information */
node.block_used[0] = bno;
node.block_used_num = 1;
node.size = 2 * sizeof(directory);
node.mode = oct2dec(1755);
time_t timer;
time(&timer);
node.creat_time = timer;
node.modify_time = timer;
node.user_id = current_user_id;

/* Save inode information */
fseek(disk, INODEPOS + new_ino * INODESIZE, SEEK_SET);
fwrite(&node, sizeof(inode), 1, disk);

/* Set basic links */
strcpy(basic_link[0].name, ".");
basic_link[0].inode_id = new_ino;
strcpy(basic_link[1].name, "..");
basic_link[1].inode_id = ino;

/* Save basic links */
fseek(disk, BLOCKPOS + bno * BLOCKSIZE, SEEK_SET);
fwrite(basic_link, sizeof(directory), 2, disk);

return FS_OK;
}

(6) 文件结点初始化 - init_file_inode
int init_file_inode(int new_ino) {
    inode node;
    fseek(disk, INODEPOS + new_ino * INODESIZE, SEEK_SET);

```

```

fread(&node, sizeof(inode), 1, disk);

/* Set new inode information */
node.block_used_num = 0;
node.size = 0;
node.mode = oct2dec(644);
time_t timer;
time(&timer);
node.creat_time = timer;
node.modify_time = timer;
node.user_id = current_user_id;

/* Save inode information */
fseek(disk, INODEPOS + new_ino * INODESIZE, SEEK_SET);
fwrite(&node, sizeof(inode), 1, disk);

return FS_OK;
}

```

4.2 实验调试及心得

4.2.1 实验调试

本次测试的内容如下表：

项目	测试	命令	测试结果
1	测试创建目录	mkdir test	通过
2	测试创建文件	touch data	通过
3	测试列出内容	ls -l	通过
4	测试向文件写数据	vim data	通过
5	测试从文件读数据	cat data	通过
6	测试移动文件	mv data testmv	通过
7	测试复制文件	cp testmv /test	通过
8	测试只读权限	chmod 444 testmv	通过
9	测试只写权限	chmod 244 testmv	通过
10	测试添加用户	useradd lnq 888888	通过

11	测试删除用户	Userdel Inq	通过
12	测试格式化	fmt	通过

(1) 测试项目 1&2&3

```
root@localhost: / > mkdir test
root@localhost: / > touch data.txt
root@localhost: / > ls -l
drwxr-xr-x  root    4 May  4 09:42 .
drwxr-xr-x  root    4 May  4 09:42 ..
drwxr-xr-x  root    2 May  4 09:42 test
-rw-r--r--  root    0 May  4 09:42 data.txt
```

图 5-3 测试项目 1&2&3

(2) 测试项目 4&5

首先在 vim 下对 data.txt 文件写入内容，再使用命令 cat 读取文件内容。

```
root@localhost: / > vim data.txt
Vim file in!!
root@localhost: / > cat data.txt
catch file in!!!

hello world!!
```

图 5-4 测试项目 4&5

(3) 测试项目 6

```
root@localhost: / > mv data.txt testmv.txt
root@localhost: / > ls -l
drwxr-xr-x  root    4 May  4 09:59 .
drwxr-xr-x  root    4 May  4 09:59 ..
drwxr-xr-x  root    2 May  4 09:42 test
-rw-r--r--  root   15 May  4 09:42 testmv.txt
```

图 5-5 测试项目 6

(4) 测试项目 7

```
root@localhost: / > cp testmv.txt test/
root@localhost: / > cd test
root@localhost: /test > ls -l
drwxr-xr-x  root    3 May  4 10:00 .
drwxr-xr-x  root    4 May  4 10:00 ..
-rw-r--r--  root   15 May  4 10:00 testmv.txt
```

图 5-6 测试项目 7

(5) 测试项目 8

更改 testmv 文件的权限为只读，执行 cat 和 vim 测试，发现 test 可以输出信息，但 vim 可读不可写

```
root@localhost: /test > chmod 444 testmv.txt
root@localhost: /test > cat testmv.txt
catch file in!!!

hello world!!
root@localhost: /test > vim testmv.txt
vim: Fail to save "testmv.txt": Insufficient privilege
```

图 5-7 测试项目 8

(6) 测试项目 9

更改 testmv 文件权限为只写，发现无法使用 cat 和 vim 获取文件内容。

```

root@localhost: /test > chmod 244 testmv.txt
root@localhost: /test > cat testmv.txt
cat: "testmv.txt": Insufficient privilege
root@localhost: /test > vim testmv.txt
vim: Fail to open "testmv.txt": Insufficient privilege

```

图 5-8 测试项目 9

(7) 测试项目 10&11

```

root@localhost: / > useradd lnq 666666

```

图 5-9 测试项目 10&11

使用开发者模式以新用户身份进入系统，可以看到用户名已经更新了，执行开发者用命令 `users`，可以看到当前系统中存在的所有用户。

```

[Developer mode] lnq@localhost: / > users
User id: 0
User id: root

User id: 1
User id: lnq

```

图 5-10 测试项目 10&11

接着换用 `root` 身份进入系统，执行删除用户命令，再使用 `users` 查看发现刚才添加的用户已经被删掉了。

```

[Developer mode] root@localhost: / > users
User id: 0
User id: root

User id: 1
User id: lnq

[Developer mode] root@localhost: / > userdel lnq
[Developer mode] root@localhost: / > users
User id: 0
User id: root

```

图 5-11 测试项目 10&11

(12) 测试项目 12

在文件系统中建立些许文件，执行 `fmt` 命令，可以看到系统已经恢复了初始状态，完成了格式化。

```

[Developer mode] root@localhost: / > ls -l
drwxr-xr-x root 4 May 4 10:08 .
drwxr-xr-x root 4 May 4 10:08 ..
drwxr-xr-x root 3 May 4 10:00 test
-rw-r--r-- root 15 May 4 09:42 testmv.txt
[Developer mode] root@localhost: / > fmt
[Developer mode] root@localhost: / > ls -l
drwxr-xr-x root 2 May 4 10:10 .
drwxr-xr-x root 2 May 4 10:10 ..

```

图 5-12 测试项目 12

4.2.2 心得

而文件系统就需要大量的设计工作了。起初我实验的是链接文件来管理块，而在我写了两天后发现 bug 越来越多，最后不得已重构了代码重新设计。为此我专门花了半天的时间，在几张草稿纸上画好了文件卷分区，目录结构设计，相关操作的算法流程，重新选用位示图作为管理空闲块的办法。

附录 实验代码

❖ Main.c

```
#include "prompt.h"

#define FS_PATH "lnq_disk"

int main(int argc, char **argv) {
    disk = fopen(FS_PATH, "r+");
    if (disk == NULL)
        return FS_NO_EXIST;
    Start_Shell(argc, argv);
    return 0;
}
```

❖ Prompt.h

```
#include "exec.h"

/* Constants */
#define MAXLINE 1024
#define MAXARGS 128

/* Global variables */

char theme[] = "Here to control file system";

/* Function prototypes */
void eval(char *cmdline);
int parseline(const char *cmdline, char **argv);
```

```

/* Print file system version */
void print_version() {
    char load[] = ".....";
    char ch[20];
    for (int i = 0; i < 19; i++) {
        memset(ch, 0, sizeof(ch));
        strncpy(ch, load, i + 1);
        printf("loading %s\n", ch);
        fflush(stdout);
        usleep(100000);
        printf("\033[1A\033[K");
    }
    printf("loading ..... \n\n");
}

/* Print usage messages */
void print_usage(void) {
    printf("Usage: ./test4 [options]\n");
    printf("Options:\n");
    printf("\t-h: print this massges\n");
    printf("\t-p: hide the prompt\n");
    printf("\t-d: use developer mode\n");
    printf("\tdefault: start shell\n");
}

int Start_Shell(int argc, char **argv) {
    int ret;
    char ch;
    /* The command line */
    char cmdline[MAXLINE];
    /* Decide whether print a prompt, default yes*/
    int emit_prompt = 1;

    /* Parse the command line */

```



```

while ((ch = getopt(argc, argv, "hpd")) != EOF) {
    switch(ch) {
        case 'h':
            print_usage();
            exit(0);
        case 'p':
            emit_prompt = 0;
            break;
        case 'd':
            developer = 0;
            break;
        default:
            print_usage();
            exit(1);
    }
}

```

```

/* Login */
print_version();
while (login() != FS_LOGIN) {}
ret = load_super_block();
if (ret == FS_RD_ERROR)
    return FS_RD_ERROR;

```

```

/* Print informations */
printf("%s\n", theme);
printf("\n");

```

```

/* Excute the shell's read/eval loop */
while (1) {

```

```

    /* Print prompt */
    if (emit_prompt) {
        if (developer == 0)
            printf("\e[1;31m[Developer mode] \e[0m");

```

