華中科技大學

嵌入式操作系统课程设计报告

姓　　名：　　潘翔

学　　院：　　计算机科学与技术学院

专　　业：　　物联网工程

班　　级：　　IOT1601

学　　号：　　U201614898

指导教师：　　石柯

分数	
教师签名	

2018 年 9 月 15 日

目 录

1 设计目的

1. 掌握 Linux 操作系统的使用方法
2. 了解 Linux 系统内核代码结构
3. 掌握实例操作系统的实现方法

2 文件拷贝

2.1 设计目的

熟悉和理解 Linux 编程环境

2.2 实验要求和内容

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

2.3 环境及步骤

2.3.1 开发环境

1) 操作系统：Arch Linux x64
2) 内核版本：4.18.5-arch1-1-ARCH
3) 编译工具：gcc (GCC) 8.2.0

2.3.2 开发步骤

1) 解析调用参数，判断 copy 类型，支持递归，软链接，硬链接
2) 检查参数数量是否正确
3) 解析参数，获取源文件地址和目标文件地址
4) 检查是否支持 copy 类型
5) 进行 copy 操作

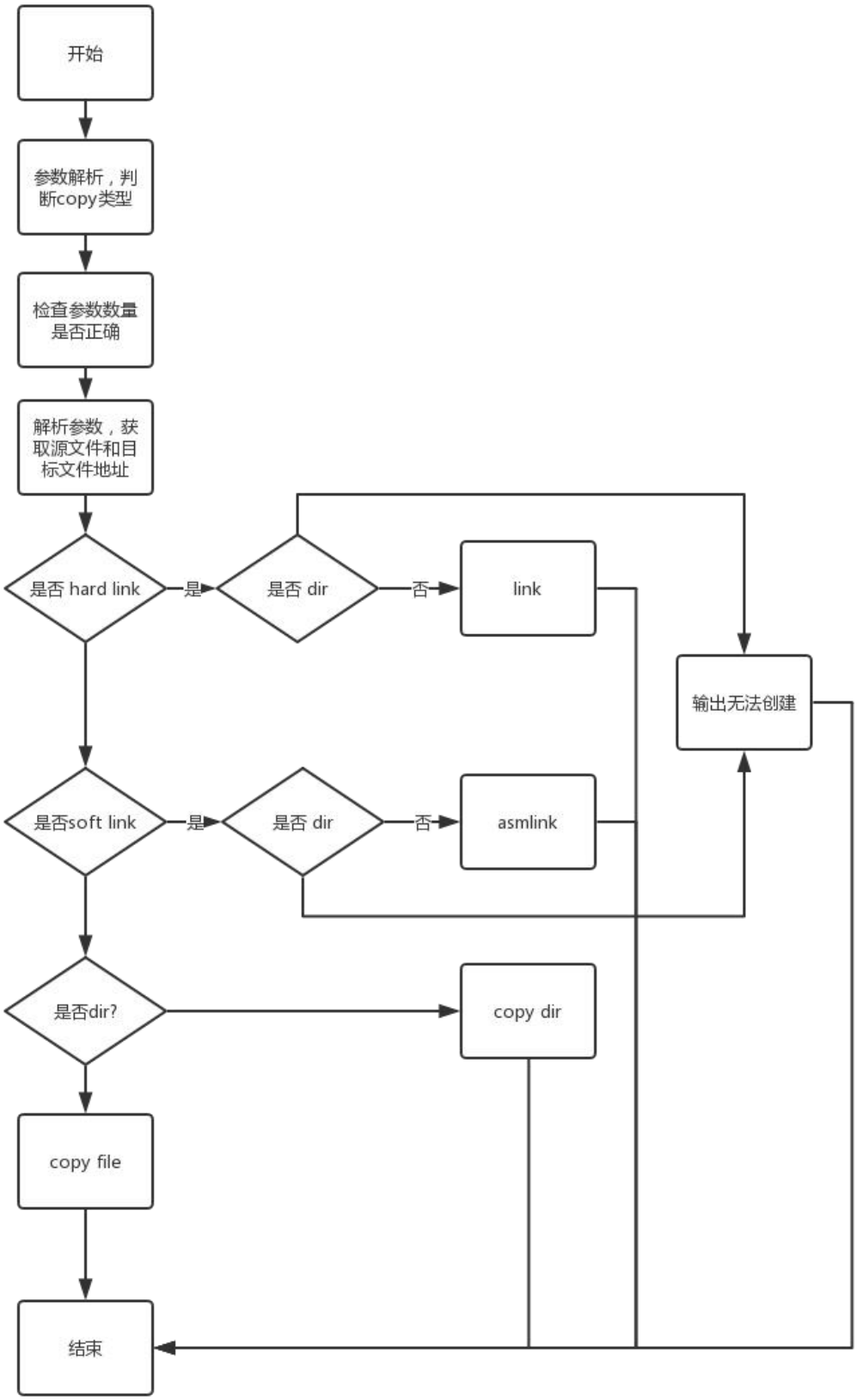


图 2-1 copy 函数流程图

2.4 关键代码

2.4.1 copyF2F

1) 函数原型

int copyF2F(char *src_file, char *dest_file)

2) 函数流程

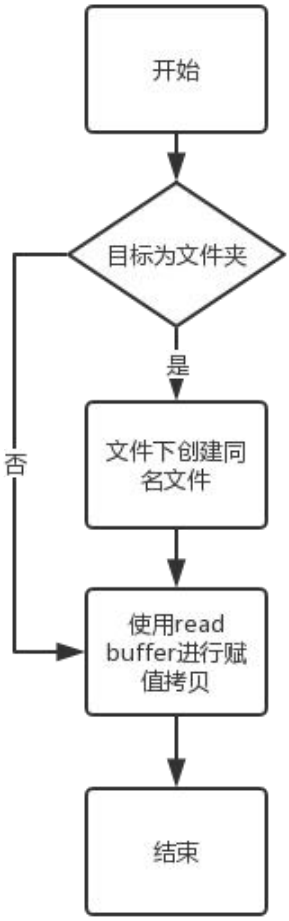


图 2-2 copyF2F 函数流程图

3) 关键代码

```
while ((n_chars = read(in_fd, buf, BUFFERSIZE)) > 0)
    {
        if (write(out_fd, buf, n_chars) != n_chars)
        {
            printf("%s write file fail！", dest_file);
            return 1;
        }
        if (n_chars == -1)
```

```
        {
            printf("%s read file fail！", src_file);
            return 1;
        }
    }
```

2.4.2 copyD2D

1) 函数原型

int copyD2D(char *src_dir, char *dest_dir)函数流程

2) 函数流程

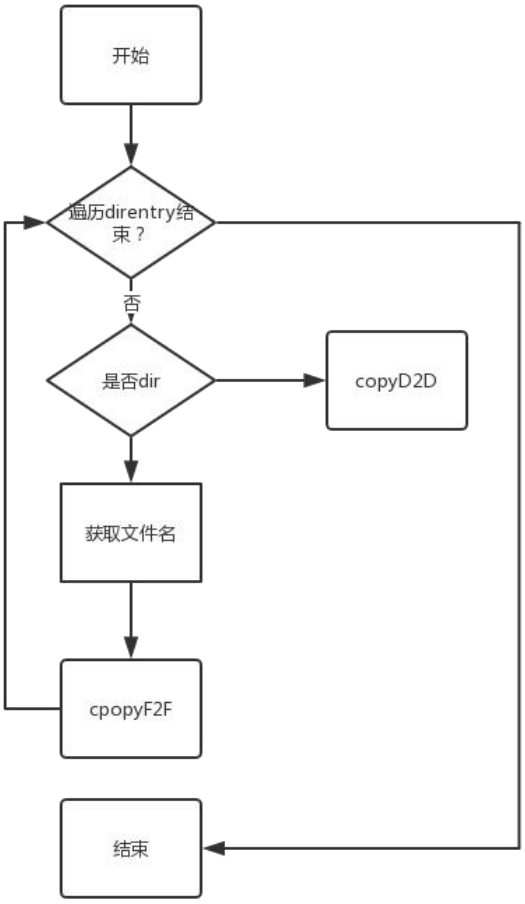


图 2-3 copyD2D 函数流程图

3) 关键代码

```
//open dir
    if ((dp = opendir(src_dir)) == NULL)
        return 1;
    else
    {
```

```
        //get dirent
        while ((dirp = readdir(dp)))
        {
            struct stat file_stat;
            if (!isdir(dirp->d_name))
            {
                //link name
                strcat(tempDest, dirp->d_name);
                strcat(tempSrc, dirp->d_name);

                //copy file
                copyF2F(tempSrc, tempDest);

                //recover name
                strcpy(tempDest, dest_dir);
                strcpy(tempSrc, src_dir);
            }
        }
        //close dir
        closedir(dp);
        return 0;
    }
```

2.5 调试记录及运行结果

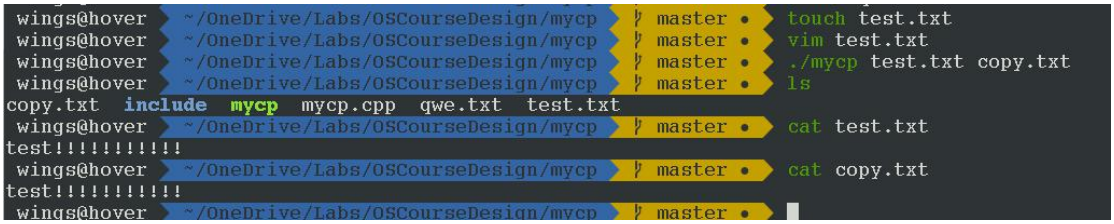


图 2-4 copyF2F Test

图 2-5 copyD2D Test

```
 wings@hover  ~/OneDrive/Labs/OSCourseDesign/mycp  master ●  ./mycp -l test.txt link.txt
 wings@hover  ~/OneDrive/Labs/OSCourseDesign/mycp  master ●  ls
copy_dir  copy.txt  include_copy  link.txt  mycp  mycp.cpp  test_dir  test.txt
 wings@hover  ~/OneDrive/Labs/OSCourseDesign/mycp  master ●  ls -al
total 60
drwxr-xr-x  5 wings wings  4096 Sep 11 23:52 .
drwxr-xr-x 13 wings wings  4096 Sep 11 22:50 ..
drwxr-xr-x  2 wings wings  4096 Sep 11 23:50 copy_dir
-rw-r--r--  1 wings wings    16 Sep 11 23:41 copy.txt
drwxr-xr-x  2 wings wings  4096 Sep 11 23:44 include_copy
-rw-r--r--  2 wings wings    16 Sep 11 23:41 link.txt
-rwxr-xr-x  1 wings wings 17648 Sep 11 23:51 mycp
-rw-r--r--  1 wings wings  6783 Sep 11 23:51 mycp.cpp
drwxr-xr-x  2 wings wings  4096 Sep 11 23:52 test_dir
-rw-r--r--  2 wings wings    16 Sep 11 23:41 test.txt
```

