華中科技大學

操作系统原理课程设计报告

姓　　名：　　徐　永　鑫

学　　院：　　计算机科学与技术学院

专　　业：　　计算机科学与技术

班　　级：　　计算机卓越 1601

学　　号：　　U201610002

指导教师：　　郑　然

分数	
教师签名	

2019 年 3 月 18 日

目　录

1 实验一 文件拷贝与并发程序

1.1 实验要求与内容

掌握 Linux 的使用方法，熟悉和理解 Linux 编程环境。

1.编写一个 C 程序，用 fread、fwrite 等库函数实现文件拷贝功能。命令形式：copy <源文件名> <目标文件名>

2.编写一个 C 程序，用 QT 或 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时间，一个窗口按行显示/etc/fstab 文件的内容，一个窗口显示 1 到 1000 的累加求和过程，每秒刷新一次)。

1.2 实验设计

1.2.1 开发环境

操作系统	**Ubuntu Linux 18.04 LTS (Linux version 4.15.0-43-generic)**
编译器	gcc (Ubuntu 7.3.0-27ubuntu1~18.04) 7.3.0
编译指令	gcc exp1.c -lpthread a.out
集成开发环境	CLion 2018.3.2
CMake 版本	3.13.2(3.5 以上即可)
GUI 版本	GTK+ 2.0

1.2.2 实验设计

为了加快文件复制的速度，本次实验采用了用两个线程共享进程的公有数据构成的环形缓冲，将源文件复制到目标文件，实现两个线程的誊抄。

由同一个进程创建的两个线程共享进程代码段、进程的公有数据(利用这些共享的数据，线程很容易的实现相互之间的通讯)、进程打开的文件描述符、信号的处理器、进程的当前目录和进程用户 ID 与进程组 ID。

但由于两个线程共享上述的内容，所以对线程临界区域的访问需要严格限制，防止出现与时间有关的错误。

环形缓冲的原理：

缓冲是用来缓和 CPU 执行与 IO 设备读写速度不匹配而诞生的，其原理如下：

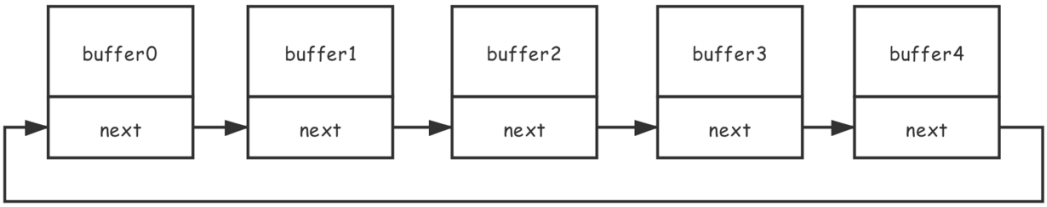

图 1.1 环形缓冲原理图

如图所示，假设环形缓冲中有 5 个缓冲区，假设 CPU 负责读，IO 设备写，那么 IO 设备会循环地按顺序地向环形缓冲区中写入数据，当 CPU 来取数据时，也会顺序地从缓冲区中读入已经写好的数据，由此提高读写效率。

可以推断出，环形缓冲整个(buffer + next)都应该设置为共享内存（由于是线程，所以在此次实验中不需要专门实现共享内存），在 CPU 读写和 IO 读写也需要同步控制。

程序设计

这里用到的环形缓冲区是以链表形式存储的，与前文不同的是，增加了 length 属性，详见下图：

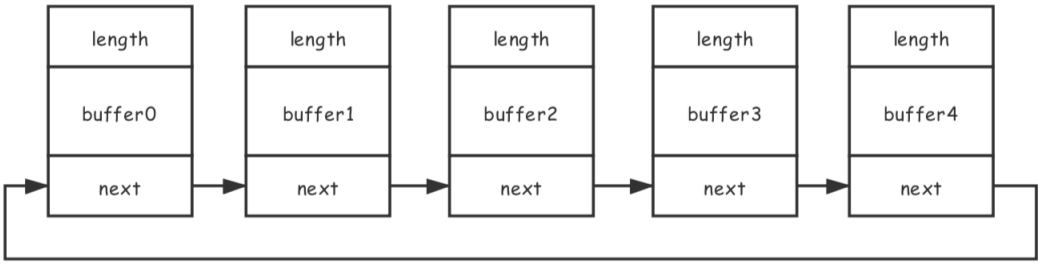

图 1.2 环形缓冲实现

如上图 1.2 所示，length 表明了缓冲区实际存的 buffer 宽度，若等于 buffer 宽度说明正常读写，若小于 buffer 宽度则是读写的最后一块，若大于 buffer(是写 buffer 进程手动添加的，用于明确标识结束，为了防止最后一块的 length 正好等于 buffer 的 size，而不知道停止造成死锁)。

进程的产生仍然使用子进程，父进程 fork 两个子进程，然后两个兄弟进程之间相互通信。

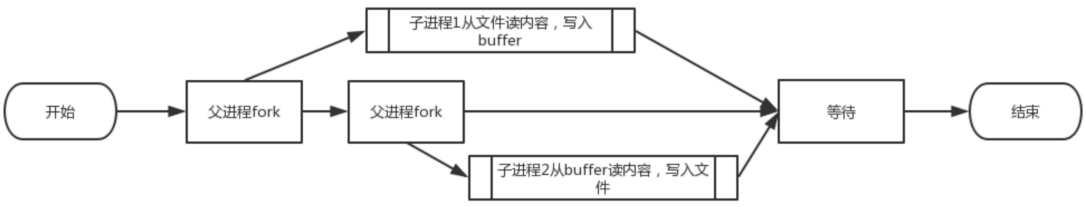

图 1.3 整体框架

对于子进程 1 和子进程 2 的同步伪代码如下（这里缓冲数量设为 10，每个缓冲区 buffer 大小设置为 4096B，也就是 4KB）：

```
int main() {
    semaphore SEM_WRITE_BUFFER = 10 /* how many buffers can child1 write */
    semaphore SEM_READ_BUFFER = 0;  /* how many buffers can child2 read */
    cobegin:
        write_to_buffer();  /* child process 1 */
        read_from_buffer(); /* child process 2 */
    coend
}
write_to_buffer() {
    ptr = head;
    P(SEM_WRITE_BUFFER);
    ptr->length = get_file_size(fd);
    ptr = ptr->next;
    V(SEM_READ_BUFFER);
    while(true) {
        P(SEM_WRITE_BUFFER);
        len = write(fd, ptr->buffer, 4096);
        ptr->length = len;
        if(len > 4096)
ptr->length);
            break;
        ptr = ptr->next;
        V(SEM_READ_BUFFER);
    }
    P(SEM_WRITE_BUFFER);
    ptr->length = 4096 + 1;
    ptr = ptr->next;
    V(SEM_READ_BUFFER);
}
```

```
read_from_buffer() {
    ptr = head;
    P(SEM_READ_BUFFER);
    len = ptr->length;
    ptr = ptr->next;
    V(SEM_WRITE_BUFFER);
    while(true) {
        P(SEM_READ_BUFFER);
        if(ptr->length > 4096)
            break;
        read(fd, ptr->buffer,

        ptr = ptr->next;
        V(SEM_WRITE_BUFFER);
    }
 }
```

写 buffer 的进程第一次写入所有数据的总长度，最后一次写入一个长度比 buffer_size 还要大的数据块来标识已经读完了。读 buffer 的进程第一次读取所有数据的长度，读到一个长度比 buffer_size 还要打的数据块表示已经读完了。

此外还有进度调的设置，根据当前已经写入数据的大小和总大小的关系，在屏幕上打印进度条展示速度。

对于共享内存的使用步骤如下：

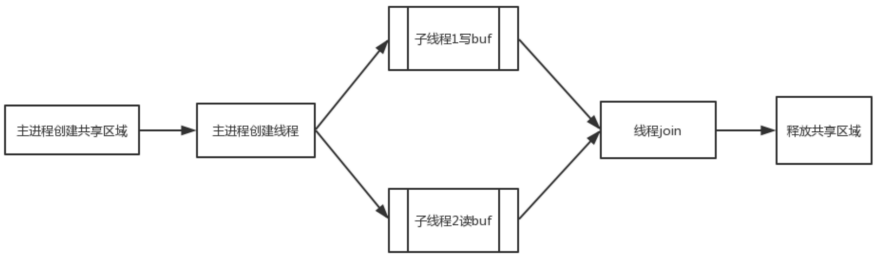

图 1.4 共享区域控制

对于多进程并发显示不同的内容，首先根据选择的 GUI 方法(GTK+2.0)利用 fork 函数的特性来实现功能需求。

首先，通过在主进程使用 fork()函数来创建三个子进程，对于每个不同的进程传入不同的参数已让它们实现不同的功能。

具体来说，设计的方法如下，对于每次 fork 的子进程，让他们调用 wind_proc()函数，这个函数根据调用的参数，以及第几次被调用来选择窗口实现的功能，在每个子进程中使用两个线程，第一个线程用于 GTK 窗口主事件循环，第二个线程用于提供动态显示，在每次刷新显示后，休眠一段时间让窗口主事件进行循环。

对于第一个子进程，读取本机的时间，通过 time()函数读取整个结构体，在利用格式化输出，每次输出后休眠 1 秒，再下一秒重新读入进行刷新。

对于第二个子进程，读取/etc/fstab 的内容。通过每隔一秒打开/etc/fstab 的内容，创建一个类似于文件位置指针的 static 指针(本文利用的是计算输出的行数)，来动态刷新每行的内容，然后利用 strcat 补充到之前读取的内容之后。为了彰显读取/etc/fstab 的内容，本文还利用表格解析了整个/etc/fstab 的内容。(详情可见后文截图)

对于第三个子进程，利用全局变量记录当前加数已经加到了第几个数以及总和是多少，再每个 1 秒刷新新的加数和被加数即可。

值得一提的是，上述的三个子进程都必须通过 gdk_thread_enter()和 gdk_thread_leave()进行子线程的插入与离开，本文还加入了 start 和 stop 键，子线程必须得到 start 信号后才开始运作，得到 stop 信号后结束进程(循环结束)。

1.3 实验调试

本次实验用的是 CLion IDE 实现的，所以在 CLion 环境中测试，首先建立一个 C 语言工程，CMakeList 如下：

```
cmake_minimum_required(VERSION 3.13)
project(design1_2 C)

set(CMAKE_C_STANDARD 11)

add_executable(design1_2 main.c)

find_package(Threads)
target_link_libraries(design1_2 ${CMAKE_THREAD_LIBS_INIT})
```

编译后，首先测试一个内容较小的 txt 文件，以下所有的拷贝操作均是从 osexp3 中的文件拷贝到 osexp2 中。

由于运行速度过快，没有截图到传输的进度条，运行输出如下：
对比两个 txt，发现内容相同，如下图。

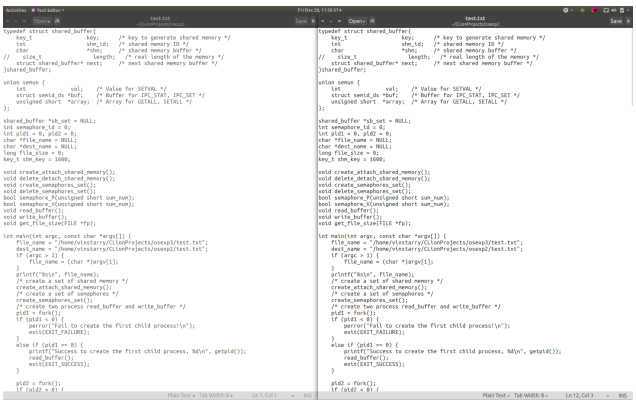

图 1.5 txt 对比效果

首先，他们的大小完全一样，再对比里面的字符，也是一模一样。可以用 diff 指令比较两个文件的区别。

结果依然是相同的。

再传输一个较大的电影文件进行比较：

用 Linux diff 指令比较两个文件的差别，显示无差别，则正确。

最后再打开视频查看是否可以正常播放，结果是可以的，截图如下：

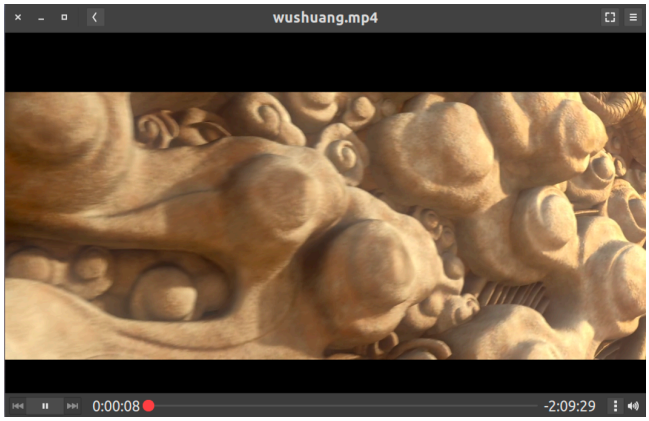

图 1.6 播放测试

下面两个截图展示了多进程显示窗口的测试结果：

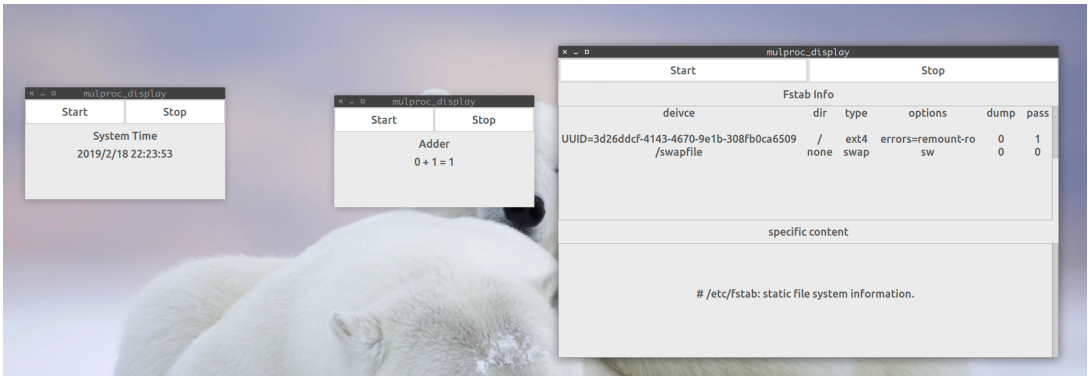

图 1.6 多进程并发测试

下图展示了按下 stop 的结果

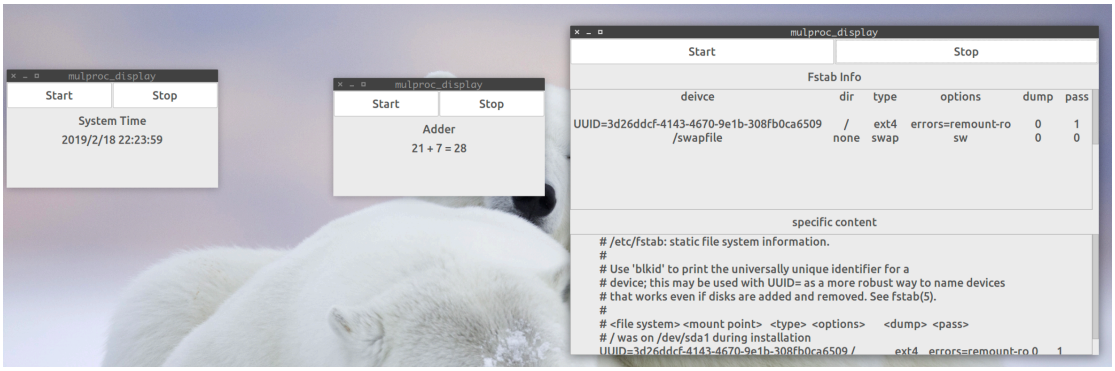

图 1.7 按下 stop，运行暂停

2 实验二 添加系统调用

2.1 实验要求与内容

掌握添加系统调用的方法。
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。
编写一个应用程序，测试新加的系统调用。

2.2 实验设计

2.2.1 开发环境

操作系统	**Ubuntu Linux 18.04 LTS (Linux version 4.15.0-43-generic)**
编译器	gcc (Ubuntu 7.3.0-27ubuntu1~18.04) 7.3.0
编译指令	gcc exp1.c -o a.out
集成开发环境	CLion 2018.3.2
CMake 版本	3.13.2(3.5 以上即可)
新内核版本	Linux-4.4.174

2.2.2 实验设计

首先需要了解系统调用的原理：

系统调用和普通库函数调用很相似，仅仅是系统调用由操作系统核心提供，执行于核心态。而普通的函数调用由函数库或用户自己提供。执行于用户态。

系统调用在用户空间进程和硬件设备之间加入了一个中间层。该层主要作用有三个：

(1).它为用户空间提供了一种统一的硬件的抽象接口。

(2).系统调用保证了系统的稳定和安全。作为硬件设备和应用程序之间的中间人，内核能够基于权限和其它一些规则对须要进行的访问进行裁决。

(3).每一个进程都执行在虚拟系统中，而在用户空间和系统的其余部分提供这样一层公共接口。也是出于这样的考虑。假设应用程序能够任意访问硬件而内核又对此一无所知的话，差点儿就没法实现多任务和虚拟内存，当然也不可能实现良好的稳定性和安全性。

之后需要了解 Linux 系统调用的过程：

应用程序通过软中断的方式，引发一个异常来促使系统切换到内核态去执行异常处理程序。此时的异常处理程序实际上就是系统调用处理程序。x86 系统上的软中断由 int $0x80 指令产生，这条指令会触发一个异常导致系统切换到内核态并执行 system_call()。系统调用号从 eax 中得到系统调用号。参数传递通过 ebx、ecx、edx、esi 和 edi 按照顺序存放前 5 个参数，需要 6 个及以上的参数情况不多见。此时，用一个单独的寄存器存放所有指向这些参数的再用户空间地址的指针。

2.3 实验调试

理解原理了之后便对系统调用进行实现：

首先，选择了 4.4.17 的 long-term Linux 版本，因为其稳定性和完整性都非常好。

然后，像写普通 C 语言程序一样，对需要实现的功能 my_copy 进行声明，在 src 目录下的./include/linux/syscalls.h 下，可以对需要添加的系统调用进行声明，声明语句截图如下：

```
asmlinkage long sys_bpf(int cmd, union bpf_attr *attr, unsigned int size);

asmlinkage long sys_execveat(int dfd, const char __user *filename,
                        const char __user *const __user *argv,
                        const char __user *const __user *envp, int flags);

asmlinkage long sys_membarrier(int cmd, int flags);

asmlinkage long sys_mlock2(unsigned long start, size_t len, int flags);

asmlinkage long sys_mycopy(const char *__src_file, const char *__dest_file);
```

图 2.1 系统调用 my_copy()声明

这里的类型 asmlinkage 是一个限定词，它是一个编译指令，通知编译器仅仅从栈中提取该函数的参数，所有的系统调用都需要这个限定词。其次，函数返回 long。为了保证 32 位和 64 位的兼容，系统调用在用户空间和内和空间都有不同的返回类型，前者为 32 位，后者为 64 位。

这里定义了 mycopy 函数，通过两个参数（第一个是源文件的路径名，一个是目的文件的路径名）。

下面需要对声明的函数进行实现，具体实现如下图 2.2

```
asmlinkage long sys_mycopy(const char *__src_file, const char *__dest_file)
{
        int src_fd, dst_fd;
        int count;
        char buf[1024];

        mm_segment_t fs;
        fs = get_fs();
        set_fs(get_ds());
        if ((src_fd = sys_open(__src_file, O_RDONLY, 0)) == -1)
        {
                return 1;
        }
        if ((dst_fd = sys_open(__dest_file, O_WRONLY | O_CREAT, S_IRUSR | S_IWUSR)) == -1)
        {
                return 2;
        }
        while ((count = sys_read(src_fd, buf, 1024)) > 0)
        {
                if (sys_write(dst_fd, buf, count) != count)
                        return 3;
        }
        if (count < 0)
                return 4;
        sys_close(src_fd);
        sys_close(dst_fd);
        set_fs(fs);
        return 0;
}
```

图 2.1 系统调用 my_copy()实现

实现的思想是首先定义两个文件描述符，然后利用 get_fs()函数保存当前的 fs，在用 set_fs(get_ds())来得到内核的 fs，从而避免内存保护检查错误，暂时将访问限制值设置为内核的内存访问范围。文件操作应使用对应的内核函数 sys_open、sys_read 等。确认两个文件名都是合法有效之后，进行循环读写完成拷贝工作。

注意：此前已经完成过了简单的系统调用输出功能等，所以此处不再用 printk 进行调试信息输出。

最后，需要在系统调用表（为每个系统调用分配唯一号码）添加新的系统调用，如下图：

```
317     common  seccomp                 sys_seccomp
318     common  getrandom               sys_getrandom
319     common  memfd_create            sys_memfd_create
320     common  kexec_file_load         sys_kexec_file_load
321     common  bpf                     sys_bpf
322     64      execveat                stub_execveat
323     common  userfaultfd             sys_userfaultfd
324     common  membarrier              sys_membarrier
325     common  mlock2                  sys_mlock2
326     64      mycopy                  sys_mycopy
```

图 2.2 系统表添加函数

在系统调用表里添加 mycopy 函数，上图中总共四列，第一列是系统调用号，直接根据之前最高的 325，再后面添加了 326 号系统调用。第二列表示的是 64 位系统编译，第三列和第四列分别定义了系统调用号的名称和系统调用的入口。

完成之后，对内核进行编译，选择新的内核版本：

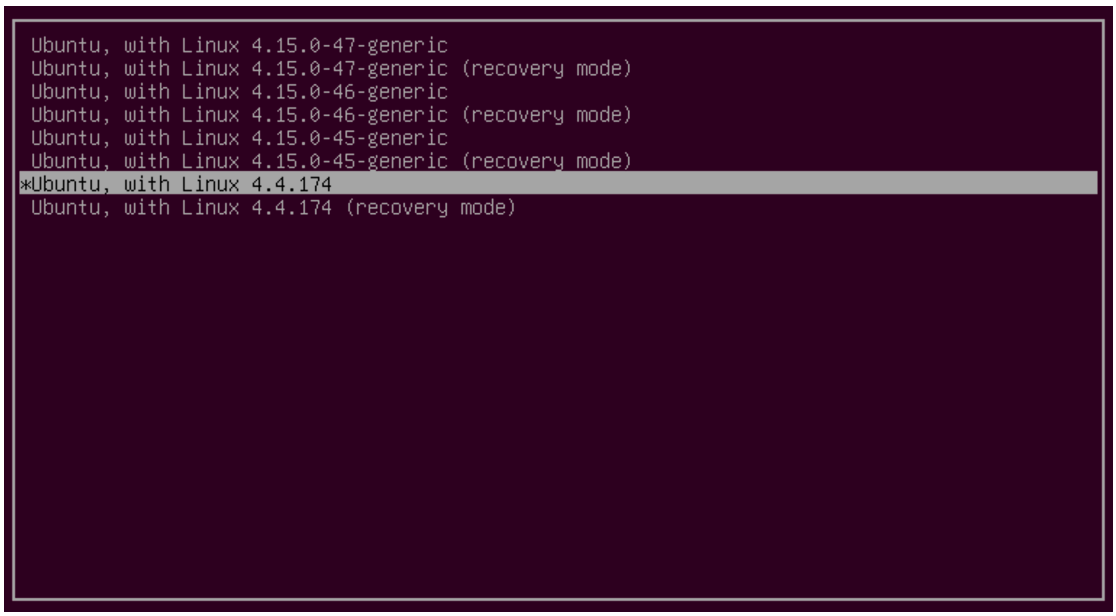

图 2.3 选择新的内核版本

然后编写测试程序，测试程序的基本思路是选择一个源文件，将其复制到另一个地址，然后同时用两个文件指针打开它们，二进制地比较文件内部的差距。

```
#include <stdio.h>
#include <linux/kernel.h>
#include <sys/syscall.h>
#include <unistd.h>

int main()
{
        long int copy_file = syscall(326, "paper.pdf", "temp/paper.pdf");
        printf("the syscall function returned %ld\n", copy_file);

        FILE *fp_src, *fp_dst;
        fp_src = fopen("paper.pdf", "rb");
        fp_dst = fopen("temp/paper.pdf", "rb");

        int result = 1;
        char ch1, ch2;

        while ( (ch1 = (char)fgetc(fp_src) != EOF) && (ch2 = (char)fgetc(fp_dst)!=EOF )) {
                if (ch1 != ch2) {
                        result = 0;
                        break;
                }
        }
        if (result == 1)
                printf("src file and dst file are the same.\n");
        else
                printf("not same!\n");
        fclose(fp_src);
        fclose(fp_dst);
        return 0;
}
```

图 2.4 测试程序截图

测试结果为源文件和目的文件没有区别，在此不再截图。

3 实验三 添加设备驱动

3.1 实验要求与内容

掌握添加设备驱动程序的方法。

采用模块方法，添加一个新的字符设备的驱动程序，实现打开/关闭、读/写等基本操作。

编写一个应用程序，测试添加的驱动程序。

3.2 实验设计

3.2.1 开发环境

操作系统	**Ubuntu Linux 18.04 LTS (Linux version 4.15.0-43-generic)**
编译器	gcc (Ubuntu 7.3.0-27ubuntu1~18.04) 7.3.0
编译指令	gcc exp1.c -o a.out
集成开发环境	CLion 2018.3.2
CMake 版本	3.13.2(3.5 以上即可)

3.2.2 实验设计

内核驱动设备原理

Linux 内核中的设备驱动程序是一组常驻内存的具有特权的共享库，是低级硬件处理例程。对用户程序而言，设备驱动程序隐藏了设备的具体细节，对各种不同设备提供了一致的接口，一般来说是把设备映射为一个特殊的设备文件，用户程序可以象对其它文件一样对此设备文件进行操作。

Linux 支持 3 种设备：字符设备、块设备和网络设备。

设备由一个主设备号和一个次设备号标识。主设备号唯一标识了设备类型，即设备驱动程序类型，它是块设备表或字符设备表中设备表项的索引。次设备号仅由设备驱动程序解释,一般用于识别在若干可能的硬件设备中，I/O 请求所涉及到的那个设备。

一个典型的驱动程序,大体上可以分为这么几个部分:

①注册设备：

在系统初启,或者模块加载时候,必须将设备登记到相应的设备数组,并返回设备的主设备号；

②定义功能函数：

对于每一个驱动函数来说，都有一些和此设备密切相关的功能函数。以最常用的块设备或者字符设备来说，都存在着诸如 open()、read()这一类的操作。当系统调用这些调用时，将自动的使用驱动函数中特定的模块。来实现具体的操作；