```

        printf("\e[1;32m%s \e[0m", path);
        fflush(stdout);
    }
    if ((fgets(cmdline, MAXLINE, stdin) == NULL) && ferror(stdin)) {
        printf("fgets error\n");
        exit(1);
    }
    if (feof(stdin)) {
        printf("[EXIT] Parser reached end-of-file. Terminated!\n");
        fflush(stdout);
        exit(0);
    }

    /* Evaluate the command line */
    eval(cmdline);
    fflush(stdout);
}

return 0;
}

/*
 * eval - Evaluate the command line that the user has just typed in
 *
 * If the user has requested a built-in command, then execute it immediately.
 */
void eval(char *cmdline) {
    char *argv[MAXARGS];
    char buf[MAXLINE];
    int argc;

    /* Parse command */
    strcpy(buf, cmdline);
    argc = parseline(buf, argv);

```

```

    if (argv[0] == NULL)
        return;
    if (!developer_cmd(argc, argv)) {
        if (!builtin_cmd(argv)) {
            if (!py_execute(argv[0], argc, argv))
                printf("sh: command not found: %s\n", argv[0]);
            // else
                // printf("[INFO] Operation finished.\n\n");
        }
    }
}

/*
 * parseline - Parse the command line and build the argv array.
 */
int parseline(const char *cmdline, char **argv) {
    /* Holds local copy of command line */
    static char array[MAXLINE];
    char *buf = array;
    char *delim;
    int argc;

    strcpy(buf, cmdline);
    buf[strlen(buf) - 1] = ' ';
    /* Ignore leading spaces */
    while (*buf && (*buf == ' '))
        buf++;

    /* Build the argv list */
    argc = 0;
    if (*buf == "\\") {
        buf++;
        delim = strchr(buf, "\\");
    }
    else {

```

```

        delim = strchr(buf, ' ');
    }

    while(delim) {
        argv[argc++] = buf;
        *delim = '\0';
        buf = delim + 1;
        while (*buf && (*buf == ' '))
            buf++;

        if (*buf == "\\") {
            buf++;
            delim = strchr(buf, "\\");
        }
        else {
            delim = strchr(buf, ' ');
        }
    }
    argv[argc] = NULL;
    return argc;
}

```

❖ Exec.h

```

#include "filesystem.h"

/* Decide whether use developer APIs */
int developer = 1;

/*
 * print_help - Print help messages
 */
void print_help(void) {
    printf("Supported cmd:\n");
    printf("    Directory and File operations:\n");
}

```

```

    printf("    %-8s %-8s %-8s %-8s %-8s\n", "mkdir", "rmdir", "cd", "ls
[-l]", "touch");
    printf("    %-8s %-8s %-8s %-8s\n", "vim", "cat", "cp", "mv");
    printf("    User operations:\n");
    printf("    %-8s %-8s %-8s\n", "useradd", "userdel", "passwd");
    printf("    Other operations:\n");
    printf("    %-8s %-8s %-8s %-8s\n", "fmt", "chmod", "help", "exit");
    if (developer == 0) {
        printf("    Developer operations:\n");
        printf("    %-8s %-8s %-8s %-8s %-8s\n", "reset", "puid", "pino",
"dirnum", "show");
        printf("    %-8s %-8s %-8s\n", "users", "superi", "superb");
    }
}

/*
 * builtin_cmd - Judge builtin command
 */
int developer_cmd(int argc, char **argv) {
    if (developer == 1)
        return 0;
    if (!strcmp(argv[0], "reset")) {
        reset_disk();
        return 1;
    }
    if (!strcmp(argv[0], "puid")) {
        print_current_user_id();
        return 1;
    }
    if (!strcmp(argv[0], "pino")) {
        print_current_inode_id();
        return 1;
    }
    if (!strcmp(argv[0], "dirnum")) {
        print_current_dir_num();

```

```

        return 1;
    }
    if (!strcmp(argv[0], "show")) {
        show_files_info();
        return 1;
    }
    if (!strcmp(argv[0], "users")) {
        show_users_info();
        return 1;
    }
    if (argc == 2 && !strcmp(argv[0], "superi")) {
        print_superblk_inode_info(atoi(argv[1]));
        return 1;
    }
    if (argc == 2 && !strcmp(argv[0], "superb")) {
        print_superblk_block_info(atoi(argv[1]));
        return 1;
    }
    /* Not a developer command */
    return 0;
}

/*
 * developer_cmd - Judge developer command
 */
int builtin_cmd(char **argv) {
    if (!strcmp(argv[0], "fmt")) {
        if (current_user_id == 0)
            format_disk();
        else
            printf("fmt: You need root privilege!\n");
        return 1;
    }
    if (!strcmp(argv[0], "passwd")) {
        user_pwd();
    }
}

```

```

        return 1;
    }
    if (!strcmp(argv[0], "exit")) {
        printf("[EXIT] User-exit. Terminated!\n");
        close_disk();
        fclose(disk);
        exit(0);
    }
    if (!strcmp(argv[0], "help")) {
        print_help();
        return 1;
    }
    /* Not a built-in command */
    return 0;
}

/*
 * py_execute - The main execute function
 */
int py_execute(char *func , int argc, char **argv) {
    int ret;
    if (!strcmp(func, "mkdir")) {
        if (argc != 2) {
            printf("Usage: mkdir [dirname]\n");
            return 1;
        }
        ret = dir_creat(current_inode_id, TYPE_DIR, argv[1]);
        if (ret == FS_FILE_EXIST)
            printf("mkdir: Fail to create directory \"%s\": File already exists\n",
argv[1]);
        else if (ret == FS_NO_PRIVILAGE)
            printf("mkdir: Fail to create directory \"%s\": Insufficient
privilege\n", argv[1]);
        else if (ret != FS_OK)
            printf("mkdir: Fail to create directory \"%s\": No enough space\n",

```

```

argv[1]);
    return 1;
}
if (!strcmp(func, "rmdir")) {
    if (argc != 2) {
        printf("Usage: rmdir [dirname]\n");
        return 1;
    }
    ret = dir_rm(current_inode_id, TYPE_DIR, argv[1]);
    if (ret == FS_INVALID)
        printf("rmdir: Fail to delete \"%s\": Invalid operation\n", argv[1]);
    else if (ret == FS_NO_PRIVILAGE)
        printf("rmdir: Fail to delete \"%s\": Insufficient privilege\n",
argv[1]);
    else if (ret == FS_NO_EXIST)
        printf("rmdir: Fail to delete \"%s\": File not exists\n", argv[1]);
    else if (ret == FS_NO_PRIVILAGE)
        printf("rmdir: Fail to delete \"%s\": Insufficient privilege\n",
argv[1]);
    else if (ret == FS_ISNOT_DIR)
        printf("rmdir: Fail to delete \"%s\": Not a directory\n", argv[1]);
    else if (ret == FS_DIR_NOEMPTY)
        printf("rmdir: Fail to delete \"%s\": Directory not empty\n", argv[1]);
    return 1;
}
if (!strcmp(func, "cd")) {
    if (argc != 2) {
        printf("Usage: cd [dirname]\n");
        return 1;
    }
    int old_inode_id = current_inode_id;
    ret = dir_cd(current_inode_id, argv[1]);
    if (ret == FS_NO_EXIST)
        printf("cd: No such file or directory: \"%s\"\n", argv[1]);
    else if (ret == FS_NO_PRIVILAGE)

```



```

        printf("cd: Insufficient privilege: \"%s\"\n", argv[1]);
    else if (ret == FS_ISNOT_DIR)
        printf("cd: Not a directory: \"%s\"\n", argv[1]);
    else if (ret == FS_OK)
        path_change(old_inode_id, argv[1]);
    return 1;
}
if (!strcmp(func, "ls")) {
    if (argc == 2 && !strcmp(argv[1], "-l"))
        dir_ls_l();
    else
        dir_ls();
    return 1;
}
if (!strcmp(func, "touch")) {
    if (argc != 2) {
        printf("Usage: touch [filename]\n");
        return 1;
    }
    ret = dir_creat(current_inode_id, TYPE_FILE, argv[1]);
    if (ret == FS_FILE_EXIST)
        mtime_change(current_inode_id, argv[1]);
    else if (ret == FS_NO_PRIVILAGE)
        printf("touch: Fail to create file \"%s\": Insufficient privilege\n",
argv[1]);
    else if (ret != FS_OK)
        printf("touch: Fail to create file \"%s\": No enough space\n",
argv[1]);
    return 1;
}
if (!strcmp(func, "rm")) {
    if (argc != 2) {
        printf("Usage: rm [filename]\n");
        return 1;
    }
}

```