图 2-6 link Test

```
 wings@hover  ~/OneDrive/Labs/OSCourseDesign/mycp  master ●  ./mycp -s test.txt slink.txt
 wings@hover  ~/OneDrive/Labs/OSCourseDesign/mycp  master ●  ls -al
total 60
drwxr-xr-x  5 wings wings  4096 Sep 11 23:53 .
drwxr-xr-x 13 wings wings  4096 Sep 11 22:50 ..
drwxr-xr-x  2 wings wings  4096 Sep 11 23:50 copy_dir
-rw-r--r--  1 wings wings    16 Sep 11 23:41 copy.txt
drwxr-xr-x  2 wings wings  4096 Sep 11 23:44 include_copy
-rw-r--r--  2 wings wings    16 Sep 11 23:41 link.txt
-rwxr-xr-x  1 wings wings 17648 Sep 11 23:51 mycp
-rw-r--r--  1 wings wings  6783 Sep 11 23:51 mycp.cpp
lrwxrwxrwx  1 wings wings     8 Sep 11 23:53 slink.txt -> test.txt
drwxr-xr-x  2 wings wings  4096 Sep 11 23:52 test_dir
-rw-r--r--  2 wings wings    16 Sep 11 23:41 test.txt
```

图 2-7 syblink Test

2.6 设计感想

参照 linux cp 进行设计，其中 D2D 进行递归操作，类似于 ls，需要注意栈空间的占用，设计过程中，学习不同 Linux 不同的文件格式，以及了解软链接和硬链接背后的实现过程，为后面文件系统做准备。

Linux 的硬链接不允许为目录建立硬链接，但是链接可以存在递归，而软链接可以跨越文件系统，故提供了灵活的“拷贝”机制。

3 图形化进程并发

3.1 实验要求和内容

要求：熟悉和理解 Linux 编程环境

内容：

编写一个C程序，使用图形编程库 (QT/GTK)分窗口显示三个并发进程的运行(一个窗口实时显示当前系统时间，一个窗口循环显示 0 到 9，一个窗口做 1 到 1000 的累加求和，刷新周期均为 1 秒)。

3.2 实验过程与结果

1) 利用 fork 进行进程并发
2) 利用 QTimer 进行定时，同事利用信号槽机制进行窗口间的状态同步

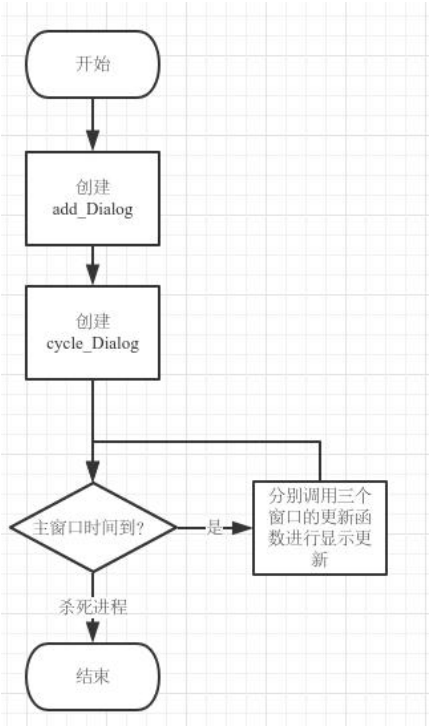


图 3-1 三窗口流程图

3.3 实验结果分析

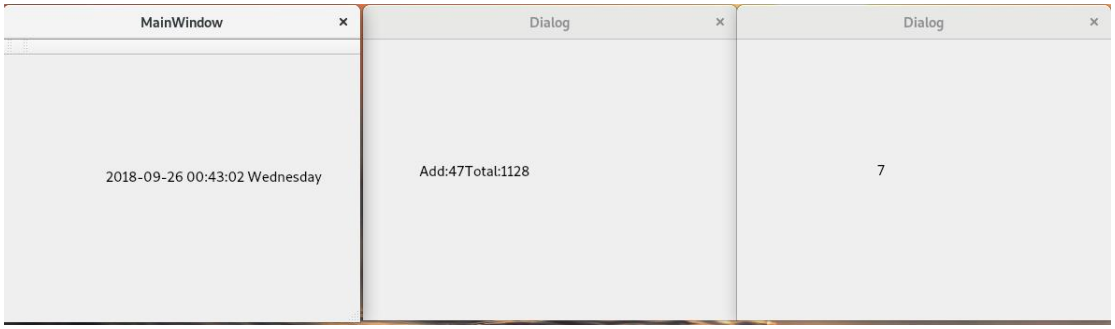


图 3-2 三窗口测试

3.4 心得与体会

进行进程并发，Qt 如果使用同一窗口，不同的 widget 那么不同的 widget 之间为三个线程，而如果为进程的话，可以创建三个 Project 在一个主 Project 中进行 QProcess 启动三个程序，或者直接利用 fork 进行进程创建。

4 添加系统调用

4.1 实验要求和内容

要求： 掌握添加系统调用的方法

内容:

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

4.2 实验步骤

4.2.1 添加源代码

编写添加到内核中的源程序，函数名以 sys_开头。

如：mycall(int num)，在 arch/kernel/sys.c 文件中添加如下调用源码：

SYSCALL_DEFINE2(mycopy, const char *, src, const char *, dst)

系统调用分析：在 arch 中采用宏定义的方式提供了函数参数检查和一些安全性保护，其中如果想要返回某些数值，而在返回前已经设置了空间范围，宏将会自动完成返回值从内核空间到用户空间的转换。

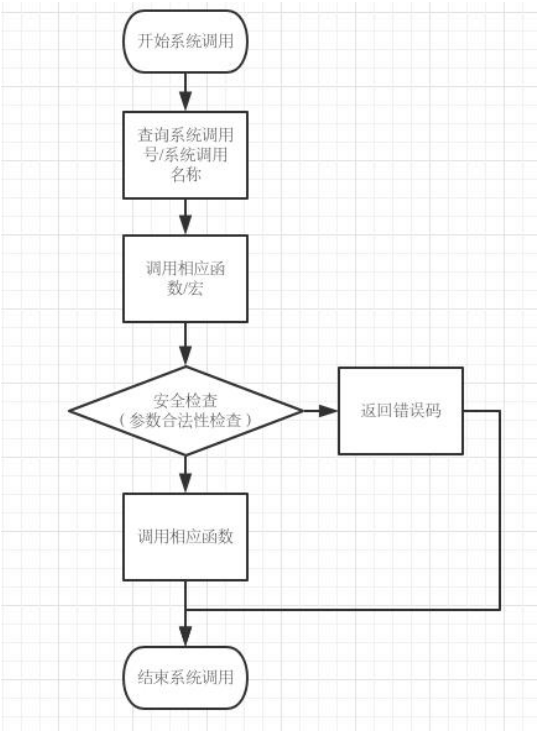


图 4-1 系统调用分析

4.2.2 添加新的系统调用

在传统的 Linux 系统调用需要更改两个文件使内核的其余部分知道该系统调用的存在。:

a) include/linux/syscalls.h ——系统调用定义

增加新系统调用的函数定义

asmlinkage long sys_mysyscall(int number);

b) arch/x86/syscalls/syscall_64.tbl ——系统调用表

在系统调用表中为新增的系统调用分配一个系统调用号和系统调用名。

在 arch 中采用宏定义的方式进行系统调用的方式进行封装，故仅仅需要依据系统默认的规则进行系统调用的声明：

<number> <abi> <name> <entry point>

其中，__x64 为宏定义中的最终的接口形式，当使用宏定义的时候，最终一层的系统调用入口与为__64_sys。

4.2.3 编译 Linux 内核

在内核的编译过程中，arch 提供了两种机制，一种是传统的内核编译机制，在内核完整编译之后修改 Boot 进行启动，另一种基于 arch 的包管理机制，从源码编译生成包，然后卸载原内核（包），安装新生成的包，此处采用传统编译方式

1) 编译内核

make

2) 编译内核模块

Make modules

3) 生成内核配置文件

make menuconfig

此处可以修改不同的驱动配置，文件系统配置，否则生成出的为裸内核，不支持任何外界设备以及默认的集成显卡驱动，同时系统本身带有内核的配置文件，可以将其拷贝过来。

4) 编译内核映像

make bzImage

5) 编译内核模块

make modules

6) 生成并安装模块

make modules_install

7) 安装新的系统

   make install

8) 重启，选择新修改的内核

9) 编写应用程序，测试新增系统调用

4.3 实验过程与结果

4.3.1 查看内核版本

cat /proc/version

```
wings@hover  ~  cat /proc/version
Linux version 4.17.2-1-ARCH (builduser@heftig-9574) (gcc version 8.1.1 20180531 (GCC)) #1 SMP PREEMPT Sat Jun 16 11:08:59 UTC 2018
```

图 4-1 查看内核版本

4.3.2 获取内核源码

访问 https://www.kernel.org/ 下载 4.18.5 版本内核

4.3.3 添加源代码

在 arch/kernel/sys.c 中添加调用服务例程定义

1. 添加调用函数声明

在 include/linux/syscalls.h 中添加调用函数声明

```
SYSCALL_DEFINE2(mycopy, const char *, src, const char *, dst)
{
    int MAX_LENGTH=256;
    struct kstat k_buf;
    char buf[MAX_LENGTH];
    int read_fd, write_fd;
    long read_num;

    //save old fs
    mm_segment_t old_fs =get_fs();
    set_fs(KERNEL_DS);

    int i=0;
    //copy src name
    char _src[MAX_LENGTH];
    i=copy_from_user(_src,src,MAX_LENGTH*sizeof(char));
    if(i==0)
    {
        printk("copy src from user sus!");
    }
    else
    {
        printk("copy src from user fail!");
        set_fs(old_fs);
        return -1;
    }
```

图 4-2 系统调用声明截图

2. 系统调用表

在/arch/x86/entry/syscalls/syscall_64.tbl 系统调用表中为新增的系统调用分配一个系统调用号和系统调用名。

```
335 common  mycall          __x64_sys_mycall
336 common  mycopy          __x64_sys_mycopy
```

图 4-3 系统调用表截图

3. 编译内核

```
KERNEL_VERSION="418"

# set compile arg
kernel_num=8
# Back edited file
cp arch/x86/entry/syscalls/syscall_64.tbl ../
cp kernel/sys.c ../

# make image
# make mrproper
# make menuconfig
make bzImage –j $KERNEL_VERSION

# make modules
make modules –j $kernel_num
make modules_install –j $kernel_num

# make install
make install –j $kernel_num

# copy kernel image to boot
cp arch/x86_64/boot/bzImage /boot/vmlinuz-linux$KERNEL_VERSION

# build initramfs
mkinitcpio -p linux$KERNEL_VERSION

# update-grub
grub-mkconfig -o /boot/grub/grub.cfg
```