③卸载设备：

在不用这个设备时,可以将它卸载，主要是从/proc 中取消这个设备的特殊文件。

3.3 实验调试

首先，编写 Makefile 文件如下：

```
ifneq ($(KERNELRELEASE),)
obj-m := zcydriver.o        //obj-m表示编译连接后将生成driver.o模块
else
PWD :=$(shell pwd)          //PWD为当前目录
KVER :=$(shell uname -r)        //KVER为当前系统内核版本
KDIR :=/lib/modules/$(KVER)/build
all:
    $(MAKE) -C $(KDIR) M=$(PWD) //调用内核模块编译
clean:
#   rm -f *.cmd *.o *.mod *.ko
    rm -rf .*.cmd *.o *.mod.c *.ko .tmp_versions
#   $(MAKE) -C $(KDIR) M=$(PWD) clean
endif
```

调用 Makefile 文件之后，其具体过程如下：

①KERNELRELEASE 是在内核源码的顶层 Makefile 中定义的一个变量，在第一次读取执行此 Makefile 时，KERNELRELEASE 没有被定义，所以 make 将读取执行 else 之后的内容；

②如果 make 的目标是 clean，直接执行 clean 操作，然后结束。

③当 make 的目标为 all 时，-C $(KDIR)指明跳转到内核源码目录下读取那里的 Makefile；M=$(PWD) 表明然后返回到当前目录继续读入、执行当前的 Makefile。

④当从内核源码目录返回时，KERNELRELEASE 已被定义，内核的 build 程序 Kbuild 也被启动去解析 kbuild 语法的语句，make 将继续读取 else 之前的内容。

⑤else 之前的内容为 kbuild 语法的语句，指明模块源码中各文件的依赖关系，以及要生成的目标模块名。

然后编写驱动程序，驱动程序需要编写 init_module 初始化模块和清空模块 cleanup_module 程序，再自定义一些文件操作，为了实现基本功能，本文定义了以下函数

```c
static int dev_open(struct inode *inode, struct file *file)   {
    // 打开设备
}
static int dev_release(struct inode *inode, struct file *file) {
    // 释放设备
}
static ssize_t dev_read(struct file *file, char __user *user, size_t t, loff_t *f) {
    // 设备读操作
}
static ssize_t dev_write(struct file *file, const char __user *user, size_t t, loff_t
*f) {
    // 设备写操作
}
```

具体的源代码可以参考附录，最后，把实现好的函数装在入设备文件操作的指针中即可。

```c
struct file_operations pStruct = { open:dev_open, release:dev_release, read:dev_read,
write:dev_write, };
```

（4）设备加载，安装过程如下：

①进入Makefile文件和zcydriver.c文件所在目录，清除make产生的残留文件。命令为：

```
make  clean
```

②删除先前可能加载过的模块，命令为：

```
rmmod  /dev/xyxdriver
```

③卸载设备：

```
rm   /dev/xyxdriver
```

④编译设备文件，产生模块文件

```
make
```

⑤加载模块

```
sudo insmod  xyxdriver.ko
```

⑥加载设备，分配设别号

```
sudo mknod  /dev/xyxdriver c 240 0
```

⑦更改用户对设备的操作权限为可读、可写

```
chmod  666 /dev/xyxdriver
```

（5）运行测试程序，检验设备是否可读写：检测程序见报告附录的源代码，测试过程如下图 3-1 所示：

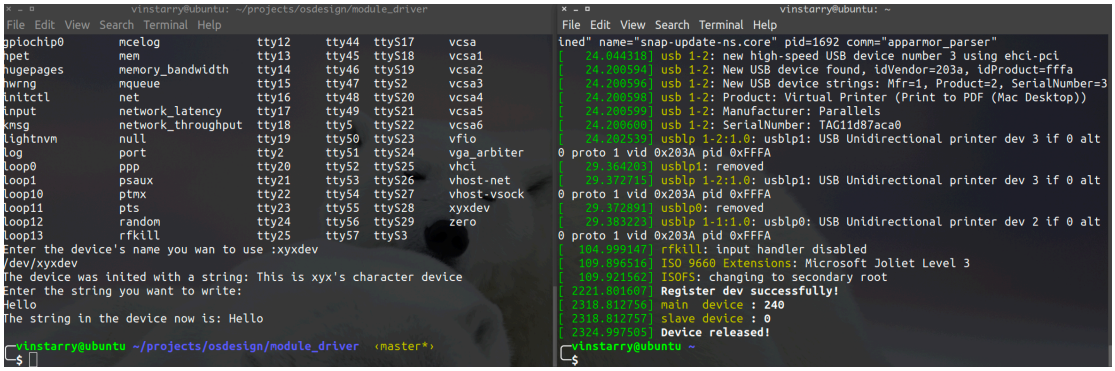

图 3-1 设备驱动测试程序信息(左)和内核输出信息(右)

如图 3-1 所示，左边展示的是对于设备的读写操作，一开始在设备中预留的信息已经被读出，再往设备中写字符串 Hello，也被接收到，且若之后再次读取驱动信息，可以看到预留信息被改为 Hello，比较右图和实现的设备驱动程序，可以验证程序的正确性。

4 实验四 Proc 文件分析

4.1 实验要求与内容

理解和分析/proc 文件。

了解/proc 文件的特点和使用方法。

周期性监控系统状态，显示系统部件的使用情况。

用图形界面监控系统状态，包括 CPU、内存、磁盘和网络使用情况（要求可排序）、进程信息等(可自己补充、添加其他功能)。

4.2 实验设计

4.2.1 开发环境

操作系统	**Ubuntu Linux 18.04 LTS (Linux version 4.15.0-43-generic)**
编译器	gcc (Ubuntu 7.3.0-27ubuntu1~18.04) 7.3.0
编译指令	gcc exp1.c -o a.out
集成开发环境	CLion 2018.3.2
CMake 版本	3.13.2(3.5 以上即可)

4.2.2 实验设计

(1)设计需要显示的模块。

本文准备通过对 proc 里的文件进行分析，主要动态刷新并显示三个主要模块以及从属子模块的信息，文件模块类型设计如下表 4-1：

表 4-1 proc 文件分析内容与文件路径表格

子模块	所属模块	主要显示内容	动态刷新
系统概要	系统模块	版本与时间	是
系统总体信息	系统模块	主机、CPU 等信息	否
CPU 具体信息	系统模块	CPU 详细参数	否
进程总体	进程模块	CPU 或内存利用率	是
进程具体	进程模块	所有进程摘要信息	是
内存具体信息	内存模块	内存详细参数	是

由于多数 CPU 信息的参数、以及系统的具体信息几乎不会改变，所以设置成了静态模块，且系统摘要的时间改变较慢（当使用分钟或小时计数时），所以刷新频率也比较低。

通过查阅资料，把需要分析的 proc 文件和内容还有设计的分组罗列如下表 4-2：

表 4-1 proc 文件分析内容与文件路径表格

路径名	文件内容	所属子模块	所属模块
/proc/version	内核和编译器版本	系统概要	系统模块
/proc/uptime	系统开启时间等	系统概要	系统模块
/proc/cpuinfo	CPU 具体	CPU 具体信息	系统模块
/proc/sys/kernel/hostname	主机名	系统总体	系统模块
/proc/cpuinfo	CPU 具体	CPU 占用率	进程模块
/proc/meminfo	内存具体	内存占用率	进程模块
/proc/stat	进程总体信息	进程总体	进程模块
/proc/pid/stat	进程具体信息	进程具体	进程模块
/proc/meminfo	内存具体信息	内存具体	内存模块

(2)设计需要显示的模块后，进行具体的显示 UI 设计，详情见下图 4.1。

图 4.1 UI 具体显示设计

如图所示，标签页有三个，分别表示显示系统相关的信息、进程相关的信息和内存相关的信息等。在每个标签页所代表的页面中，都由几个独立的模块构成，每个模块内有标题和主体部分，根据子模块的类型选择主体部分刷新与否。

进程模块是本文设计的核心模块，首先需要调研进程文件的/proc/pid/stat 文件，以 1 号进程为例子，文件的内容截图如下

```
vinstarry@ubuntu ~
$ cat /proc/1/stat
1 (systemd) S 0 1 1 0 -1 4194560 16163 1792003 88 1641 44 169 5427 1963 20 0 1 0
 1 163725312 2334 18446744073709551615 1 1 0 0 0 0 671173123 4096 1260 0 0 0 17
0 0 0 107 0 0 0 0 0 0 0 0 0 0
```

图 4.2 进程信息明细图(以 1 号进程为例)

查阅网上资料，对以上的内容进行解释如下：

pid=1 表示进程(包括轻量级进程，即线程)号；

systemd 表示当前进程是系统初始化的；

task_state=S 任务的状态，R:runnign, S:sleeping (TASK_INTERRUPTIBLE), D:disk sleep (TASK_UNINTERRUPTIBLE), T: stopped, T:tracing stop,Z:zombie, X:dead；

ppid=0 表示父进程 ID；

pgid=1 表示线程组号；

sid=1 c 该任务所在的会话组 ID；

tty_nr=0(pts/3)表示该任务的 tty 终端的设备号，INT（34817/256）=主设备号，（34817-主设备号）=次设备号；

tty_pgrp=-1 终端的进程组号，当前运行在该任务所在终端的前台任务(包括 shell 应用程序)的 PID。

task->flags=4194560 进程标志位，查看该任务的特性

min_flt=16163 该任务不需要从硬盘拷数据而发生的缺页（次缺页）的次数

cmin_flt=1792003 累计的该任务的所有的 waited-for 进程曾经发生的次缺页的次数目。

maj_flt=88 该任务需要从硬盘拷数据而发生的缺页（主缺页）的次数。

cmaj_flt=0 累计的该任务的所有的 waited-for 进程曾经发生的主缺页的次数。

utime=1641 该任务在用户态运行的时间，单位为 jiffies

stime=1 该任务在核心态运行的时间，单位为 jiffies

cutime=0 累计的该任务的所有的 waited-for 进程曾经在用户态运行的时间，单位为 jiffies

cstime=0 累计的该任务的所有的 waited-for 进程曾经在核心态运行的时间，单位为 jiffies

priority=25 任务的动态优先级

nice=0 任务的静态优先级

num_threads=3 该任务所在的线程组里线程的个数

it_real_value=0 由于计时间隔导致的下一个 SIGALRM 发送进程的时延，以 jiffy 为单位.

start_time=5882654 该任务启动的时间，单位为 jiffies

vsize=1409024（page） 该任务的虚拟地址空间大小

rss=56(page) 该任务当前驻留物理地址空间的大小

rlim=4294967295（bytes） 该任务能驻留物理地址空间的最大值

start_code=134512640 该任务在虚拟地址空间的代码段的起始地址

end_code=134513720 该任务在虚拟地址空间的代码段的结束地址

start_stack=3215579040 该任务在虚拟地址空间的栈的结束地址

kstkesp=0 esp(32 位堆栈指针) 的当前值, 与在进程的内核堆栈页得到的一致.

kstkeip=2097798 指向将要执行的指令的指针, EIP(32 位指令指针)的当前值.

pendingsig=0 待处理信号的位图，记录发送给进程的普通信号

block_sig=0 阻塞信号的位图

sigign=0 忽略的信号的位图

sigcatch=082985 被俘获的信号的位图

wchan=0 如果该进程是睡眠状态，该值给出调度的调用点

nswap 被 swapped 的页数，当前没用

cnswap 所有子进程被 swapped 的页数的和，当前没用

exit_signal=17 该进程结束时，向父进程所发送的信号

task_cpu(task)=0 运行在哪个 CPU 上

task_rt_priority=0 实时进程的相对优先级别

task_policy=0 进程的调度策略，0=非实时进程，1=FIFO 实时进程；2=RR 实时进程

在了解了所有进程信息所表示的意义之后，查询了一下 Liunx 系统主要的进程信息。

在命令行输入 ps -aux，得到输出信息如下：

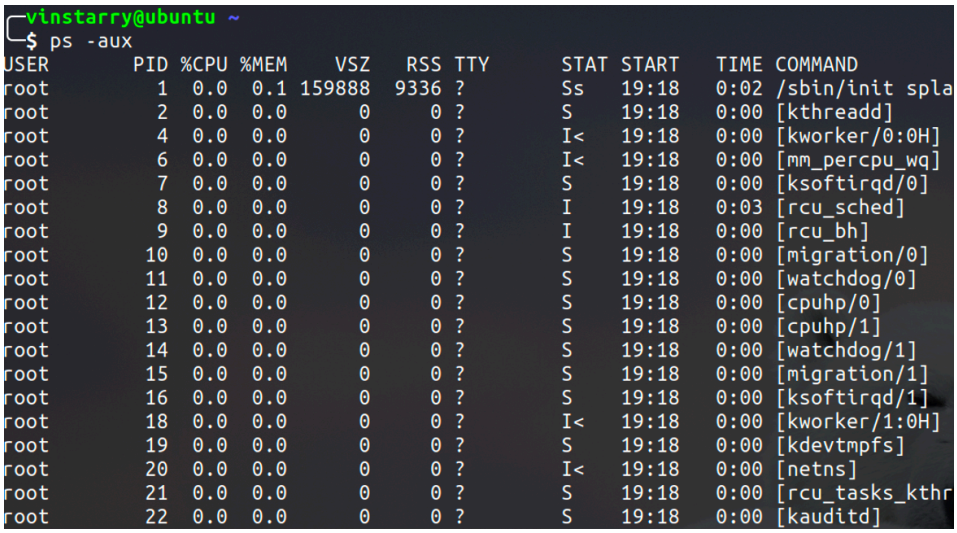

图 4.3 系统 ps -aux 函数如下

其中第一列为用户，第二列为进程 ID，后面两列为 CPU 占用率和内存占用率等。除了 CPU 和内存占用率，其他的数据都可以直接读出。首先进入/proc 目录下，对于所有数字 (即 PID)命名的目录，对于 stat 文件进行规格化读入即可。

此外，还需要支持可以对于某类信息进行排序，所以需要用一个结构体列表进行信息的存储，然后利用链表排序根据关键词进行排序，链表排序算法的伪代码如下：

```
链表指针 temp_head = head;
for i in rows :
    链表指针 base = temp_head->next;
    bool change = false;
    mps *min = base;
    for j = i to rows - 1
        bool smaller = false;
        根据特殊的关键词进行排序
    if (change):
        链表指针 *temp_prev = head;
        while(temp_prev->next != min)
            temp_prev = temp_prev->next;
        temp_prev->next = min->next;
        min->next = temp_head->next;
        temp_head->next = min;
    temp_head = temp_head->next;
```

图 4.4 链表排序伪代码

对于每个进程的利用率，用以下方法进行计算：

1. 计算指定进程的 cpu 使用率

①.计算系统的运行时间

```
cat /proc/uptime
```

得到系统运行的总时间 up_time

②. 计算 process 的运行时间

```
cat /proc/[pid]/stat
```

得到进程运行的总时间 total_time 和进程的启动时间 start_time。

其中，total_time = stime + utime

③.获取系统时钟频率

```
hz = sysconf(_SC_CLK_TCK);
```

④.计算 cpu 使用率

```
seconds = up - (start_time / hz);
```

从进程启动到当前的时间，单位为秒

```
pcpu = (((total_time * 1000ULL) / hz) / seconds);
```

2. 计算指定进程的内存使用率

①. 获取系统的 memsize

```
cat /proc/meminfo
```

②.获取进程的 page 数

```
cat /proc/[pid]/stat
```

③.获取系统的 page size(单位是 Byte)

```
sysconf(_SC_PAGE_SIZE)
```

④.计算内存使用率

```
rss = page_cnt * page_size / 1024;
pmem = (rss * 1000UL) / memsize;
```

对于动态监测总体的 CPU 利用率和内存利用率，需要首先确定利用率的计算方法。

对于内存的占用率的统计方法，只需要动态地读取同一时刻的/proc/info 中的 MemTotal 和 MemFree，即总内存和空闲内存，得到内存利用率公式如下：

Mem% = (MemTotal – MemFree) / MemTotal

在 Linux 系统中，可以用/proc/stat 文件来计算 CPU 的利用率。这个文件包含了所有 CPU 活动的信息，该文件中的所有值都是从系统启动开始累计到当前时刻。

例如：利用 cat /proc/stat，得到以下输出图，(部分输出信息被截取)。

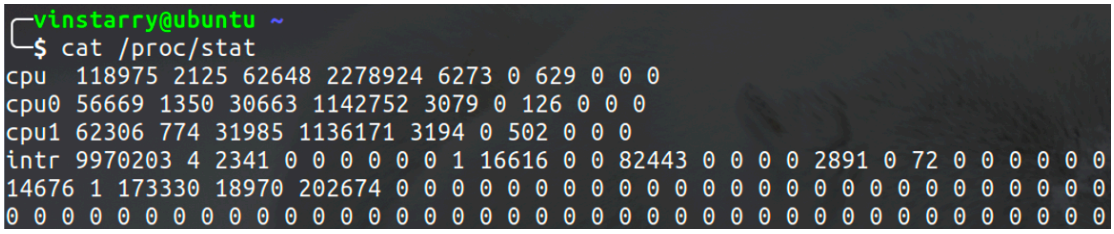

图 4.5 /proc/stat 信息

下面对上图的内容信息进行分析(/proc/stat)：

user (426215)从系统启动开始累计到当前时刻，用户态的 CPU 时间（单位：jiffies）不包含 nice 值为负进程。1jiffies=0.01 秒

nice(701)从系统启动开始累计到当前时刻，nice 值为负的进程所占用的 CPU 时间（单位：jiffies）

system (115732) 从系统启动开始累计到当前时刻，核心时间（单位：jiffies）

idle (2023866)从系统启动开始累计到当前时刻，除硬盘 IO 等待时间以外其它等待时间（单位：jiffies）

iowait (27329)从系统启动开始累计到当前时刻，硬盘 IO 等待时间（单位：jiffies）。

irq (4)从系统启动开始累计到当前时刻，硬中断时间（单位：jiffies）

softirq (557)从系统启动开始累计到当前时刻，软中断时间（单位：jiffies）

CPU 时间=user+system+nice+idle+iowait+irq+softirq

“intr”这行给出中断的信息，第一个为自系统启动以来，发生的所有的中断的次数；然后每个数对应一个特定的中断自系统启动以来所发生的次数。

“ctxt”给出了自系统启动以来 CPU 发生的上下文交换的次数。

“btime”给出了从系统启动到现在为止的时间，单位为秒。

“processes (total_forks) 自系统启动以来所创建的任务的个数目。

“procs_running”：当前运行队列的任务的数目。

“procs_blocked”：当前被阻塞的任务的数目。

那么 CPU 利用率可以使用以下两个方法。先取两个采样点，然后计算其差值：

cpu usage=(idle2-idle1)/(total_2 - total_1)*100

cpu usage=[(user_2 +sys_2+nice_2) - (user_1 + sys_1+nice_1)]/(total_2 - total_1)*100

对于 CPU 和内存的使用率，用折线图的方式动态展示。画图的方法如下：

① 首先，生成一个画布，并在上面画出水平线(y 轴刻度线)。

② 然后开辟一段定长的数组(本文开辟的长度为 10)，每隔一秒获取一个新的 CPU 利用率，放入数组中，若数组已满，则循环向前移动一位，并将第 0 位移出。

③ 将 10 个点输出到画布上，并对每相邻的两个点连线。

4.3 实验调试

根据上述的实验设计，完成工程并进行调试。

首先，调试 proc 文件分析项目的主界面 UI：

System 模块的界面如下：

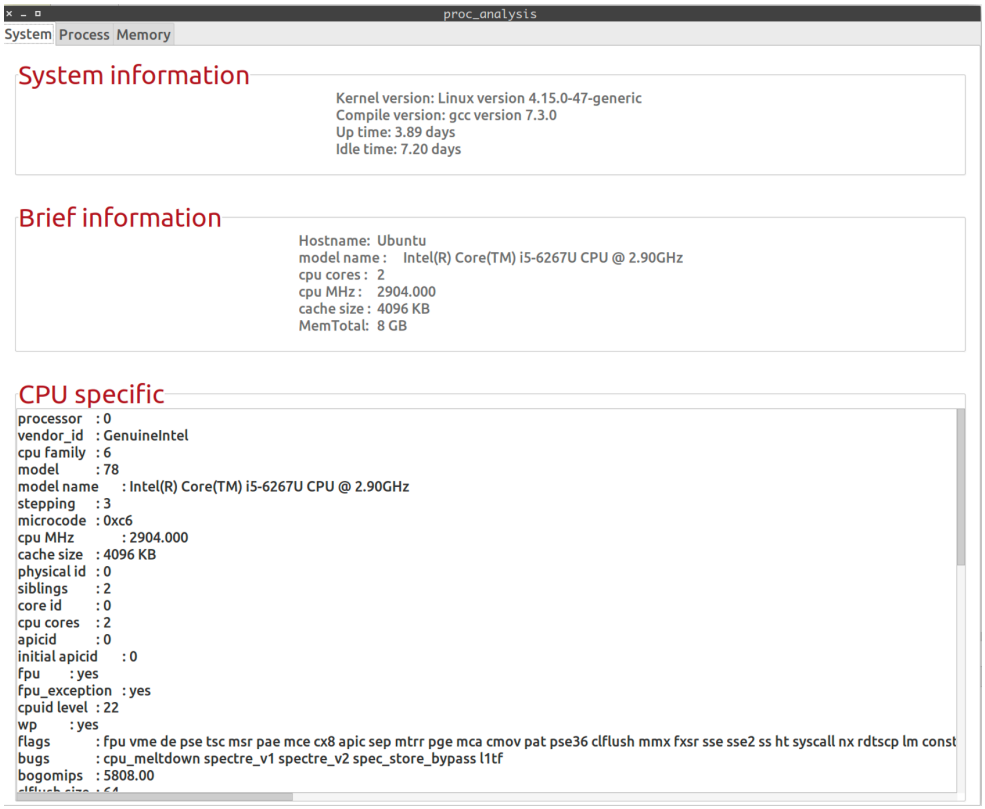

图 4.6 System 模块界面

如上图所示，System 模块由三栏内容组成，参考前文的子模块划分，第一栏是系统的基本信息，这个信息时动态刷新的，第二个和第三个模块分别是简介信息和 CPU 详细信息，这两个模块是静态的，符合设计要求。

下面测试 Process 模块，如下图 4.7 所示：

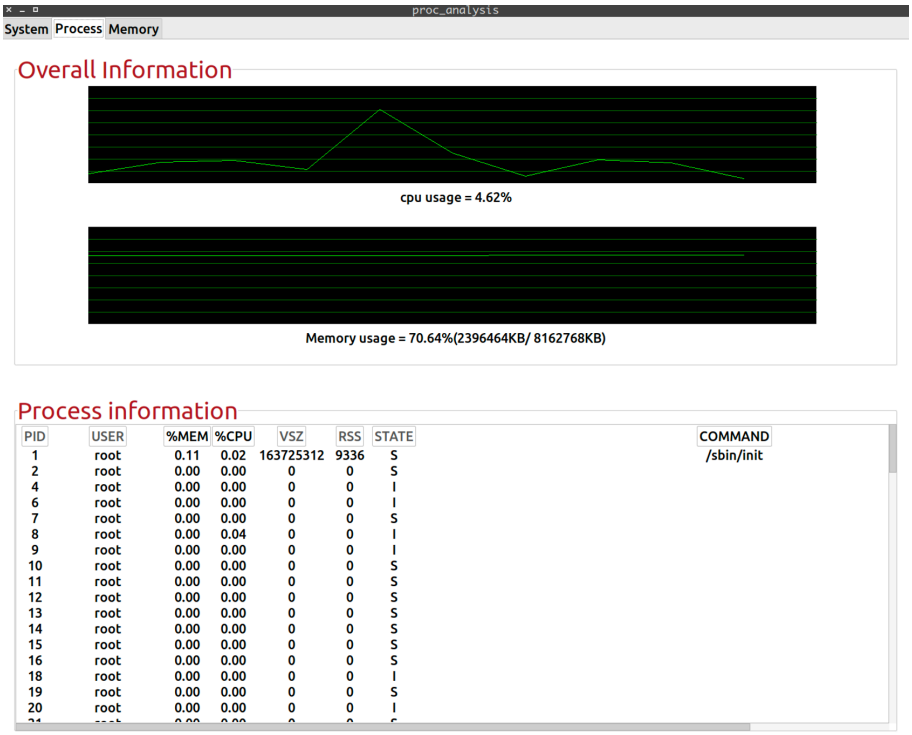

图 4.7 Process 模块界面

对于 Process 界面，可以看到 CPU 和内存的变化曲线图。在第二个子模块进程的详细信息模块，可以看到一些进程的信息(由于读取进程时是按照进程号从小到大读的，所以进程是按照进程号从小到大排序的)。

下面对进程按照不同关键词进行排序的测试：

图 4.8 排序测试(根据 User 名称排序左，根据内存占用排序右)

可以看到，以上测试结果无误，下面进行对 CPU 占用率，虚拟页占用和实际内存占用的排序：

Process information

PID	USER	%MEM	%CPU	VSZ	RSS	STATE	COMMAND
9500	vinstarry	12.02	4.87	5153443840981028		S	/home/vinstarry/.local/share/JetBrains/Toolbox/apps/CLion/ch-0/183.4
16962	vinstarry	0.36	2.81	515395584	29600	R	/home/vinstarry/projects/osdesign/proc_analysis/cmake-build-deb
7191	vinstarry	3.37	2.07	3496652800274784		S	/usr/bin/gnome-shell
7031	root	1.49	1.86	398344192	121332	S	/usr/lib/xorg/Xorg
7421	vinstarry	1.96	1.42	896028672	160280	S	albert
7499	vinstarry	2.04	0.90	3696029696166228		S	/home/vinstarry/vivado/xic/tps/lnx64/jre/bin/java
7423	vinstarry	0.10	0.68	521613312	8180	S	/usr/bin/prlcc
684	messagebus	0.08	0.40	52903936	6136	S	/usr/bin/dbus-daemon
7796	vinstarry	1.23	0.22	1286234112100608		S	usr/share/jetbrains-toolbox/jetbrains-toolbox
6185	root	0.12	0.18	112054272	10180	S	/usr/sbin/cupsd
17223	root	0.00	0.14	0	0	I	
753	avahi	0.05	0.12	48906240	4208	S	avahi-daemon: running [ubuntu.local]
1941	root	0.26	0.12	534953984	21488	S	/usr/lib/packagekit/packagekitd
7495	vinstarry	0.22	0.07	496484352	17876	S	/usr/bin/prlsga
1458	gdm	1.64	0.06	3187470336134164		S	/usr/bin/gnome-shell
9886	vinstarry	3.28	0.05	448700416	267632	S	/home/vinstarry/.local/share/JetBrains/Toolbox/apps/CLion/ch-0/183.4886.
774	root	0.12	0.05	310747136	9744	S	/usr/lib/policykit-1/polkitd