```

ret = dir_rm(current_inode_id, TYPE_FILE, argv[1]);
if (ret == FS_INVALID)
    printf("rmdir: Fail to delete \"%s\": Invalid operation\n", argv[1]);
else if (ret == FS_NO_EXIST)
    printf("rmdir: Fail to delete \"%s\": File not exists\n", argv[1]);
else if (ret == FS_NO_PRIVILAGE)
    printf("rmdir: Fail to delete \"%s\": Insufficient privilege\n",
argv[1]);
else if (ret == FS_ISNOT_FILE)
    printf("rmdir: Fail to delete \"%s\": Not a file\n", argv[1]);
return 1;
}
if (!strcmp(func, "vim")) {
    if (argc != 2) {
        printf("Usage: vim [filename]\n");
        return 1;
    }
    int pid, status;
    char *vim_arg[] = {"vim", BUFFERFILE, NULL};
    if (check_if_readonly(current_inode_id, argv[1]) == TRUE) {
        printf("vim: Fail to save \"%s\": Insufficient privilege\n", argv[1]);
        return 1;
    }
    ret = file_open(current_inode_id, argv[1]);
    if (ret == FS_IS_DIR) {
        printf("vim: Fail to open \"%s\": Is a directory\n", argv[1]);
        return 1;
    }
    else if (ret == FS_NO_PRIVILAGE) {
        printf("vim: Fail to open \"%s\": Insufficient privilege\n", argv[1]);
        return 1;
    }
    else if (ret == FS_NO_EXIST) {
        ret = dir_creat(current_inode_id, TYPE_FILE, argv[1]);
        if (ret == FS_NO_PRIVILAGE) {

```

```

        printf("vim: Fail to creat \"%s\": Insufficient privilege\n", argv[1]);
        return 1;
    }
    file_open(current_inode_id, argv[1]);
}
if((pid = fork()) == 0) {
    execvp("vim", vim_arg);
}
wait(&status);
file_close(current_inode_id, argv[1]);
return 1;
}
if (!strcmp(func, "cat")) {
    if (argc != 2) {
        printf("Usage: cat [filename]\n");
        return 1;
    }
    ret = file_open(current_inode_id, argv[1]);
    if (ret == FS_IS_DIR)
        printf("cat: \"%s\": Is a directory\n", argv[1]);
    else if (ret == FS_NO_EXIST)
        printf("cat: \"%s\": No such file or directory\n", argv[1]);
    else if (ret == FS_NO_PRIVILAGE)
        printf("cat: \"%s\": Insufficient privilege\n", argv[1]);
    else {
        file_cat();
        file_close(current_inode_id, argv[1]);
    }
    return 1;
}
if (!strcmp(func, "useradd")) {
    if (argc != 3) {
        printf("Usage: useradd [username] [userpwd]\n");
        return 1;
    }
}

```

```

    if (current_user_id != 0)
        printf("useradd: You need root privilege!\n");
    else {
        ret = user_add(argv[1], argv[2]);
        if (ret == FS_USER_EXIST)
            printf("useradd: Failed to add user \"%s\": User already exists\n",
argv[1]);
        }
        return 1;
    }
    if (!strcmp(func, "userdel")) {
        if (argc != 2) {
            printf("Usage: userdel [username]\n");
            return 1;
        }
        if (current_user_id != 0)
            printf("userdel: You need root privilege!\n");
        else {
            ret = user_del(argv[1]);
            if (ret == FS_USER_NOT_EXIST)
                printf("userdel: Failed to delete user \"%s\": User not exists\n",
argv[1]);
            }
            return 1;
        }
        if (!strcmp(func, "chmod")) {
            if (argc != 3) {
                printf("Usage: chmod [mod] [filename]\n");
                return 1;
            }
            ret = mode_change(atoi(argv[1]), argv[2]);
            if (ret == FS_NO_PRIVILAGE)
                printf("chmod: Change the permissions of \"%s\": Invalid
operation\n", argv[2]);
            else if (ret == FS_NO_EXIST)

```

```

        printf("chmod: Unable to access \"%s\": No such file or directory\n",
argv[2]);
    else if (ret == FS_INVALID_MODE)
        printf("chmod: Invalid operation: \"%s\"\n", argv[1]);
    return 1;
}
if (!strcmp(func, "mv")) {
    if (argc != 3) {
        printf("Usage: mv [srcfile] [dstfile | dstdir/]\n");
        return 1;
    }
    ret = file_mv(current_inode_id, argv[1], argv[2]);
    if (ret == FS_NO_EXIST)
        printf("mv: Unable to get file status for \"%s\" or \"%s\" (stat) : No
such file or directory\n", argv[1], argv[2]);
    else if (ret == FS_NO_PRIVILAGE)
        printf("mv: Unable to move \"%s\": Insufficient privilege\n",
argv[1]);
    else if (ret == FS_IS_DIR)
        printf("mv: Unable to move \"%s\": Not a file\n", argv[1]);
    else if (ret == FS_IS_FILE)
        printf("mv: Unable to move \"%s\" into \"%s\": Not a directory\n",
argv[1], argv[2]);
    else if (ret == FS_FILE_EXIST)
        printf("mv: Unable to move \"%s\": Target file exists\n", argv[1]);
    return 1;
}
if (!strcmp(func, "cp")) {
    if (argc != 3) {
        printf("Usage: cp [srcfile] [dstfile | dstdir/]\n");
        return 1;
    }
    ret = file_cp(current_inode_id, argv[1], argv[2]);
    if (ret == FS_NO_EXIST)
        printf("cp: Unable to get file status for \"%s\" or \"%s\" (stat) : No

```

```

such file or directory\n", argv[1], argv[2]);
    else if (ret == FS_NO_PRIVILAGE)
        printf("cp: Unable to copy \"%s\": Insufficient privilege\n", argv[1]);
    else if (ret == FS_IS_DIR)
        printf("cp: Unable to copy \"%s\": Not a file\n", argv[1]);
    else if (ret == FS_IS_FILE)
        printf("cp: Unable to copy \"%s\" into \"%s\": Not a directory\n",
argv[1], argv[2]);
    else if (ret == FS_FILE_EXIST)
        printf("cp: Unable to move \"%s\": Target file exists\n", argv[1]);
    return 1;
}
return 0;
}

```

❖ Filesystem.h

```

#ifndef lnq_FILESYSTEM
#define lnq_FILESYSTEM

#include "../lib/sys.h"
#include "fs_error.h"
#include <time.h>
#include <wait.h>

/* Disk parameters */
#define BLOCKSIZE 1024 /* The block size */
#define INODENUM 1024 /* The Inode nums */
#define BLOCKNUM (1024 * 64) /* The disk size 64 MB */
#define FILEBLKMAX (1024 * 4) /* The file max size is 4 MB */
#define FILENAMEMAX 20 /* File and directory name max
length */
#define USERNAMEMAX 20 /* User name max length */
#define USERPWDMAX 20 /* User password max length */
#define USERNUMMAX 10

```

```

#define INODESIZE sizeof(inode) /* The
inode size */
#define SUPERPOS sizeof(sys_users) /* The #1
is super block */
#define INODEPOS (SUPERPOS + sizeof(super_block)) /* The #2
~ #1025 are inodes */
#define BLOCKPOS (INODEPOS + INODESIZE * INODENUM) /*
The #1026 is first free block */

#define TYPE_DIR 0
#define TYPE_FILE 1

#define DIRMAXINBLK (BLOCKSIZE / sizeof(directory)) /* Directory
entry num per block */
#define DIRMAXNUM (FILEBLKMAX * DIRMAXINBLK) /*
File num per directory */

#define CAN_READ 0
#define CAN_WRITE 1

#define BUFFERFILE "/tmp/lnq_disk_buf"

/* File system data structures */
typedef struct super_block {
    int inode_map[INODENUM];
    int block_map[BLOCKNUM];
    int inode_free_num;
    int block_free_num;
} super_block;

typedef struct inode {
    int block_used[FILEBLKMAX];
    int block_used_num;

```

```

    int size;
    int mode;
    time_t creat_time;
    time_t modify_time;
    int user_id;
} inode;

typedef struct directory {
    char name[FILENAMEMAX];
    int inode_id;
} directory;

typedef struct user {
    char user_name[USERNAMEMAX];
    char user_pwd[USERPWDMAX];
} user;

typedef struct sys_users {
    int user_map[USERNUMMAX];
    int user_num;
    user users[USERNUMMAX];
} sys_users;

/* Global variables */
extern FILE *disk;
extern super_block super;
extern int current_inode_id;
extern inode current_inode;
extern directory current_dir_content[DIRMAXNUM];
extern int current_dir_num;
extern int current_user_id;

extern char path[128];

/* Developer functions */

```



```

void reset_disk();
void print_current_user_id(void);
void print_current_inode_id(void);
void print_current_dir_num(void);
void show_files_info();
void print_superblk_inode_info(int pos);
void print_superblk_block_info(int pos);
void show_users_info(void);