4.4 实验结果分析

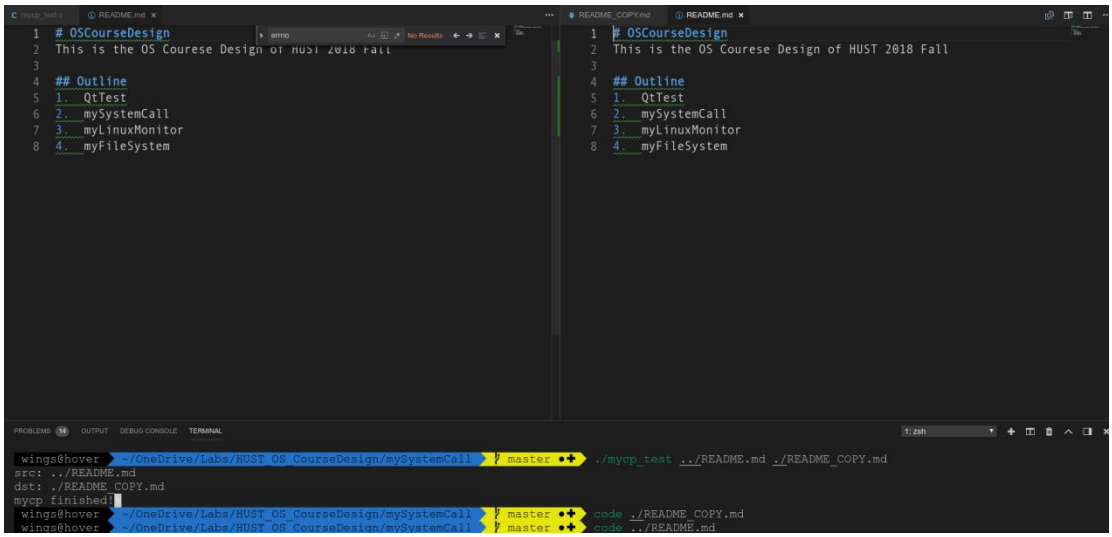


图 4-4 系统调用测试

4.5 心得与体会

整个内核实验过程中，学习了《Linux 内核分析》学习使用 GDB 进行内核函数的跟踪调试，而通常使用 printk 和 errno 错误代码记录也是较为有效的方式。

许多教程采用了每次编译后 make clean/make mrproper 的方式，导致编译时间很长，而实际上因为采用增量编译，则 make 仅仅需要对修改的部分进行重新编译和链接，故直接 make 就好，通常不会出现问题，这在一些较大的工程中是常采用的方式，同时 -j $kernel_num 在 make 执行过程中可以同时执行的指令数目，相当于并行，在有大量独立模块且依赖较上的时候，能够大大加快底层的编译速度。

开始进行内核调用的时候参考传统教程，修改系统 sys.h，sys.c 和 syscall_64.tbl。而对于 ArchLinux 采用宏定义的方式包装相应函数，目的是提供一定的安全性机制，包括参数检查，返回值的空间转换。

而当使用宏定义后，系统自动生成的函数调用接口为__sys_x64 格式，此时对于仍然保留 sys.h 中函数定义，发现并不会冲突，因为函数入口的格式变化，这种机制保证了新旧函数调用的过渡和向后兼容，可以看到，在 ArchLinux 中仍抱有少量的旧式函数调用。

5 设备驱动

5.1 设计目的和内容

要求： 掌握添加设备驱动程序的方法

内容：

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

写一个应用程序， 测试添加的驱动程序

5.2 环境及步骤

5.2.1 开发环境

1) 操作系统：Arch Linux x64
2) 内核版本：4.18.5-arch1-1-ARCH
3) TextEditor: Visual Studio Code
4) 编译工具： gcc (GCC) 8.2.0

5.2.2 开发步骤

1) 编写设备驱动程序 mydev.c
2) 设备驱动模块的编译 Makefile 文件的使用
3) 加载设备驱动模块: insmod mydev.ko
4) 生成设备文件： mknod /dev/test c 254 0

5.3 设计实现及关键代码

5.3.1 编写设备驱动程序

Linux 设备驱动程序利用 file_operations 提供调用接口，当程序调用设备驱动程序对设备进行操作的时候，通过该结构体接口，将相应操作交给设备驱动程序的函数，完成操作。

```
struct file_operations pStruct=
{
    open : my_open,
    release : my_release,
    read : my_read,
    write : my_write,
```

};

5.3.2 设备驱动 Makefile

设备驱动独立于内核便于修改，但编写好的设备驱动需要 build 进最终的内核所在的文件夹，才能够调用系统命令进行设备驱动的装载。2

```
ifeq ($(KERNELRELEASE),)

KERNELDIR := /lib/modules/$(shell uname -r)/build
PWD := $(shell pwd)

modules:
	$(MAKE) -C $(KERNELDIR) M=$(PWD) modules

modules_install:
	$(MAKE) -C $(KERNELDIR) M=$(PWD) modules_install

clean:
	rm -rf *.o .depend .*.cmd *.ko *.mod.c .tmp_versions modules.*

else

obj-m := myDevDriver.o

endif
```

5.3.3 加载设备驱动模块

insmod myDevDrive.ko

图 5-1 设备驱动装载结果

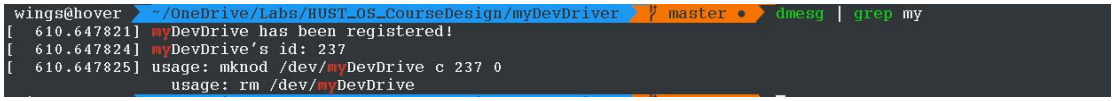


图 5-2 设备驱动调试信息

```
✗ wings@hover  ~/OneDrive/Labs/HUST_OS_CourseDesign/myDevDriver  master ●  sudo mknod /dev/myDev c 237 0
 wings@hover  ~/OneDrive/Labs/HUST_OS_CourseDesign/myDevDriver  master ●  ls
a.out  Makefile        Module.symvers  myDevDriver.ko      myDevDriver.mod.o  myDevDriver_test.c
back   modules.order   myDevDriver.c   myDevDriver.mod.c   myDevDriver.o
 wings@hover  ~/OneDrive/Labs/HUST_OS_CourseDesign/myDevDriver  master ●  ls /dev | grep my
myDev
 wings@hover  ~/OneDrive/Labs/HUST_OS_CourseDesign/myDevDriver  master ●
```

图 5-3 分配设备号

5.3.4 设备驱动测试

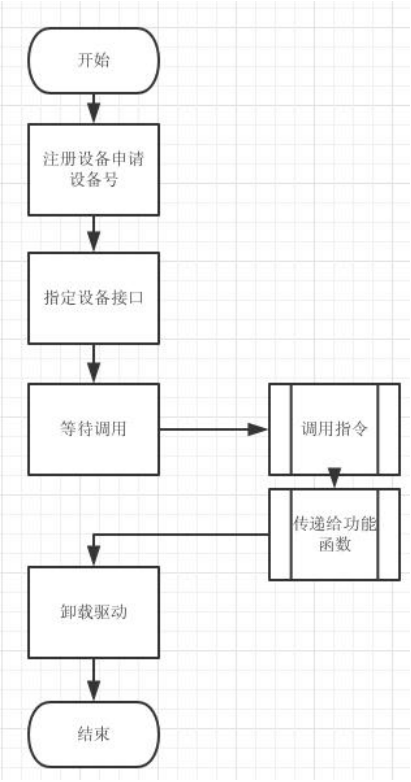


图 5-4 设备驱动测试流程图

```
✗ wings@hover  ~/OneDrive/Labs/HUST_OS_CourseDesign/myDevDriver  master ●  sudo ./a.out
acpi_thermal_rel  dri           input           net                 rfkill  sdd     tty1   tty21  tty33  tty45  tty57  ttyS2     vcs1   vfio
autofs            drm_dp_aux0   kmsg            network_latency     rtc     sg0     tty10  tty22  tty34  tty46  tty58  ttyS3     vcs2   vga_arbiter
block             fb0           kvm             network_throughput  rtc0    sg1     tty11  tty23  tty35  tty47  tty59  uhid      vcs3   vhci
bsg               fd            lightnvm        null                sda     sg2     tty12  tty24  tty36  tty48  tty6   uinput    vcs4   vhost-net
btrfs-control     freefall      log             nvidia0             sda1    sg3     tty13  tty25  tty37  tty49  tty60  urandom   vcs5   vhost-vsock
bus               full          loop-control    nvidiactl           sda2    shm     tty14  tty26  tty38  tty5   tty61  usb       vcs6   video0
char              fuse          mapper          nvidia-modeset      sda3    snapshot tty15 tty27  tty39  tty50  tty62  userio    vcsa   video1
console           hidraw0       media0          port                sda4    snd     tty16  tty28  tty4   tty51  tty63  v4l       vcsa1  zero
core              hidraw1       mei0            ppp                 sda5    stderr  tty17  tty29  tty40  tty52  tty7   vboxdrv   vcsa2
cpu               hidraw2       mem             psaux               sdb     stdin   tty18  tty3   tty41  tty53  tty8   vboxdrvu  vcsa3
cpu_dma_latency   hpet          memory_bandwidth ptmx               sdb1    stdout  tty19  tty30  tty42  tty54  tty9   vboxnetctl vcsa4
cuse              hugepages     mqueue          pts                 sdc     tty     tty2   tty31  tty43  tty55  ttyS0  vboxusb   vcsa5
disk              initctl       myDev           random              sdc1    tty0    tty20  tty32  tty44  tty56  ttyS1  vcs       vcsa6
Please input the device's name: myDev
---------Hover'sDriver---------
Please input a string : change dev
device Message : change dev
```

图 5-5 设备驱动测试结果

5.5 设计感想

在调试过程中，发现存在无法卸载驱动的情况，出现驱动繁忙而无法卸载的情况，此时可能为驱动在 insmod 的时候出现了 NULL 指针异常或者在 exit 函数的时候没有正常退出，导致驱动虽然被加载了(kerneloops 驱动的结点已经被插入到内核设备树中)，但是驱动运行过程中却导致内核段错误，设备引用计数没有被正常的清除，导致无法卸载，可以采用重新启动系统的方式，但是很麻烦，于是与决定看看有没有解决办法。

Rmmod 进行的系统调用为 sys_delet_module，此时可以重新编写 debug 版本的驱动代码，重新注册 exit 函数并使用 atomic_set(&mod->refcnt, 1)重置驱动引用，便可以安全删除，仅仅作为 debug 使用，在正式版本中不能强制更改引用，容易造成设备故障。

Linux 系统为微内核架构，故文件系统，网络管理的模块尽可能的放在内核之外，而驱动本质仍为 VFS 的一部分虽然增加了一部分的调度开销，但是能够使内核精简，方便的添加驱动程序，修改文件系统。

6 QT 系统监控器

6.1 设计目的

1. 了解/proc 文件的特点和使用方法
2. 监控系统状态，显示系统部件的使用情况
3. 用图形界面监控系统状态，包括 CPU 和内存利用率、所有进程信息等(可自己补充、添加其他功能)

6.2 设计内容

1. 监控系统功能：通过读取 proc 文件系统，获取系统各种信息，并以比较容易理解的方式显示出来
2. C 语言开发，图形界面直观展示
3. 参照 WINDOWS 的任务管理器，实现其中的部分功能