图 4.9 进程根据 CPU 占用率排序测试

Process information

PID	USER	%MEM	%CPU	VSZ	RSS	STATE	COMMAND
9500	vinstarry	12.02	4.88	5153443840980824		S	/home/vinstarry/.local/share/JetBrains/Toolbox/apps/CLion/ch-0/183.4
7499	vinstarry	2.04	0.90	3696029696166228		S	/home/vinstarry/vivado/xic/tps/lnx64/jre/bin/java
7191	vinstarry	3.37	2.06	3496652800274776		S	/usr/bin/gnome-shell
1458	gdm	1.64	0.06	3187470336134164		S	/usr/bin/gnome-shell
7842	vinstarry	0.93	0.01	1807429632	75556	S	/tmp/.mount_jetbraM4p2GJ/usr/share/jetbrains-toolbox/CEF/jetbr
7396	vinstarry	0.41	0.00	1517129728	33768	S	/usr/lib/tracker/tracker-extract
7203	vinstarry	0.15	0.01	1468751872	12004	S	/usr/bin/pulseaudio
8069	vinstarry	2.22	0.04	1437585408180944		S	/usr/bin/gnome-software
7242	vinstarry	0.32	0.00	1338175488	25784	S	/usr/lib/evolution/evolution-source-registry
7901	vinstarry	1.37	0.05	1305677824111540		S	/tmp/.mount_jetbraM4p2GJ/usr/share/jetbrains-toolbox/CEF/jetbrains-toolb
7796	vinstarry	1.23	0.22	1286234112100608		S	usr/share/jetbrains-toolbox/jetbrains-toolbox
1966	gdm	0.29	0.00	1240035328	23796	S	/usr/lib/gnome-settings-daemon/gsd-media-keys
7350	vinstarry	0.29	0.00	1239560192	23620	S	/usr/lib/gnome-settings-daemon/gsd-media-keys
7688	vinstarry	0.77	0.00	1238974464	62672	S	/usr/lib/evolution/evolution-calendar-factory-subproc
7406	vinstarry	0.73	0.00	1187397632	59948	S	/usr/lib/evolution/evolution-alarm-notify
1560	gdm	0.14	0.00	1181106176	11396	S	/usr/bin/pulseaudio
7721	vinstarry	0.33	0.00	1108910080	26600	S	/usr/lib/evolution/evolution-addressbook-factory-subpr

图 4.10 进程根据 VSZ 排序测试

由于根据其他列数据元素排序结果与上述截图相似，且本质原理一样，在这里不再对测试结果进行截图展示。

内存模块的动态显示功能也与 CPU 详细展示功能相似，此处不再赘述。

5 实验五 文件系统设计与实现

5.1 实验要求与内容

理解和掌握文件系统的设计方法。
设计、实现一个模拟的文件系统。
多用户、多级目录、用户登录。
文件/目录创建/删除，目录显示，文件打开/读/写/关闭等基本功能。
可自行扩充权限控制、读写保护等其他功能。

5.2 实验设计

5.2.1 开发环境

操作系统	MacOS Mojave 10.14.3
编译器	gcc 4.2.1
编译指令	gcc exp1.c -o a.out
集成开发环境	CLion 2018.3.2
CMake 版本	3.13.2(3.5 以上即可)

5.2.2 实验设计

整体设计：

首先，需要对整个文件系统进行整体设计，整体设计主要包括将如何实现文件系统，将如何管理块等方面。进行整体设计之后，进入文件系统的具体设计，包括文件系统各层的数据结构设计，各层之间的接口，以及最上层指令的映射设计。

对于文件系统的整体设计思路如下：

申请一个 100MB 的文件，模拟文件系统的存储空间就是该文件的存储空间，并将文件分块，用位示图的方法管理文件块。块的大小为 1KB，并且用 0 号块做管理块，前面 1~13 号块做位示图。

由于每一个二进制位管理了一个 1KB 的块是否被使用，总共有

$$100MB/1KB=1024000$$

个块，用 $1024000/8/1024 \approx 13$，得出需要的管理块的数量，也就是说，从第一块开始的前 13 个二进制位永远是 1(因为它们要一直被占用)。

使用 Unix 经典思想中的 inode 思想来简历块与目录之间的索引，对于文件中的数据块，和 inode 的存放，都是对 100MB 的文件进行操作的。本文建立了一个经济的策略：在申请 inode 存储空间时，从第 14 块开始遍历找到一个未被利用完的块(利用的空间不足 1KB)进行申请。当申请文件的数据块时，从第 1023999 块开始从后往前遍历，对于文件的数据块，每一次分配都分配一个整块，记录文件对此块占用的实际大小。示意图如下图 5.1：

SB0	BM0	BM1	BM2	BM3	BM4	BM5	BM6	BM7	BM8
BM9	BM10	BM11	BM12	IN0	IN1	IN2	…		
						…	DB2	DB1	DB0

图 5.1 块管理图

(其中，SB0 位 super block0，存储了超级块的信息，BM0~BM12 为 13 个位示图管理块，INx 存放的是 i 节点的信息，注意：一个块可以存放多个 i 节点，DB 为 Data block，一个块只存一个信息)

模拟文件系统分为四层：

1、块管理层：负责块的分配/回收、块的读/写等，由超级快(superblock)的数据结构实现。

2、文件控制块层：文件系统目录功能，目录文件由目录(dentry)配合 inode 的数据结构实现。

3、文件管理层：文件操作由 file 数据结构实现。

4、命令解释层，实现多种命令对上述底层文件与接口的映射。

管理数据结构设计

超级块数据结构设计

```
struct super_block {
   unsigned long s_blocknumbers;   // 块的总数量
   unsigned long s_blocksize;      // 块的大小
   void *s_bdev;       // 存储块
   int s_bitmap_blks;  // 位示图的块数量
};
```

超级块是主要进行块管理的数据，超级块的数据结构包含了块的总数量，在实例化过程中，块的总数量为 1024000 块，块的大小为 1024B，即 1024KB，存储块成员变量指向 100MB 的空间，用 void*指针的原因是方便进行类型转换。最后一个成员函数是位示图的块的数量，就像前文分析的那样，在这里这个值等于 13。

超级块操作设计

```
struct super_block *init_block(struct dentry *root);    //初始化超级块
struct super_block *free_block(struct super_block *sb); //释放超级块
bool save_block(struct super_block *sb);               //保存块到文件
struct super_block *load_block(struct dentry *root);    //从文件读取块
struct inode *alloc_inode(struct super_block *sb);      //分配 i 节点
bool occupy_block(blkcnt_t blk_no, struct super_block *sb); //位示图修改 占用
bool release_block(blkcnt_t blk_no, struct super_block *sb);//位示图修改 释放
void *free_block_for_inode(struct super_block *sb, struct inode *inode); //释放 i 节点
unsigned long alloc_data_block(struct super_block *sb); //为数据文件分配块
void free_data_block(unsigned long blk_no, struct super_block *sb); //释放为数据文件分配的块
size_t write_block(struct super_block *sb, unsigned long block_no, char *stream, size_t size);  // 对块进行写操作
size_t read_block(struct super_block *sb, unsigned long block_no, char *stream, size_t size);   // 对块进行读操作
```

对于超级块的操作，主要分以下几对，首先是超级块的初始化和超级块的释放，超级块的初始化表示了超级块的格式化过程，具体来说就是动态申请 100MB 的空间，并将位示图初始化，用第 0 块存放超级块数据结构的信息，1~13 块存放位示图，超级块的释放即将内存释放置空。

接着是超级块与文件的连接，由于块是用动态申请的 100MB 内存模拟的，所以将其信息存放在一个 100MB 大小的文件中。

接着是分配 i 节点与释放 i 节点,由于对 i 节点的分配与释放也是通过对内存进行修改完成的,所以 i 节点的释放与分配也是超级块应尽的责任。

配套 i 节点分配与释放的，需要对位示图进行修改和释放的函数，这个函数主要是修改位示图中管理某一块的位操作，本文另外在辅助函数中定义了判断当前块是否被占用，占用了多少空间等函数，详见附录。

接下来，还要考虑块对于文件数据块的支持，由于文件数据块与 i 节点的分配方式是不同的(前者是从后往前遍历一个空闲块，后者是从前往后遍历一个没有被用完的块)，所以对数据块的申请与释放单独设计了两个函数。

最后，为了支持文件对数据块进行读写，增加了 write_block 和 read_block 两个函数，参数是参照 Linux 系统的 write 和 read 进行设计的。

inode 数据结构设计

```
struct inode {
    struct super_block *i_sb;   // 节点对应的超级块
    struct blk_lists *i_list;   // i 节点所对应的块链表
    unsigned long i_nlink;      // i 节点被硬链接的数量
    unsigned long i_no;         // i 节点的编号
    unsigned long i_uid;        // i 节点所属的用户的 id
    blkcnt_t i_blocks;          // i 节点占用的块数量
    unsigned long i_btyes;      // i 节点的大小，以 Byte 为单位
    u_mode_t mode;              // 模式
};
```

i 节点的数据结构由以上部分组成。由于一个系统中可能有多个超级块，所以此处添加了 i 节点的超级块成员函数，实际使用时只用到了一个超级块。I 节点所对应的块链表表示 i 节点所占的块。后续的一些成员如注释所示。最后的成

员 mode 表现了 inode 所处的模式，用于权限控制，详情可见后文对用户和权限控制的说明。

inode 操作设计

```
void get_inode_memory_by_num(struct inode *inode, struct super_block *sb);  // 根据 i 节点的序号得到 i 节点
void put_inode_memory_by_num(struct inode *inode, struct super_block *sb);  // 根据 i 节点的序号写入 i 节点的信息
bool test_block_free_by_inode_num(unsigned long inode_num, struct super_block *sb); // 查询 i 节点号所对应的块是否空闲
void fill_block_by_inode_num(unsigned long inode_num, struct super_block *sb);  // 写位示图
```

由于 i 节点不能离开目录项和块操作而单独存在，所以专门对 i 节点的操作是不多的，此处对 i 节点的操作主要是从块中读出 i 节点的信息，以及将 i 节点的信息修改入块，后两个函数是用来检测是否存储出现了问题，通过测试 i 节点对应的块是否为空闲得到读取内容是否与之前存储的不一致，或者读错文件等问题。

目录项数据结构设计

```
struct dentry {
    char type;                // 目录项的类型
    struct inode *d_inode;     // 对应的 i 节点
    struct dir_hash_table *subdirs[HASH_TABLE_ROW]; // 子目录 hash 表
    struct tm *d_time;         // 创建的时间
    struct super_block *d_sb;   // 对应的超级块
    struct dentry *parent;     // 父目录
    char d_iname[DNAME_LEN];    // 目录项的名称
};
```

目录项包含的第一个成员 type 表示了目录项的类型，本文支持的类型有 l：硬链接，d：目录，f：普通文件。每个目录项都包含了自己的子目录，子目录是一个 hash 表，详情可见后文的存储数据结构设计。为了方便整个文件树的互相访问，设计了父目录成员函数。

目录项操作

```
bool save_entry(struct dentry *root, struct super_block *sb);   // 保存目录项到文件
bool load_entry(struct dentry *root, struct super_block *sb);   // 从文件读取目录项
struct dir_hash_table *d_hash(struct dentry *dentry);          // 计算目录项的 hash 值
void hash_table_insert(struct dentry *dentry, struct dentry *sub_dentry);   // 插入目录项到哈希表
void hash_table_delete(struct dentry *dentry, struct dentry *sub_dentry);   // 从哈希表删除目录项
struct dentry *search_by_str(struct dentry *current_dentry, const char *dname); // 查询 hash 表
void packet_user_dlist(dlist *dl, struct user_linked_list *head, struct dentry *root);  // 将目录项与用户名对应
struct dentry *change_home_dir(dlist *dl, struct usr_ptr *current_user);    // 改变当前路径
void get_current_path_str(struct dentry *dentry, char *str);    // 得到当前的路径
void travel_dentry(queue *q, slist *str_list, struct dentry *begin_dentry, size_t num); // 遍历目录项
```

主要需要说明的是，由于子目录是利用哈希表存储的，所以目录项操作中有许多与哈希表相关的，包括计算哈希值，插入，删除等，它们都是通过调用基本的 hash 函数来实现的。

文件操作设计

由于文件操作的难度过大，需要动态监测键盘上下左右、backspace 键等，所以本文的设计仅仅是一个非常原始的文件打开和读写操作。利用了目录项和 i 节点的索引操作和超级块中对块的读写操作对文件进行简单地读写。

辅助存储数据结构设计

本次实验用于辅助存储与操作的数据结构有基本链表、双向队列以及哈希链表等，由于基本链表很常见，下面只对双向队列与哈希链表进行说明。

双向队列数据结构与操作设计

```
typedef struct _queue_node {
    qtype val;                // value of the node
    int aux;                  // auxiliary info
    struct _queue_node *next;   // next the node
}qnode;
queue *init_queue(void); // 初始化队列，分配内存
queue *free_queue(queue **q); // 销毁队列，回收
qtype get_front_queue(const queue *q); // 得到队列的第一个元素
qtype get_rear_queue(const queue *q); // 得到队列的最后一个元素
bool queue_empty(const queue *q); // 判断队列是否为空
bool enqueue(qtype item, queue *q); // 入队列
qtype dequeue(queue *q); // 出队列
qtype dequeue_rear(queue *q); // 获得队列的尾部
size_t get_size_of_queue(const queue *q); // 获得队列的大小
void print_queue(const queue *q); // 输出队列元素，用于
```

双向队列主要来解决遍历哈希表，解析路径名等操作。

用链表解决冲突的哈希表设计

本文设计了一个哈希表的数据结构用于保存一个目录项的子目录，方便快速查找。对于一个目录项的文件名，首先将其 ASCII 码想加，得到一个哈希值，然后用它对 10 的余数作为哈希表的入口，解决冲突的方法是用链表将结果链起来，示意图如下：

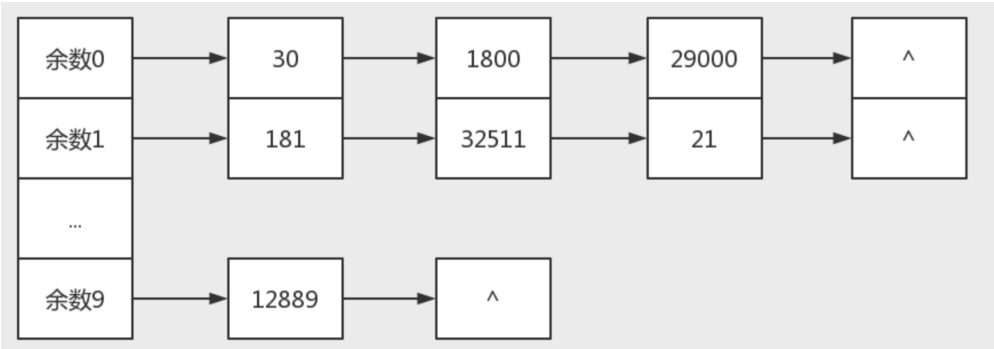

图 5.2 哈希表示意图

哈希表数据结构设计

```
struct dir_hash_table {
    int hash_key;       // 哈希值
    char *dname;        // 目录名
    struct dentry *corres_dentry;   // 相对应的目录
```

```
    struct dir_hash_table *next;    // 链表元素
};
```

哈希表的操作斗鱼目录项操作有关，详情请见上文。

多用户模式与权限控制设计

由于是简单的文件系统模拟，本次课程设计中没有实现用户组的概念，仅仅对用户的等级进行了区分。

对用户等级的按照优先级分成三类，第一类的用户优先级为 0，是根用户，优先级最高，一般一个系统只有一个根用户(root)，需要密码认证；第二类的用户优先级为 1，是管理员用户，需要密码认证；第三类用户是访客用户，不需要进行密码认证。下标展示了用户优先级说明。

表 5-1 用户优先级说明

优先级	用户类别	备注
0	根用户 root	每个系统只有一个
1	管理员用户 admin	可以有多个
2	访客用户 guest	需要验证密码

对于权限，设计了如下的格式的模式

rwx_rwx_rwx，其中前三位表示对于 0 级优先级的用户的读、写可执行权限；中间三位表示对于 1 级优先级的用户的读、写可执行权限；后三位表示对于 2 级优先级的用户的读、写可执行权限。

存储模式的时候，使用一个 unsigned long 类型，利用末尾的 9 位二进制数，分别表示 rwx_rwx_rwx 的 9 位。每三位为 1 组，4 代表 r，2 代表 w，1 代表 x，比如说 751 就表示：0 级用户可读可写可执行，1 级用户可读可执行，2 级用户只可以执行。权限控制需要用掩码的方式得到相应位数，以确定权限。

命令映射设计

为了模拟真实环境，本文设计了以下 11 个指令：

表 5-2 命令与功能设计

指令名称	功能	扩展
shutdown	保存并关闭文件系统	无
su	切换当前用户	无
ls	展示当前的目录	-R 递归 -l 具体
rename	修改文件名	无
rm	删除文件/目录	无
mkdir	创建目录	无
chmod	修改权限	无
cd	更改当前目录	无
edit	编辑文件	无
clear	清空显示屏	无
cat	输出当前文件内容	无

对于 shutdown 指令，是修改了整个主函数中的无限循环，让其退出循环，并进入到检查和保存环节。

对于 su 指令，让其进入检测模式，如果输入的字符串与某个已经家在进入系统中的用户名匹配，则进入验证密码模式，如果切换的用户等级为 2(guest)，则不需要输入密码，否则需要匹配密码。

ls 有三个选项，若不做任何扩展，即遍历当前目录下的子目录 hash 表。-R 利用双向队列进行递归，-l 表示需要输出更多的信息。

rename 是修改当前目录或文件名，需要进行权限检测。

rm 指令需要做一下几个操作：

1、 首先判断权限是否满足，不满足返回 permission denied。

2、 利用队列查找到当前需要删除的目录或数据文件，未查找到直接退出，返回名称不合法，否则对目录进行检查。

3、 如果目录下还有别的文件，则提示不删除(本文只支持单目录删除)，否则查看目录的种类，若是一个硬连接，则只需要在上层目录中的子目录哈希表中删除此目录。若是一个数据文件，则释放数据块、释放”.”和”..”子目录、释放 i 节点并且释放其父目录下的此目录指针。

mkdir 是实现中最核心，最需要注意的一个映射，但归根到底，它主要做了以下几个操作。

1、 首先找到 mkdir 指令后的文件目录，查找是否已经有该目录，若有，则返回创建失败，已经存在。

2、 若不存在，则在当前目录下新建一个 dentry，类型为 d，然后串联 i 节点，串联后将相应信息写入 i 节点。

3、 把子目录插入父目录的哈希表中，子目录的父指针指向父目录。

4、 在子目录下创建”.”和”..”的硬链接。

chmod 需要判断当前的用户，为了简化设计，只允许 root 级别的用户修改，然后根据前面设计的用户模式，与 Linux 一样，采用 chmod dir xxx 的格式改变目录的读写模式。

cd 首先需要检测输入是否合法，如果不合法，则退出，如果合法，则将它转换成从当前目录开始的多级目录队列，每次访问队列的头元素，如果最开始头元素不为”/”则在当前目录的表中查找，对于..和.依然做了处理(因为在 mkdir 时使用了硬链接.和..文件，所以同样可以在当前目录的子目录哈希表下查找”.”和”..”文件)。若队列最开始的头元素为”/”，则从根目录开始查找。每次查找完后用 temp 指针进入那个目录，并且头元素出队列。结束的情况为 1、无法找到目录(显示失败，temp 指针置空返回失败)，2、队列为空，查找成功，temp 指针赋值给 pwd 指针。

edit 指令首先需要判断是否已经存在当前需要操作的文件，如果不存在，根据 mkdir 函数的功能，创建一个数据文件目录，首先给 i 节点连接一个块，然后进入输入功能区，根据输入数量的大小再判断关闭时是否需要继续串联下面的块。若文件已经存在，首先按照其 i 节点串联的块，以及每块所占的大小，读入字符串数据，并显示到屏幕上，后续根据之前所述的功能继续添加即可。

clear 函数调用了 system(“clear”)作为一种清空界面的 UI 小功能。

cat 指令相当于是 edit 指令的操作子集，需要按照文件的内容找到数据块链表，按照每个块所占的大小读入字符串，并输出到控制台中即可。

5.3 实验调试

在本设计中，预先存放了以下的用户：

表 5-3 预先存储的用户

用户名	优先级	权限	密码
root	0	root	root123456
cat	1	admin	cat123456
dog	1	admin	dog123456
lion	1	admin	lion123456
guest	2	guest	guest123456

总共 1 个根用户，3 个 admin 和一个 guest。

预先写了一些目录与文件，组织形式如下：

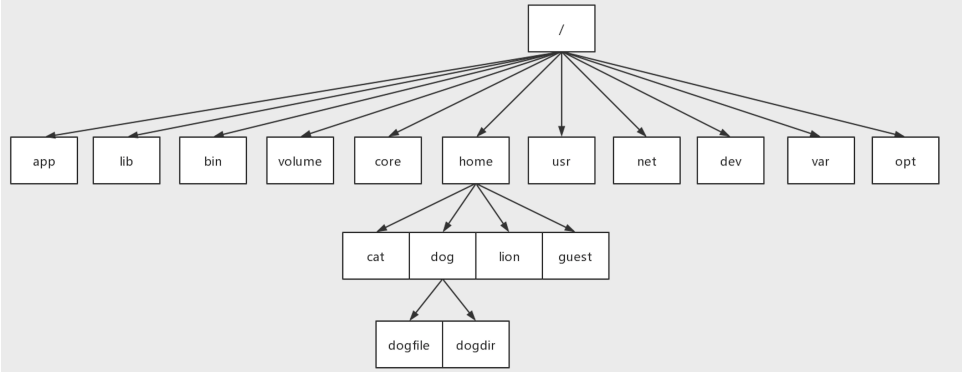

图 5.3 预留目录示意图

首先测试密码登录，以登录 dog 用户为例：

```
┌─yongxinxu@yongxindeMacBook-Pro ~/Desktop/Projects/vfs  <master*>
└─$ ./vfs
please input user name!
dog
input password of dog
dog12345
sorry, try again!
please input user name!
dog
input password of dog
dog123456
dog@vfs /home/dog $
```

图 5.4 用户登录测试

可以看到，必须匹配 dog123456 的密码就可以登录，登录后显示了 dog 的家目录/home/dog，（登录 cat 时会进入到/home/cat，登录 root 时会进入/）

然后测试一下 cd 功能：

```
dog@vfs /home/dog $cd ..
..
dog@vfs /home $cd /home/dog
/ --> home --> dog
dog@vfs /home/dog $cd ../cat
.. --> cat
dog@vfs /home/cat $cd .././
.. --> .
dog@vfs /home $cd ./dog
. --> dog
dog@vfs /home/dog $
```

图 5.5 切换目录测试

可以看到，分别以直接寻址和间接寻址测试了多种功能，结果都符合要求。

之后测试 ls 的功能，分别有 ls，ls -R ls -l：

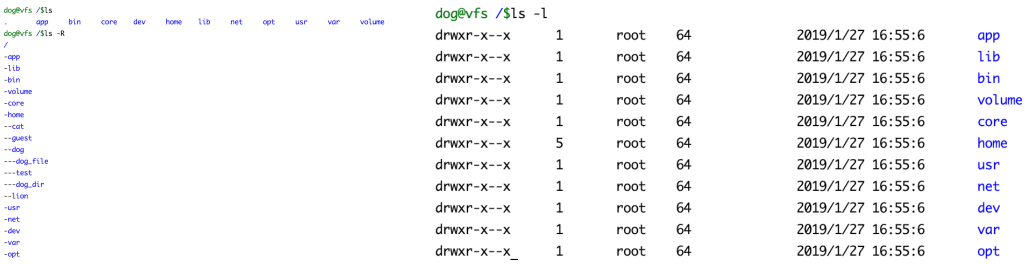

图 5.6 list file 测试(-R 左,-l 右)

可以看到，ls -R 的输出与之前的设计相符,-l 也输出了额外信息。

接下来测试 rm、su 和权限

```
dog@vfs /$rm lib
permission denied!
dog@vfs /$su root
input password of root
root123456
root@vfs        /$rm lib
root@vfs        /$ls
.       app     bin     core    dev     home    net     opt     usr     var     volume
root@vfs        /$
```

图 5.7 综合测试

可以看到，首先以 dog 的身份进入，删除 lib，但是由于 lib 是一个不具备 1 级用户读写的目录，所以不能删除，在切换成 root 之后删除成功。

```
root@vfs        /home/dog $mkdir a_dir
root@vfs        /home/dog $ls
.       ..      a_dir   dog_dir dog_file
root@vfs        /home/dog $cd a_dir
a_dir
root@vfs        /home/dog/a_dir $ls
.       ..
root@vfs        /home/dog/a_dir $cd ..
..
root@vfs        /home/dog $rename a_dir b_dir
root@vfs        /home/dog $ls
.       ..      b_dir   dog_dir dog_file
root@vfs        /home/dog $
```

图 5.8 mkdir 和 rename 综合测试

如图，创建了一个 a_dir 的目录，并改名成 b_dir，辅助 ls 和 cd，发现功能实现成功。

下面进行文件读写操作，首先是写文件

```
test:
hello
root@vfs          /home/dog $cat test
hello
```

图 5.9 写文件测试

再进行文件追加操作测试：

```
test:
hello
another hello
root@vfs        /home/dog $cat test
hello
another hello
```

图 5.10 读写文件追加测试

最后再对一些边界情况进行测试：

```
dog@vfs /home/dog $mkdir test
dog@vfs /home/dog $ls
.       ..      dog_dir dog_file        test
dog@vfs /home/dog $mkdir test
file exists!
dog@vfs /home/dog $cd .../../
... --> ..
No such file or directory: .../../
dog@vfs /home/dog $cd /fejw/wf
/ --> fejw --> wf
No such file or directory: /fejw/wf
```

图 5.11 边界测试

如上图，对已存在的 dir 继续 mkdir，报错，cd 进入不存在的目录，报错，设计成功！

6 附录 实验代码

文件拷贝源代码

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <string.h>
#include <pthread.h>
#include <stdbool.h>

#define BUFFER_SIZE 4096
#define BUFFER_NUM 10
#define READ_SEM 0
#define WRITE_SEM 1

int sem_id = 0;

typedef struct buffer_str {
    size_t length;
    char buffer[BUFFER_SIZE];
    struct buffer_str *next;
}shared_buffer;

union semun {
    int            val;    /* Value for SETVAL */
    struct semid_ds *buf;    /* Buffer for IPC_STAT, IPC_SET */
    unsigned short  *array;  /* Array for GETALL, SETALL */
};

shared_buffer *buf_ptr = NULL;

enum ERR_NUM {SUCCEED = 0,
        SRC_FILE_IS_DIR,
        DST_FILE_IS_NOT_DIR,
        SRC_OPEN_FAILED,
        DST_OPEN_FAILED,
        CREATE_SEM_ERROR,
        SET_SEM_VALUE_ERROR,
        };
```