/* Core function */
int inode_alloc(void);
int inode_free(int ino);
int init_dir_inode(int new_ino, int ino);
int init_file_inode(int new_ino);
int block_alloc(void);
int block_free(int bno);

/* User function */
int login(void);
int user_pwd(void);
int user_add(char *name, char *pwd);
int user_del(char *name);

/* Disk function */
int load_super_block(void);
int format_disk(void);
int close_disk(void);

/* File and directory function */
int dir_open(int ino);
int dir_close(int ino);
int dir_creat(int ino, int type, char *name);
int dir_rm(int ino, int type, char *name);
int dir_cd(int ino, char *path);
int dir_ls(void);

```

```

int dir_ls_l(void);
int file_open(int ino, char *name);
int file_close(int ino, char *name);
int file_cat(void);
int file_mv(int ino, char *srcname, char *dstname);
int file_cp(int ino, char *srcname, char *dstname);

```

```

/* Assist function */

```

```

int oct2dec(int oct_number);
int check_name(char *name);
int check_type(int ino, int type);
int check_mode(int mode, int operation);
void path_change(int old_inode_id, char *name);
int mtime_change(int ino, char *name);
void get_modestr(char *modstr, int mode);
int mode_change(int mode, char *name);
int check_if_readonly(int ino, char *name);

```

```

#endif

```

❖ Filesystem.c

```

#include "filesystem.h"

```

```

FILE *disk;
super_block super;
int current_inode_id;
inode current_inode;
directory current_dir_content[DIRMAXNUM];
int current_dir_num;
int current_user_id;
char path[128];

```

```

/*****DEVELOPER

```

```

FUNCTIONS*****/
/*
 * Warning: These functions is only for developer, they will not
 *          be called in the formal edition.
 */

/*
 * reset_disk - Totally reset the disk, you can call it to reset
 *              disk or make a new file become a disk file for this
 *              program.
 *
 * root initial password: 123456
 */
void reset_disk(void) {
    sys_users all_users;
    /* Reset user */
    fseek(disk, 0, SEEK_SET);
    fread(&all_users, sizeof(sys_users), 1, disk);
    memset(&all_users, 0, sizeof(sys_users));

    /* Set root initial password */
    all_users.user_map[0] = 1;
    all_users.user_num = 1;
    strcpy(all_users.users[0].user_name, "root");
    strcpy(all_users.users[0].user_pwd, "123456");

    /* Save data */
    fseek(disk, 0, SEEK_SET);
    fwrite(&all_users, sizeof(sys_users), 1, disk);

    /* Format the disk */
    format_disk();
}

```

```

/*
 * print_current_user_id - See the current user id
 */
void print_current_user_id(void) {
    printf("%d\n", current_user_id);
}

/*
 * print_current_inode_id - See the current inode id
 */
void print_current_inode_id(void) {
    printf("%d\n", current_inode_id);
}

/*
 * print_current_inode_id - See the current directory num
 */
void print_current_dir_num(void) {
    printf("%d\n", current_dir_num);
}

/*
 * show_files_info - See the files inode information in directory
 */
void show_files_info(void) {
    int pos;
    inode node;
    for (pos = 0; pos < current_dir_num; pos++) {
        fseek(disk, INODEPOS + current_dir_content[pos].inode_id *
INODESIZE, SEEK_SET);
        fread(&node, sizeof(node), 1, disk);
        printf("pos: %d  ", pos);
        printf("name: %-10s  ", current_dir_content[pos].name);
        printf("inode id: %d  ", current_dir_content[pos].inode_id);
    }
}

```

```

        printf("user id: %d\n\n", node.user_id);
    }
}

/*
 * print_superblk_inode_info - See inode information in super block
 */
void print_superblk_inode_info(int pos) {
    printf("Super block inode:\n");
    printf("Pos %d: %d\n", pos, super.inode_map[pos]);
    printf("Free num: %d\n", super.inode_free_num);
}

/*
 * print_superblk_block_info - See block information in super block
 */
void print_superblk_block_info(int pos) {
    printf("Super block block:\n");
    printf("Pos %d: %d\n", pos, super.block_map[pos]);
    printf("Free num: %d\n", super.block_free_num);
}

void show_users_info(void) {
    int pos;
    sys_users all_users;

    /* Read users information */
    fseek(disk, 0, SEEK_SET);
    fread(&all_users, sizeof(sys_users), 1, disk);

    for (pos = 0; pos < USERNUMMAX; pos++) {
        if (all_users.user_map[pos] == 1) {
            printf("User id: %d\n", pos);
            printf("User id: %s\n\n", all_users.users[pos].user_name);
        }
    }
}

```

```

    }
}

/*****CORE FUNCTIONS*****/

/*
 * inode_alloc - Alloc a new free inode from inode map
 */
int inode_alloc(void) {
    int ino;
    if (super.inode_free_num <= 0)
        return FS_NO_INODE;
    super.inode_free_num--;
    for (ino = 0; ino < INODENUM; ino++) {
        if (super.inode_map[ino] == 0) {
            super.inode_map[ino] = 1;
            break;
        }
    }
    return ino;
}

/*
 * inode_free - Free a inode and the file's all block
 */
int inode_free(int ino) {
    inode node;
    fseek(disk, INODEPOS + ino * INODESIZE, SEEK_SET);
    fread(&node, sizeof(inode), 1, disk);
    for (int i = 0; i < node.block_used_num; i++)
        block_free(node.block_used[i]);
    super.inode_map[ino] = 0;
    super.inode_free_num++;
    return FS_OK;
}

```

```

/*
 * init_dir_inode - Initial directory inode
 */
int init_dir_inode(int new_ino, int ino) {
    int bno;
    inode node;
    directory basic_link[2];

    fseek(disk, INODEPOS + new_ino * INODESIZE, SEEK_SET);
    fread(&node, sizeof(inode), 1, disk);
    bno = block_alloc();

    /* Set new inode information */
    node.block_used[0] = bno;
    node.block_used_num = 1;
    node.size = 2 * sizeof(directory);
    node.mode = oct2dec(1755);
    time_t timer;
    time(&timer);
    node.creat_time = timer;
    node.modify_time = timer;
    node.user_id = current_user_id;

    /* Save inode information */
    fseek(disk, INODEPOS + new_ino * INODESIZE, SEEK_SET);
    fwrite(&node, sizeof(inode), 1, disk);

    /* Set basic links */
    strcpy(basic_link[0].name, ".");
    basic_link[0].inode_id = new_ino;
    strcpy(basic_link[1].name, "..");
    basic_link[1].inode_id = ino;

    /* Save basic links */

```

```

        fseek(disk, BLOCKPOS + bno * BLOCKSIZE, SEEK_SET);
        fwrite(basic_link, sizeof(directory), 2, disk);

    return FS_OK;
}

/*
 * init_file_inode - Initial file inode
 */
int init_file_inode(int new_ino) {
    inode node;
    fseek(disk, INODEPOS + new_ino * INODESIZE, SEEK_SET);
    fread(&node, sizeof(inode), 1, disk);

    /* Set new inode information */
    node.block_used_num = 0;
    node.size = 0;
    node.mode = oct2dec(644);
    time_t timer;
    time(&timer);
    node.creat_time = timer;
    node.modify_time = timer;
    node.user_id = current_user_id;

    /* Save inode information */
    fseek(disk, INODEPOS + new_ino * INODESIZE, SEEK_SET);
    fwrite(&node, sizeof(inode), 1, disk);

    return FS_OK;
}

/*
 * block_alloc - Alloc a new free block from block map
 */
int block_alloc(void) {

```



```

int bno;
if (super.block_free_num <= 0)
    return FS_NO_BLOCK;
super.block_free_num--;
for (bno = 0; bno < BLOCKNUM; bno++) {
    if (super.block_map[bno] == 0) {
        super.block_map[bno] = 1;
        break;
    }
}
return bno;
}

/*
 * block_free - Free a block
 */
int block_free(int bno) {
    super.block_free_num++;
    super.block_map[bno] = 0;
    return FS_OK;
}

/*****USER FUNCTIONS*****/

/*
 * login - Login for the disk
 */
int login(void) {
    sys_users all_users;
    char uname[USERNAMEMAX];
    char upwd[USERPWDMAX];
    printf("localhost login: ");
    scanf("%s", uname);

```

```

printf("\nPassword: ");
scanf("%s", upwd);
getchar();

/* Read users information */
fseek(disk, 0, SEEK_SET);
fread(&all_users, sizeof(sys_users), 1, disk);