   主机名、系统启动时间、系统运行时间、版本号、所有进程信息、CPU 类型、CPU 的使用率、内存使用率

6.3 环境及步骤

6.3.1 开发环境

1) 操作系统： Arch Linux x64
2) 内核版本： 4.18.5-arch1-1-ARCH
3) IDE: Qt Creator 4.7.0(Based onQt5.11.1)
4) 编译工具： gcc (GCC) 8.2.0

6.3.2 运行环境

基于 Qt 跨平台特性，可运行于基于 Linux 的平台

6.3.3 开发步骤

1) 创建窗口，进行初始化
2) CPU:
   a) 获取 CPU 的时间信息，利用间隔时间进行 CPU 利用率计算
   b) 将新的 CPU 信息点加入图表，进行刷新
3) Process
   a) 定时读取进程信息

b) 维护 proc 内存池，进行 process 列表更新

c) 进行 table 展示的更新

d) 设置焦点和当前所在的页面

4) Net:

a) 获取上一秒和当期秒当前网卡的数据包信息

b) 进行数据负载的计算

c) 维护 chart 展示，计算 60s 之类的峰值，进行表格的适当展示

5) Mem:

a) 读取内存信息

b) 维护 chart 图表

6.4 设计实现及关键代码

6.4.1 CPU

1) 数据结构描述

a) cpu_info

```
/*
 * utime: user
 * stime: system time
 * ntime: nice time, the time for modefiy the priority of cpu
 * itime: idle time
 * iowtime: io waiting time
 * irqtime: interuption time
 * sirqtime: soft interuption time
*/
struct cpu_info
{
        long unsigned utime, ntime, stime, itime;
        long unsigned iowtime, irqtime, sirqtime;
};
```

b) proc_info

```
struct proc_info
{
        struct proc_info *next;
        pid_t pid;
        pid_t tid;
        uid_t uid;
        gid_t gid;
```

```
        char name[PROC_NAME_LEN];
        char tname[THREAD_NAME_LEN];
        char state;
        long unsigned utime;
        long unsigned stime;
        long unsigned delta_utime;
        long unsigned delta_stime;
        long unsigned delta_time;
        long vss;
        long rss;
        int num_threads;
        char policy[32];
    };
```

c) proc_list

```
    struct proc_list
    {
        struct proc_info **array;
        int size;
    };
```

2) 更新过程

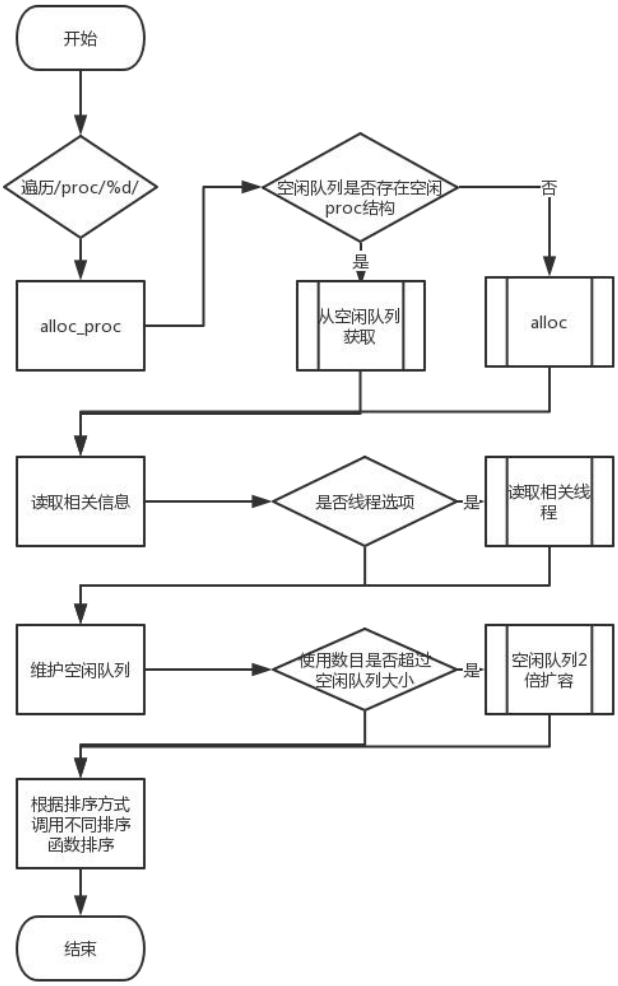


图 5-1 CPU 信息更新过程流程图

3) 利用率计算

a) 两次更新之间的总时间 Total_delta_time

total_delta_time =

(new_cpu.utime + new_cpu.ntime + new_cpu.stime+new_cpu.itime+ new_cpu.iowtime + new_cpu.irqtime + new_cpu.sirqtime)-

(old_cpu.utime + old_cpu.ntime + old_cpu.stime + old_cpu.itime + old_cpu.iowtime + old_cpu.irqtime + old_cpu.sirqtime);

b) 系统态和用户态时间计算

cpu_user= ((new_cpu.utime + new_cpu.ntime) - (old_cpu.utime + old_cpu.ntime)) * 100 / total_delta_time;

cpu_sys= ((new_cpu.stime) - (old_cpu.stime)) * 100 / total_delta_time;

4) 图表绘制

图 5-2 CPU 绘图过程流程图

6.4.2 Process

1) 更新列表
2) 列表维护
3) 列表绘制

保存更新前的焦点 pid，获取更新后的表格行数，进行焦点设置，利用表格行数除以总行数获取页面相对位置，进行页面位置设置。

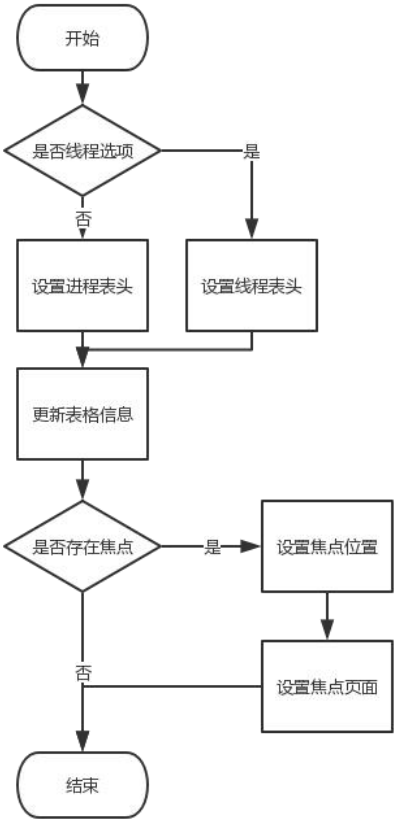


图 5-3 CPU 模块流程图

6.4.3 Memory

1) 更新内存信息
   a) 打开/proc/meminfo 文件
   b) 定位不同的标识符，更新内存信息
   c) 关闭文件
2) 利用率计算

   在 Linux 内存管理中，存在已经释放但是仍然存在与内存的缓存，故 MemFree<MemAvailable,故进行计算利用率时采用 MemTotal-MemAvailable 作为 MemUsed
3) 图表绘制

   绘制内存曲线图和饼图
4) 双击触发

   当点击表格，使用 nautiulus 系统调用打开对应的文件系统。

6.4.4 Net

1) 读取网络信息

   读取/proc/net/dev 中的网卡信息

2) 利用率计算

   计算两个时间差之间的包数量，从而近似计算当前网络的利用率

3) 图表绘制

   需要较为合适的绘制速率曲线，计算 60s 内大于峰值的最小模 512 整数作为 y 轴最高值进行曲线绘制

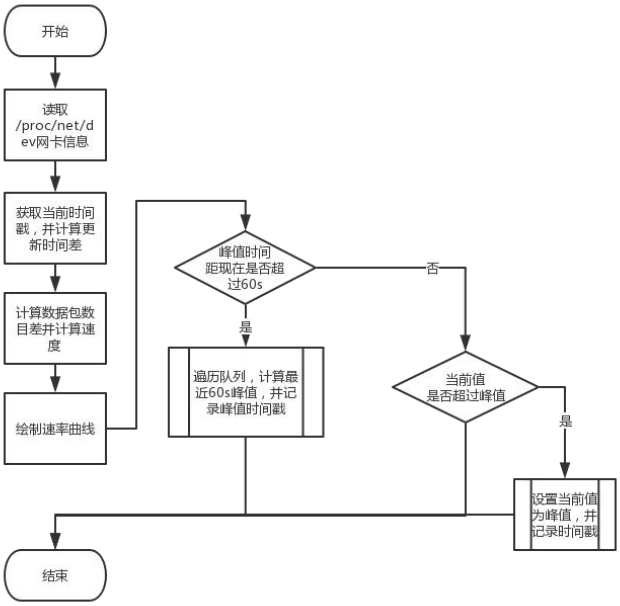


图 5-4 Net 模块流程图

6.4.5 FileSystem

1) 读取/etc/mtab 信息，并判断非交换文件和临时文件系统
2) 更新 FileSystablelist，并计算利用率
   a) 硬盘整个的空间大小不等于 blocks 数目，还含有 inode 节点和前面的空余信息，故整个硬盘的大小为 disktotal=used+avail
   b) blocks_percent_used=blocks_used/disktotal;
3) 调用绘图绘制利用率曲线图

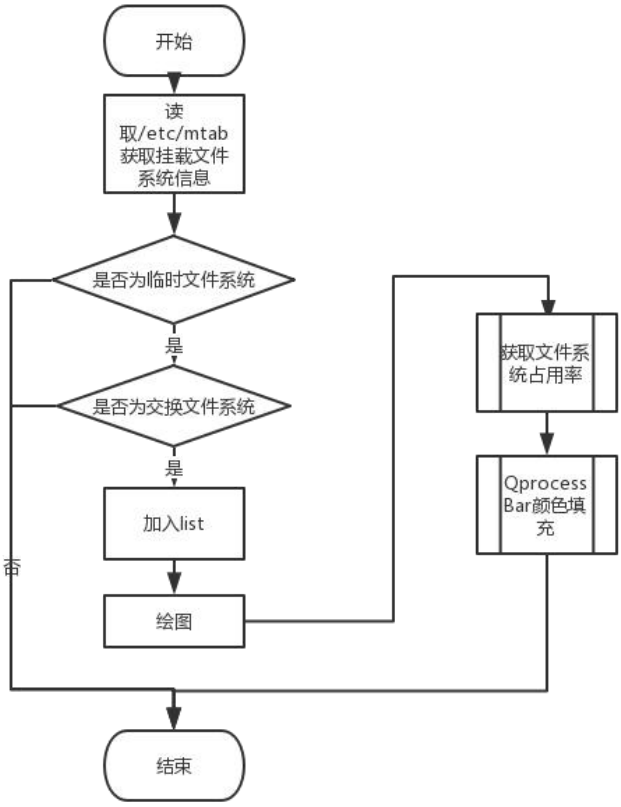
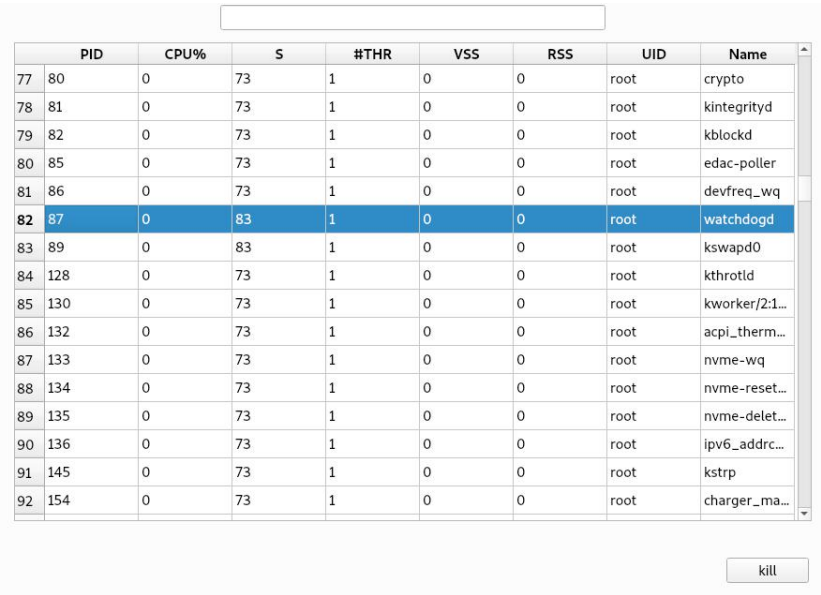