```c
/* get the name of the file, stripe the path */
char *get_file_name(char *filepath);
/* copy file from src to dst */
int copy_file(char *src_file_name, char *dst_file_name);
/* return error number */
void error_catch(int rstate);
/* thread1: read from src */
void *reader_function(void *read_fp);
/* thread2: write to dst */
void *writer_function(void *writer_fp);
/* semaphore P operation */
bool  semaphore_P(unsigned short sum_num);
/* semaphore V operation */
bool  semaphore_V(unsigned short sum_num);
/* set initial value for semaphore */
bool  set_value_semaphore(void);
/* delete semaphore set */
void  delete_semaphore(void);
/* get shared buffer linked list */
shared_buffer *malloc_circular_list(int size);
/* remove shared buffer linked list */
shared_buffer *free_circular_list(shared_buffer *target, int size);

int main(int argc, char *argv[]) {
    int return_state = 0;   // return val, used to catch error
    char *src = NULL;      // src file name
    char *dst = NULL;      // dst file name(directory)
    char *filename = NULL;
    struct stat s_buf;
    if (argc != 3) {    // 3 argument is required
        perror("Invalid input!");
    }
    else {
        src = argv[1];
        dst = argv[2];
        stat(src, &s_buf);
        if (S_ISDIR(s_buf.st_mode)) {   // src cannot be a directory (do not support recursive copy)
            printf("%s is a directory\n", src);
            return_state = SRC_FILE_IS_DIR;
        }
        else {
            stat(dst, &s_buf);
            if (S_ISDIR(s_buf.st_mode)) {   // dst should be a directory
                filename = get_file_name(src);  // stripe the filename of the source file
```

```
//              printf("file name is %s\n", filename);
                if (dst[strlen(dst) - 1] != '/')    // add left dash if needed
                    dst = strcat(dst, "/");
                char *dest_name = strcat(dst,filename); // get destination file name
//              printf("destination file name is %s\n", dest_name);
                copy_file(src, dest_name);  // finally, copy file
            }
            else {  // dst is not a directory
                printf("%s is not a directory\n", dst);
                return_state = DST_FILE_IS_NOT_DIR;
            }
        }
    }
    /* error catch and display */
    error_catch(return_state);
    return return_state;
}

/*
 * stripe path to filename
 * e.g: filepath is /tmp/hello/abc
 * result: return "abc"
 */
char *get_file_name(char *filepath) {
    char *filename = NULL;
    size_t len = strlen(filepath);
    int left_slash_pos = -1;
    for (int i = (int)len - 1; i >= 0; i--) {
        if (filepath[i] == '/') {
            left_slash_pos = i;
            break;
        }
    }
    filename = (char *)malloc(sizeof(char) * (len - left_slash_pos));
    for (int i = 0; i < len - left_slash_pos; i++) {
        filename[i] = filepath[left_slash_pos + 1 + i];
    }
    filename[len-left_slash_pos-1] = '\0';
    return filename;
}

/*
 * copy source file to destination
 * use two threads and shared buffer
```

```
 * use semaphore to syn threads
 */
int copy_file(char *src_file_name, char *dst_file_name) {
    printf("Copy %s to %s.\n", src_file_name, dst_file_name);
    FILE *src = fopen(src_file_name, "rb");
    FILE *dst = fopen(dst_file_name, "wb");
    if (src == NULL)
        return SRC_OPEN_FAILED;
    if (dst == NULL)
        return DST_OPEN_FAILED;
    sem_id = semget(IPC_PRIVATE, 2, IPC_CREAT | 0666);

    if (sem_id == -1) {
        return CREATE_SEM_ERROR;
    }
    if (set_value_semaphore() == false) {
        delete_semaphore();
        return SET_SEM_VALUE_ERROR;
    }
    buf_ptr = malloc_circular_list(BUFFER_NUM);
    pthread_t reader, writer;
    int iret1 = 0, iret2 = 0;
    /* Create 2 independent threads each of which will execute function */
    if ((iret1 = pthread_create(&reader, NULL, reader_function, src)) != 0) {
        // error creating the first thread
        perror("Create thread for reader failed!\n");
        delete_semaphore();
        exit(1);
    }

    if ((iret2 = pthread_create(&writer, NULL, writer_function, dst)) != 0) {
        // error creating the second thread
        perror("Create thread for writer failed!\n");
        delete_semaphore();
        exit(1);
    }
    pthread_join(reader, NULL);
    pthread_join(writer, NULL);
    buf_ptr = free_circular_list(buf_ptr, BUFFER_NUM);
    fclose(src);
    fclose(dst);
    return SUCCEED;
}
```

```
void *reader_function(void *read_fp) {
    FILE *fp = (FILE *) read_fp;
    shared_buffer *ptr = buf_ptr;
    fseek(read_fp, 0, SEEK_SET);
    semaphore_P(READ_SEM);
    size_t len = fread(ptr->buffer, sizeof(char), BUFFER_SIZE * sizeof(char), fp);
    ptr->length = len;
//    printf("reader %lu\t", ptr->length);
    ptr = ptr->next;
    semaphore_V(WRITE_SEM);
    while (len != 0) {
        semaphore_P(READ_SEM);
        len = fread(ptr->buffer, sizeof(char), BUFFER_SIZE * sizeof(char), fp);
        ptr->length = len;
//        printf("reader %lu\t", ptr->length);
        ptr = ptr->next;
        semaphore_V(WRITE_SEM);
    }
    return NULL;
}

void *writer_function(void *writer_fp) {
    FILE *fp = (FILE *) writer_fp;
    semaphore_P(WRITE_SEM);
    shared_buffer *ptr = buf_ptr;
    size_t len = fwrite(ptr->buffer, sizeof(char), ptr->length * sizeof(char), fp);
//    printf("writer %lu\t", len);
    ptr = ptr->next;
    semaphore_V(READ_SEM);
    while(ptr->length > 0 && ptr->length <= BUFFER_SIZE * sizeof(char)) {
        semaphore_P(WRITE_SEM);
        len = fwrite(ptr->buffer, sizeof(char), ptr->length * sizeof(char), fp);
//        printf("writer %lu\t", len);
        ptr = ptr->next;
        semaphore_V(READ_SEM);
    }
    return NULL;
}

bool semaphore_P(unsigned short sum_num) {
    struct sembuf sem;
    sem.sem_num = sum_num;
    sem.sem_op = -1;
    sem.sem_flg = 0;
```

```
    if (semop(sem_id, &sem, 1) == -1) {
        perror("semaphore P failed!\n");
        return false;
    }
    return true;
}

bool semaphore_V(unsigned short sum_num) {
    struct sembuf sem;
    sem.sem_num = sum_num;
    sem.sem_op = 1;
    sem.sem_flg = 0;
    if (semop(sem_id, &sem, 1) == -1) {
        perror("semaphore V failed!\n");
        return false;
    }
    return true;
}

bool set_value_semaphore(void) {
    union semun sem_union;

    sem_union.val = BUFFER_NUM;
    if (semctl(sem_id, 0, SETVAL, sem_union) == -1) {
        perror("Fail to set value for semaphore!\n");
        return false;
    }
    sem_union.val = 0;
    if (semctl(sem_id, 1, SETVAL, sem_union) == -1) {
        perror("Fail to set value for semaphore!\n");
        return false;
    }
    return true;
}

void delete_semaphore(void) {
    union semun sem_union;

    if (semctl(sem_id, 0, IPC_RMID, sem_union) == -1) {
        perror("Failed to delete semaphore!\n");
    }
}

shared_buffer *malloc_circular_list(int size) {
```

```
    shared_buffer *head = NULL, *current = NULL;
    for (int i = 0; i < size; i++) {
        if (i == 0) {
            head = (shared_buffer *)malloc(sizeof(shared_buffer));
            head->next = NULL;
            head->length = 0;
            current = head;
        }
        else if (i != size - 1) {
            current->next = (shared_buffer *)malloc(sizeof(shared_buffer));
            current = current->next;
            current->next = NULL;
            current->length = 0;
        }
        else {
            current->next = (shared_buffer *)malloc(sizeof(shared_buffer));
            current = current->next;
            current->next = head;
            current->length = 0;
        }
    }
    return head;
}

shared_buffer *free_circular_list(shared_buffer *target, int size) {
    shared_buffer *current = target, *previous = target;
    for (int i = 0; i < size; i++) {
        current = ( i == size - 1 ) ? NULL : current->next;
        free(previous);
        previous = NULL;
    }
    return NULL;
}

void error_catch(int rstate) {
    switch (rstate) {
        case SRC_FILE_IS_DIR:
            printf("source file is a directory!\n");
            break;
        case DST_FILE_IS_NOT_DIR:
            printf("destination file is not a directory!\n");
        case SRC_OPEN_FAILED:
            printf("open source file failed!\n");
        case DST_OPEN_FAILED:
```

```
            printf("open destination file failed!\n");
        case CREATE_SEM_ERROR:
            printf("error creating semaphores!\n");
        case SET_SEM_VALUE_ERROR:
            printf("error setting value for semaphores!\n");
        default:
            break;
    }
}
```

多进程显示代码

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include "window_proc.h"

int pid1, pid2, pid3;

void killForks(int sig_no);
void childrenKilled(int sig_no);

int main(int argc, char *argv[]) {

    signal(SIGINT, killForks);

    pid1 = fork();
    if (pid1 < 0) {
        /* failed to create the first child process */
#ifdef _DEBUG
        perror("Fail to create the first child process!\n");
#endif
    }
    else if (pid1 == 0) {
        /* this is child process 1 executing */
#ifdef _DEBUG
        printf("Success to create the first child process, %d\n", getpid());
#endif
        signal(SIGUSR1, childrenKilled);
        wind_proc(argc, argv, 0);
    }
    else {
        pid2 = fork();
        if (pid2 < 0) {
```

```
            /* failed to create the second child process */
#ifdef _DEBUG
            perror("Fail to create the second child process!\n");
#endif
        }
        else if (pid2 == 0) {
            /* this is child process 2 executing */
#ifdef _DEBUG
            printf("Success to create the second child process, %d\n", getpid());
#endif
            signal(SIGUSR1, childrenKilled);
            wind_proc(argc, argv, 1);
        }
        else {
            pid3 = fork();
            if (pid3 < 0) {
                /* failed to create the third child process */
#ifdef _DEBUG
                perror("Fail to create the third child process!\n");
#endif
            }
            else if (pid3 == 0) {
                /* this is child process 3 executing */
#ifdef _DEBUG
                printf("Success to create the third child process, %d\n", getpid());
#endif
                signal(SIGUSR1, childrenKilled);
                wind_proc(argc, argv, 2);
            }
        }
        // this is main process
        // wait two child process to end
        waitpid(pid1, NULL, 0);
        waitpid(pid2, NULL, 0);
        waitpid(pid3, NULL, 0);
#ifdef _DEBUG
        printf("Parent Process is killed!\n");
#endif
    }
    return 0;
}



/**
```

```
    Send SIGUSR1 signal to two processes
    @param sig_no: specify which process to kill
*/
void killForks(int sig_no) {
    switch (sig_no) {
        case SIGINT:
#ifdef _DEBUG
            printf("Receive SIGINT signal, kill forks\n");
#endif
            kill(pid1, SIGUSR1);
            kill(pid2, SIGUSR1);
            kill(pid3, SIGUSR1);
            break;

        default:
#ifdef _DEBUG
            printf("Something wrong, kill forks!\n");
#endif

            kill(pid1, SIGUSR1);
            kill(pid2, SIGUSR1);
            kill(pid3, SIGUSR1);
            break;
    }
}

/**
    kill two child processes, use sig_no to determine which process to kill
    @param sig_no: specify which process to kill
*/
void childrenKilled(int sig_no) {
    if (pid1 == 0) {
#ifdef _DEBUG
        printf("Child Process 1 is Killed by Parent!\n");
#endif
        _exit(0);
    }
    if (pid2 == 0) {
#ifdef _DEBUG
        printf("Child Process 2 is Killed by Parent!\n");
#endif
        _exit(0);
    }
    if (pid3 == 0) {
```

```
#ifdef _DEBUG
       printf("Child Process 3 is Killed by Parent!\n");
#endif
       _exit(0);
   }
}
//
// Created by vinstarry on 2/8/19.
//

#ifndef MULPROC_DISPLAY_WINDOW_PROC_H
#define MULPROC_DISPLAY_WINDOW_PROC_H

#include <gtk/gtk.h>
#include <malloc.h>
#include <stdbool.h>
#include "table_op.h"

/* use argv and argc and rank (0,1,2) to create window proc */
void wind_proc(int argc, char *argv[], int rank);
/* destroy singal handle */
gboolean destroy_handle(GtkWidget *self, GdkEvent *event, gpointer data);
/* get title name from rank */
char *title_name(int rank);
/* start button pressed, call back start_display */
void start_display(GtkWidget *self, GdkEvent *event, gpointer data);
/* stop button pressed, call back stop_display */
void stop_display(GtkWidget *self, GdkEvent *event, gpointer data);
/* get the system time */
char *get_system_time(void);
/* get adder info */
char *get_adder_info(int *number, int *sum);
/* get cpu usage by all time and idle time */
char *get_cpu_usager(long all1, long all2, long idle1, long idle2);
/* refresh fstab content */
char *refresh_content(int line_no);
/* thread1 function */
void *thread_func1(void *arg);
/* thread2 function */
void *thread_func2(void *arg);
/* thread3 function */
void *thread_func3(void *arg);

#endif //MULPROC_DISPLAY_WINDOW_PROC_H
```

```
//
// Created by vinstarry on 2/9/19.
//

#include "window_proc.h"

#define BUF_LEN 50
#define WINDOW_WIDTH 400
#define WINDOW_HEIGHT 200
#define HUGE_WINDOW_WIDTH 1000
#define HUGE_WINDOW_HEIGHT 600
#define BUTTON_WIDTH 200
#define BUTTON_HEIGHT 50
#define FSTAB_INFO_COL 6
#define FSTAB_INFO_ROW 20
#define MAX_SPEC 2000
#define MAX_LINE_C 200

typedef struct _re_data {
    GtkWidget *label;
    int rank;
}g_data;

bool endSignal = false;
GtkWidget *content = NULL;
int proc_num = 0;
long all1 = 0, all2 = 0, idle1 = 0, idle2 = 0;

cross_linked_list *table_head = NULL;
GtkWidget *label_output = NULL;

void wind_proc(int argc, char *argv[], int rank) {
    gtk_init(&argc, &argv);
    // main window
    GtkWidget *window = gtk_window_new(GTK_WINDOW_TOPLEVEL);

    // window no.1 is a huge window
    if (rank != 1)
        gtk_widget_set_size_request(window, WINDOW_WIDTH, WINDOW_HEIGHT);
    else
        gtk_widget_set_size_request(window, HUGE_WINDOW_WIDTH, HUGE_WINDOW_HEIGHT);
    gtk_window_set_resizable(GTK_WINDOW(window), TRUE);
    gtk_window_set_position(GTK_WINDOW(window),GTK_WIN_POS_CENTER);
```

```
    g_signal_connect(window,"destroy",G_CALLBACK(destroy_handle),NULL);

    char *title = title_name(rank);
    proc_num = rank;
    // layouts
    GtkWidget *vbox = gtk_vbox_new(FALSE, 10);
    GtkWidget *button_box = gtk_hbox_new(FALSE, 0);

    GtkWidget *label = gtk_label_new(title);

    gtk_box_pack_start(GTK_BOX(vbox), GTK_WIDGET(button_box), FALSE, FALSE, 0);
    gtk_box_pack_start(GTK_BOX(vbox), GTK_WIDGET(label), FALSE, FALSE, 0);

    if (rank != 1) {
        content = gtk_label_new("display here");
        gtk_box_pack_start(GTK_BOX(vbox), GTK_WIDGET(content), FALSE, FALSE, 0);
    }
    else {
        GtkWidget *scrolled = gtk_scrolled_window_new(NULL,NULL);
        gtk_container_add(GTK_CONTAINER(vbox),scrolled);
        gtk_scrolled_window_set_policy(GTK_SCROLLED_WINDOW(scrolled), GTK_POLICY_AUTOMATIC,
GTK_POLICY_ALWAYS);

        table_head = (cross_linked_list *)malloc(sizeof(cross_linked_list));

        GtkWidget* pHtable = gtk_table_new(FSTAB_INFO_ROW, FSTAB_INFO_COL, false);
        gtk_table_set_col_spacings(GTK_TABLE(pHtable), 10);
        gtk_scrolled_window_add_with_viewport (GTK_SCROLLED_WINDOW (scrolled), pHtable);

        init_table(pHtable, table_head);

        GtkWidget *specific_content = gtk_label_new("specific content");
        gtk_box_pack_start(GTK_BOX(vbox), GTK_WIDGET(specific_content), FALSE, FALSE, 0);

        label_output = gtk_label_new("");
        GtkWidget *scrolled2 = gtk_scrolled_window_new(NULL,NULL);
        gtk_container_add(GTK_CONTAINER(vbox), scrolled2);
        gtk_scrolled_window_set_policy(GTK_SCROLLED_WINDOW(scrolled2), GTK_POLICY_AUTOMATIC,
GTK_POLICY_ALWAYS);

        gtk_scrolled_window_add_with_viewport (GTK_SCROLLED_WINDOW (scrolled2), label_output);
    }
```

```
    gtk_container_add(GTK_CONTAINER(window), vbox);

    // two butttons
    GtkWidget *start_button = gtk_button_new_with_label("Start");
    GtkWidget *stop_button = gtk_button_new_with_label("Stop");

    gtk_widget_set_size_request(start_button, BUTTON_WIDTH, BUTTON_HEIGHT);
    gtk_widget_set_size_request(stop_button, BUTTON_WIDTH, BUTTON_HEIGHT);
    gtk_box_pack_start(GTK_BOX(button_box), GTK_WIDGET(start_button), TRUE, TRUE, 0);
    gtk_box_pack_start(GTK_BOX(button_box), GTK_WIDGET(stop_button), TRUE, TRUE, 0);

    g_signal_connect(start_button,"clicked",G_CALLBACK(start_display), NULL);
    g_signal_connect(stop_button,"clicked",G_CALLBACK(stop_display), NULL);

    gtk_widget_show_all(window);

    gdk_threads_init();

    gdk_threads_enter();
    gtk_main();
    gdk_threads_leave();
}

gboolean destroy_handle(GtkWidget *self, GdkEvent *event, gpointer data) {
    endSignal = true;
    gtk_main_quit();
    return FALSE;
}

char *title_name(int rank) {
    char *title = NULL;
    switch (rank) {
        case 0:
            title = "System Time";
            break;
        case 1:
            title = "Fstab Info";
            break;
        case 2:
            title = "Adder";
            break;
        default:
            title = "ERROR";
            break;
```

```
    }
    return title;
}

char *get_system_time() {
    char *time_text = (char *)malloc(sizeof(char) * BUF_LEN);
    time_t local_time;
    struct tm *lt;
    time(&local_time);
    lt = localtime (&local_time);
    int len = sprintf(time_text, "%d/%d/%d %d:%d:%d\n", lt->tm_year+1900, lt->tm_mon, lt->tm_mday, lt->tm_hour,
lt->tm_min, lt->tm_sec);
    time_text[len] = '\0';
    return time_text;
}

char *get_adder_info(int *number, int *sum) {
    char *adder_text = (char *)malloc(sizeof(char) * BUF_LEN);
    sprintf(adder_text, "%d + %d = %d\n", *sum, *number, *sum + *number);
    *sum = *sum + *number;
    *number = *number + 1;
    return adder_text;
}

char *refresh_content(int line_no) {
    FILE *fp_fstab = NULL;
    char *message = (char *)malloc(sizeof(char) * MAX_LINE_C);
    fp_fstab = fopen("/etc/fstab", "r");
    if (fp_fstab == NULL)
        strcpy(message, "Cannot open /etc/fstab.\n");
    else {
        for (int i = 0; i <= line_no; i++) {
            if (fgets(message, MAX_LINE_C, fp_fstab) == NULL)
                strcpy(message, "");
        }
    }
    return message;
}

void start_display(GtkWidget *self, GdkEvent *event, gpointer data) {
    endSignal = false;
    GThread *th;
    switch (proc_num) {
        case 0:
```

```
            th = g_thread_new ("a", thread_func1, NULL);
            break;
        case 1:
            th = g_thread_new ("b", thread_func2, NULL);
            break;
        case 2:
            th = g_thread_new ("c", thread_func3, NULL);
            break;
        default:
            break;
    }
}

void stop_display(GtkWidget *self, GdkEvent *event, gpointer data) {
    endSignal = true;
}

void *thread_func1(void *arg) {
    while(!endSignal) {
        sleep(1);
        gdk_threads_enter();
        gtk_label_set_text(GTK_LABEL(content), get_system_time());
        gdk_threads_leave();
    }
    return NULL;
}

void *thread_func2(void *arg) {
    char specific[MAX_SPEC] = {'\0'};
    memset(specific, 0, MAX_SPEC);
    int line_number = 0;
    while(!endSignal) {
        sleep(1);
        gdk_threads_enter();
        get_fstab_info(table_head);
        char *temp = refresh_content(line_number);
        line_number++;
        strcat(specific, temp);
        gtk_label_set_text(GTK_LABEL(label_output), specific);
        gdk_threads_leave();
    }
    return NULL;
}
```

```
void *thread_func3(void *arg) {
    int num = 0, sum = 0;
    while(!endSignal) {
        sleep(1);
        if (num > 1000)
            continue;
        gdk_threads_enter();
        gtk_label_set_text(GTK_LABEL(content), get_adder_info(&num, &sum));
        gdk_threads_leave();
    }
    return NULL;
}
//
// Created by vinstarry on 2/18/19.
//

#ifndef MULPROC_DISPLAY_TABLE_OP_H
#define MULPROC_DISPLAY_TABLE_OP_H

#include <gtk/gtk.h>

/* linked list of text */
typedef struct __label_list {
    GtkWidget *lb;
    struct __label_list *next;
}lb_list;

/* cross linked list, row and column respectively */
typedef struct __cross_linked_list {
    lb_list *content_head;
    struct __cross_linked_list *next;
}cross_linked_list;

/* initialize table with title */
void init_table(GtkWidget *table, cross_linked_list *tb_head);
/* add fstab's info table */
void add_fstab_title(lb_list *title_head);
/* fill table according to the label_list */
void fill_table(GtkWidget *table, cross_linked_list *tb_head);
/* get fstab's info, and add it into label list */
void get_fstab_info(cross_linked_list *tb_head);

#endif //MULPROC_DISPLAY_TABLE_OP_H
```

```
//
// Created by vinstarry on 2/18/19.
//

#include "table_op.h"

#define ROW_MAX 19
#define COL_MAX 6
#define LINE_MAX_CHARA 1024

void init_table(GtkWidget *table, cross_linked_list *tb_head) {
    if (tb_head) {
        cross_linked_list *cur = tb_head;
        cur->next = (cross_linked_list *)malloc(sizeof(lb_list));
        cur = cur->next;
        cur->content_head = (lb_list *)malloc(sizeof(lb_list));
        add_fstab_title(cur->content_head);
        for (int i = 0; i < ROW_MAX; i++) {
            cur->next = (cross_linked_list *)malloc(sizeof(lb_list));
            cur = cur->next;
            cur->content_head = (lb_list *)malloc(sizeof(lb_list));
            lb_list *ptr = cur->content_head;
            for (int j = 0; j < COL_MAX; j++) {
                ptr->next = (lb_list *)malloc(sizeof(lb_list));
                ptr = ptr->next;
                ptr->lb = gtk_label_new("");
            }
        }
        fill_table(table, tb_head);
    }
}

void add_fstab_title(lb_list *title_head) {
    if (title_head) {
        lb_list *temp = title_head;
        title_head->next = (lb_list *)malloc(sizeof(lb_list));
        temp = title_head->next;
        temp->lb = gtk_label_new("deivce");
        temp->next = (lb_list *)malloc(sizeof(lb_list));
        temp = temp->next;
        temp->lb = gtk_label_new("dir");
        temp->next = (lb_list *)malloc(sizeof(lb_list));
        temp = temp->next;
        temp->lb = gtk_label_new("type");
```

```
        temp->next = (lb_list *)malloc(sizeof(lb_list));
        temp = temp->next;
        temp->lb = gtk_label_new("options");
        temp->next = (lb_list *)malloc(sizeof(lb_list));
        temp = temp->next;
        temp->lb = gtk_label_new("dump");
        temp->next = (lb_list *)malloc(sizeof(lb_list));
        temp = temp->next;
        temp->lb = gtk_label_new("pass");
        temp->next = (lb_list *)malloc(sizeof(lb_list));
        temp->next = NULL;
    }
}

void fill_table(GtkWidget *table, cross_linked_list *tb_head) {
    cross_linked_list *cur = tb_head;
    for (int i = 0; i < ROW_MAX; i++) {
        cur = cur->next;
        lb_list *temp = cur->content_head;
        for (int j = 0; j < COL_MAX; j++) {
            temp = temp->next;
            gtk_table_attach(GTK_TABLE(table), temp->lb, (guint)j, (guint)j+1, (guint)i, (guint)i+1, GTK_EXPAND,
GTK_EXPAND, 0, 0);
        }
    }
}

void get_fstab_info(cross_linked_list *tb_head) {
    FILE *fp = fopen("/etc/fstab", "r");
    char temp[LINE_MAX_CHARA];
    if (fp) {
        cross_linked_list *cur = tb_head->next;
        cur = cur->next;
        while ((fgets(temp, LINE_MAX_CHARA, fp)) != NULL) {
            if (temp[0] == '#')
                continue;
            else {
                cur = cur->next;
                lb_list *temp_ptr = cur->content_head;
                char dev[200], dir[200], type[20], options[50];
                char dump[2], pass[2];
                if (6 != (sscanf(temp, "%s %s %s %s %s %s", dev, dir, type, options, dump, pass)))
                    break;
                temp_ptr = temp_ptr->next;
```

```
            gtk_label_set_text(GTK_LABEL(temp_ptr->lb), dev);
            temp_ptr = temp_ptr->next;
            gtk_label_set_text(GTK_LABEL(temp_ptr->lb), dir);
            temp_ptr = temp_ptr->next;
            gtk_label_set_text(GTK_LABEL(temp_ptr->lb), type);
            temp_ptr = temp_ptr->next;
            gtk_label_set_text(GTK_LABEL(temp_ptr->lb), options);
            temp_ptr = temp_ptr->next;
            gtk_label_set_text(GTK_LABEL(temp_ptr->lb), dump);
            temp_ptr = temp_ptr->next;
            gtk_label_set_text(GTK_LABEL(temp_ptr->lb), pass);
          }
       }
    }
    fclose(fp);
}
```

系统调用测试代码

```
#include <stdio.h>
#include <linux/kernel.h>
#include <sys/syscall.h>
#include <unistd.h>

int main()
{
      long int copy_file = syscall(326, "paper.pdf", "temp/paper.pdf");
      printf("the syscall function returned %ld\n", copy_file);

      FILE *fp_src, *fp_dst;
      fp_src = fopen("paper.pdf", "rb");
      fp_dst = fopen("temp/paper.pdf", "rb");

      int result = 1;
      char ch1, ch2;

      while ( (ch1 = (char)fgetc(fp_src) != EOF) && (ch2 = (char)fgetc(fp_dst)!=EOF )) {
            if (ch1 != ch2) {
                  result = 0;
                  break;
            }
      }
      if (result == 1)
            printf("src file and dst file are the same.\n");
```

```
	else
		printf("not same!\n");
	fclose(fp_src);
	fclose(fp_dst);
	return 0;
}
```