/* Check user */
for (int i = 0; i < USERNUMMAX; i++) {
    if (strcmp(all_users.users[i].user_name, uname) == 0) {
        if (strcmp(all_users.users[i].user_pwd, upwd) == 0) {
            current_user_id = i;
            sprintf(path, "%s@localhost: / >", uname);
            printf("\033[2J");
            return FS_LOGIN;
        }
    }
}

usleep(500000);
printf("Login incorrect\n\n");
return FS_LOGIN_ERROR;
}

/*
 * user_pwd - Change a user's password
 */
int user_pwd(void) {
    int i;
    sys_users all_users;
    char    current_pwd[USERPWDMAX],    new_pwd[USERPWDMAX],
    new_pwd_2[USERPWDMAX];

    /* Read users information */

```

```

fseek(disk, 0, SEEK_SET);
fread(&all_users, sizeof(sys_users), 1, disk);

printf("Change          password          for          %s\n",
all_users.users[current_user_id].user_name);
printf("Current password: ");
scanf("%s", current_pwd);
getchar();

if (strcmp(all_users.users[current_user_id].user_pwd, current_pwd) != 0)
{
    printf("passwd: Identification failure\n");
    printf("passwd: Password not changed\n");
    return FS_INVALID;
}

printf("New password: ");
scanf("%s", new_pwd);
printf("New password again: ");
scanf("%s", new_pwd_2);
getchar();

if (strcmp(new_pwd, new_pwd_2) != 0) {
    printf("Sorry, the password does not match\n");
    printf("passwd: Password service preliminary check failed\n");
    printf("passwd: Password not changed\n");
    return FS_INVALID;
}

strcpy(all_users.users[current_user_id].user_pwd, new_pwd);
printf("password: Password successfully updated\n");

/* Save users information */
fseek(disk, 0, SEEK_SET);
fwrite(&all_users, sizeof(sys_users), 1, disk);

```

```

    return FS_OK;
}

/*
 * user_add - Add a user
 */
int user_add(char *name, char *pwd) {
    sys_users all_users;

    /* Read users information */
    fseek(disk, 0, SEEK_SET);
    fread(&all_users, sizeof(sys_users), 1, disk);

    /* Check user */
    for (int i = 0; i < USERNUMMAX; i++) {
        if (strcmp(all_users.users[i].user_name, name) == 0) {
            return FS_USER_EXIST;
        }
    }

    for (int i = 0; i < USERNUMMAX; i++) {
        if (all_users.user_map[i] == 0) {
            all_users.user_map[i] = 1;
            strcpy(all_users.users[i].user_name, name);
            strcpy(all_users.users[i].user_pwd, pwd);
            break;
        }
    }

    /* Save users information */
    fseek(disk, 0, SEEK_SET);
    fwrite(&all_users, sizeof(sys_users), 1, disk);

    return FS_OK;
}

```

```

}

/*
 * user_del - Delete a user
 */
int user_del(char *name) {
    sys_users all_users;
    int i;

    /* Read users information */
    fseek(disk, 0, SEEK_SET);
    fread(&all_users, sizeof(sys_users), 1, disk);

    /* Check user */
    for (i = 0; i < USERNUMMAX; i++) {
        if (strcmp(all_users.users[i].user_name, name) == 0) {
            memset(all_users.users[i].user_name, 0, USERNAMEMAX);
            memset(all_users.users[i].user_pwd, 0, USERPWDMAX);
            all_users.user_map[i] = 0;
            break;
        }
    }
    if (i == USERNUMMAX)
        return FS_USER_NOT_EXIST;

    /* Save users information */
    fseek(disk, 0, SEEK_SET);
    fwrite(&all_users, sizeof(sys_users), 1, disk);

    return FS_OK;
}

/*****DISK FUNCTIONS*****/

```

```

/*
 * load_super_block - Load super block information from disk
 */
int load_super_block(void) {
    int ret;

    /* Load super block */
    fseek(disk, SUPERPOS, SEEK_SET);
    ret = fread(&super, sizeof(super_block), 1, disk);
    if (ret != 1)
        return FS_RD_ERROR;

    /* Open root dir directory */
    current_inode_id = 0;
    ret = dir_open(current_inode_id);
    if (ret != FS_OK)
        return ret;

    return FS_OK;
}

/*
 * format_disk - Format and initialize the disk
 */
int format_disk(void) {
    int ret;

    /* Set inode map */
    memset(super.inode_map, 0, sizeof(super.inode_map));
    super.inode_map[0] = 1;
    super.inode_free_num = INODENUM - 1;

    /* Set block map */

```

```

memset(super.block_map, 0, sizeof(super.block_map));
super.block_map[0] = 1;
super.block_free_num = BLOCKNUM - 1;

/* Set root inode */
current_inode_id = 0;
fseek(disk, INODEPOS, SEEK_SET);
ret = fread(&current_inode, sizeof(inode), 1, disk);
if (ret != 1)
    return FS_RD_ERROR;
current_inode.block_used[0] = 0;
current_inode.block_used_num = 1;
current_inode.size = 2 * sizeof(directory);
current_inode.mode = oct2dec(1755);
time_t timer;
    time(&timer);
current_inode.creat_time = timer;
current_inode.modify_time = timer;
current_inode.user_id = 0;

/* Set basic link of root file */
current_dir_num = 2;
strcpy(current_dir_content[0].name, ".");
current_dir_content[0].inode_id = 0;
strcpy(current_dir_content[1].name, "..");
current_dir_content[1].inode_id = 0;

strcpy(path, "root@localhost: / >");

return FS_OK;
}

/*
* close_disk - Load super block information from disk

```

```

    */
int close_disk(void) {
    int ret;

    /* Save super block */
    fseek(disk, SUPERPOS, SEEK_SET);
    ret = fwrite(&super, sizeof(super_block), 1, disk);
    if (ret != 1)
        return FS_WR_ERROR;

    /* Close current directory */
    ret = dir_close(current_inode_id);
    if (ret != 1)
        return ret;

    return FS_OK;
}

/*****DIR FUNCTIONS*****/

/*
 * dir_open - Open a directory, and read its data in memory
 */
int dir_open(int ino) {
    int i, ret;
    int end_block_dirnum;
    fseek(disk, INODEPOS + ino * INODESIZE, SEEK_SET);
    ret = fread(&current_inode, sizeof(inode), 1, disk);
    if (ret != 1)
        return FS_RD_ERROR;

    /* Read all directory entry */
    for (i = 0; i < current_inode.block_used_num - 1; i++) {

```



```

        fseek(disk, BLOCKPOS + current_inode.block_used[i] * BLOCKSIZE,
SEEK_SET);

        fread(current_dir_content + i * DIRMAXINBLK, sizeof(directory),
DIRMAXINBLK, disk);
    }
    end_block_dirnum = current_inode.size / sizeof(directory) -
DIRMAXINBLK * (current_inode.block_used_num - 1);
    fseek(disk, BLOCKPOS + current_inode.block_used[i] * BLOCKSIZE,
SEEK_SET);
    fread(current_dir_content + i * DIRMAXINBLK, sizeof(directory),
end_block_dirnum, disk);

    time_t timer;
    time(&timer);
    current_inode.modify_time = timer;
    current_dir_num = i * DIRMAXINBLK + end_block_dirnum;
    return FS_OK;
}

/*
 * dir_close - Close a directory, and write its data back to disk
 */
int dir_close(int ino) {
    int i, ret;
    int end_block_dirnum;

    /* Write all directory entry back to disk */
    for (i = 0; i < current_inode.block_used_num - 1; i++) {
        fseek(disk, BLOCKPOS + current_inode.block_used[i] * BLOCKSIZE,
SEEK_SET);
        fwrite(current_dir_content + i * DIRMAXINBLK, sizeof(directory),
DIRMAXINBLK, disk);
    }
    end_block_dirnum = current_dir_num - i * DIRMAXINBLK;
    fseek(disk, BLOCKPOS + current_inode.block_used[i] * BLOCKSIZE,

```

```

SEEK_SET);
    fwrite(current_dir_content + i * DIRMAXINBLK, sizeof(directory),
end_block_dirnum, disk);

    current_inode.size = current_dir_num * sizeof(directory);
    fseek(disk, INODEPOS + ino * INODESIZE, SEEK_SET);
    ret = fwrite(&current_inode, sizeof(inode), 1, disk);
    if (ret != 1)
        return FS_WR_ERROR;
    return FS_OK;
}

/*
 * dir_creat - Creat a directory or a file and initial the inode
 */
int dir_creat(int ino, int type, char *name) {
    int new_ino;
    int block_need = 1;
    if (current_dir_num >= DIRMAXNUM)
        return FS_DIR_FULL;
    if (check_name(name) != FS_OK)
        return FS_FILE_EXIST;

    /* Check mode */
    if (check_mode(ino, CAN_WRITE) == 0)
        return FS_NO_PRIVILAGE;