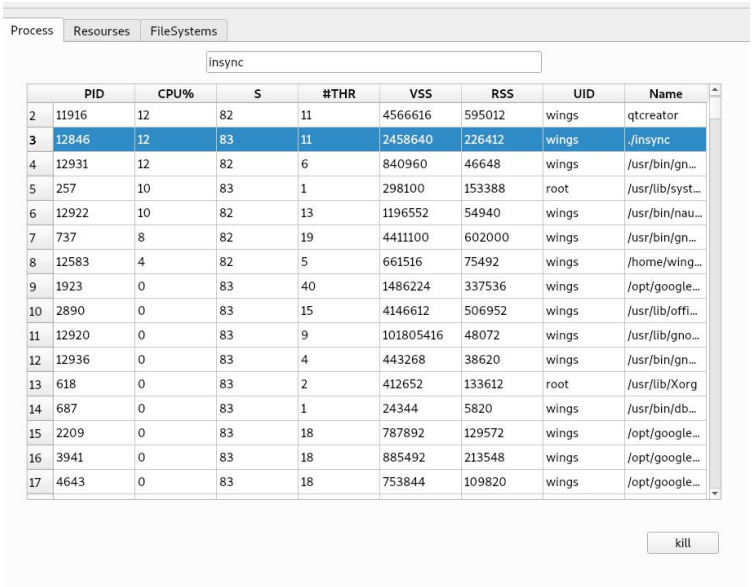
图 5-5 FileSystem 模块流程图

6.6 调试记录及运行结果



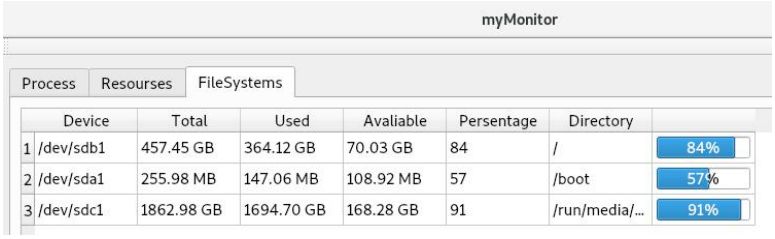
	PID	CPU%	S	#THR	VSS	RSS	UID	Name
77	80	0	73	1	0	0	root	crypto
78	81	0	73	1	0	0	root	kintegrityd
79	82	0	73	1	0	0	root	kblockd
80	85	0	73	1	0	0	root	edac-poller
81	86	0	73	1	0	0	root	devfreq_wq
82	87	0	83	1	0	0	root	watchdogd
83	89	0	83	1	0	0	root	kswapd0
84	128	0	73	1	0	0	root	kthrotld
85	130	0	73	1	0	0	root	kworker/2:1...
86	132	0	73	1	0	0	root	acpi_therm...
87	133	0	73	1	0	0	root	nvme-wq
88	134	0	73	1	0	0	root	nvme-reset...
89	135	0	73	1	0	0	root	nvme-delet...
90	136	0	73	1	0	0	root	ipv6_addrc...
91	145	0	73	1	0	0	root	kstrp
92	154	0	73	1	0	0	root	charger_ma...

图 5-6 Process 模块排序测试图



	PID	CPU%	S	#THR	VSS	RSS	UID	Name
2	11916	12	82	11	4566616	595012	wings	qtcreator
3	12846	12	83	11	2458640	226412	wings	./insync
4	12931	12	82	6	840960	46648	wings	/usr/bin/gn...
5	257	10	83	1	298100	153388	root	/usr/lib/syst...
6	12922	10	82	13	1196552	54940	wings	/usr/bin/nau...
7	737	8	82	19	4411100	602000	wings	/usr/bin/gn...
8	12583	4	82	5	661516	75492	wings	/home/wing...
9	1923	0	83	40	1486224	337536	wings	/opt/google...
10	2890	0	83	15	4146612	506952	wings	/usr/lib/offi...
11	12920	0	83	9	101805416	48072	wings	/usr/lib/gno...
12	12936	0	83	4	443268	38620	wings	/usr/bin/gn...
13	618	0	83	2	412652	133612	root	/usr/lib/Xorg
14	687	0	83	1	24344	5820	wings	/usr/bin/db...
15	2209	0	83	18	787892	129572	wings	/opt/google...
16	3941	0	83	18	885492	213548	wings	/opt/google...
17	4643	0	83	18	753844	109820	wings	/opt/google...

图 5-7 Process 模块搜索测试图



	Device	Total	Used	Avaliable	Persentage	Directory	
1	/dev/sdb1	457.45 GB	364.12 GB	70.03 GB	84	/	84%
2	/dev/sda1	255.98 MB	147.06 MB	108.92 MB	57	/boot	57%
3	/dev/sdc1	1862.98 GB	1694.70 GB	168.28 GB	91	/run/media/...	91%

图 5-7 FileSystem 模块外接设备测试图

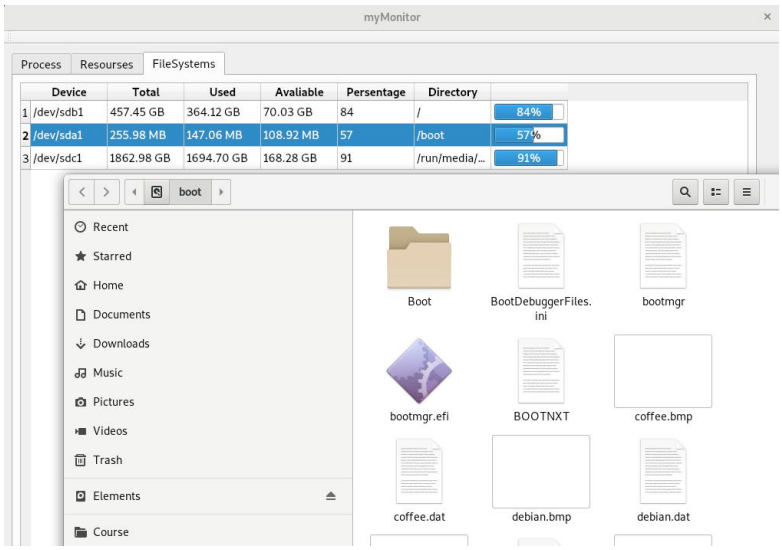


图 5-7 FileSystem 模块双击打开测试图

图 5-7 资源管理器模块测试图

6.7 设计感想

设计过程中，参考 GNOME-MONITOR，使用 Qt 的定时器机制进行操作，定时更新，因为 Process 和绘制图像的即时性不同，故采用不同的定时时长，进行更行。

更新机制上，获取数据部分和绘图部分分离，可以采用不同的定时器时长，在绘图的间隙中间进行数据的获取，减少卡顿。

Process 表格维护上，采用了内存池机制，避免频繁的内存申请和释放，同时尝试了原生的 model 和自己手动实现了焦点追踪进行比较，因为原生的 model 进行的实现的时候，对于添加项的数据如果已经存在，仅仅进行 update 操作，故所在的 item 并不会有构造和销毁操作，焦点不会丢失，同时数字不会闪烁，尝试手动实现，计算焦点所在的列数

Net 表格绘制，为了维护适当的坐标轴上的最大刻度，维护 60s 内的峰值，并设置过时机制，使能够尽可能的合理显示网络曲线。

FileSystem 模块，因为 QProcessBar 为 widget 的子类，故不能嵌套进 item 之下，故采用自己继承原 UI 类进行重载，手写绘图函数进行块填充，完成图像的绘制。

7 模拟文件系统设计

7.1 设计目的

熟悉 Linux 文件系统

7.2 设计内容

1. 用磁盘中的一个文件（大小事先指定）来模拟一个磁盘
2. 确定文件目录项的结构
3. 空闲块的管理（每个块 = 连续的N个文件字节）
4. 扩充系统调用命令实现文件的操作：open、 close、 read、 write、 cp、 rm 等

7.3 环境及步骤

7.3.1 开发环境

1) 操作系统： Arch Linux x64
2) 内核版本： 4.18.5-arch1-1-ARCH
3) 编译器： gcc (GCC) 8.2.0
4) 编译工具： CMake 3.12.1
5) 编辑器: Visual Studio Code

7.3.2 开发步骤

1) 初始化文件系统，并对相应的类
2) 载入 superblock，同时输出界面信息
3) 对于相应的操作，进行相应的读写操作
4) 函数退出，将缓存写回

7.3.2 系统结构

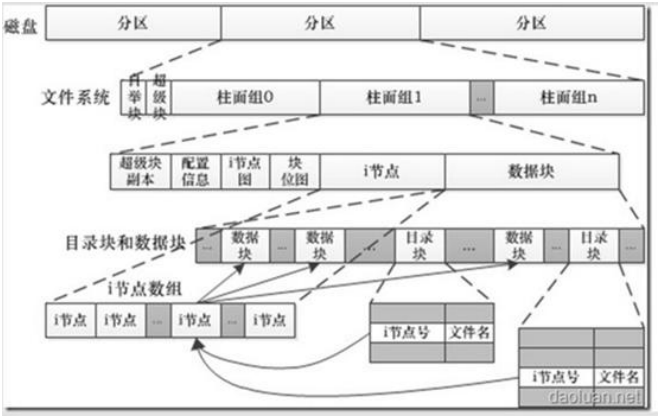


图 7-1 系统结构图

7.4 内存版本设计实现

7.4.1 FreeNode

1) 类描述

所删除的文件之后，其 inode 和数据块需要同时被释放，此时将其打包为 FreeNode，故当下一次需要创建文件，先从 FreeNodeList 中进行寻找，减少了 inode 类构造和销毁的开销，同时减少创建文件需要遍历位图的开销