设备驱动测试代码

```
#include <sys/types.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#define MAX_SIZE 1024
#define FNAME_SIZE 256

int main(void) {
	int fd;
	char buf[MAX_SIZE];
	char get[MAX_SIZE];
	char devName[FNAME_SIZE];
	char filename[FNAME_SIZE] = "/dev/";
	system("ls /dev/");
	printf("Enter the device's name you wan to use :");
	fgets(devName, FNAME_SIZE, stdin);
	devName[strlen(devName) - 1] = '\0';
	strcat(filename, devName);
	printf("%s\n", filename);
	fd = open(filename, O_RDWR | O_NONBLOCK);
	if (fd != -1) {
		read(fd, buf, sizeof(buf));
		printf("The device was inited with a string: %s\n", buf);

		printf("Enter the string you want to write:\n");
		fgets(get, MAX_SIZE, stdin);
		write(fd, get, sizeof(get));

		read(fd, buf, sizeof(buf));
		printf("The string in the device now is: %s\n", buf);
		close(fd);
		return 0;
	}
```

```
    else {
        printf("Device open failed\n");
        return -1;
    }
    return 0;
}
```

Proc 文件分析代码

```
#include <stdio.h>
#include <gtk/gtk.h>
#include "sys_page.h"
#include "mem_page.h"
#include "ps_page.h"

#define WINDOW_WIDTH 1500
#define WINDOW_HIGHT 1200
#define LINE_CHART_WIDTH 1500
#define LINE_CHART_HEIGHT 160
#define ROW_NUMBER_MAX 400

gboolean destroy_handle(GtkWidget *self, GdkEvent *event, gpointer data);
void switch_page_handle(GtkNotebook *notebook, gpointer page, guint page_num, gpointer data);
void *thread_func(void *arg);

bool endSignal = true;
GThread *th;

GtkWidget *vbox, *second_vbox, *third_vbox;
lb_list *lb_head;

int main(int argc, char *argv[]) {
    gtk_init(&argc, &argv);

    // main window
    GtkWidget *window = gtk_window_new(GTK_WINDOW_TOPLEVEL);

    gtk_widget_set_size_request(window, WINDOW_WIDTH, WINDOW_HIGHT);
    gtk_window_set_resizable(GTK_WINDOW(window), TRUE);
    gtk_window_set_position(GTK_WINDOW(window),GTK_WIN_POS_CENTER);

    g_signal_connect(window,"destroy",G_CALLBACK(destroy_handle),NULL);

    // notebook
    GtkWidget *notebook = gtk_notebook_new();
    gtk_container_add(GTK_CONTAINER(window), notebook);
```

```
gtk_notebook_set_tab_pos(GTK_NOTEBOOK(notebook), GTK_POS_TOP);

// layouts
vbox = system_general_info();

// page one
GtkWidget *label = gtk_label_new("System");

gtk_notebook_append_page(GTK_NOTEBOOK(notebook), vbox, label);

// page two
label = gtk_label_new("Process");
second_vbox = gtk_vbox_new(FALSE, 5);
GtkWidget *overall_frame_title = gtk_label_new("");
gtk_label_set_markup(
       GTK_LABEL(overall_frame_title),
       "<span foreground='brown' font_desc='24'>Overall Information</span>");

GtkWidget *overall_frame = gtk_frame_new("");
gtk_frame_set_label_widget(GTK_FRAME(overall_frame), overall_frame_title);
gtk_container_set_border_width(GTK_CONTAINER(overall_frame), 20);
gtk_box_pack_start(GTK_BOX(second_vbox), overall_frame, false, false, 5);

GtkWidget *cpu_usage = gtk_label_new("CPU usage: 0.0%");
GtkWidget *usage_info_vbox = gtk_vbox_new(false, 0);

init_cpu_array();
GtkWidget *cpu_usage_drawing_area = gtk_drawing_area_new();
gtk_widget_set_size_request(cpu_usage_drawing_area, LINE_CHART_WIDTH, LINE_CHART_HEIGHT);

gtk_box_pack_start(GTK_BOX(usage_info_vbox), cpu_usage_drawing_area, false, false, 5);
gtk_box_pack_start(GTK_BOX(usage_info_vbox), cpu_usage, false, false, 5);

g_timeout_add(1000, update_cpu_usage, cpu_usage);
g_signal_connect(G_OBJECT(cpu_usage_drawing_area), "expose_event", G_CALLBACK(usage_cpu_draw), NULL);

GtkWidget *mem_usage = gtk_label_new("Memory usage: 0.0%");

GtkWidget *mem_usage_drawing_area = gtk_drawing_area_new();
gtk_widget_set_size_request(mem_usage_drawing_area, LINE_CHART_WIDTH, LINE_CHART_HEIGHT);

gtk_box_pack_start(GTK_BOX(usage_info_vbox), mem_usage_drawing_area, false, false, 5);
gtk_box_pack_start(GTK_BOX(usage_info_vbox), mem_usage, false, false, 5);
```

```
    gtk_container_add(GTK_CONTAINER(overall_frame), usage_info_vbox);

    g_timeout_add(1000, update_mem_usage, mem_usage);
    g_signal_connect(G_OBJECT(mem_usage_drawing_area), "expose_event", G_CALLBACK(usage_mem_draw), NULL);

    GtkWidget *ps_frame_title = gtk_label_new("");
//    GtkWidget *proc_frame =  get_proc_info();
    GtkWidget *proc_frame = gtk_frame_new("");
    gtk_label_set_markup(
           GTK_LABEL(ps_frame_title),
           "<span foreground='brown' font_desc='24'>Process information</span>");
    gtk_frame_set_label_widget(GTK_FRAME(proc_frame), ps_frame_title);
    gtk_container_set_border_width(GTK_CONTAINER(proc_frame), 20);

    GtkWidget *table = gtk_table_new(ROW_NUMBER_MAX, 7, false);
    GtkWidget *tbscrolled = gtk_scrolled_window_new(NULL,NULL);
    gtk_container_add(GTK_CONTAINER(proc_frame),tbscrolled);
    gtk_scrolled_window_set_policy(GTK_SCROLLED_WINDOW(tbscrolled), GTK_POLICY_AUTOMATIC, GTK_POLICY_ALWAYS);

    gtk_scrolled_window_add_with_viewport (
           GTK_SCROLLED_WINDOW (tbscrolled), table);

    GtkWidget *pid_button = gtk_button_new_with_label("PID"),
             *user_button = gtk_button_new_with_label("USER"),
             *mem_per_button = gtk_button_new_with_label("%MEM"),
             *cpu_per_button = gtk_button_new_with_label("%CPU"),
             *vsz_button = gtk_button_new_with_label("VSZ"),
             *rss_button = gtk_button_new_with_label("RSS"),
             *state_button = gtk_button_new_with_label("STATE"),
             *cmdline_button = gtk_button_new_with_label("COMMAND");

    gtk_table_attach(GTK_TABLE(table), pid_button, 0, 1, (guint)0, (guint)1, GTK_EXPAND, GTK_EXPAND, 0, 0);
    gtk_table_attach(GTK_TABLE(table), user_button, 1, 2, (guint)0, (guint)1, GTK_EXPAND, GTK_EXPAND, 0, 0);
    gtk_table_attach(GTK_TABLE(table), mem_per_button, 2, 3, (guint)0, (guint)1, GTK_EXPAND, GTK_EXPAND, 0, 0);
    gtk_table_attach(GTK_TABLE(table), cpu_per_button, 3, 4, (guint)0, (guint)1, GTK_EXPAND, GTK_EXPAND, 0, 0);
    gtk_table_attach(GTK_TABLE(table), vsz_button, 4, 5, (guint)0, (guint)1, GTK_EXPAND, GTK_EXPAND, 0, 0);
    gtk_table_attach(GTK_TABLE(table), rss_button, 5, 6, (guint)0, (guint)1, GTK_EXPAND, GTK_EXPAND, 0, 0);
    gtk_table_attach(GTK_TABLE(table), state_button, 6, 7, (guint)0, (guint)1, GTK_EXPAND, GTK_EXPAND, 0, 0);
    gtk_table_attach(GTK_TABLE(table), cmdline_button, 7, 8, (guint)0, (guint)1, GTK_EXPAND, GTK_EXPAND, 0, 0);

    init_proc_table(table);

    gtk_container_add(GTK_CONTAINER(second_vbox), proc_frame);
```

```
g_signal_connect(G_OBJECT(pid_button), "pressed", G_CALLBACK(sort_by_pid), NULL);
g_signal_connect(G_OBJECT(user_button), "pressed", G_CALLBACK(sort_by_user), NULL);
g_signal_connect(G_OBJECT(mem_per_button), "pressed", G_CALLBACK(sort_by_memper), NULL);
g_signal_connect(G_OBJECT(cpu_per_button), "pressed", G_CALLBACK(sort_by_cpuper), NULL);
g_signal_connect(G_OBJECT(vsz_button), "pressed", G_CALLBACK(sort_by_vsz), NULL);
g_signal_connect(G_OBJECT(rss_button), "pressed", G_CALLBACK(sort_by_rss), NULL);

gtk_notebook_append_page(GTK_NOTEBOOK(notebook), second_vbox, label);

g_timeout_add(5000, update_proc_table, table);

// page three
label = gtk_label_new("Memory");
third_vbox = gtk_vbox_new(TRUE, 5);

GtkWidget *mem_frame_title = gtk_label_new("");
gtk_label_set_markup(
       GTK_LABEL(mem_frame_title),
       "<span foreground='brown' font_desc='24'>Memory specific</span>");

GtkWidget *mem_frame = gtk_frame_new("");
gtk_frame_set_label_widget(GTK_FRAME(mem_frame), mem_frame_title);
gtk_container_set_border_width(GTK_CONTAINER(mem_frame), 20);
gtk_container_add(GTK_CONTAINER(third_vbox), mem_frame);

GtkWidget *scrolled = gtk_scrolled_window_new(NULL,NULL);
gtk_container_add(GTK_CONTAINER(mem_frame),scrolled);
gtk_scrolled_window_set_policy(GTK_SCROLLED_WINDOW(scrolled), GTK_POLICY_AUTOMATIC, GTK_POLICY_ALWAYS);
lb_head = (lb_list *)malloc(sizeof(lb_head));
lb_head->next = NULL;
unsigned int lines = memory_info_lable_attach(lb_head);
GtkWidget* pHtable = gtk_table_new(lines + 1, 2, false);
gtk_scrolled_window_add_with_viewport (
       GTK_SCROLLED_WINDOW (scrolled), pHtable);
attach_memory_info(pHtable, lb_head);

gtk_notebook_append_page(GTK_NOTEBOOK(notebook), third_vbox, label);

g_signal_connect(notebook, "switch-page", G_CALLBACK(switch_page_handle), NULL);

gtk_notebook_set_current_page(GTK_NOTEBOOK(notebook),1);

gtk_widget_show_all(window);
```

```
    gdk_threads_init();

    gdk_threads_enter();
    gtk_main();
    gdk_threads_leave();

    return 0;
}

gboolean destroy_handle(GtkWidget *self, GdkEvent *event, gpointer data) {
    free_mps();
    endSignal = true;
    gtk_main_quit();
    return FALSE;
}

void switch_page_handle(GtkNotebook *notebook, gpointer page, guint page_num, gpointer data)
{
    printf("this is page no. %d\n", page_num + 1);
    switch (page_num) {
        case 0:
            endSignal = true;
            break;
        case 1:
            endSignal = true;
            break;
        case 2:
            endSignal = false;
            th = g_thread_new("a", thread_func, NULL);
            break;
        default:
            break;
    }
}

void *thread_func(void *arg) {
    while (!endSignal) {
        sleep(2);
        gdk_threads_enter();
        refresh_memory_info(lb_head);
        gdk_threads_leave();
    }
    return NULL;
}
```

```c
//
// Created by vinstarry on 2/20/19.
//

#ifndef PROC_ANALYSIS_PS_PAGE_H
#define PROC_ANALYSIS_PS_PAGE_H

#include <gtk/gtk.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/stat.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
#include <sys/types.h>
#include <dirent.h>
#include <pwd.h>

typedef struct ps_info {
    double cpu_usage;
    double mem_usage;
    char pname[20];
    char user[20];
    int  pid;
    int  ppid;
    int pgid;
    int sid;
    int tty_nr;
    int tty_pgrp;
    int min_flt;
    int cmin_flt;
    int maj_flt;
    int cmaj_flt;
    int utime;
    int stime;
    int cutime;
    int cstime;
    int priority;
    int nice;
    int num_threads;
    int it_real_value;
    int start_time;
    unsigned long vsize;
```

```
    unsigned long rss;
    unsigned long task_flags;
    char state;
    char cmdline[100];
    struct ps_info *next;
}mps;

GtkWidget *get_proc_info(void);
mps *trav_dir(char *dir);
int read_info(char *d_name, struct ps_info *p1);
GtkWidget* print_ps(struct ps_info *head);
bool str_pure_num(char *str);
void uid_to_name(uid_t uid, struct ps_info *p1);        //由进程uid得到进程的所有者user
gboolean update_cpu_usage (gpointer user_data);
gboolean update_mem_usage (gpointer user_data);
gboolean usage_cpu_draw(GtkWidget *widget,GdkEventExpose *event,gpointer data);
gboolean usage_mem_draw(GtkWidget *widget,GdkEventExpose *event,gpointer data);
gboolean fresh_cpu_record(GtkWidget *widget);
gboolean fresh_mem_record(GtkWidget *widget);
gboolean update_proc_table(gpointer table);
void add_cpu_usage_into_array(double usage);
void add_mem_usage_into_array(double usage);
void debug_print_cpu_array(void);
void init_cpu_array(void);
void init_proc_table(GtkWidget *table);
void free_mps(void);
long get_up_time(void);
void get_system_config(void);
gboolean sort_by_pid(GtkWidget *widget,GdkEventExpose *event,gpointer data);
gboolean sort_by_user(GtkWidget *widget,GdkEventExpose *event,gpointer data);
gboolean sort_by_memper(GtkWidget *widget,GdkEventExpose *event,gpointer data);
gboolean sort_by_cpuper(GtkWidget *widget,GdkEventExpose *event,gpointer data);
gboolean sort_by_rss(GtkWidget *widget,GdkEventExpose *event,gpointer data);
gboolean sort_by_vsz(GtkWidget *widget,GdkEventExpose *event,gpointer data);
void struct_sort(int col_num, int rows);

#endif //PROC_ANALYSIS_PS_PAGE_H

//
// Created by vinstarry on 2/20/19.
//

#ifndef PROC_ANALYSIS_MEM_PAGE_H
#define PROC_ANALYSIS_MEM_PAGE_H
```

```c
#include <gtk/gtk.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/stat.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
#include <sys/types.h>
#include <dirent.h>
#include <pwd.h>

typedef struct _label_list{
    GtkWidget *lb1;
    GtkWidget *lb2;
    struct _label_list *next;
}lb_list;

void refresh_memory_info(lb_list *head);
unsigned int memory_info_lable_attach(lb_list *head);
void attach_memory_info(GtkWidget *table, lb_list *head);

#endif //PROC_ANALYSIS_MEM_PAGE_H

//
// Created by vinstarry on 2/20/19.
//

#include "mem_page.h"

#define VALUE_MAX_LEN 100

void attach_memory_info(GtkWidget *table, lb_list *head) {
    int i = 0;
    lb_list *ptr = head->next;
    GtkWidget *title_1 = gtk_label_new("Name");
    GtkWidget *title_2 = gtk_label_new("Value");
    gtk_table_attach(GTK_TABLE(table), title_1, 0, 1, (guint)i, (guint)i+1, GTK_EXPAND, GTK_EXPAND, 0, 0);
    gtk_table_attach(GTK_TABLE(table), title_2, 1, 2, (guint)i, (guint)i+1, GTK_EXPAND, GTK_EXPAND, 0, 0);
    i++;
    while (ptr) {
        gtk_table_attach(GTK_TABLE(table), ptr->lb1, 0, 1, (guint)i, (guint)i+1, GTK_EXPAND, GTK_EXPAND, 0, 0);
        gtk_table_attach(GTK_TABLE(table), ptr->lb2, 1, 2, (guint)i, (guint)i+1, GTK_EXPAND, GTK_EXPAND, 0, 0);
        i++;
```

```
        ptr = ptr->next;
    }
}

void refresh_memory_info(lb_list *head) {
    lb_list *ptr = head->next;
    char name[VALUE_MAX_LEN] = {0};
    char value[VALUE_MAX_LEN] = {0};
    char ch;
    FILE *fp = fopen("/proc/meminfo", "r");
    while ((ch = (char)fgetc(fp)) != EOF) {
        int i = 0, sp = 0, j = 0;
        while (ch != ':') {
            name[i++] = ch;
            ch = (char)fgetc(fp);
        }
        for (sp = i - 1; sp >= 0; sp--) {
            if(name[sp] != ' ' && name[sp] != '\t')
                break;
        }
        name[sp + 1] = '\0';
        fgetc(fp);
        ch = (char)fgetc(fp);
        while (ch != '\n') {
            value[j++] = ch;
            ch = (char)fgetc(fp);
        }
        value[j] = '\0';
        gtk_label_set_text(GTK_LABEL(ptr->lb2), value);
        ptr = ptr->next;
    }
    fclose(fp);
}

unsigned int memory_info_lable_attach(lb_list *head) {
    lb_list *ptr = head;
    char name[VALUE_MAX_LEN] = {0};
    char value[VALUE_MAX_LEN] = {0};
    char ch;
    unsigned int lines = 0;
    FILE *fp = fopen("/proc/meminfo", "r");
    while ((ch = (char)fgetc(fp)) != EOF) {
        ptr->next = (lb_list *)malloc(sizeof(lb_list));
        ptr = ptr->next;
```

```
        ptr->next = NULL;
        int i = 0, sp = 0, j = 0;
        while (ch != ':') {
            name[i++] = ch;
            ch = (char)fgetc(fp);
        }
        for (sp = i - 1; sp >= 0; sp--) {
            if(name[sp] != ' ' && name[sp] != '\t')
                break;
        }
        name[sp + 1] = '\0';
        fgetc(fp);
        ch = (char)fgetc(fp);
        while (ch != '\n') {
            value[j++] = ch;
            ch = (char)fgetc(fp);
        }
        value[j] = '\0';
        ptr->lb1 = gtk_label_new(name);
        ptr->lb2 = gtk_label_new(value);
        lines++;
    }
    fclose(fp);
    return lines;
}

//
// Created by vinstarry on 2/11/19.
//

#ifndef PROC_ANALYSIS_SYS_PAGE_H
#define PROC_ANALYSIS_SYS_PAGE_H

#include <gtk/gtk.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/stat.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
#include <sys/types.h>
#include <dirent.h>
#include <pwd.h>
```

```
GtkWidget *system_general_info(void);
char *get_cpu_info_by_key(char *key);
char *get_kernel_version(int rank);
char *get_specific_info(void);
char *get_uptime(int rank);
char *get_total_memory(void);
char *get_hostname(void);

#endif //PROC_ANALYSIS_SYS_PAGE_H

//
// Created by vinstarry on 2/11/19.
//

#include "sys_page.h"

#define WINDOW_WIDTH 1200
#define WINDOW_HIGHT 800
#define VALUE_MAX_LEN 100
#define SPECIFIC_LENGTH 1000

GtkWidget *system_general_info(void) {
    GtkWidget *vbox = gtk_vbox_new(false, 0);
    char sys_specific[SPECIFIC_LENGTH];
    char cm_brief[SPECIFIC_LENGTH];
    memset(sys_specific, 0, sizeof(sys_specific)/ sizeof(char));
    memset(cm_brief, 0, sizeof(cm_brief)/ sizeof(char));

    GtkWidget *sys_frame_title = gtk_label_new("");
    gtk_label_set_markup(
           GTK_LABEL(sys_frame_title),
           "<span foreground='brown' font_desc='24'>System information</span>");

    GtkWidget *cm_frame_title = gtk_label_new("");
    gtk_label_set_markup(
           GTK_LABEL(cm_frame_title),
           "<span foreground='brown' font_desc='24'>Brief information</span>");

    GtkWidget *cpu_frame_title = gtk_label_new("");
    gtk_label_set_markup(
           GTK_LABEL(cpu_frame_title),
           "<span foreground='brown' font_desc='24'>CPU specific</span>");

    GtkWidget *system_frame = gtk_frame_new("");
```

```
gtk_frame_set_label_widget(GTK_FRAME(system_frame), sys_frame_title);
gtk_container_set_border_width(GTK_CONTAINER(system_frame), 20);

GtkWidget *cm_frame = gtk_frame_new("");
gtk_frame_set_label_widget(GTK_FRAME(cm_frame), cm_frame_title);
gtk_container_set_border_width(GTK_CONTAINER(cm_frame), 20);

GtkWidget *cpu_specific = gtk_frame_new("");
gtk_frame_set_label_widget(GTK_FRAME(cpu_specific), cpu_frame_title);
gtk_container_set_border_width(GTK_CONTAINER(cpu_specific), 20);

gtk_box_pack_start(GTK_BOX(vbox), system_frame, false, false, 0);
gtk_box_pack_start(GTK_BOX(vbox), cm_frame, false, false, 0);
gtk_box_pack_start(GTK_BOX(vbox), cpu_specific, false, false, 0);

GtkWidget *label = gtk_label_new("");
gtk_container_add(GTK_CONTAINER(system_frame), label);

sprintf(sys_specific, "%s\n%s\n%s\n%s\n",
       get_kernel_version(0),
       get_kernel_version(1),
       get_uptime(0),
       get_uptime(1));

gtk_label_set_text(GTK_LABEL(label), sys_specific);

GtkWidget *brief_label = gtk_label_new("");

gtk_container_add(GTK_CONTAINER(cm_frame), brief_label);

sprintf(cm_brief, "%s\n%s\n%s\n%s\n%s\n%s\n",
       get_hostname(),
       get_cpu_info_by_key("model name"),
       get_cpu_info_by_key("cpu cores"),
       get_cpu_info_by_key("cpu MHz"),
       get_cpu_info_by_key("cache size"),
       get_total_memory());

gtk_label_set_text(GTK_LABEL(brief_label), cm_brief);

GtkWidget *scrolled = gtk_scrolled_window_new(NULL,NULL);
gtk_container_add(GTK_CONTAINER(cpu_specific),scrolled);
gtk_scrolled_window_set_policy(GTK_SCROLLED_WINDOW(scrolled), GTK_POLICY_AUTOMATIC, GTK_POLICY_ALWAYS);
```

```
    GtkWidget *text=gtk_text_view_new();
    gtk_scrolled_window_add_with_viewport(GTK_SCROLLED_WINDOW(scrolled),text);
    gtk_widget_set_size_request(scrolled, 0, 600);

    GtkTextBuffer *buffer = gtk_text_view_get_buffer(GTK_TEXT_VIEW(text));
    char *specfics = get_specific_info();

    GtkTextIter start,end;

    gtk_text_buffer_get_bounds(GTK_TEXT_BUFFER(buffer),&start,&end);
    gtk_text_buffer_insert(GTK_TEXT_BUFFER(buffer),&start,specfics,(gint)strlen(specfics));
    return vbox;
}

char *get_cpu_info_by_key(char *key) {
    char name[VALUE_MAX_LEN] = {0};
    char value[VALUE_MAX_LEN] = {0};
    char ch;
    FILE *fp = fopen("/proc/cpuinfo", "r");
    while ((ch = (char)fgetc(fp)) != EOF) {
        int i = 0, sp = 0, j = 0;
        while (ch != ':') {
            name[i++] = ch;
            ch = (char)fgetc(fp);
        }
        for (sp = i - 1; sp >= 0; sp--) {
            if(name[sp] != ' ' && name[sp] != '\t')
                break;
        }
        name[sp + 1] = '\0';
        fgetc(fp);
        ch = (char)fgetc(fp);
        while (ch != '\n') {
            value[j++] = ch;
            ch = (char)fgetc(fp);
        }
        value[j] = '\0';
        if (!strcmp(key, name)) {
            char *rtn = (char *)malloc(sizeof(char) * VALUE_MAX_LEN);
            strcpy(rtn, key);
            strcat(rtn, " :\t");
            strcat(rtn, value);
            return rtn;
        }
```

```
    }
    fclose(fp);
    return NULL;
}

char *get_kernel_version(int rank) {
    FILE *fp = fopen("/proc/version", "r");
    char *rtn = (char *)malloc(sizeof(char) * VALUE_MAX_LEN);
    char ch;
    int i = 0;
    if (rank == 0) {
        strcpy(rtn, "Kernel version:");
        for (i = 0; i < VALUE_MAX_LEN; i++)
            if (rtn[i] == '\0') {
                rtn[i] = ' ';
                break;
            }
        rtn[i++] = ' ';
        ch = (char)getc(fp);
        while (ch != EOF) {
            if (ch == '(') {
                rtn[i] = '\0';
                break;
            }
            rtn[i++] = ch;
            ch = (char)getc(fp);
        }
    }
    else {
        strcpy(rtn, "Compile version:");
        for (i = 0; i < VALUE_MAX_LEN; i++)
            if (rtn[i] == '\0') {
                rtn[i] = ' ';
                break;
            }
        rtn[i++] = ' ';
        ch = (char)getc(fp);
        int par_count = 0;
        while (ch != EOF) {
            if (ch == '(') {
                par_count++;
                ch = (char)getc(fp);
                continue;
            }
```

```
            if (par_count != 2) {
                ch = (char) getc(fp);
                continue;
            }
            if (ch == ')') {
                rtn[i] = '\0';
                break;
            }
            rtn[i++] = ch;
            ch = (char)getc(fp);
        }
    }
    fclose(fp);
    return rtn;
}

char *get_specific_info(void) {
    FILE *fp = fopen("/proc/cpuinfo", "r");
    long length = 0;
    char ch;
    while ((ch = (char)fgetc(fp)) != EOF) {
        length++;
    }
    length++;
    rewind(fp);
    char *rtn = (char *)malloc(sizeof(char) * length);
    int index = 0;
    while ((ch = (char)fgetc(fp)) != EOF) {
        rtn[index++] = ch;
    }
    rtn[index] = '\0';
    fclose(fp);
    return rtn;
}

char *get_uptime(int rank) {
    FILE *fp = fopen("/proc/uptime", "r");
    double up = 0, idle = 0;
    char *up_time = (char *)malloc(sizeof(char) * VALUE_MAX_LEN);
    char *idle_time = (char *)malloc(sizeof(char) * VALUE_MAX_LEN);

    if (fp == NULL) {
        strcpy(up_time, "Open /proc/uptime error");
        strcpy(idle_time, "Open /proc/uptime error");
```

```
    }
    else {
        fscanf(fp, "%lf %lf", &up, &idle);

        if (up <= 60) {
            sprintf(up_time, "Up time: %.2lf seconds", up);
        }
        else if (up <= 60 * 60) {
            sprintf(up_time, "Up time: %.2lf minutes", up / 60);
        }
        else{
            sprintf(up_time, "Up time: %.2lf days", up / 60 / 60);
        }

        if (idle <= 60) {
            sprintf(idle_time, "Idle time: %.2lf seconds", idle);
        }
        else if (idle <= 60 * 60) {
            sprintf(idle_time, "Idle time: %.2lf minutes", idle / 60);
        }
        else{
            sprintf(idle_time, "Idle time: %.2lf days", idle / 60 / 60);
        }

    }

    fclose(fp);
    if (rank == 0) {
        free(idle_time);
        idle_time = NULL;
        return up_time;
    }
    else {
        free(up_time);
        up_time = NULL;
        return idle_time;
    }
}

char *get_total_memory(void) {
    FILE *fp = fopen("/proc/meminfo", "r");
    double up = 0, idle = 0;
    char *rtn = (char *)malloc(sizeof(char) * VALUE_MAX_LEN);
    char memtotal[VALUE_MAX_LEN / 2];
```

```
    unsigned long sz;

    if (fp == NULL) {
        strcpy(rtn, "Open /proc/meminfo error");
    }
    else {
        fscanf(fp, "%s %lu", memtotal, &sz);
        if (sz <= 1024) {
            sprintf(rtn, "%s\t%lu KB", memtotal, sz);
        }
        else if (sz <= 1024 * 1024) {
            sprintf(rtn, "%s\t%lu MB", memtotal, (unsigned long)(sz / 1024.0 + 0.5));
        }
        else {
            sprintf(rtn, "%s\t%lu GB", memtotal, (unsigned long)(sz / 1024.0 / 1024.0 + 0.5));
        }
    }
    fclose(fp);

    return rtn;
}

char *get_hostname(void) {
    FILE *fp = fopen("/proc/sys/kernel/hostname", "r");
    double up = 0, idle = 0;
    char *rtn = (char *)malloc(sizeof(char) * VALUE_MAX_LEN);
    char hostname[VALUE_MAX_LEN / 2];

    if (fp == NULL) {
        strcpy(rtn, "Open /proc/sys/kernel/hostname error");
    }
    else {
        fscanf(fp, "%s", hostname);
        if (hostname[0] < 'z' && hostname[0] > 'a')
            hostname[0] += ('A' - 'a');
        sprintf(rtn, "Hostname:\t%s", hostname);
    }

    fclose(fp);

    return rtn;
}
```