    /* If need more blocks */
    if (current_dir_num / DIRMAXINBLK != (current_dir_num + 1) /
DIRMAXINBLK)
        block_need++;
    if (block_need > super.block_free_num)
        return FS_NO_BLOCK;
    if (block_need == 2)
        current_inode.block_used[++current_inode.block_used_num] =

```

```

block_alloc();
    new_ino = inode_alloc();
    if (new_ino == FS_NO_INODE)
        return FS_NO_INODE;

    /* Initial new inode */
    if (type == TYPE_DIR)
        init_dir_inode(new_ino, ino);
    else
        init_file_inode(new_ino);

    /* Register new inode */
    strcpy(current_dir_content[current_dir_num].name, name);
    current_dir_content[current_dir_num].inode_id = new_ino;

    /* Update modify time */
    time_t timer;
    time(&timer);
    current_inode.modify_time = timer;

    current_dir_num++;
    return FS_OK;
}

/*
 * dir_rm - Delete a empty directory or a file
 */
int dir_rm(int ino, int type, char *name) {
    int rm_inode;
    int ret;
    inode node;

    /* Check mode */
    if (check_mode(ino, CAN_WRITE) == 0)
        return FS_NO_PRIVILAGE;

```

```

/* Can't delete . and .. */
if (!strcmp(name, ".") || !strcmp(name, ".."))
    return FS_INVALID;

/* Check if the directory or file exists */
for (rm_inode = 0; rm_inode < current_dir_num; rm_inode++) {
    if (strcmp(name, current_dir_content[rm_inode].name) == 0)
        break;
}
if (rm_inode == current_dir_num)
    return FS_NO_EXIST;

rm_inode = current_dir_content[rm_inode].inode_id;
/* Read inode information */
fseek(disk, INODEPOS + rm_inode * INODESIZE, SEEK_SET);
fread(&node, sizeof(node), 1, disk);

/* Check user */
if (node.user_id != current_user_id && current_user_id != 0)
    return FS_NO_PRIVILAGE;

/* Check type */
ret = check_type(node.mode, type);
if (ret == FS_ISNOT_DIR || ret == FS_ISNOT_FILE)
    return ret;

if (ret == FS_IS_DIR) {
    dir_cd(ino, name);
    if (current_dir_num != 2) {
        dir_cd(rm_inode, "..");
        return FS_DIR_NOEMPTY;
    }
    dir_cd(rm_inode, "..");
}
}

```

```

int pos;
inode_free(rm_inode);
for (pos = 0; pos < current_dir_num; pos++) {
    if (strcmp(current_dir_content[pos].name, name) == 0)
        break;
}
for (; pos < current_dir_num - 1; pos++) {
    current_dir_content[pos] = current_dir_content[pos + 1];
}
current_dir_num--;

/* Free last block if need */
if (current_dir_num / DIRMAXINBLK != (current_dir_num - 1) /
DIRMAXINBLK) {
    current_inode.block_used_num--;

block_free(current_inode.block_used[current_inode.block_used_num]);
}

/* Update modify time */
time_t timer;
time(&timer);
current_inode.modify_time = timer;

return FS_OK;
}

/*
 * dir_cd - Enter into a directory
 */
int dir_cd(int ino, char *name) {
    int i;
    int cd_inode;
    inode node;

```

```

/* Check if the directory or file exists */
for (i = 0; i < current_dir_num; i++) {
    if (strcmp(current_dir_content[i].name, name) == 0)
        break;
}
if (i == current_dir_num)
    return FS_NO_EXIST;
cd_inode = current_dir_content[i].inode_id;

/* Check if this is a directory */
fseek(disk, INODEPOS + cd_inode * INODESIZE, SEEK_SET);
fread(&node, sizeof(node), 1, disk);
if (check_type(node.mode, TYPE_FILE) == FS_IS_FILE)
    return FS_ISNOT_DIR;

/* Check mode */
if (check_mode(cd_inode, CAN_READ) == 0)
    return FS_NO_PRIVILAGE;

dir_close(ino);
current_inode_id = cd_inode;
dir_open(cd_inode);

return FS_OK;
}

/*
 * dir_ls - List all files in directory
 */
int dir_ls(void) {
    /* Save current status */
    dir_close(current_inode_id);
    dir_open(current_inode_id);

```

```

    int pos;
    inode node;
    for (pos = 0; pos < current_dir_num; pos++) {
        fseek(disk, INODEPOS + current_dir_content[pos].inode_id *
INODESIZE, SEEK_SET);
        fread(&node, sizeof(node), 1, disk);
        if (check_type(node.mode, TYPE_DIR) == FS_IS_DIR)
            printf("\e[1;34m%s\t\e[0m", current_dir_content[pos].name);
        else
            printf("%s\t", current_dir_content[pos].name);
    }
    printf("\n");

    return FS_OK;
}

/*
 * dir_ls_l - List all files and its information in a directory
 */
int dir_ls_l(void) {
    /* Save current status */
    dir_close(current_inode_id);
    dir_open(current_inode_id);

    int pos;
    sys_users all_users;
    inode node;
    char modstr[11];
    char *time;

    /* Read users information */
    fseek(disk, 0, SEEK_SET);
    fread(&all_users, sizeof(sys_users), 1, disk);

    for (pos = 0; pos < current_dir_num; pos++) {

```

```

        fseek(disk, INODEPOS + current_dir_content[pos].inode_id *
INODESIZE, SEEK_SET);
        fread(&node, sizeof(node), 1, disk);

        get_modestr(modstr, node.mode);
        time = ctime(&node.modify_time);

        if (check_type(node.mode, TYPE_DIR) == FS_IS_DIR) {
            printf("%s %-s \e[1;34m%6ld\e[0m %.12s \e[1;34m%-s\e[0m\n",
                    modstr, all_users.users[node.user_id].user_name, node.size /
sizeof(directory), time + 4, current_dir_content[pos].name);
        }
        else {
            printf("%s %-s %6d %.12s %-s\n",
                    modstr, all_users.users[node.user_id].user_name, node.size,
time + 4, current_dir_content[pos].name);
        }
    }

    return FS_OK;
}

/*
 * file_open - Open a file, read its contains into buffer tmp file
 */
int file_open(int ino, char *name) {
    int open_inode;
    int bno, pos;
    inode node;
    char block[BLOCKSIZE];
    FILE *buf_fp = fopen(BUFFERFILE, "w+");

    /* Check if the directory or file exists */
    for (open_inode = 0; open_inode < current_dir_num; open_inode++) {
        if (strcmp(current_dir_content[open_inode].name, name) == 0)

```



```

        break;
    }
    if (open_inode == current_dir_num)
        return FS_NO_EXIST;

    open_inode = current_dir_content[open_inode].inode_id;

    /* Check mode */
    if (check_mode(open_inode, CAN_READ) == 0)
        return FS_NO_PRIVILAGE;

    /* Read inode information */
    fseek(disk, INODEPOS + open_inode * INODESIZE, SEEK_SET);
    fread(&node, sizeof(node), 1, disk);

    /* Check type */
    if (check_type(node.mode, TYPE_DIR) == FS_IS_DIR)
        return FS_IS_DIR;

    if (node.size == 0) {
        fclose(buf_fp);
        return FS_OK;
    }

    /* Read data from disk */
    for (pos = 0; pos < node.block_used_num - 1; pos++) {
        memset(block, 0, BLOCKSIZE);
        bno = node.block_used[pos];
        fseek(disk, BLOCKPOS + bno * BLOCKSIZE, SEEK_SET);
        fread(block, sizeof(char), BLOCKSIZE, disk);
        fwrite(block, sizeof(char), BLOCKSIZE, buf_fp);
        block_free(bno);
        node.size -= BLOCKSIZE;
    }
    bno = node.block_used[pos];

```

```

fseek(disk, BLOCKPOS + bno * BLOCKSIZE, SEEK_SET);
fread(block, sizeof(char), node.size, disk);
fwrite(block, sizeof(char), node.size, buf_fp);
block_free(bno);
node.size = 0;
node.block_used_num = 0;

/* Save inode */
fseek(disk, INODEPOS + open_inode * INODESIZE, SEEK_SET);
fwrite(&node, sizeof(inode), 1, disk);

fclose(buf_fp);
return FS_OK;
}

/*
 * file_close - Close a file, write buffer file contents to disk
 */
int file_close(int ino, char *name) {
    int close_inode;
    int bno, read_num;
    inode node;
    char block[BLOCKSIZE];
    FILE *buf_fp = fopen(BUFFERFILE, "r");

    /* Check if the directory or file exists */
    for (close_inode = 0; close_inode < current_dir_num; close_inode++) {
        if (strcmp(current_dir_content[close_inode].name, name) == 0)
            break;
    }
    if (close_inode == current_dir_num)
        return FS_NO_EXIST;

    close_inode = current_dir_content[close_inode].inode_id;

```

```

/* Read inode information */
fseek(disk, INODEPOS + close_inode * INODESIZE, SEEK_SET);
fread(&node, sizeof(node), 1, disk);