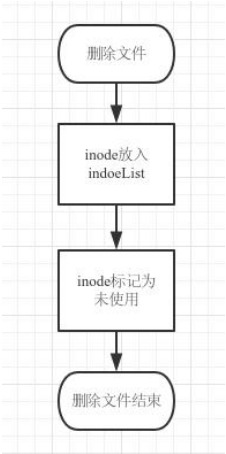


图 7-2 删除文件操作

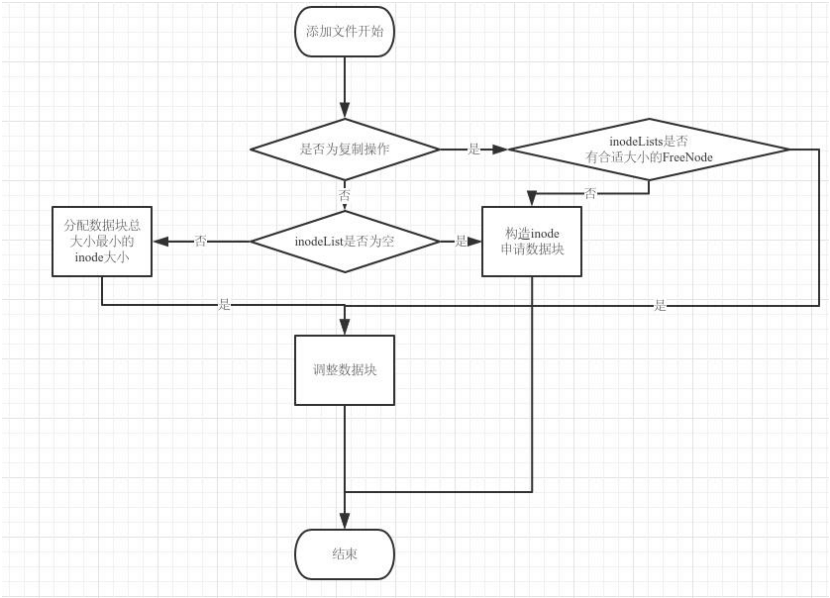


图 7-3 添加文件操作

2) 关键成员

数据块个数： uint block_num; //free node num

起始数据块个数：uint pos; //start pos

7.4.2 Inode

(1) 类描述

Inode 类，索引节点，记录文件相关信息，包括 inode 号，创建时间，文件类型等。文件的数据块有两种形式，一种是采用数据块串联链表形式，因为流式文件多为顺序读取，另一种直接在 ionde 中存储指针数组，此处采用后者。

(2) 关键成员

Inode 号： uint inode_num;

起始数据块位置：uint sec_num;

7.4.3 DirEntry

(1) 类描述

文件的数据块类，包含文件的类型（数据/目录）及相应的构造函数接口，利用 c++的 lock 机制提供访问保护。

(2) 关键成员

a) 目录项/数据项构造函数

static shared_ptr<DirEntry>
make_de_dir(const string name,const shared_ptr<DirEntry> parent);

```
static shared_ptr<DirEntry>
make_de_file(const string name,const shared_ptr<DirEntry> parent,
const shared_ptr<Inode> &inode=nullptr);
```

b) 目录入口

```
weak_ptr<DirEntry> parent; // .
weak_ptr<DirEntry> self; // ..
list<shared_ptr<DirEntry>> contents; // dir entry
```

c) 锁变量

```
bool is_locked; // lock
```

d) 添加子项操作

```
shared_ptr<DirEntry> find_child(const string name) const;
shared_ptr<DirEntry> add_dir(const string name);
shared_ptr<DirEntry> add_file(const string name);
```

7.4.4 myFs(file system operation)

(1) 类描述

相应的界面输出函数以及参数解析，同时利用底层类的接口实现相应的常用文件操作，同时维护当前打开的文件及其描述信息。

(2) 关键成员

a) 打开文件描符

```
struct Descriptor
{
    Mode mode;          // open mode
    uint byte_pos;      //now pos
    weak_ptr<Inode> inode;
    weak_ptr<DirEntry> from;
    uint fd;
};
```

b) 当前位置描述符

```
struct PathRet
{
    bool invalid_path = false;
    string final_name;
    shared_ptr<DirEntry> parent_node;
    shared_ptr<DirEntry> final_node;
};
```

c) 基础读/写操作

真正的读写操作，向上提供服务接口，不提供保护功能

bool basic_open(Descriptor *d, vector <string> args);
unique_ptr<string> basic_read(Descriptor &desc, const uint size);
uint basic_write(Descriptor &desc, const string data);
bool basic_close(uint fd);

(3) 操作函数

所有操作函数接口包括：

文件操作：open,read,write,seek.close,

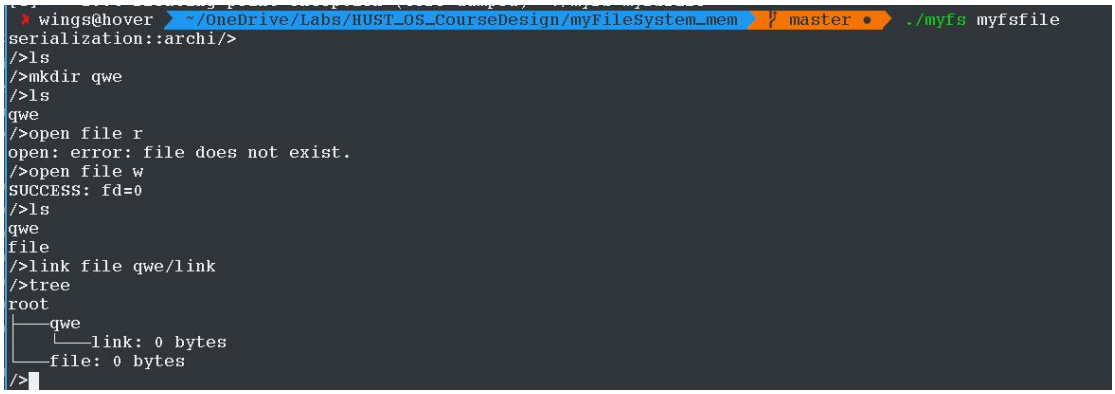
文件夹操作：mkdir,rmdir

文件系统信息查看：cd,stat,ls,cat,pwd,tree,

本地文件系统交互：import,FS_export

文件拷贝：link,unlink,cp

7.4.5 系统测试



7-4 系统调用接口测试

7.5 硬盘版本设计实现

7.5.1 Buffer

1) 类描述

提供 cache 机制,向上提供虚拟读写接口，供功能函数进行调用，在进行调用时，维护 bufferList，每一次调用将更新当前读写块的优先级，然后调用底层的读写接口进行真正的磁盘读写操作

2) 关键成员

a) 函数接口

bool write_disk(const BufferNode& node);

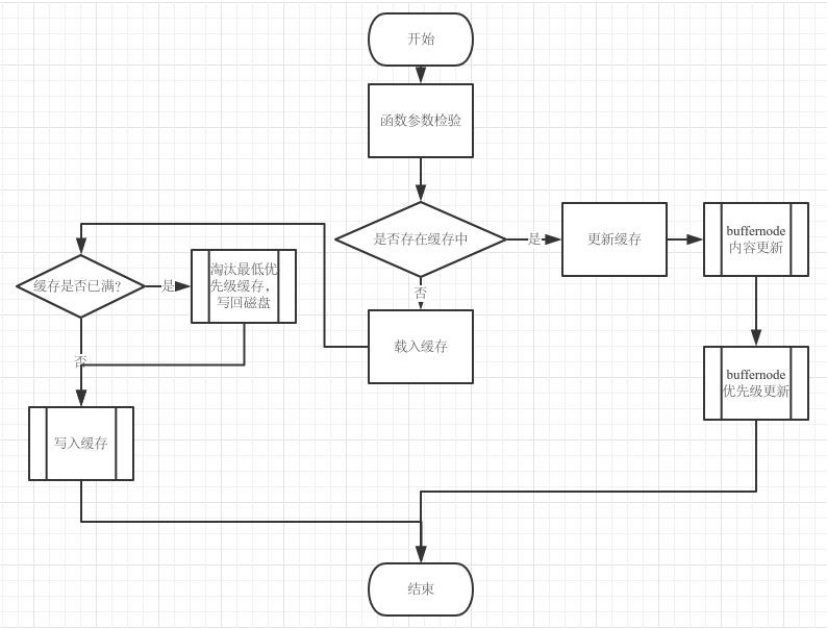


图 7-5 写磁盘操作流程图

bool read_disk(int sec_num, BufferNode& node);

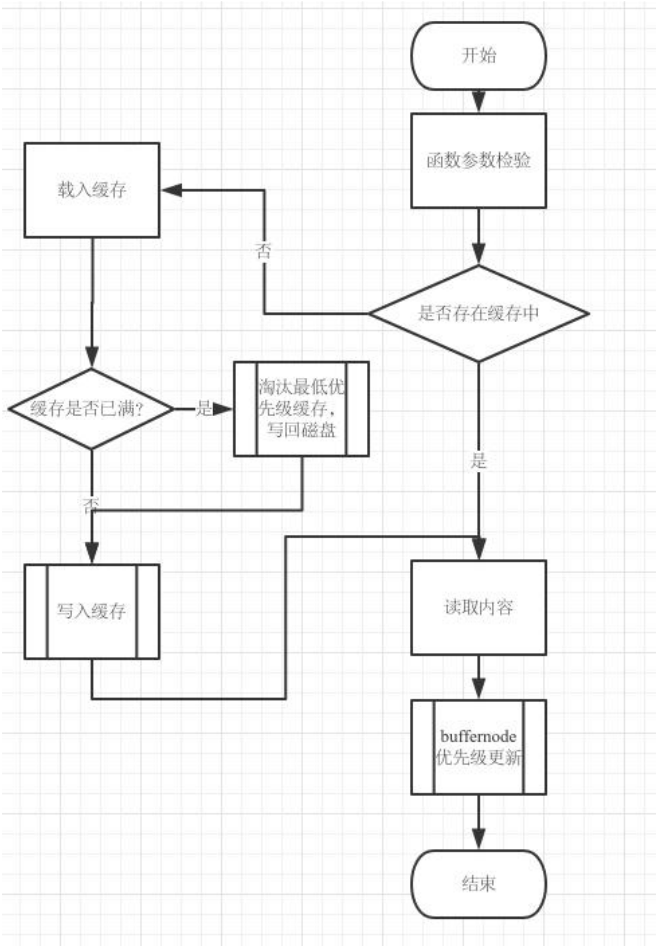


图 7-6 读磁盘操作流程图

void all_write_to_disk();

b) 底层操作函数

```
bool real_disk_write(const BufferNode& node);
bool real_disk_read(int sec_num, BufferNode& node);
int has_sec(int sec_number);
int is_full();
```

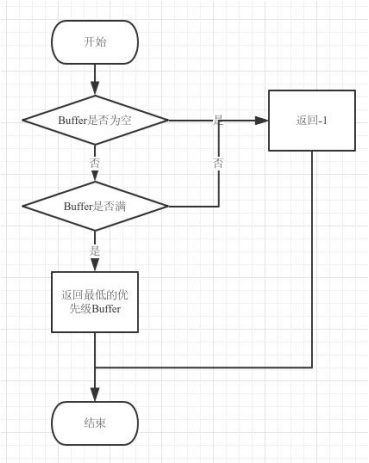


图 7-7 查找最低优先级缓冲流程图

7.5.2 SuperBlock

1) 类描述

维护当前文件系统 inode 位图和 block 位图，同时提供 inode 和 block 的检索函数，获取到空闲的块

2) 关键成员

Inode 位图：bool inode_bitmap[INODE_NUM];