文件系统源代码

主函数 main.c

```
#include <stdio.h>
#include "data_structures.h"
#include "user.h"
#include "super_block.h"
#include "dentry.h"
#include "../include/instruction_handle.h"

//#define __debug_mode

struct usr_ptr *current_usr = NULL;
struct dentry *root_dir = NULL;
struct dentry *current_dir = NULL;
dlist *usr_dir_list = NULL;

int main(int argc, const char *argv[]) {

#ifdef __debug_mode
    printf("%lu\n", sizeof(struct inode));
#endif

#ifndef __debug_mode
    struct super_block *sb = NULL;
    root_dir = (struct dentry *)malloc(sizeof(struct dentry));
    if (argc == 1) {
        sb = load_block(root_dir);
    }
    else {
        sb = init_block(root_dir);
    }
    struct user_linked_list *head = load_users_info();
    load_entry(root_dir, sb);

    current_usr = (struct usr_ptr *)malloc(sizeof(current_usr));
    current_dir = root_dir;

    usr_dir_list = init_dlist();
    packet_user_dlist(usr_dir_list, head, root_dir);

    char init_user_name[USER_NAME_MAX_LEN];
    bool logon = false;
    while (!logon) {
        printf("please input user name!\n");
```

```
    fgets(init_user_name, USER_NAME_MAX_LEN, stdin);
    init_user_name[strlen(init_user_name) - 1] = '\0';
    logon = get_user_by_name(head, current_usr, init_user_name);
}

current_dir = change_home_dir(usr_dir_list, current_usr);

bool endSignal = false;

while (!endSignal) {
    char line[LINE_MAX_LEN];
    char instr[INSTR_MAX_LEN];
    char text_dir[LINE_MAX_LEN];
    char input_usr_name[USER_NAME_MAX_LEN];
    char redundent[LINE_MAX_LEN];
    char option[LINE_MAX_LEN];
    unsigned int input_mode;

    printf("\x1b[32m""%s@vfs\t""\x1b[0m", current_usr->name);
    get_current_path_str(current_dir, text_dir);
    fflush(stdout);

    fgets(line, LINE_MAX_LEN, stdin);
    sscanf(line, "%s", instr);
    instr_type type_instr = raw_instruction_handle(instr);

    switch (type_instr) {
        case __shutdown:
            endSignal = true;
            printf("System shut down by user :%s\n", current_usr->name);
            break;
        case __change__mode:
            if (!strcmp(current_usr->name, "root")) {
                if (sscanf(line, "%s%u%s", input_usr_name, &input_mode, input_usr_name) == 3)
                    chmod_handle(current_dir, input_mode, input_usr_name);
                else
                    type_instr = __error_instr;
            }
            else {
                printf("permission denied!\n");
            }
            break;
        case __swap_user:
            if (sscanf(line, "%s%s%s", input_usr_name, input_usr_name, redundent) == 2) {
```

```
        if (get_user_by_name(head, current_usr, input_usr_name) == true)
            current_dir = change_home_dir(usr_dir_list, current_usr);
    }
    break;
case __stdout_clear:
    system("clear");
    break;
case __list_file:
    if (sscanf(line, "%s%s", input_usr_name, option) == 1)
        ls_handle(current_dir, root_dir, 'n', head);
    else if (strlen(option) > 1)
        ls_handle(current_dir, current_dir, option[1], head);
    else
        type_instr = __error_instr;
    break;
case __make_directory:
    if (sscanf(line, "%s%s", input_usr_name, redundent) == 2)
        mkdir_handle(current_dir, redundent, current_usr, sb);
    else
        type_instr = __error_instr;
    break;
case __change_directory:
    if (sscanf(line, "%s%s", input_usr_name, redundent) == 2)
        current_dir = cd_handle(current_dir, redundent);
    else
        type_instr = __error_instr;
    break;
case __rename_file:
    if (sscanf(line, "%s%s%s", text_dir, input_usr_name, redundent) == 3)
        rename_handle(current_dir, input_usr_name, redundent);
    else
        type_instr = __error_instr;
    break;
case __remove_file:
    if (sscanf(line, "%s%s", input_usr_name, redundent) == 2)
        rm_handle(current_dir, redundent, sb, current_usr);
    else
        type_instr = __error_instr;
    break;
case __edit_file:
    if (sscanf(line, "%s%s", input_usr_name, redundent) == 2)
        edit_handle(current_dir, redundent, sb, current_usr);
    else
        type_instr = __error_instr;
```

```
                break;
            case __cat_file:
                if (sscanf(line, "%s%s", input_usr_name, redundent) == 2)
                    cat_handle(current_dir, redundent, sb, current_usr);
                else
                    type_instr = __error_instr;
                break;
            case __error_instr:
                break;
            default:
                printf("Some unknown error occurs!\n");
                break;
        }
        if (type_instr == __error_instr)
            printf("Cannot resolve instruction %s\n", instr);
    }

    save_users_info(head);
    free_user_info(head);
    save_entry(root_dir, sb);
    save_block(sb);
    free_block(sb);
    fflush(stdin);
#endif
    return 0;
}
```

指令处理头文件 instruction_handle.h

```
//
// Created by 永鑫  徐 on 2019-02-27.
//

#ifndef VFS_INSTRUCTION_HANDLE_H
#define VFS_INSTRUCTION_HANDLE_H

#include <string.h>
#include "data_structures.h"
#include "user.h"
#include "dentry.h"
#include "super_block.h"

#define INSTR_MAX_LEN 50
#define LINE_MAX_LEN 2048
#define DEFAULT_FILE_BLK 64
```

```
enum INSTR_TYPE {
    __shutdown, __swap_user, __list_file, __rename_file,
    __change_directory, __remove_file,  __change__mode,
    __edit_file, __stdout_clear, __make_directory, __cat_file,
    __error_instr,
};

typedef int instr_type;

instr_type raw_instruction_handle(char instr[INSTR_MAX_LEN]);
void ls_handle(struct dentry *dentry, struct dentry *begin_dentry, char option, struct
user_linked_list *head);
struct dentry *cd_handle(struct dentry *dentry, const char *path);
void chmod_handle(struct dentry *cur_dir, unsigned int mode_num, char *name);

slist *init_slist(void);
void insert_slist(slist *sl, char *str, char type);
void sort_slist(slist *sl);
bool str_in_slists(slist *sl, char *str);
void free_slist(slist **sl);
void print_slist(slist *sl);
bool mkdir_handle(struct dentry *parent_dir, const char *dir_name, struct usr_ptr *user, struct
super_block *sb);
bool rename_handle(struct dentry *parent_dir, const char *dir_name, const char *new_name);
bool rm_handle(struct dentry *parent_dir, const char *dir_name, struct super_block *sb, struct
usr_ptr *user);
bool edit_handle(struct dentry *parent_dir, const char *dir_name, struct super_block *sb, struct
usr_ptr *user);
bool cat_handle(struct dentry *parent_dir, const char *dir_name, struct super_block *sb, struct
usr_ptr *user);

void str_get_priority(struct inode *inode1, char *buf);
unsigned long priority_get_by_usr(struct usr_ptr *user);
bool permit_write(u_mode_t f_mode, unsigned long priority);
bool permit_read(u_mode_t f_mode, unsigned long priority);

#endif //VFS_INSTRUCTION_HANDLE_H
```

指令处理源文件 instruction_handle.c

```
//
// Created by 永鑫  徐 on 2019-02-27.
//

#include "../include/instruction_handle.h"
```

```
#define ANSI_COLOR_RED     "\x1b[31m"
#define ANSI_COLOR_GREEN   "\x1b[32m"
#define ANSI_COLOR_YELLOW  "\x1b[33m"
#define ANSI_COLOR_BLUE    "\x1b[34m"
#define ANSI_COLOR_MAGENTA "\x1b[35m"
#define ANSI_COLOR_CYAN    "\x1b[36m"
#define ANSI_COLOR_RESET   "\x1b[0m"

#define PRINT_LEN 100

slist *init_slist(void) {
    slist *sl = (slist *)malloc(sizeof(slist));
    sl->str = NULL;
    sl->next = NULL;
    return sl;
}

void insert_slist(slist *sl, char *str, char type) {
    slist *tmp = sl;
    while (tmp->next) {
        tmp = tmp->next;
    }
    tmp->next = (slist *)malloc(sizeof(slist));
    tmp = tmp->next;
    tmp->str = (char *)malloc(sizeof(char) * PRINT_LEN);
    tmp->type = type;
    strcpy(tmp->str, str);
    tmp->next = NULL;
}

void sort_slist(slist *sl) {
    slist *tmp = sl;
    int len = 0;
    while(tmp->next) {
        tmp = tmp->next;
        len++;
    }
    slist *base = sl;
    slist *min = base, *prev = base;
    for (int i = 0; i < len - 1; i++) {
        bool change = false;
        min = base->next;
        tmp = base;
```

```
        for (int j = i; j < len; j++) {
            if (strcmp(tmp->next->str, min->str) < 0) {
                min = tmp->next;
                prev = tmp;
                change = true;
            }
            tmp = tmp->next;
        }
        if (change) {
            prev->next = min->next;
            min->next = base->next;
            base->next = min;
        }
        base = base->next;
    }
}

bool str_in_slists(slist *sl, char *str) {
    bool found = false;
    slist *tmp = sl;
    while (tmp->next) {
        tmp = tmp->next;
        if (!strcmp(tmp->str, str)) {
            found = true;
            break;
        }
    }
    return found;
}

void free_slist(slist **sl) {
    slist *tmp = *sl;
    slist *prev = *sl;

    while (tmp) {
        prev = tmp;
        tmp = tmp->next;
        free(prev->str);
        free(prev);
    }
    tmp = NULL;
    *sl = NULL;
}
```

```
void print_slist(slist *sl) {
    slist *tmp = sl;
    while (tmp->next) {
        tmp = tmp->next;
        if (tmp->type == __directory)
            printf(ANSI_COLOR_BLUE "%s" ANSI_COLOR_RESET "\t", tmp->str);
        else if (tmp->type == __link)
            printf(ANSI_COLOR_GREEN "%s" ANSI_COLOR_RESET "\t", tmp->str);
        else
            printf("%s\t", tmp->str);
    }
    putchar('\n');
}

instr_type raw_instruction_handle(char instr[INSTR_MAX_LEN]) {
    instr_type rtn;
    if (!strcmp(instr, "shutdown"))
        rtn = __shutdown;
    else if (!strcmp(instr, "su"))
        rtn = __swap_user;
    else if (!strcmp(instr, "ls"))
        rtn = __list_file;
    else if (!strcmp(instr, "mkdir"))
        rtn = __make_directory;
    else if (!strcmp(instr, "chmod"))
        rtn = __change__mode;
    else if (!strcmp(instr, "cd"))
        rtn = __change_directory;
    else if (!strcmp(instr, "rename"))
        rtn = __rename_file;
    else if (!strcmp(instr, "rm"))
        rtn = __remove_file;
        // todo:
    else if (!strcmp(instr, "edit"))
        rtn = __edit_file;
    else if (!strcmp(instr, "clear"))
        rtn = __stdout_clear;
        // todo:
    else if (!strcmp(instr, "cat"))
        rtn = __cat_file;
    else
        rtn = __error_instr;
    return rtn;
}
```

```
void ls_handle(struct dentry *dentry, struct dentry *begin_dentry, char option, struct
user_linked_list *head) {
    if (option == 'n') {
        slist *str_list = init_slist();
        for (int i = 0; i < HASH_TABLE_ROW; i++) {
            struct dir_hash_table *ptr = dentry->subdirs[i];
            while(ptr->next) {
                ptr = ptr->next;
                insert_slist(str_list, ptr->dname, ptr->corres_dentry->type);
            }
        }
        sort_slist(str_list);
        print_slist(str_list);
        free_slist(&str_list);
    }
    else if (option == 'l') {
        slist *str_list = init_slist();
        for (int i = 0; i < HASH_TABLE_ROW; i++) {
            struct dir_hash_table *ptr = dentry->subdirs[i];
            while(ptr->next) {
                ptr = ptr->next;
                if (!strcmp(ptr->dname, ".") || !strcmp(ptr->dname, ".."))
                    continue;
                char *buf = (char *)malloc(sizeof(char) * 11);
                str_get_priority(ptr->corres_dentry->d_inode, buf);

                if (ptr->corres_dentry->type == __directory) {
                    buf[0] = __directory;
                    printf("%s\t", buf);
                }
                else if (ptr->corres_dentry->type == __link) {
                    buf[0] = __link;
                    printf("%s\t", buf);
                }
                else {
                    buf[0] = __file;
                    printf("%s\t", buf);
                }

                if (ptr->corres_dentry->type == __directory || ptr->corres_dentry->type == __link)
                    printf("%lu\t%s\t%lu\t\t", ptr->corres_dentry->d_inode->i_nlink,
get_user_by_user_id(head, ptr->corres_dentry->d_inode->i_uid),
                           sizeof(struct inode));
```

```
            else
                printf("%lu\t%s\t%lu\t\t", ptr->corres_dentry->d_inode->i_nlink,
get_user_by_user_id(head, ptr->corres_dentry->d_inode->i_uid)
                       ,                ptr->corres_dentry->d_inode->i_btyes);
            print_time(ptr->corres_dentry->d_time);
            if (ptr->corres_dentry->type == __directory)
                printf("\t" ANSI_COLOR_BLUE "%s" ANSI_COLOR_RESET "\n", ptr->dname);
            else if (ptr->corres_dentry->type == __link)
                printf("\t" ANSI_COLOR_GREEN "%s" ANSI_COLOR_RESET "\n", ptr->dname);
            else
                printf("\t""%s\n", ptr->dname);
        }
    }
}
else if (option == 'R') {
    slist *str_list = init_slist();
    queue *q = init_queue();
    enqueue(begin_dentry, q);
    q->front->aux = 0;
    int last_level = 0;
    struct dentry *cur_entry = begin_dentry;

    size_t num = 0;

    for (int i = 0; i < HASH_TABLE_ROW; i++) {
        struct dir_hash_table *ptr = cur_entry->subdirs[i];
        while (ptr->next) {
            ptr = ptr->next;
            if (!strcmp(ptr->dname, ".") || !strcmp(ptr->dname, ".."))
                continue;
            num++;
        }
    }

    size_t num_of_first_level = 0;

    while (cur_entry != NULL) {
        struct dentry *temp_save = cur_entry;
        bool found = false;
        for (int i = 0; i < HASH_TABLE_ROW; i++) {
            struct dir_hash_table *ptr = cur_entry->subdirs[i];
            while (ptr->next) {
                ptr = ptr->next;
                if (!strcmp(ptr->dname, ".."))
```

```
                    continue;
                if (!strcmp(ptr->dname, "."))
                    continue;
                if (str_in_slists(str_list, ptr->dname)) {
                    continue;
                }
                enqueue(ptr->corres_dentry, q);
                insert_slist(str_list, ptr->dname, ptr->corres_dentry->type);
                found = true;
                last_level++;
                q->rear->aux = last_level;
                cur_entry = ptr->corres_dentry;
                break;
            }
            if (found == true)
                break;
        }
        if (found == false) {
            cur_entry = temp_save->parent;
            last_level--;
//              hash_table_delete(cur_entry, temp_save);
        }
        if (temp_save == begin_dentry) {
            num_of_first_level++;
            if (num_of_first_level == num) {
                break;
            }
        }
    }

    free_slist(&str_list);

    while (!queue_empty(q)) {
        for (int i = 0 ; i < q->front->aux; i++) {
            putchar('-');
        }
        if (((struct dentry *)q->front->val)->type == __directory)
            printf(ANSI_COLOR_BLUE "%s" ANSI_COLOR_RESET "\n", ((struct dentry
*)q->front->val)->d_iname);
        else if (((struct dentry *)q->front->val)->type == __link)
            printf(ANSI_COLOR_GREEN "%s" ANSI_COLOR_RESET "\n", ((struct dentry
*)q->front->val)->d_iname);
        else
            printf("%s\n", ((struct dentry *)q->front->val)->d_iname);
```

```
            dequeue(q);
        }
    }

}

struct dentry *cd_handle(struct dentry *dentry, const char *path) {
    queue *dir_path = resolve_path_to_queue(path);
    if (dir_path == NULL) {
        printf("cannot resolve path: %s\n", path);
        return dentry;
    }
    struct dentry *temp = dentry;
    if (!strcmp(get_front_queue(dir_path), "/")) {
        while (temp->parent)
            temp = temp->parent;
        dequeue(dir_path);
    }
    while (!queue_empty(dir_path)) {
        char *change_to_str = get_front_queue(dir_path);
        if (!strcmp(".", change_to_str))
            temp = temp;
        else if (!strcmp("..", change_to_str))
            temp = temp->parent;
        else
            temp = search_by_str(temp, change_to_str);
        if (!temp) {
            printf("No such file or directory: %s\n", path);
            return dentry;
        }
        dequeue(dir_path);
    }
    return temp;
}

void chmod_handle(struct dentry *cur_dir, unsigned int mode_num, char *name) {
    struct dentry *rtn = search_by_str(cur_dir, name);
    rtn->d_inode->mode = priority_get_by_number(mode_num);
}

void str_get_priority(struct inode *inode1, char *buf) {
    char paylod[] = "-rwxrwxrwx";
    const unsigned long mask1 = 0x01;
    unsigned long t = inode1->mode;
```

```
    size_t len = strlen(paylod);
    for (int i = 0; i < 9; i++) {
        unsigned long temp = mask1 << i;
        if (((temp & t) >> i) != mask1)
            paylod[len - 1 - i] = '-';
    }
    strcpy(buf, paylod);
}

bool mkdir_handle(struct dentry *parent_dir, const char *dir_name, struct usr_ptr *user, struct
super_block *sb) {
    if (search_by_str(parent_dir, dir_name) != NULL) {
        printf("file exists!\n");
        return false;
    }
    else {
        struct dentry *new_dir = (struct dentry *)malloc(sizeof(struct dentry));
        new_dir->d_inode = alloc_inode(sb);
        new_dir->d_inode->i_uid = user->u_id;
        new_dir->d_inode->mode = priority_get_by_usr(user);
        new_dir->d_inode->i_btyes = 0;
        new_dir->d_inode->i_sb = sb;
        new_dir->d_inode->i_blocks = 0;

        if (new_dir->d_inode == NULL)
            return false;
        else {
            new_dir->d_time = get_local_time();
            strcpy(new_dir->d_iname, dir_name);
            new_dir->type = __directory;
            new_dir->parent = parent_dir;
            for (int i = 0; i < HASH_TABLE_ROW; i++) {
                new_dir->subdirs[i] = (struct dir_hash_table *)malloc(sizeof(struct
dir_hash_table));
                new_dir->subdirs[i]->dname = NULL;
                new_dir->subdirs[i]->next = NULL;
                new_dir->subdirs[i]->corres_dentry = NULL;
            }
            new_dir->d_sb = sb;
        }

        struct dentry *new_dir_self = (struct dentry *)malloc(sizeof(struct dentry));
        new_dir_self->parent = NULL;
        new_dir_self->d_time = new_dir->d_time;
```

```
        new_dir_self->type = __link;
        strcpy(new_dir_self->d_iname, ".");
        new_dir_self->d_inode = new_dir->d_inode;

        new_dir_self->d_inode->i_nlink++;

        hash_table_insert(new_dir, new_dir_self);

        hash_table_insert(parent_dir, new_dir);
        struct dentry *new_dir_parent = (struct dentry *)malloc(sizeof(struct dentry));
        new_dir_parent->parent = NULL;
        new_dir_parent->d_time = parent_dir->d_time;
        new_dir_parent->type = __link;

        strcpy(new_dir_parent->d_iname, "..");
        new_dir_parent->d_inode = parent_dir->d_inode;
        new_dir_parent->d_inode->i_nlink++;

        hash_table_insert(new_dir, new_dir_parent);
    }
    return true;
}

unsigned long priority_get_by_usr(struct usr_ptr *user) {
    if (user->priority == 0)
        return priority_get_by_number(751);
    else if (user->priority == 1)
        return priority_get_by_number(771);
    else
        return priority_get_by_number(777);
}

bool rename_handle(struct dentry *parent_dir, const char *dir_name, const char *new_name) {
    struct dentry *current_dir = search_by_str(parent_dir, dir_name);
    if (current_dir == NULL) {
        printf("file doesn't exist!\n");
        return false;
    }
    else {
        hash_table_delete(parent_dir, current_dir);
        strcpy(current_dir->d_iname, new_name);
        hash_table_insert(parent_dir, current_dir);
    }
    return true;
```

```
}

bool rm_handle(struct dentry *parent_dir, const char *dir_name, struct super_block *sb, struct
usr_ptr *user) {
    struct dentry *target_dir = search_by_str(parent_dir, dir_name);
    if (target_dir == NULL) {
        printf("file doesn't exist\n");
        return false;
    }
    else {
        unsigned long prio = user->priority;
        if (!permit_write(target_dir->d_inode->mode, prio)) {
            printf("permission denied!\n");
            return false;
        }
        if (target_dir->type == __directory){
            size_t num = 0;

            for (int i = 0; i < HASH_TABLE_ROW; i++) {
                struct dir_hash_table *ptr = target_dir->subdirs[i];
                while (ptr->next) {
                    ptr = ptr->next;
                    if (!strcmp(ptr->dname, ".") || !strcmp(ptr->dname, ".."))
                        continue;
                    num++;
                }
            }

            if (num > 0) {
                printf("rm: %s: is a directory\n", dir_name);
                return false;
            }

            else {
                hash_table_delete(parent_dir, target_dir);
                for (int i = 0; i < HASH_TABLE_ROW; i++) {
                    struct dir_hash_table *ptr = target_dir->subdirs[i], *prev;
                    while (ptr) {
                        prev = ptr;
                        if (prev->dname) {
                            if (!strcmp(prev->dname, ".."))
                                prev->corres_dentry->parent->d_inode->i_nlink--;
                            else if (!strcmp(prev->dname, "."))
                                prev->corres_dentry->d_inode->i_nlink--;
```

```
                    free(prev->dname);
                }
                ptr = ptr->next;
                free(prev);
                prev = NULL;
            }
        }
        if (target_dir->d_inode->i_nlink == 0) {
            free_block_for_inode(sb, target_dir->d_inode);
        }
        free(target_dir);
    }
}
else if (target_dir->type == __file){
    hash_table_delete(parent_dir, target_dir);
    free_block_for_inode(sb, target_dir->d_inode);
    free(target_dir);
}
else {
    printf("there are some problems with rm_handle()\n");
    return false;
}
}
return true;
}

bool cat_handle(struct dentry *parent_dir, const char *dir_name, struct super_block *sb, struct
usr_ptr *user) {
    struct dentry *target_dir = search_by_str(parent_dir, dir_name);
    if (target_dir == NULL) {
        printf("file doesn't exist!\n");
        return false;
    }
    else {
        if (target_dir->type == __directory || target_dir->type == __link) {
            printf("%s is not a file\n", dir_name);
            return false;
        }
        else {
            char *content = (char *)malloc(sizeof(char) * target_dir->d_inode->i_btyes);
            struct blk_lists *blk_ptr = target_dir->d_inode->i_list;
            if (target_dir->d_inode->i_blocks == 0 | target_dir->d_inode->i_btyes == 0) {
                return false;
            }
```

```
            unsigned long offset = 0;
            while (blk_ptr->next_blk) {
                blk_ptr = blk_ptr->next_blk;
                size_t rtn = read_block(sb, blk_ptr->blk_no, content + offset, blk_ptr->blk_len);
                offset += rtn;
            }
            printf("%s", content);
            fflush(stdout);
            free(content);
        }
    }
    return true;
}

bool edit_handle(struct dentry *parent_dir, const char *dir_name, struct super_block *sb, struct
usr_ptr *user) {
    struct dentry *target_dir = search_by_str(parent_dir, dir_name);
    fflush(stdin);
    if (target_dir == NULL) {
        struct dentry *new_dir = (struct dentry *)malloc(sizeof(struct dentry));
        new_dir->d_inode = alloc_inode(sb);
        new_dir->d_inode->i_uid = user->u_id;
        new_dir->d_inode->mode = priority_get_by_usr(user);
        new_dir->d_inode->i_btyes = 0;
        new_dir->d_inode->i_sb = sb;
        new_dir->d_inode->i_blocks = 0;

        if (new_dir->d_inode == NULL)
            return false;
        else {
            new_dir->d_time = get_local_time();
            strcpy(new_dir->d_iname, dir_name);
            new_dir->type = __file;
            new_dir->parent = parent_dir;
            for (int i = 0; i < HASH_TABLE_ROW; i++) {
                new_dir->subdirs[i] = NULL;
            }
            new_dir->d_sb = sb;
        }

        hash_table_insert(parent_dir, new_dir);
        char *content = (char *)malloc(sizeof(char) * DEFAULT_FILE_BLK);
        char ch = 0;
        long bytes = 0;
```

```
    int round = 1;
    system("clear");
    printf("%s:\n", dir_name);
    while ( ch != '\n') {
        ch = (char)getchar();
        content[bytes++] = ch;
        if (bytes > round * DEFAULT_FILE_BLK) {
            round++;
            content = (char *)realloc(content, sizeof(char) * (round * DEFAULT_FILE_BLK));
        }
    }
    content[bytes] = '\0';

    new_dir->d_inode->i_btyes = (unsigned long)bytes;
    new_dir->d_inode->i_list = (struct blk_lists *)malloc(sizeof(struct blk_lists));
    struct blk_lists *blk_ptr = new_dir->d_inode->i_list;
    unsigned long offset = 0;
    while (bytes > 0) {
        blk_ptr->next_blk = (struct blk_lists *)malloc(sizeof(struct blk_lists));
        blk_ptr = blk_ptr->next_blk;
        blk_ptr->blk_no = alloc_data_block(sb);
        new_dir->d_inode->i_blocks++;
        if (bytes > sb->s_blocksize) {
            blk_ptr->blk_len = sb->s_blocksize;
        }
        else {
            blk_ptr->blk_len = (unsigned long)bytes;
        }
        size_t rtn = write_block(sb, blk_ptr->blk_no, content + offset, blk_ptr->blk_len);
        blk_ptr->next_blk = NULL;
        bytes -= sb->s_blocksize;
        offset += rtn;
    }
}
else {
    if (!permit_write(target_dir->d_inode->mode, user->priority)) {
        printf("permission denied!\n");
        return false;
    }
    else {
        struct blk_lists *blk_ptr = target_dir->d_inode->i_list;
        char *content = (char *)malloc(sizeof(char) * target_dir->d_inode->i_btyes);
        char ch = 0;
        long bytes = target_dir->d_inode->i_btyes;
```

```
            int round = 1;
            system("clear");
            printf("%s:\n", dir_name);
            unsigned long offset = 0;
            while (blk_ptr->next_blk) {
                blk_ptr = blk_ptr->next_blk;
                size_t rtn = read_block(sb, blk_ptr->blk_no, content + offset, blk_ptr->blk_len);
                bytes -= sb->s_blocksize;
                offset += rtn;
            }
            printf("%s", content);
            fflush(stdin);
            char *new_content = (char *)malloc(sizeof(char) * DEFAULT_FILE_BLK);
            int t = 0;
            round = 1;
            while ( ch != '\n') {
                ch = (char)getchar();
                new_content[t++] = ch;
                if (t > round * DEFAULT_FILE_BLK) {
                    round++;
                    new_content = (char *)realloc(new_content, sizeof(char) * (round *
DEFAULT_FILE_BLK));
                }
            }
            new_content[t] = '\0';
            long new_len = t + target_dir->d_inode->i_btyes;
            char *new_str = (char *)malloc((unsigned long)new_len);
            strcpy(new_str, content);
            strcat(new_str, new_content);
            target_dir->d_inode->i_btyes = (unsigned long)new_len;
            struct blk_lists *tmp_ptr = target_dir->d_inode->i_list;
            offset = 0;
            while (new_len > 0) {
                if (tmp_ptr->next_blk == NULL) {
                    tmp_ptr->next_blk = (struct blk_lists *)malloc(sizeof(struct blk_lists));
                    tmp_ptr = tmp_ptr->next_blk;
                    tmp_ptr->blk_no = alloc_data_block(sb);
                    target_dir->d_inode->i_blocks++;
                    if (bytes > sb->s_blocksize) {
                        tmp_ptr->blk_len = sb->s_blocksize;
                    }
                    else {
                        tmp_ptr->blk_len = (unsigned long)new_len;
                    }
```

```
                size_t rtn = write_block(sb, tmp_ptr->blk_no, content + offset,
tmp_ptr->blk_len);
                tmp_ptr->next_blk = NULL;
                new_len -= sb->s_blocksize;
                offset += rtn;
            }
            else {
                tmp_ptr = tmp_ptr->next_blk;
                if (new_len > sb->s_blocksize)
                    tmp_ptr->blk_len = sb->s_blocksize;
                else
                    tmp_ptr->blk_len = (unsigned long)new_len;
                size_t rtn = write_block(sb, tmp_ptr->blk_no, new_str + offset,
tmp_ptr->blk_len);
                new_len = new_len - rtn;
            }
        }
    }
    return true;
}

bool permit_write(u_mode_t f_mode, unsigned long priority) {
    const unsigned long a_mask = 0x01;
    unsigned long shift = 1 + (2 - priority) * 3;
    unsigned long temp = (a_mask << shift) & f_mode;
    if ((temp >> shift) == a_mask) {
        return true;
    }
    return false;
}

bool permit_read(u_mode_t f_mode, unsigned long priority) {
    const unsigned long a_mask = 0x01;
    unsigned long shift = 2 + (2 - priority) * 3;
    unsigned long temp = (a_mask << shift) & f_mode;
    if ((temp >> shift) == a_mask) {
        return true;
    }
    return false;
}
```