/* Check type */
if (check_type(node.mode, TYPE_DIR) == FS_IS_DIR)
    return FS_ISNOT_FILE;

/* Read data from buffer file */
memset(block, 0, BLOCKSIZE);
read_num = fread(block, sizeof(char), BLOCKSIZE, buf_fp);
while(read_num != 0) {
    bno = block_alloc();
    if (bno == FS_NO_BLOCK)
        break;
    fseek(disk, BLOCKPOS + bno * BLOCKSIZE, SEEK_SET);
    fwrite(block, sizeof(char), BLOCKSIZE, disk);
    node.block_used[node.block_used_num] = bno;
    node.block_used_num++;
    node.size += read_num;

    memset(block, 0, BLOCKSIZE);
    read_num = fread(block, sizeof(char), BLOCKSIZE, buf_fp);
}

/* Save inode */
fseek(disk, INODEPOS + close_inode * INODESIZE, SEEK_SET);
fwrite(&node, sizeof(inode), 1, disk);

fclose(buf_fp);
return FS_OK;
}

/*
* file_cat - Get a file's content

```

```

*/
int file_cat(void) {
    int read_num;
    FILE *buf_fp = fopen(BUFFERFILE, "r");
    char block[BLOCKSIZE];
    memset(block, 0, BLOCKSIZE);

    /* Read data from buffer file */
    while((read_num = fread(block, sizeof(char), BLOCKSIZE, buf_fp) !=
0))
        printf("%s", block);

    fclose(buf_fp);
    return FS_OK;
}

/*
* file_mv - Move file from srcname to dstname
*/
int file_mv(int ino, char *srcname, char *dstname) {
    int pos;
    int src_inode;
    inode node;

    /* Check if the directory or file exists */
    for (pos = 0; pos < current_dir_num; pos++) {
        if (strcmp(current_dir_content[pos].name, srcname) == 0)
            break;
    }
    if (pos == current_dir_num)
        return FS_NO_EXIST;

    src_inode = current_dir_content[pos].inode_id;

    /* Check mode */

```

```

if (check_mode(current_inode_id, CAN_READ) == 0)
    return FS_NO_PRIVILAGE;

/* Read inode information */
fseek(disk, INODEPOS + src_inode * INODESIZE, SEEK_SET);
fread(&node, sizeof(node), 1, disk);

/* Check type */
if (check_type(node.mode, TYPE_DIR) == FS_IS_DIR)
    return FS_IS_DIR;

/* Move to other directory */
if (dstname[strlen(dstname) - 1] == '/') {
    dstname[strlen(dstname) - 1] = '\0';
    int dstpos, dst_node;
    inode dstnode;

    /* Check if the directory or file exists */
    for (dstpos = 0; dstpos < current_dir_num; dstpos++) {
        if (strcmp(current_dir_content[dstpos].name, dstname) == 0)
            break;
    }
    if (dstpos == current_dir_num)
        return FS_NO_EXIST;

    dst_node = current_dir_content[dstpos].inode_id;

    /* Check mode */
    if (check_mode(dst_node, CAN_WRITE) == 0)
        return FS_NO_PRIVILAGE;

    /* Read inode information */
    fseek(disk, INODEPOS + dst_node * INODESIZE, SEEK_SET);
    fread(&dstnode, sizeof(dstnode), 1, disk);

```

```

/* Check type */
if (check_type(dstnode.mode, TYPE_FILE) == FS_IS_FILE)
    return FS_IS_FILE;

/* Check same name */
dir_cd(current_inode_id, dstname);
if (check_name(srcname) == FS_FILE_EXIST) {
    dir_close(current_inode_id);
    current_inode_id = ino;
    dir_open(ino);
    return FS_FILE_EXIST;
}
dir_close(current_inode_id);
current_inode_id = ino;
dir_open(ino);

/* Delete entry */
for (; pos < current_dir_num - 1; pos++) {
    current_dir_content[pos] = current_dir_content[pos + 1];
}
current_dir_num--;

/* Free last block if need */
if (current_dir_num / DIRMAXINBLK != (current_dir_num - 1) /
DIRMAXINBLK) {
    current_inode.block_used_num--;

block_free(current_inode.block_used[current_inode.block_used_num]);
}

/* Create new entry */
dir_cd(current_inode_id, dstname);
dir_creat(current_inode_id, TYPE_FILE, srcname);

/* Copy inode */

```

```

    for (dstpos = 0; dstpos < current_dir_num; dstpos++) {
        if (strcmp(current_dir_content[dstpos].name, srcname) == 0)
            break;
    }
    dst_node = current_dir_content[dstpos].inode_id;
    fseek(disk, INODEPOS + dst_node * INODESIZE, SEEK_SET);
    fwrite(&node, sizeof(node), 1, disk);

    /* Return to src directory */
    dir_close(current_inode_id);
    current_inode_id = ino;
    dir_open(ino);
}
/* Move to current directory */
else {
    /* Check mode */
    if (check_mode(current_inode_id, CAN_WRITE) == 0)
        return FS_NO_PRIVILAGE;

    /* Check same name */
    if (check_name(dstname) == FS_FILE_EXIST)
        return FS_FILE_EXIST;

    strcpy(current_dir_content[pos].name, dstname);
    /* Save data */
    dir_close(current_inode_id);
    dir_open(ino);
}

return FS_OK;
}

/*
 * file_cp - Copy file from srcname to dstname
 */

```

```

int file_cp(int ino, char *srcname, char *dstname) {
    int pos, bno;
    int src_inode;
    inode node;
    char block[BLOCKSIZE];
    FILE *buf_fp = fopen(BUFFERFILE, "w+");

    /* Check if the directory or file exists */
    for (pos = 0; pos < current_dir_num; pos++) {
        if (strcmp(current_dir_content[pos].name, srcname) == 0)
            break;
    }
    if (pos == current_dir_num)
        return FS_NO_EXIST;

    src_inode = current_dir_content[pos].inode_id;

    /* Check mode */
    if (check_mode(current_inode_id, CAN_READ) == 0)
        return FS_NO_PRIVILAGE;

    /* Read inode information */
    fseek(disk, INODEPOS + src_inode * INODESIZE, SEEK_SET);
    fread(&node, sizeof(node), 1, disk);

    /* Check type */
    if (check_type(node.mode, TYPE_DIR) == FS_IS_DIR)
        return FS_IS_DIR;

    /* Copy file content */
    if (node.size == 0) {
        fclose(buf_fp);
        return FS_OK;
    }
    for (pos = 0; pos < node.block_used_num - 1; pos++) {

```



```

    memset(block, 0, BLOCKSIZE);
    bno = node.block_used[pos];
    fseek(disk, BLOCKPOS + bno * BLOCKSIZE, SEEK_SET);
    fread(block, sizeof(char), BLOCKSIZE, disk);
    fwrite(block, sizeof(char), BLOCKSIZE, buf_fp);
}
bno = node.block_used[pos];
fseek(disk, BLOCKPOS + bno * BLOCKSIZE, SEEK_SET);
fread(block, sizeof(char), node.size, disk);
fwrite(block, sizeof(char), node.size, buf_fp);
fclose(buf_fp);

/* Copy to other directory */
if (dstname[strlen(dstname) - 1] == '/') {
    dstname[strlen(dstname) - 1] = '\0';
    int dstpos, dst_node;
    inode dstnode;

    /* Check if the directory or file exists */
    for (dstpos = 0; dstpos < current_dir_num; dstpos++) {
        if (strcmp(current_dir_content[dstpos].name, dstname) == 0)
            break;
    }
    if (dstpos == current_dir_num)
        return FS_NO_EXIST;

    dst_node = current_dir_content[dstpos].inode_id;

    /* Check mode */
    if (check_mode(dst_node, CAN_WRITE) == 0)
        return FS_NO_PRIVILAGE;

    /* Read inode information */
    fseek(disk, INODEPOS + dst_node * INODESIZE, SEEK_SET);
    fread(&dstnode, sizeof(dstnode), 1, disk);

```

```

/* Check type */
if (check_type(dstnode.mode, TYPE_FILE) == FS_IS_FILE)
    return FS_IS_FILE;

/* Check same name */
dir_cd(current_inode_id, dstname);
if (check_name(srcname) == FS_FILE_EXIST) {
    dir_close(current_inode_id);
    current_inode_id = ino;
    dir_open(ino);
    return FS_FILE_EXIST;
}
dir_close(current_inode_id);
current_inode_id = ino;
dir_open(ino);