Block 位图：bool block_bitmap[BLOCK_NUM];

磁盘文件： fstream disk;

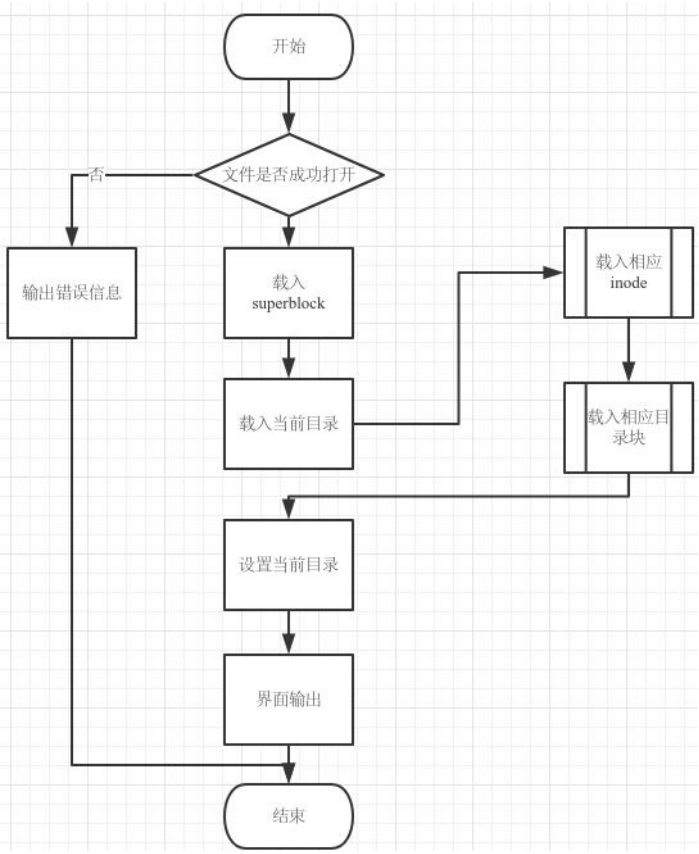


图 7-8 系统初始化流程图

7.5.3 myFs(file system operation)

1) 类描述

相应的界面输出函数以及参数解析，同时利用底层类的接口实现相应的常用文件操作，同时维护当前打开的文件及其描述信息。

2) 关键成员

Vim:使用 vim 编辑器编辑文件：

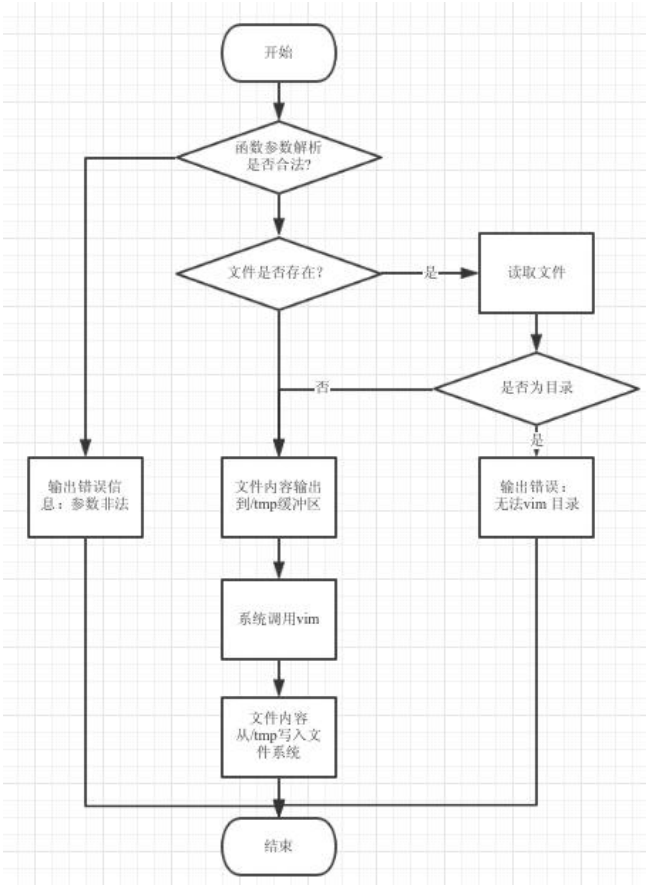


图 7-9 vim 系统调用流程图

7.5.4 系统测试

```
✗ wings@hover  ~/OneDrive/Labs/HUST_OS_CourseDesign/myFileSystem  master ●  ./myfs myfsfile
File has opened

****************** Hover's FileSystem ******************
/>format
create new inode0 , begin sec: 66
create new inode1 , begin sec: 67
create new inode2 , begin sec: 68
create new inode3 , begin sec: 69
create new inode4 , begin sec: 70
create new inode5 , begin sec: 71
inode write back , inode num0, sec num: 2
inode write back , inode num1, sec num: 3
inode write back , inode num2, sec num: 4
inode write back , inode num3, sec num: 5
inode write back , inode num4, sec num: 6
inode write back , inode num5, sec num: 7
/>ls
bin etc home dev
/>mkdir testdir
create new inode6 , begin sec: 72
mkdir inode num num6inode write back , inode num6, sec num: 8
/>ls
bin etc home dev testdir
/>cd testdir
inode write back , inode num0, sec num: 2
inode num:6
root/testdir>touch qwe
touch file
create new inode7 , begin sec: 73
inode write back , inode num7, sec num: 9
```

图 7-10 目录函数测试

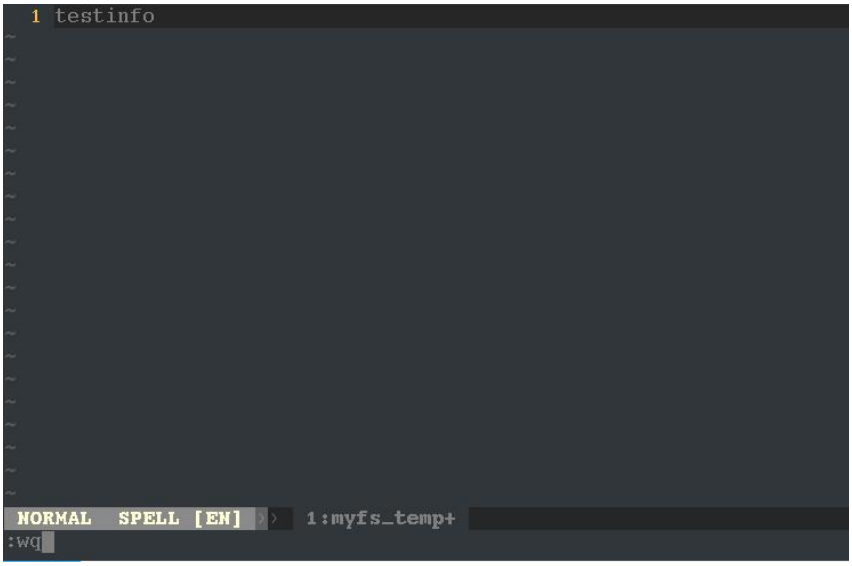


图 7-11 vim 系统调用测试 1

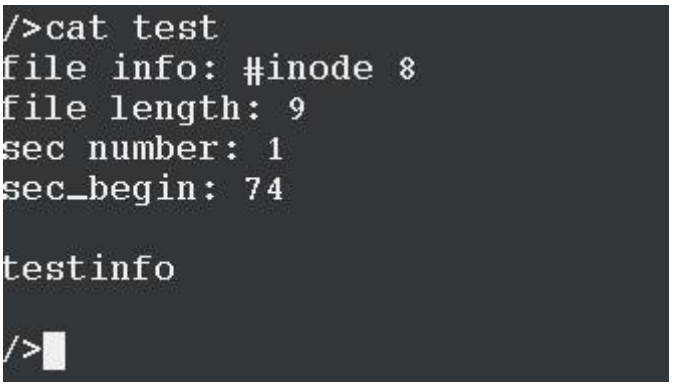


图 7-12 vim 系统调用测试 2

图 7-13 vim 系统调用测试 3

```
/>ls
bin etc home dev qwe asd
/>rmdir bin
inode num of the dir is : 1
delete inode, inode num1
delate inode, delete sector
/>ls
 etc home dev qwe asd
```

图 7-14 rm 测试

7.5 实验总结

开始了解到关于内核编译文件系统模块，所参考的教程较老，且内核编译较为复杂，故不采用，后学习 FUSE，但调用的为系统底层 API，上层加壳实现其他的目的（如网络文件系统、音乐库文件系统），故最后采用手动编写文件系统。

模拟文件系统类似资源管理工具，如游戏引擎中的资源管理器，本质为磁盘和内存之间信息的交换，故文件系统中遇到的问题也大多相似：脏数据，减少读写次数和读写频率，内存空间限制，涉及到操作系统中关于虚拟存储的常见算法：如缓存调度，其设计思想终究为目录结构，仅仅对于特定的目录内容进行载入。

在编写过程中，考虑参数传入，使用 C++ STL 模板类进行传参，同时对于参数的的个数进行动态的解析，在每一个函数的入口进行参数个数判断，最少参数数目及最多参数数目，并对操作对象和操作方法之间的合法性进行检验，确保程序的鲁棒性。

而所有的系统调用函数，因为之前实现过不同的函数，故仅仅是针对自己的文件系统适当更改，其中接口因为参照系统接口，故更改不大，体现了接口统一的重要性。

对象的持久化过程中，尝试了两种思路，一种是使用 Boost 库进行对象持久化，但因为在 Class 中采用变长的 vector 容器进行数据的存储，故仅仅在文件系统读取和最终关闭的时候进行 IO,同时需要占用较大的内存空间，故采用定长的分块管理机制。

性能优化上，采用了 FreeNodeList 对释放的 Node 节点进行缓存，类似于内存池机制，便于下一次 malloc，实现了自己的 malloc 机制。在硬盘 I/O 上，采用 Buffer 队列进行缓冲，对外提供 read/write 接口，对硬盘实现 read_disk/write_disk 操作，完成了缓冲机制，减少了硬盘 I/O，提高了读写效率。

参考文献

[1] Qt 官方文档

[2] Unix 环境高级编程

[3] Linux 内核分析[MOOC]中国科技大学

[4] Writing-a-linux-kernel-module
http://derekmolloy.ie/writing-a-linux-kernel-module-part-1-introduction/

[5] Boost 库 Tutorial
https://www.boost.org/doc/libs/1_57_0/libs/filesystem/doc/tutorial.html