数据结构整体定义头文件 data_structure.h

```
//
// Created by 永鑫   徐 on 2019-02-26.
//

#ifndef VFS_DATA_STRUCTURES_H
#define VFS_DATA_STRUCTURES_H

#define DNAME_LEN 100
#define HASH_TABLE_ROW 10

typedef unsigned long blkcnt_t;
typedef unsigned long u_mode_t;

#include "trival_helper.h"
#include "user.h"

enum d_type {
    __directory = 'd',
    __link = 'l',
    __file = '-'
};

struct blk_lists {
    blkcnt_t blk_no;
    unsigned long blk_len;
    struct blk_lists *next_blk;
};

typedef struct ___dentry__list {
    struct dentry *dir;
    struct ___dentry__list *next;
}dlist;

typedef struct ___str__list {
    char *str;
    char type;
    struct ___str__list *next;
}slist;

struct dir_hash_table {
    int hash_key;
    char *dname;
    struct dentry *corres_dentry;
    struct dir_hash_table *next;
```

```
};

struct super_block {
    unsigned long s_blocknumbers;   // 块的总数量
    unsigned long s_blocksize;      // 块的大小
//    struct inode_linked_list *s_inodes; // 超级块的空闲 inode 链表
//    unsigned long s_inodes_num;     // 空闲 inode 数量
    void *s_bdev;       // 存储块
    int s_bitmap_blks;  // 位示图的块数量
};

struct inode {
    struct super_block *i_sb;
    struct blk_lists *i_list;
    unsigned long i_nlink;
    unsigned long i_no;
    unsigned long i_uid;
    blkcnt_t i_blocks;
    unsigned long i_btyes;
    u_mode_t mode;
};

struct dentry {
    char type;
    struct inode *d_inode;
    struct dir_hash_table *subdirs[HASH_TABLE_ROW];
    struct tm *d_time;
    struct super_block *d_sb;
    struct dentry *parent;
    char d_iname[DNAME_LEN];
};

#endif //VFS_DATA_STRUCTURES_H
```

超级块具体函数定义 super_block.h

```
//
// Created by 永鑫   徐 on 2019-02-25.
//

#ifndef VFS_SUPER_BLOCK_H
#define VFS_SUPER_BLOCK_H

#include <stdio.h>
#include <memory.h>
```

```c
#include <stdbool.h>
#include "../include/data_structures.h"
#include "../include/inode.h"
#include "../include/dentry.h"

#define BLK_SIZE 1024
#define BLK_NUM 102400

struct super_block *init_block(struct dentry *root);
struct super_block *free_block(struct super_block *sb);
bool save_block(struct super_block *sb);
struct super_block *load_block(struct dentry *root);
struct inode *alloc_inode(struct super_block *sb);
void destroy_inode(struct inode *inode);
bool occupy_block(blkcnt_t blk_no, struct super_block *sb);
bool release_block(blkcnt_t blk_no, struct super_block *sb);
bool alloc_block_for_inode(struct super_block *sb, struct inode *inode);
void *free_block_for_inode(struct super_block *sb, struct inode *inode);
unsigned long alloc_data_block(struct super_block *sb);
void free_data_block(unsigned long blk_no, struct super_block *sb);
size_t write_block(struct super_block *sb, unsigned long block_no, char *stream, size_t size);
size_t read_block(struct super_block *sb, unsigned long block_no, char *stream, size_t size);

#endif //VFS_SUPER_BLOCK_H
```

超级块实现 super_block.c

```c
//
// Created by 永鑫  徐 on 2019-02-25.
//

#include "../include/super_block.h"

const char * const save_path = "./data/blk_data.dat";

struct super_block *init_block(struct dentry *root) {
    struct super_block *sb = (struct super_block *)malloc(sizeof(struct super_block));
    sb->s_bdev = (char *)malloc(sizeof(char) * BLK_NUM * BLK_SIZE);
    sb->s_blocknumbers = BLK_NUM;
    sb->s_blocksize = BLK_SIZE;
    sb->s_bitmap_blks = (int)(BLK_NUM * 1.0 / BLK_SIZE / 8 + 0.5);
//    sb->s_inodes = NULL;
//    sb->s_inodes_num = 0;

    root->parent = NULL;
```

```
    root->type = __directory;
    root->d_time = get_local_time();
    root->d_sb = sb;
    strcpy(root->d_iname, "/");
    root->d_inode = alloc_inode(sb);
    for (int i = 0; i < HASH_TABLE_ROW; i++) {
        root->subdirs[i] = (struct dir_hash_table *)malloc(sizeof(struct dir_hash_table));
        root->subdirs[i]->dname = NULL;
        root->subdirs[i]->next = NULL;
        root->subdirs[i]->corres_dentry = NULL;
    }
    struct dentry *root_self = (struct dentry *)malloc(sizeof(struct dentry));
    root_self->d_sb = sb;
    root_self->parent = NULL;
    root_self->d_time = root->d_time;
    root_self->type = __link;
    strcpy(root_self->d_iname, ".");
    root_self->d_inode = root->d_inode;
    root->d_inode->i_nlink++;
    hash_table_insert(root, root_self);

    unsigned long offset = 0;
    memcpy(sb->s_bdev + offset, &sb->s_blocknumbers, sizeof(unsigned long));
    offset += sizeof(unsigned long);
    memcpy(sb->s_bdev + offset, &sb->s_blocksize, sizeof(unsigned long));
    offset += sizeof(unsigned long);
    memcpy(sb->s_bdev + offset, &sb->s_bitmap_blks, sizeof(int));
    offset += sizeof(int);
//    memcpy(sb->s_bdev + offset, &sb->s_inodes_num, sizeof(unsigned long));
//    offset += sizeof(unsigned long);

    int number_of_chars =  (sb->s_bitmap_blks + (sizeof(char) - 1)) / sizeof(char);
    char mask = 1;
    unsigned round = 0;
    const char char_mask = 0x01;
    int temp = sb->s_bitmap_blks;
    for (int i = 0; i < number_of_chars; i++) {
        for (int j = 0; j < sizeof(char) * 8; j++) {
            if (temp > 0) {
                temp--;
                mask = (mask << 1) | char_mask;
            }
            else {
                mask = (mask << 1) & (~char_mask);
```

```
            }
        }
        memcpy(sb->s_bdev + BLK_SIZE + round * sizeof(char), &mask, sizeof(char));
        round++;
    }
    memcpy(sb->s_bdev + BLK_SIZE + round * sizeof(char), 0, sizeof(char) * (sb->s_bitmap_blks
- number_of_chars) * BLK_SIZE);
    return sb;
}

struct super_block *free_block(struct super_block *sb) {
    if (sb) {
        if (sb->s_bdev)
            free(sb->s_bdev);
//        if (sb->s_inodes)
//            inode_list_free(sb->s_inodes);
    }
    return NULL;
}

bool save_block(struct super_block *sb) {
    FILE *fp = fopen(save_path, "wb");
    if (!fp)
        return false;
    fwrite(sb->s_bdev, sb->s_blocknumbers * sb->s_blocksize, 1, fp);
    fclose(fp);
    return true;
}

struct super_block *load_block(struct dentry *root) {
    FILE *fp = fopen(save_path, "rb");
    if (!fp)
        return NULL;
    struct super_block *sb = (struct super_block *)malloc(sizeof(struct super_block));
    sb->s_bdev = (char *)malloc(sizeof(char) * BLK_NUM * BLK_SIZE);
    fread(sb->s_bdev, BLK_NUM * BLK_SIZE, 1, fp);
    fclose(fp);
    unsigned long offset = 0;
    memcpy(&sb->s_blocknumbers, sb->s_bdev + offset, sizeof(unsigned long));
    offset += sizeof(unsigned long);
    memcpy(&sb->s_blocksize, sb->s_bdev + offset, sizeof(unsigned long));
    offset += sizeof(unsigned long);
    memcpy(&sb->s_bitmap_blks, sb->s_bdev + offset, sizeof(int));
    offset += sizeof(int);
```

```
//    memcpy(&sb->s_inodes_num, sb->s_bdev + offset, sizeof(unsigned long));
//    offset += sizeof(unsigned long);

//    sb->s_inodes = NULL;

    root->parent = NULL;
    root->type = __directory;
    root->d_sb = sb;
    root->d_inode = alloc_inode(sb);
    for (int i = 0; i < HASH_TABLE_ROW; i++) {
        root->subdirs[i] = (struct dir_hash_table *)malloc(sizeof(struct dir_hash_table));
        root->subdirs[i]->dname = NULL;
        root->subdirs[i]->next = NULL;
        root->subdirs[i]->corres_dentry = NULL;
    }
    struct dentry *root_self = (struct dentry *)malloc(sizeof(struct dentry));
    root_self->d_sb = sb;
    root_self->parent = NULL;
    root_self->d_time = root->d_time;
    root_self->type = __link;
    strcpy(root_self->d_iname, ".");
    root_self->d_inode = root->d_inode;
    root->d_inode->i_nlink++;
    hash_table_insert(root, root_self);

    return sb;
}

struct inode *alloc_inode(struct super_block *sb) {
    bool found = false;
    struct inode *new_inode = (struct inode *)malloc(sizeof(struct inode));
    new_inode->i_blocks = 0;
    new_inode->i_sb = sb;
    new_inode->i_nlink = 0;
    new_inode->i_btyes = 0;
    new_inode->i_blocks = 0;
    new_inode->i_list = NULL;
    for (unsigned long j = 0; j < sb->s_blocknumbers; j++) {
        if (test_block_free_by_inode_num(j, sb) == true) {
            new_inode->i_no = j;
            found = true;
            break;
        }
    }
```

```
    if (found == false){
        free(new_inode);
        new_inode = NULL;
        return NULL;
    }

    return new_inode;
}

bool occupy_block(blkcnt_t blk_no, struct super_block *sb) {
    unsigned long begin_pos = sb->s_blocksize;
    unsigned long char_pos = (blk_no) / 8 + (sb->s_bitmap_blks + 4) / 8 + begin_pos;
    int char_bit = (int)blk_no % 8 ;
//    printf("%x\n", ((char *)sb->s_bdev)[char_pos]);
    char *test_char = (char *)sb->s_bdev + char_pos;
    if (test_bit_char(*test_char, 7 - char_bit) == true)
        return false;
    char result = fill_bit_char(*test_char, 7 - char_bit);
    ((char *)sb->s_bdev)[char_pos] = result;
//    printf("%x\n", ((char *)sb->s_bdev)[char_pos]);
    return true;
}

bool release_block(blkcnt_t blk_no, struct super_block *sb) {
    unsigned long begin_pos = sb->s_blocksize;
    unsigned long char_pos = (blk_no) / 8 + (sb->s_bitmap_blks + 4) / 8 + begin_pos;
    int char_bit = (int)blk_no % 8 ;
//    printf("%x\n", ((char *)sb->s_bdev)[char_pos]);
    char *test_char = (char *)sb->s_bdev + char_pos;
    if (test_bit_char(*test_char, 7 - char_bit) == false)
        return false;
    char result = release_bit_char(*test_char, sizeof(char) - 1 - char_bit);
    ((char *)sb->s_bdev)[char_pos] = result;
//    printf("%x\n", ((char *)sb->s_bdev)[char_pos]);
    return true;
}

bool alloc_block_for_inode(struct super_block *sb, struct inode *inode) {
    if (!inode)
        return false;
    if (inode->i_blocks == 0) {
        inode->i_list = (struct blk_lists *) malloc(sizeof(struct blk_lists));
        inode->i_list->blk_len = 0;
        inode->i_list->next_blk = NULL;
```

```
        inode->i_nlink = 0;
        inode->i_btyes = 0;
        inode->i_sb = sb;
    }
    unsigned long blk_max_no = sb->s_blocknumbers - sb->s_bitmap_blks;
    unsigned long blk_char_pos = sb->s_blocksize * (sb->s_bitmap_blks + 1);
    for (unsigned long i = 0; i < blk_max_no; i++) {
        if (occupy_block(i, sb)) {
            blk_char_pos = blk_char_pos + i * sb->s_blocksize;
            inode->i_no = i;
            struct blk_lists *ptr = inode->i_list;
            while (ptr->next_blk) {
                ptr = ptr->next_blk;
            }
            ptr->next_blk = sb->s_bdev + blk_char_pos;
            ptr->next_blk->next_blk = NULL;
            inode->i_blocks++;
            return true;
        }
    }
    return false;
}

void *free_block_for_inode(struct super_block *sb, struct inode *inode) {
    if (!inode)
        return NULL;
    unsigned long num = inode->i_blocks;
    if (!inode->i_list) {
        printf("Critical error, the memory be freed is not allocated!\n");
        return NULL;
    }
    struct blk_lists *ptr = inode->i_list, *prev;
    ptr = ptr->next_blk;
    unsigned long blk_max_no = sb->s_blocknumbers - sb->s_bitmap_blks;
    unsigned long blk_char_pos = sb->s_blocksize * (sb->s_bitmap_blks + 1);
    while(ptr) {
        prev = ptr;
        release_block(ptr->blk_no, sb);
        free(prev);
        ptr = ptr->next_blk;
    }
    release_block(inode->i_no, sb);
    inode->i_list = NULL;
    inode->i_sb = NULL;
```

```
    free(inode);
    inode = NULL;
    return NULL;
}

unsigned long alloc_data_block(struct super_block *sb) {
    unsigned long blk_max_no = sb->s_blocknumbers - sb->s_bitmap_blks - 1;
    unsigned long blk_char_pos = sb->s_blocksize * (sb->s_bitmap_blks + 1);
    for (unsigned long i = blk_max_no - 1; i >= 0; i++) {
        if (test_block_free_by_inode_num(i, sb)) {
            blk_char_pos = blk_char_pos + i * sb->s_blocksize;
            return i;
        }
    }
    return 0;
}

void free_data_block(unsigned long blk_no, struct super_block *sb) {
    if (test_block_free_by_inode_num(blk_no, sb)) {
        printf("critical error! free_data_block()\n");
        exit(1);
    }
    else {
        release_block(blk_no, sb);
    }
}

size_t write_block(struct super_block *sb, unsigned long block_no, char *stream, size_t size)
{
    unsigned long blk_char_pos = sb->s_blocksize * (sb->s_bitmap_blks + 1);

    if (test_block_free_by_inode_num(block_no, sb)) {
        occupy_block(block_no, sb);
    }
    blk_char_pos = blk_char_pos + block_no * sb->s_blocksize;
    memcpy(sb->s_bdev + blk_char_pos, stream, size);
    return size;
}

size_t read_block(struct super_block *sb, unsigned long block_no, char *stream, size_t size) {
    unsigned long blk_char_pos = sb->s_blocksize * (sb->s_bitmap_blks + 1);

    if (test_block_free_by_inode_num(block_no, sb)) {
        printf("use a free block!\n");
```

```
        exit(1);
    }
    blk_char_pos = blk_char_pos + block_no * sb->s_blocksize;
    memcpy(stream, sb->s_bdev + blk_char_pos, size);
    return size;
}
```

i 节点函数定义 inode.h

```
//
// Created by 永鑫   徐 on 2019-02-25.
//

#ifndef VFS_INODE_H
#define VFS_INODE_H

#include <stdlib.h>
#include "../include/data_structures.h"

struct inode_linked_list {
    struct inode *inode;
    struct inode_linked_list *next;
};

struct inode_linked_list *init_inode_list(size_t num, struct super_block *sb);
struct inode_linked_list *inode_list_free(struct inode_linked_list *ilist);
void get_inode_memory_by_num(struct inode *inode, struct super_block *sb);
void put_inode_memory_by_num(struct inode *inode, struct super_block *sb);
bool test_block_free_by_inode_num(unsigned long inode_num, struct super_block *sb);
void fill_block_by_inode_num(unsigned long inode_num, struct super_block *sb);

#endif //VFS_INODE_H
```

i 节点实现 inode.c

```
//
// Created by 永鑫   徐 on 2019-02-25.
//

#include "../include/inode.h"

struct inode_linked_list *init_inode_list(size_t num, struct super_block *sb) {
    struct inode_linked_list *head = (struct inode_linked_list *)malloc(sizeof(struct
inode_linked_list));
    head->next = NULL;
    head->inode = NULL;
```

```
    struct inode_linked_list *ptr = head;
    for (size_t i = 0; i < num; i++) {
        ptr->next = (struct inode_linked_list *)malloc(sizeof(struct inode_linked_list));
        ptr = ptr->next;
        ptr->next = NULL;
        ptr->inode = (struct inode *)malloc(sizeof(struct inode));
        ptr->inode->i_no = i;
        ptr->inode->i_sb = sb;
        ptr->inode->i_btyes = 0;
        ptr->inode->i_blocks = 0;
    }
    return head;
}

struct inode_linked_list *inode_list_free(struct inode_linked_list *ilist) {
    if (!ilist)
        return NULL;
    while (ilist) {
        struct inode_linked_list *temp = ilist;
        ilist = ilist->next;
        free(temp->inode);
        free(temp);
    }
    return NULL;
}

void get_inode_memory_by_num(struct inode *inode, struct super_block *sb) {
    unsigned long ino = inode->i_no;
    unsigned long begin_pos = (sb->s_bitmap_blks + 1) * sb->s_blocksize + ino * (sizeof(struct
inode));
    unsigned long offset = 0;
    memcpy(&inode->i_uid, sb->s_bdev + begin_pos + offset, sizeof(unsigned long));
    offset += sizeof(unsigned long);
    memcpy(&inode->i_no, sb->s_bdev + begin_pos + offset, sizeof(unsigned long));
    offset += sizeof(unsigned long);
    memcpy(&inode->i_btyes, sb->s_bdev + begin_pos + offset, sizeof(unsigned long));
    offset += sizeof(unsigned long);
    memcpy(&inode->mode, sb->s_bdev + begin_pos + offset, sizeof(unsigned long));
    offset += sizeof(unsigned long);
    memcpy(&inode->i_blocks, sb->s_bdev + begin_pos + offset, sizeof(unsigned long));
    offset += sizeof(unsigned long);

}
```

```
void put_inode_memory_by_num(struct inode *inode, struct super_block *sb) {
    unsigned long ino = inode->i_no;
    unsigned long begin_pos = (sb->s_bitmap_blks + 1) * sb->s_blocksize + ino * (sizeof(struct
inode));
    unsigned long offset = 0;
    memcpy(sb->s_bdev + begin_pos + offset, &inode->i_uid, sizeof(unsigned long));
    offset += sizeof(unsigned long);
    memcpy(sb->s_bdev + begin_pos + offset, &inode->i_no, sizeof(unsigned long));
    offset += sizeof(unsigned long);
    memcpy(sb->s_bdev + begin_pos + offset, &inode->i_btyes, sizeof(unsigned long));
    offset += sizeof(unsigned long);
    memcpy(sb->s_bdev + begin_pos + offset, &inode->mode, sizeof(unsigned long));
    offset += sizeof(unsigned long);
    memcpy(sb->s_bdev + begin_pos + offset, &inode->i_blocks, sizeof(unsigned long));
    offset += sizeof(unsigned long);

    fill_block_by_inode_num(inode->i_no, sb);
}

bool test_block_free_by_inode_num(unsigned long inode_num, struct super_block *sb) {
    unsigned long begin_pos = sb->s_blocksize;
    unsigned long char_pos = (inode_num) / 8 + (sb->s_bitmap_blks + 4) / 8 + begin_pos;
    int char_bit = (int)inode_num % 8 ;
//    printf("%x\n", ((char *)sb->s_bdev)[char_pos]);
    char *test_char = (char *)sb->s_bdev + char_pos;
    if (test_bit_char(*test_char, 7 - char_bit) == true)
        return false;   // occupied
    return true;    // free
}

void fill_block_by_inode_num(unsigned long inode_num, struct super_block *sb) {
    unsigned long begin_pos = sb->s_blocksize;
    unsigned long char_pos = (inode_num) / 8 + (sb->s_bitmap_blks + 4) / 8 + begin_pos;
    int char_bit = (int)inode_num % 8 ;
//    printf("%x\n", ((char *)sb->s_bdev)[char_pos]);
    char *test_char = (char *)sb->s_bdev + char_pos;
    char result = fill_bit_char(*test_char, 7 - char_bit);
    ((char *)sb->s_bdev)[char_pos] = result;
}
```

目录项定义 dentry.h

```
//
// Created by 永鑫   徐 on 2019-02-25.
//
```

```c
#ifndef VFS_DENTRY_H
#define VFS_DENTRY_H

typedef void * qtype;
#include "data_structures.h"
#include "inode.h"

typedef struct _queue_node {
    qtype val;               // value of the node
    int aux;                 // auxiliary info
    struct _queue_node *next;   // next the node
}qnode;

typedef struct _simple_queue {
    size_t size;    // number of nodes of the queue
    qnode *front;   // front of the queue
    qnode *rear;    // rear of the queue
}queue;

struct __dentry_with_level {
    struct dentry *dir;
    int level;
};

bool save_entry(struct dentry *root, struct super_block *sb);
bool load_entry(struct dentry *root, struct super_block *sb);
struct dir_hash_table *d_hash(struct dentry *dentry);
void hash_table_insert(struct dentry *dentry, struct dentry *sub_dentry);
void hash_table_delete(struct dentry *dentry, struct dentry *sub_dentry);
struct dentry *search_by_str(struct dentry *current_dentry, const char *dname);
void packet_user_dlist(dlist *dl, struct user_linked_list *head, struct dentry *root);
struct dentry *change_home_dir(dlist *dl, struct usr_ptr *current_user);
void get_current_path_str(struct dentry *dentry, char *str);
void travel_dentry(queue *q, slist *str_list, struct dentry *begin_dentry, size_t num);

// initialize queue, allocate memory, return the initialized queue pointer
queue *init_queue(void);
// destroy the whole queue, free memory allocated, return null pointer
queue *free_queue(queue **q);
// return value of the front of the queue, return NULL if val is null or queue is empty
qtype get_front_queue(const queue *q);
// return value of the rear of the queue, return NULL if val is null or queue is empty
qtype get_rear_queue(const queue *q);
```

```
// return true if the queue is empty, false otherwise
bool queue_empty(const queue *q);
// enqueue a item, return false if overflow
bool enqueue(qtype item, queue *q);
// dequeue the front of the queue, return NULL if the queue is empty or val is null
qtype dequeue(queue *q);
// dequeue the rear of the queue, return NULL if the queue is empty or val is null
qtype dequeue_rear(queue *q);
// return the size of the queue
size_t get_size_of_queue(const queue *q);
// print_queue, used for debugging
void print_queue(const queue *q);

/*
 * resolve path into a queue
 * example: resolve "../user1/project/foo/main.c"
 * into: ".." --> "user1" --> "project" --> "foo" --> "main.c"
 * where ".." is the front of the queue, and "main.c" is the rear
 */

// resolve path into a queue, return NULL if the path is invalid
queue *resolve_path_to_queue(const char *path);

dlist *init_dlist(void);
void insert_dlist(dlist *dl, struct dentry *dentry);
void free_dlist(dlist **dl);
void print_dlist(dlist *dl);
struct dentry *look_up_dlist(dlist *dl, const char *name);

#endif //VFS_DENTRY_H
```