/* Create new entry */
dir_cd(current_inode_id, dstname);
dir_creat(current_inode_id, TYPE_FILE, srcname);

/* Save file content */
file_close(current_inode_id, srcname);

/* Return to src directory */
dir_close(current_inode_id);
current_inode_id = ino;
dir_open(ino);
}
/* Copy to current directory */
else {
    /* Check mode */
    if (check_mode(current_inode_id, CAN_WRITE) == 0)
        return FS_NO_PRIVILAGE;

```

```

        /* Check same name */
        if (check_name(dstname) == FS_FILE_EXIST)
            return FS_FILE_EXIST;

        /* Create new file */
        dir_creat(ino, TYPE_FILE, dstname);

        /* Save file content */
        file_close(ino, dstname);
    }

    return FS_OK;
}

/*****ASSIST FUNCTIONS*****/

/*
 * oct2dec - Change Octal number to decimal number
 *
 * For this program use, only supply 0~7777
 */
int oct2dec(int oct_number) {
    int dec_number = 0;
    int a, b, c, d;
    a = oct_number / 1000;
    b = (oct_number % 1000) / 100;
    c = (oct_number % 100) / 10;
    d = oct_number % 10;
    dec_number = ((a * 8 + b) * 8 + c) * 8 + d;
    return dec_number;
}

/*

```

```

    * check_name - Check if the file already exists
    */
int check_name(char *name) {
    int i;
    for (i = 0; i < current_dir_num; i++) {
        if (strcmp(name, current_dir_content[i].name) == 0)
            return FS_FILE_EXIST;
    }
    return FS_OK;
}

/*
    * check_type - Check file type
    */
int check_type(int mode, int type) {
    int isdir = mode & (1 << 9);
    if (isdir == (1 << 9) && type == TYPE_FILE)
        return FS_ISNOT_FILE;
    else if (isdir == 0 && type == TYPE_DIR)
        return FS_ISNOT_DIR;
    else if (isdir == 0 && type == TYPE_FILE)
        return FS_IS_FILE;
    else
        return FS_IS_DIR;
}

/*
    * check_mode - Check if user have privilege to do operation
    */
int check_mode(int ino, int operation) {
    inode node;
    int ret;

    fseek(disk, INODEPOS + ino * INODESIZE, SEEK_SET);
    fread(&node, sizeof(node), 1, disk);

```

```

    if (operation == CAN_READ) {
        if (node.user_id == current_user_id)
            ret = ((node.mode & (1 << 8)) == (1 << 8));
        else
            ret = ((node.mode & (1 << 2)) == (1 << 2));
    }
    else {
        if (node.user_id == current_user_id)
            ret = ((node.mode & (1 << 7)) == (1 << 7));
        else
            ret = ((node.mode & (1 << 1)) == (1 << 1));
    }
    return ret;
}

/*
 * path_cd - Change path when enter a directory
 */
void path_change(int old_inode_id, char *name) {
    int pos;
    if (!strcmp(name, ".") || (!strcmp(name, "..") && (old_inode_id == 0)))
        return;
    else if (!strcmp(name, "..") && current_inode_id != 0) {
        for (pos = strlen(path) - 1; pos >= 0; pos--) {
            if (path[pos] == '/') {
                path[pos] = '\0';
                strcat(path, ">");
                break;
            }
        }
    }
    else if (!strcmp(name, "..") && current_inode_id == 0) {
        for (pos = strlen(path) - 1; pos >= 0; pos--) {
            if (path[pos] == '/') {

```

```

        path[pos + 1] = '\0';
        strcat(path, ">");
        break;
    }
}
}
else if (path[strlen(path) - 3] == '/') {
    path[strlen(path) - 2] = '\0';
    strcat(path, name);
    strcat(path, ">");
}
else {
    path[strlen(path) - 2] = '\0';
    strcat(path, "/");
    strcat(path, name);
    strcat(path, ">");
}
}

/*
 * mtime_change - Change modified time for a file
 */
int mtime_change(int ino, char *name) {
    int i;
    int ch_node;
    inode node;

    /* Check if the directory or file exists */
    for (i = 0; i < current_dir_num; i++) {
        if (strcmp(current_dir_content[i].name, name) == 0)
            break;
    }
    ch_node = current_dir_content[i].inode_id;

    /* Check if this is a directory */

```

```

fseek(disk, INODEPOS + ch_node * INODESIZE, SEEK_SET);
fread(&node, sizeof(node), 1, disk);

/* Update modify time */
time_t timer;
time(&timer);
node.modify_time = timer;

/* Save node data */
fseek(disk, INODEPOS + ch_node * INODESIZE, SEEK_SET);
fwrite(&node, sizeof(node), 1, disk);

return FS_OK;
}

/*
 * path_cd - Change path when enter a directory
 */
void get_modestr(char *modstr, int mode) {
    strcpy(modstr, "-----");
    if ((mode & (1 << 9)) == (1 << 9))
        modstr[0] = 'd';
    if ((mode & (1 << 8)) == (1 << 8))
        modstr[1] = 'r';
    if ((mode & (1 << 7)) == (1 << 7))
        modstr[2] = 'w';
    if ((mode & (1 << 6)) == (1 << 6))
        modstr[3] = 'x';
    if ((mode & (1 << 5)) == (1 << 5))
        modstr[4] = 'r';
    if ((mode & (1 << 4)) == (1 << 4))
        modstr[5] = 'w';
    if ((mode & (1 << 3)) == (1 << 3))
        modstr[6] = 'x';
    if ((mode & (1 << 2)) == (1 << 2))

```

```

        modstr[7] = 'r';
    if ((mode & (1 << 1)) == (1 << 1))
        modstr[8] = 'w';
    if ((mode & 1) == 1)
        modstr[9] = 'x';
}

/*
 * mode_change - Change a file's mode
 */
int mode_change(int mode, char *name) {
    int i;
    int change_inode;
    inode node;

    /* Invalid mode */
    if (mode < 0 || mode > 777)
        return FS_INVALID_MODE;

    /* Check if the directory or file exists */
    for (i = 0; i < current_dir_num; i++) {
        if (strcmp(current_dir_content[i].name, name) == 0)
            break;
    }
    if (i == current_dir_num)
        return FS_NO_EXIST;
    change_inode = current_dir_content[i].inode_id;

    /* Check if this is a directory */
    fseek(disk, INODEPOS + change_inode * INODESIZE, SEEK_SET);
    fread(&node, sizeof(node), 1, disk);

    /* Only owner can change the mode */
    if (node.user_id != current_user_id)
        return FS_NO_PRIVILAGE;
}

```



```

mode = oct2dec(mode);
mode |= (node.mode & (1 << 9));

node.mode = mode;

/* Save node data */
fseek(disk, INODEPOS + change_inode * INODESIZE, SEEK_SET);
fwrite(&node, sizeof(node), 1, disk);

return FS_OK;
}

/*
 * check_if_readonly - Check if a file is readonly
 */
int check_if_readonly(int ino, char *name) {
    int pid, status;
    int check_inode;
    int bno, pos;
    inode node;
    char block[BLOCKSIZE];
    char *vim_arg[] = {"vim", BUFFERFILE, NULL};
    FILE *buf_fp = fopen(BUFFERFILE, "w+");

    /* Check if the directory or file exists */
    for (check_inode = 0; check_inode < current_dir_num; check_inode++) {
        if (strcmp(current_dir_content[check_inode].name, name) == 0)
            break;
    }
    if (check_inode == current_dir_num)
        return FALSE;

    check_inode = current_dir_content[check_inode].inode_id;

```

```

/* Check mode */
if (check_mode(check_inode, CAN_READ) == 0 ||
check_mode(check_inode, CAN_WRITE) == 1)
    return FALSE;

/* Read only */

/* Read inode information */
fseek(disk, INODEPOS + check_inode * INODESIZE, SEEK_SET);
fread(&node, sizeof(node), 1, disk);

/* Check type */
if (check_type(node.mode, TYPE_DIR) == FS_IS_DIR)
    return FALSE;

/* Read data from disk */
for (pos = 0; pos < node.block_used_num - 1; pos++) {
    memset(block, 0, BLOCKSIZE);
    bno = node.block_used[pos];
    fseek(disk, BLOCKPOS + bno * BLOCKSIZE, SEEK_SET);
    fread(block, sizeof(char), BLOCKSIZE, disk);
    fwrite(block, sizeof(char), BLOCKSIZE, buf_fp);
}
bno = node.block_used[pos];
fseek(disk, BLOCKPOS + bno * BLOCKSIZE, SEEK_SET);
fread(block, sizeof(char), node.size, disk);
fwrite(block, sizeof(char), node.size, buf_fp);

fclose(buf_fp);

if ((pid = fork()) == 0) {
    execvp("vim", vim_arg);
}

wait(&status);

```

```
    return TRUE;  
}
```