目录项实现 dentry.c

```
//
// Created by 永鑫  徐 on 2019-02-25.
//

#include "../include/dentry.h"

const char * const dentry_path = "./data/dentry.txt";

bool save_entry(struct dentry *root, struct super_block *sb) {
    FILE *fp = fopen(dentry_path, "w");
    if (fp == NULL) {
        printf("An error occurs when open %s\n, save entry failed!\n", dentry_path);
```

```
    return false;
}

queue *q = init_queue();
enqueue(root, q);
q->front->aux = 0;
int last_level = 0;

struct dentry *cur_entry = root;

size_t num = 0;

for (int i = 0; i < HASH_TABLE_ROW; i++) {
    struct dir_hash_table *ptr = cur_entry->subdirs[i];
    while (ptr->next) {
        ptr = ptr->next;
        if (!strcmp(ptr->dname, ".") || !strcmp(ptr->dname, ".."))
            continue;
        num++;
    }
}

size_t num_of_first_level = 0;

while (cur_entry != NULL) {
    struct dentry *temp_save = cur_entry;
    bool found = false;
    for (int i = 0; i < HASH_TABLE_ROW; i++) {
        struct dir_hash_table *ptr = cur_entry->subdirs[i];
        while (ptr->next) {
            ptr = ptr->next;
            if (!strcmp(ptr->dname, ".."))
                continue;
            if (!strcmp(ptr->dname, "."))
                continue;
            enqueue(ptr->corres_dentry, q);
            found = true;
            last_level++;
            q->rear->aux = last_level;
            cur_entry = ptr->corres_dentry;
            break;
        }
        if (found == true)
            break;
```

```
        }
        if (found == false) {
            cur_entry = temp_save->parent;
            last_level--;
            hash_table_delete(cur_entry, temp_save);
        }
        if (temp_save == root) {
            num_of_first_level++;
            if (num_of_first_level == num)
                break;
        }
    }

    while (!queue_empty(q)) {
        fprintf(fp, "%d\n%lu %c %lu\n", q->front->aux, ((struct dentry
*)q->front->val)->d_inode->i_no, ((struct dentry *)q->front->val)->type,
                ((struct dentry *)q->front->val)->d_inode->i_uid);

        put_inode_memory_by_num(((struct dentry *)q->front->val)->d_inode, sb);

        fwrite(((struct dentry *)q->front->val)->d_time, sizeof(struct tm), 1, fp);
        fputc('\n', fp);
        fprintf(fp, "%s\n", ((struct dentry *)q->front->val)->d_iname);
        dequeue(q);
    }

    fclose(fp);
    printf("save successfully!\n");
    return true;
}

bool load_entry(struct dentry *root, struct super_block *sb) {
    FILE *fp = fopen(dentry_path, "r");
    if (fp == NULL) {
        printf("An error occurs when open %s\n, load entry failed!\n", dentry_path);
        return false;
    }
    int level = 0;
    if (root->d_inode == NULL) {
        root->d_inode = (struct inode *) malloc(sizeof(struct inode));
    }
    fscanf(fp, "%d\n%lu %c %lu\n", &level, &root->d_inode->i_no, &root->type,
&root->d_inode->i_uid);
```

```
    get_inode_memory_by_num(root->d_inode, sb);

    root->d_time = (struct tm *)malloc(sizeof(struct tm));
    fread(root->d_time, sizeof(struct tm), 1, fp);
    fscanf(fp, "%s\n", root->d_iname);

    queue *q = init_queue();
    enqueue(root, q);
    int last_level = level;
    while (!queue_empty(q)) {
        struct dentry *parent_dir = get_rear_queue(q);
        struct dentry *new_dir = (struct dentry *)malloc(sizeof(struct dentry));
        new_dir->d_inode = (struct inode *)malloc(sizeof(struct inode));
        if (4 != fscanf(fp, "%d\n%lu %c %lu\n", &level, &new_dir->d_inode->i_no, &new_dir->type,
&new_dir->d_inode->i_uid)) {
            free(new_dir);
            break;
        }
        else {
            get_inode_memory_by_num(new_dir->d_inode, sb);
            new_dir->d_time = (struct tm *)malloc(sizeof(struct tm));
            fread(new_dir->d_time, sizeof(struct tm), 1, fp);
            fscanf(fp, "%s\n", new_dir->d_iname);
            while (level != last_level + 1) {
                dequeue_rear(q);
                last_level--;
            }
            parent_dir = q->rear->val;
            last_level = level;
            enqueue(new_dir, q);
            for (int i = 0; i < HASH_TABLE_ROW; i++) {
                new_dir->subdirs[i] = (struct dir_hash_table *)malloc(sizeof(struct
dir_hash_table));
                new_dir->subdirs[i]->dname = NULL;
                new_dir->subdirs[i]->next = NULL;
                new_dir->subdirs[i]->corres_dentry = NULL;
            }

            struct dentry *new_dir_self = (struct dentry *)malloc(sizeof(struct dentry));
            new_dir_self->parent = NULL;
            new_dir_self->d_time = new_dir->d_time;
            new_dir_self->type = __link;
            strcpy(new_dir_self->d_iname, ".");
            new_dir_self->d_inode = new_dir->d_inode;
```

```
            new_dir_self->d_inode->i_nlink++;

            hash_table_insert(new_dir, new_dir_self);

            hash_table_insert(parent_dir, new_dir);
            struct dentry *new_dir_parent = (struct dentry *)malloc(sizeof(struct dentry));
            new_dir_parent->parent = NULL;
            new_dir_parent->d_time = parent_dir->d_time;
            new_dir_parent->type = __link;

            strcpy(new_dir_parent->d_iname, "..");
            new_dir_parent->d_inode = parent_dir->d_inode;
            new_dir_parent->d_inode->i_nlink++;

            hash_table_insert(new_dir, new_dir_parent);
        }
    }
    free_queue(&q);

    fclose(fp);
    return true;
}

struct dir_hash_table *d_hash(struct dentry *dentry) {
    struct dir_hash_table *item = (struct dir_hash_table *)malloc(sizeof(struct dir_hash_table));
    item->corres_dentry = dentry;
    item->next = NULL;
    item->dname = (char *)malloc(sizeof(char) * DNAME_LEN);
    strcpy(item->dname, dentry->d_iname);
    size_t len = strlen(item->dname);
    int hash_val = 0;
    for (int i = 0; i < len; i++) {
        hash_val += item->dname[i];
    }
    item->hash_key = hash_val;

    return item;
}

void hash_table_insert(struct dentry *dentry, struct dentry *sub_dentry) {
    struct dir_hash_table *item = d_hash(sub_dentry);
    struct dir_hash_table *ptr = dentry->subdirs[item->hash_key % HASH_TABLE_ROW];
    sub_dentry->parent = dentry;
```

```
    item->next = ptr->next;
    ptr->next = item;
}

void hash_table_delete(struct dentry *dentry, struct dentry *sub_dentry) {
    int hash_val = 0;
    size_t len = strlen(sub_dentry->d_iname);
    for (int i = 0; i < len; i++) {
        hash_val += sub_dentry->d_iname[i];
    }
    struct dir_hash_table *ptr = dentry->subdirs[hash_val % HASH_TABLE_ROW];
    while (ptr->next) {
        if (!strcmp(ptr->next->dname, sub_dentry->d_iname)) {
            free(ptr->next->dname);
            struct dir_hash_table *temp = ptr->next;
            ptr->next = ptr->next->next;
            free(temp);
            temp = NULL;
            break;
        }
        ptr = ptr->next;
    }
}

struct dentry *search_by_str(struct dentry *current_dentry, const char *dname) {
    struct dentry *dir = NULL;
    int hash_value = 0;
    size_t len = strlen(dname);
    for (int i = 0; i < len; i++) {
        hash_value += dname[i];
    }
    struct dir_hash_table *ptr = current_dentry->subdirs[hash_value % HASH_TABLE_ROW];
    while (ptr->next) {
        ptr = ptr->next;
        if (!strcmp(ptr->dname, dname)) {
            dir = ptr->corres_dentry;
            return dir;
        }
    }
    return NULL;
}

void packet_user_dlist(dlist *dl, struct user_linked_list *head, struct dentry *root) {
    struct user_linked_list *ptr = head;
```

```
    if (head->next) {
        ptr = head->next;
        insert_dlist(dl, root);
    }
    struct dentry *temp = root;
    temp = search_by_str(temp, "home");
    while (ptr->next) {
        ptr = ptr->next;
        temp = search_by_str(temp, ptr->name);
        if (temp)
            insert_dlist(dl, temp);
        else {
            printf("Serious bug! cannot get user %s's home dir\n", ptr->name);
            break;
        }
        temp = temp->parent;
    }
}

struct dentry *change_home_dir(dlist *dl, struct usr_ptr *current_user) {
    return look_up_dlist(dl, current_user->name);
}

void get_current_path_str(struct dentry *dentry, char *str) {
    queue *q = init_queue();
    struct dentry *temp = dentry;
    while (temp != NULL && (0 != strcmp(temp->d_iname, "/")) ) {
        enqueue(temp->d_iname, q);
        temp = temp->parent;
    }
    int i = 0;
    strcpy(str, "/");
    while (!queue_empty(q)) {
        char *tmp_str = dequeue_rear(q);
        strcat(str, tmp_str);
        if (queue_empty(q)) {
            strcat(str, " ");
            break;
        }
        else {
            strcat(str, "/");
        }
    }
    printf("\x1b[34m""%s""\x1b[32m""$""\x1b[0m", str);
```

```
}

queue *init_queue(void) {
    queue *q = (queue *)malloc(sizeof(queue));
    q->front = NULL;
    q->rear = NULL;
    q->size = 0;
    return q;
}

/*
 * destroy the whole queue, free memory allocated
 */
queue *free_queue(queue **q) {
    while ((*q)->front != NULL) {
        qnode *temp = (*q)->front;
        (*q)->front = (*q)->front->next;
        temp = NULL;
    }
    (*q)->front = NULL;
    (*q)->rear = NULL;
    (*q)->size = 0;
    free(*q);
    (*q) = NULL;
    return NULL;
}

/*
 * return the value of the front of the queue, return NULL if val is null or queue is empty
 */
qtype get_front_queue(const queue * q) {
    if (q->front)
        return q->front->val;
    else
        return NULL;
}

/*
 * return the value of the rear of the queue, return NULL if val is null or queue is empty
 */
qtype get_rear_queue(const queue * q) {
    if (q->rear)
        return q->rear->val;
    else
```

```
        return NULL;
}

/*
 * return true if the queue is empty, false otherwise
 */
bool queue_empty(const queue * q) {
    return !(q->size);
}

/*
 * enqueue a item, return false if overflow
 */
bool enqueue(qtype item, queue *q) {
    if(q->front == NULL) {
        qnode *node = (qnode *)malloc(sizeof(qnode));
        if (node == NULL)
            return false;
        node->val = item;
        node->next = NULL;
        q->front = node;
        q->rear = node;
        q->size = 1;
    }
    else {
        qnode *node = (qnode *)malloc(sizeof(qnode));
        if (node == NULL)
            return false;
        q->rear->next = node;
        node->next = NULL;
        node->val = item;
        q->rear = node;
        q->size++;
    }
    return true;
}

/*
 * dequeue the front of the queue, return NULL if the queue is an empty queue
 */
qtype dequeue(queue *q) {
    if (q->size == 0)
        return NULL;
    qtype rtn = q->front->val;
```

```
    if (q->front->next == NULL) {
        q->front = NULL;
        q->rear = NULL;
        q->size = 0;
    }
    else {
        qnode *temp = q->front->next;
        q->front = temp;
        q->size--;
    }
    return rtn;
}

/*
 * dequeue the rear of the queue, return NULL if the queue is an empty queue
 */
qtype dequeue_rear(queue *q) {
    if (q->size == 0)
        return NULL;
    qtype rtn = q->rear->val;
    qnode *t = q->front;
    if (q->rear->val == q->front->val) {
        q->rear = NULL;
        q->front = NULL;
    }
    else {
        while (t->next != q->rear) {
            t = t->next;
        }
        t->next = NULL;
        q->rear = t;
    }
    q->size--;
    return rtn;
}

/*
 * return the size of the queue
 */
size_t get_size_of_queue(const queue * q) {
    return q->size;
}

/*
```

```
 * print all elements in queue
 */
void print_queue(const queue * q) {
    qnode *temp = q->front;
    while (temp) {
        if (temp->next)
            printf("%s --> ", (char *)temp->val);
        else
            printf("%s\n", (char *)temp->val);
        temp = temp->next;
    }
}

queue *resolve_path_to_queue(const char * const path) {
    queue *q = init_queue();
    int left_slash_pos = -1;
    for (int i = 0; i < strlen(path); i++) {
        if ( i == 0 && path[i] == '/' ) {
            if (path[1] == '/') {
                // invalid path
                free_queue(&q);
                return NULL;
            }
            enqueue("/", q);
            left_slash_pos = 0;
        }
        else if (i == strlen(path) - 1 && path[i] != '/') {
            i = i + 1;
            char *temp = (char *)malloc(sizeof(char) * (i - left_slash_pos));
            if (temp == NULL) {
                free_queue(&q);
                return NULL;
            }
            for (int j = 0; j < i - left_slash_pos - 1; j++)
                temp[j] = path[left_slash_pos + 1 + j];
            temp[i - left_slash_pos - 1] = '\0';
            enqueue(temp, q);
        }
        else if (path[i] != '/')
            continue;
        else {
            // path[i] == '/'
            if (path[i + 1] == '/') {
                // invalid path
```

```
                free_queue(&q);
                return NULL;
            }
            else {
                char *temp = (char *)malloc(sizeof(char) * (i - left_slash_pos));
                if (temp == NULL) {
                    free_queue(&q);
                    return NULL;
                }
                for (int j = 0; j < i - left_slash_pos - 1; j++)
                    temp[j] = path[left_slash_pos + 1 + j];
                temp[i - left_slash_pos - 1] = '\0';
                enqueue(temp, q);
                left_slash_pos = i;
            }
        }
    }
    print_queue(q);
    return q;
}

dlist *init_dlist(void) {
    dlist *dl = (dlist *)malloc(sizeof(dlist));
    dl->next = NULL;
    dl->dir = NULL;
    return dl;
}

void insert_dlist(dlist *dl, struct dentry *dentry) {
    dlist *temp = dl;
    while(temp->next) {
        temp = temp->next;
    }
    temp->next = (dlist *)malloc(sizeof(dlist));
    temp = temp->next;
    temp->dir = dentry;
    temp->next = NULL;
}

void free_dlist(dlist **dl) {
    dlist *tmp = *dl;
    dlist *prev = *dl;

    while (tmp) {
```

```
        prev = tmp;
        tmp = tmp->next;
        free(prev);
    }
    tmp = NULL;
    *dl = NULL;
}

void print_dlist(dlist *dl) {
    dlist *temp = dl;
    while(temp->next) {
        temp = temp->next;
        printf("%s\t", temp->dir->d_iname);
    }
    putchar('\n');
}

struct dentry *look_up_dlist(dlist *dl, const char *name) {
    struct dentry *rtn = NULL;
    dlist *temp = dl;
    while(temp->next) {
        temp = temp->next;
        if (!strcmp(name, temp->dir->d_iname) || (!strcmp(name, "root")
&& !strcmp(temp->dir->d_iname, "/"))) {
            rtn = temp->dir;
            break;
        }
    }
    return rtn;
}
```

辅助函数定义 trival_helper.h

```
//
// Created by 永鑫  徐 on 2019-02-27.
//

#ifndef VFS_TRIVAL_HELPER_H
#define VFS_TRIVAL_HELPER_H

#include <time.h>
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
#include "data_structures.h"
```

```
struct tm* get_local_time(void);
void print_time(struct tm* lt);
bool test_bit_char(const char byte, const int num);
char fill_bit_char(char byte, int num);
char release_bit_char(char byte, int num);
unsigned long priority_get_by_number(int num);

#endif //VFS_TRIVAL_HELPER_H
```

辅助函数实现 trival_helper.h

```
//
// Created by 永鑫  徐 on 2019-02-27.
//

#include "../include/trival_helper.h"

const char char_mask = 0x01;

inline bool test_bit_char(const char byte, const int num) {
    if (num > (sizeof(char) * 8 - 1) || num < 0)
        return false;
    char test = char_mask << num;
    char test2 = byte & test;
    return ((test2 >> num) & char_mask);
}

inline char fill_bit_char(char byte, int num) {
    if (num > (sizeof(char) * 8 - 1) || num < 0)
        return 0;
    char rtn = byte | (char_mask << num);
    return rtn;
}

inline char release_bit_char(char byte, int num) {
    if (num > (sizeof(char) * 8 - 1) || num < 0)
        return 0;
    char rtn = byte & ~(char_mask << num);
    return rtn;
}

struct tm* get_local_time(void) {
    time_t t;
    struct tm * lt = (struct tm *)malloc(sizeof(struct tm));
```

```
    time (&t);
    lt = localtime (&t);
    return lt;
}

void print_time(struct tm* lt) {
    printf ( "%d/%d/%d %d:%d:%d",lt->tm_year+1900, lt->tm_mon, lt->tm_mday, lt->tm_hour,
lt->tm_min, lt->tm_sec);
}

unsigned long priority_get_by_number(int num) {
    unsigned long rtn = 0;
    int temp = num;
    int residue;
    unsigned int decimal = 0;
    while (temp > 0) {
        residue = temp % 10;
        rtn =  residue << (3 * decimal) | rtn;
        decimal++;
        temp = temp / 10;
    }
    return rtn;
}
```

用户相关函数定义 user.h

```
//
// Created by 永鑫   徐 on 2019-02-25.
//

#ifndef VFS_USER_H
#define VFS_USER_H

#include <stdlib.h>
#include <stdbool.h>
#include <stdio.h>
#include <string.h>

#define USER_NAME_MAX_LEN 50
#define USER_PSW_MAX_LEN 50

struct usr_ptr {
    unsigned int u_id;
    char *name;
    unsigned int priority;
```

```
};

struct user_linked_list {
    unsigned int u_id;
    char *name;
    char *psw;
    unsigned int priority;
    struct user_linked_list *next;
};

struct user_linked_list *create_usr(struct user_linked_list *usr_list_head, char *name, char
*psw, unsigned int priority);
bool change_pswd(struct user_linked_list *usr_list_head, struct user_linked_list *usr, char
*new_psw);
bool change_user_name(struct user_linked_list *usr_list_head, struct user_linked_list *usr, char
*new_name);
struct user_linked_list *load_users_info(void);
bool save_users_info(struct user_linked_list *usr_list_head);
void print_user_info(struct user_linked_list *usr_list_head);
void free_user_info(struct user_linked_list *usr_list_head);

bool get_user_by_name(struct user_linked_list *user_list, struct usr_ptr *current_user, const
char *name);
void print_current_user_info(struct usr_ptr *info);

char *get_user_by_user_id(struct user_linked_list *user_list, unsigned long user_id);

#endif //VFS_USER_H
```

用户相关函数实现 uesr.c

```
//
// Created by 永鑫  徐 on 2019-02-25.
//

#include "../include/user.h"

const char * const filename = "./data/user_info.txt";

struct user_linked_list *create_usr(struct user_linked_list *usr_list_head, char *name, char
*psw, unsigned int priority) {
    if (!usr_list_head) {
        usr_list_head = (struct user_linked_list *) malloc(sizeof(struct user_linked_list));
        usr_list_head->priority = 0;
        usr_list_head->psw = NULL;
```

```
        usr_list_head->name = NULL;
        usr_list_head->next = NULL;
    }
    struct user_linked_list *ptr = usr_list_head;
    while (ptr->next)
        ptr = ptr->next;
    ptr->next = (struct user_linked_list *)malloc(sizeof(struct user_linked_list));
    ptr = ptr->next;
    char *uname = (char *)malloc(sizeof(char) * USER_NAME_MAX_LEN);
    char *upsw = (char *)malloc(sizeof(char) * USER_PSW_MAX_LEN);
    strcpy(uname, name);
    strcpy(upsw, psw);
    ptr->next = NULL;
    ptr->name = uname;
    ptr->psw = upsw;
    ptr->priority = priority;
    ptr->u_id = (unsigned)usr_list_head->priority;
    usr_list_head->priority++;
    return usr_list_head;
}

bool change_pswd(struct user_linked_list *usr_list_head, struct user_linked_list *usr, char
*new_psw) {
    if (!usr_list_head)
        return false;
    struct user_linked_list *ptr = usr_list_head;
    while (ptr->next) {
        ptr = ptr->next;
        if (!strcmp(usr->name, ptr->name)) {
            free(ptr->psw);
            char *upsw = (char *)malloc(sizeof(char) * USER_PSW_MAX_LEN);
            strcpy(upsw, new_psw);
            ptr->psw = upsw;
            return true;
        }
    }
    return false;
}

bool change_user_name(struct user_linked_list *usr_list_head, struct user_linked_list *usr, char
*new_name) {
    if (!usr_list_head)
        return false;
    struct user_linked_list *ptr = usr_list_head;
```

```
    while (ptr->next) {
        ptr = ptr->next;
        if (!strcmp(usr->psw, ptr->psw)) {
            free(ptr->name);
            char *uname = (char *)malloc(sizeof(char) * USER_PSW_MAX_LEN);
            strcpy(uname, new_name);
            ptr->name = uname;
            return true;
        }
    }
    return false;
}

struct user_linked_list *load_users_info(void) {
    FILE *fp = fopen(filename, "r");
    if (!fp) {
        system("pwd");
        printf("failed!\n");
        return false;
    }
    struct user_linked_list *ptr = (struct user_linked_list *)malloc(sizeof(struct
user_linked_list));
    struct user_linked_list *head = ptr;
    if (!ptr)
        return false;
    ptr->name = NULL;
    ptr->psw = NULL;
    ptr->priority = 0;
    char u_id[10], u_name[USER_NAME_MAX_LEN], u_psw[USER_PSW_MAX_LEN], priority[10];
    while (4 == fscanf(fp, "%s%s%s%s\n", u_id, u_name, u_psw, priority)) {
        ptr->next = (struct user_linked_list *)malloc(sizeof(struct user_linked_list));
        ptr = ptr->next;
        ptr->next = NULL;
        sscanf(u_id, "%u", &ptr->u_id);
        char *uname = (char *)malloc(sizeof(char) * USER_NAME_MAX_LEN);
        char *upsw = (char *)malloc(sizeof(char) * USER_PSW_MAX_LEN);
        strcpy(uname, u_name);
        strcpy(upsw, u_psw);
        ptr->name = uname;
        ptr->psw = upsw;
        sscanf(priority, "%u", &ptr->priority);
    }
    return head;
}
```

```
bool save_users_info(struct user_linked_list *usr_list_head) {
    struct user_linked_list *ptr = usr_list_head;
    if (!ptr)
        return false;
    FILE *fp = fopen(filename, "w");
    if (!fp)
        return false;
    while (ptr->next) {
        ptr = ptr->next;
        fprintf(fp, "%u\t%s\t%s\t%d\n", ptr->u_id, ptr->name, ptr->psw, ptr->priority);
    }
    fclose(fp);
    return true;
}

void print_user_info(struct user_linked_list *usr_list_head) {
    struct user_linked_list *ptr = usr_list_head;
    if (!ptr)
        return;
    while (ptr->next) {
        ptr = ptr->next;
        printf("uid: %u\tname: %s\t\tpsw: %s\t\tpriority: %d\n", ptr->u_id, ptr->name, ptr->psw,
ptr->priority);
    }
}

void free_user_info(struct user_linked_list *usr_list_head) {
    struct user_linked_list *ptr = usr_list_head;
    if (!ptr)
        return ;
    while (ptr->next) {
        ptr = ptr->next;
        if (usr_list_head->name)
           free(usr_list_head->name);
        if (usr_list_head->psw)
            free(usr_list_head->psw);
        free(usr_list_head);
        usr_list_head = ptr;
    }
}

bool get_user_by_name(struct user_linked_list *user_list, struct usr_ptr *current_user, const
char *name) {
```

```
    struct user_linked_list *ptr = user_list;
    bool found = false;
    char psw[USER_PSW_MAX_LEN];
    while (ptr) {
        ptr = ptr->next;
        if (ptr) {
            if (!strcmp(ptr->name, name)) {
                if (ptr->priority < 2) {
                    printf("input password of %s\n", ptr->name);
                    fgets(psw, USER_PSW_MAX_LEN, stdin);
                    psw[strlen(psw) - 1] = '\0';
                    if (!strcmp(psw, ptr->psw)) {
                        current_user->name = ptr->name;
                        current_user->priority = ptr->priority;
                        current_user->u_id = ptr->u_id;
                        found = true;
                        break;
                    }
                    else {
                        printf("%s\n", "sorry, try again!");
                        return found;
                    }
                }
                else {
                    current_user->name = ptr->name;
                    current_user->priority = ptr->priority;
                    current_user->u_id = ptr->u_id;
                    found = true;
                    break;
                }
            }
        }
    }
    if (!found)
        printf("soort, didn't found user: %s\n", name);
    return found;
}

char *get_user_by_user_id(struct user_linked_list *user_list, unsigned long user_id) {
    struct user_linked_list *ptr = user_list;
    bool found = false;
    while (ptr) {
        ptr = ptr->next;
        if (ptr) {
```

```
            if (ptr->u_id == user_id) {
                found = true;
                break;
            }
        }
    }
    if (!found)
        return NULL;
    else {
        char *usr_name = (char *)malloc(sizeof(char) * 100);
        strcpy(usr_name, ptr->name);
        return usr_name;
    }
}

void print_current_user_info(struct usr_ptr *info) {
    if (info) {
        printf("%d\t%s\t%d" ,info->u_id, info->name, info->priority);
    }
}
```

文件系统 CMakeLists.txt

```
cmake_minimum_required(VERSION 3.12)
project(vfs C)

set(CMAKE_C_STANDARD 11)

add_definitions(-std=c++11)

include_directories(${PROJECT_SOURCE_DIR}/include)

aux_source_directory(${PROJECT_SOURCE_DIR}/src SRC_LIST)

add_executable(vfs vfs.c ${SRC_LIST})
```