附录

QTTest

QTTest.pro

```
#-------------------------------------------------
#
# Project created by QtCreator 2018-07-23T17:05:57
#
#-------------------------------------------------

QT       += core gui widgets

TARGET = QtTest
TEMPLATE = app

# The following define makes your compiler emit warnings if you use
# any feature of Qt which has been marked as deprecated (the exact warnings
# depend on your compiler). Please consult the documentation of the
# deprecated API in order to know how to port your code away from it.
DEFINES += QT_DEPRECATED_WARNINGS

# You can also make your code fail to compile if you use deprecated APIs.
# In order to do so, uncomment the following line.
# You can also select to disable deprecated APIs only up to a certain version of Qt.
#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables
all the APIs deprecated before Qt 6.0.0

CONFIG += c++11

SOURCES += \
        main.cpp \
        mainwindow.cpp \
    cycle_dialog.cpp \
    add_dialog.cpp

HEADERS += \
        mainwindow.h \
    cycle_dialog.h \
    add_dialog.h
```

```
FORMS += \
        mainwindow.ui \
    cycle_dialog.ui \
    add_dialog.ui

# Default rules for deployment.
qnx: target.path = /tmp/$${TARGET}/bin
else: unix:!android: target.path = /opt/$${TARGET}/bin
!isEmpty(target.path): INSTALLS += target
```

add_dialog.h

```
/* Author:Hover
 * E-Mail:hover@hust.edu.cn
 * GitHub:HoverWings
 * Description:QtTest
 */
#ifndef ADD_DIALOG_H
#define ADD_DIALOG_H

#include <QDialog>

namespace Ui {
class add_Dialog;
}

class add_Dialog : public QDialog
{
    Q_OBJECT

public:
    int add_num=0;
    int total=0;
    explicit add_Dialog(QWidget *parent = nullptr);
    ~add_Dialog();

private:
    Ui::add_Dialog *ui;

public slots:
    void update_add();
```

```
};

#endif // ADD_DIALOG_H
```

cycle_dialog.h

```
/* Author:Hover
  * E-Mail:hover@hust.edu.cn
  * GitHub:HoverWings
  * Description:QtTest
  */
#ifndef CYCLE_DIALOG_H
#define CYCLE_DIALOG_H

#include <QDialog>
#include <QSharedMemory>
#include <QBuffer>
#include <QString>

namespace Ui
{
	class cycle_Dialog;
}

class cycle_Dialog : public QDialog
{
	Q_OBJECT

public:
	int cycle_num=0;
	explicit cycle_Dialog(QWidget *parent = nullptr);
	~cycle_Dialog();

private:
	Ui::cycle_Dialog *ui;

public slots:
	void update_cycle();
};

#endif // CYCLE_DIALOG_H
```

mainwindow.h

```
/* Author:Hover
  * E-Mail:hover@hust.edu.cn
  * GitHub:HoverWings
  * Description:QtTest
  */
#ifndef CYCLE_DIALOG_H
#define CYCLE_DIALOG_H

#include <QDialog>
#include <QSharedMemory>
#include <QBuffer>
#include <QString>

namespace Ui
{
	class cycle_Dialog;
}

class cycle_Dialog : public QDialog
{
	Q_OBJECT

public:
	int cycle_num=0;
	explicit cycle_Dialog(QWidget *parent = nullptr);
	~cycle_Dialog();

private:
	Ui::cycle_Dialog *ui;

public slots:
	void update_cycle();
};

#endif // CYCLE_DIALOG_H
```

add_dialog.cpp

```
/* Author:Hover
  * E-Mail:hover@hust.edu.cn
```

```
 * GitHub:HoverWings
 * Description:QtTest
 */
#include "add_dialog.h"
#include "ui_add_dialog.h"

add_Dialog::add_Dialog(QWidget *parent) :
	QDialog(parent),
	ui(new Ui::add_Dialog)
{
	ui->setupUi(this);
}

add_Dialog::~add_Dialog()
{
	delete ui;
}



void add_Dialog::update_add()
{
	add_num+=1;
	total+=add_num;

ui->label->setText("Add:"+QString::number(add_num)+"Total:"+QString::number(to
tal));
	if(add_num==100)
	{
		add_num=0;
		total=0;
	}
}
```

cycle_dialog.cpp

```
/* Author:Hover
 * E-Mail:hover@hust.edu.cn
 * GitHub:HoverWings
 * Description:QtTest
 */
```

```
#include "cycle_dialog.h"
#include "ui_cycle_dialog.h"

cycle_Dialog::cycle_Dialog(QWidget *parent) :
	QDialog(parent),
	ui(new Ui::cycle_Dialog)
{
	ui->setupUi(this);



}

cycle_Dialog::~cycle_Dialog()
{
	delete ui;
}

void cycle_Dialog::update_cycle()
{
	cycle_num=(cycle_num+1)%10;
//	  ui->label->setTex
	ui->label->setText(QString::number(cycle_num));
}
```

main.cpp

```
/* Author:Hover
  * E-Mail:hover@hust.edu.cn
  * GitHub:HoverWings
  * Description:QtTest
  */
#include "mainwindow.h"
#include <sys/types.h>
#include <unistd.h>
int main(int argc, char *argv[])
{
	QApplication a(argc, argv);
	MainWindow w;
	w.show();
	return a.exec();
}
```

mainwindow.cpp

```
/* Author:Hover
  * E-Mail:hover@hust.edu.cn
  * GitHub:HoverWings
  * Description:QtTest
  */
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent) :
	QMainWindow(parent),
	ui(new Ui::MainWindow)
{
	ui->setupUi(this);
	cyc=new cycle_Dialog();
	cyc->show();

	add=new add_Dialog();
	add->show();

	QTimer *timer=new QTimer(this);
	connect(timer,SIGNAL(timeout()),this,SLOT(timerUpdate()));
	timer->start(1000);
}

MainWindow::~MainWindow()
{
	delete ui;
}

void MainWindow::timerUpdate(void)
{
	//update time
	QDateTime time = QDateTime::currentDateTime();
	QString str = time.toString("yyyy-MM-dd hh:mm:ss dddd");
	ui->time_label->setText(str);

	//update cycle
	cyc->update_cycle();
	add->update_add();
```

```
    //update add
}

void MainWindow::cycle_fun()
{
    qDebug()<<"asdasd";
}
```

mycp

```
/* FileName:       mycp.cpp
  * Author:         Hover
  * E-Mail:         hover@hust.edu.cn
  * GitHub:         HoverWings
  * Description:    the linux cp of Hover's implementation
  */
#include <string.h>
#include <unistd.h> //unix std API
#include <stdio.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <dirent.h>
#include <stdlib.h>

#define BUFFERSIZE 1024
#define COPYMORE 0644      // the owner can w/r others can only r


int copyD2D(char *src, char *dest);
int copyF2F(char *src, char *dest);
bool isdir(char *filename);
char *strrev(char *str);

int main(int argc, char **argv)
{
        bool opt_r = false; // -r/-R:copy the dir recursively
        bool opt_l = false; // create the hard link
        bool opt_s = false; // create the soft link

        char *src = NULL;
        char *dest = NULL;

        char opt;

        while ((opt = getopt(argc, argv, "rRls")) != -1)
        {
                switch (opt)
                {
                        case 'R':
```

```
            case 'r':
                opt_r = true;
                break;

            case 'l':
                opt_l = true;
                break;

            case 's':
                opt_s = true;
                break;
        }
    }

    // check the arg nums
    if (optind >= argc - 1)
    {
        printf("missing Operator \n");
        exit(1);
    }

    // get src and dest
    src = argv[optind];
    dest = argv[optind + 1];

    if (opt_l)
    {
        if (isdir(src))
        {
            printf(" dir can not create the hard link \n");
            exit(1);
        }
        if ((link(src, dest)) == 0)
        {
            return 0;
        }
        else
        {
            printf("create hard link fail\n");
            exit(1);
```

```
        }
    }

    if (opt_s)
    {
        if (isdir(src)) //
        {
            printf("dir can not create the symbol link\n");
            exit(1);
        }

        if ((symlink(src, dest)) == 0) //symlink to create sybmol link
            return 0;
        else
        {
            printf("fail to create the symbol link\n");
            exit(1);
        }
    }


    if (!isdir(src))
    {
        // if not dir, process as file
        if ((copyF2F(src, dest)) == 0)
            return 0;
        else
        {
            printf("copy file fail\n");
            exit(1);
        }
    }
    else if (isdir(src))
    {
        if (!isdir(dest))
        {
            printf("can not copy a dir to a file\n");
            exit(1);
        }
        else if (isdir(dest) && opt_r)
```

```
            {
                if (copyD2D(src, dest) != 0)
                {
                    printf("copy dir fail\n");
                    exit(1);
                }
                else
                    return 0;
            }
            else
            {
                printf("you may need -R/-r opeartor to copy a dir\n");
                exit(1);
            }
        }
        else
        {
            printf("illegal operation");
            exit(1);
        }

        return 0;
}




int copyF2F(char *src_file, char *dest_file)
{
        int in_fd, out_fd, n_chars;
        char buf[BUFFERSIZE];

        // if dest is dir, then create the same name file in the dir defautly
        if (isdir(dest_file))
        {
            char c;
            char temp[10] = { '\0' };
            char *r_temp;
            int n = strlen(src_file);
            int m = 0;
```

```
        // the final level name as the dest name
        while ((c = src_file[n - 1]) != '/')
        {
                temp[m] = c;
                m++;
                n--;
        }
        r_temp = strrev(temp);
        strcat(dest_file, r_temp);
}

if ((in_fd = open(src_file, O_RDONLY)) == -1)
{
        printf("%s read file fail！",src_file);
        return 1;
}

if ((out_fd = open(dest_file, O_WRONLY | O_CREAT, COPYMORE)) == -1)
        return 1;

while ((n_chars = read(in_fd, buf, BUFFERSIZE)) > 0)
{
        if (write(out_fd, buf, n_chars) != n_chars)
        {
                printf("%s write file fail！", dest_file);
                return 1;
        }
        if (n_chars == -1)
        {
                printf("%s read file fail！", src_file);
                return 1;
        }
}

//close file
if (close(in_fd) == -1 || close(out_fd) == -1)
{
        printf("close file fail");
```

```
            return 1;
        }
        return 0;
}

//typedef struct __dirstream DIR;
//struct __dirstream
//{
//      void        *__fd; /* `struct hurd_fd' pointer for descriptor.      */
//      char        *__data;                    /* Directory block.      */
//      int             __entry_data;       /* Entry number `__data' corresponds to.      */
//      char            *__ptr;                     /* Current pointer into the block.      */
//      int            __entry_ptr;         /* Entry number `__ptr' corresponds to.      */
//      size_t            __allocation;           /* Space allocated for the block.      */
//      size_t        __size;                     /* Total valid data in the block.      */
//      __libc_lock_define (, __lock)             /* Mutex lock for this structure.      */
//};




/*
D:is dir check
*/
bool isdir(char *filename)
{
        struct stat fileInfo;
        if (stat(filename, &fileInfo) >= 0)
        {
            if (S_ISDIR(fileInfo.st_mode))
            {
                return true;
            }
            else
            {
                return false;
            }
        }
}
```

```
/*
D:copy dir
*/
int copyD2D(char *src_dir, char *dest_dir)
{
	DIR *dp = NULL;
	struct dirent *dirp;
	char tempDest[256];
	char tempSrc[256];
	strcpy(tempDest, dest_dir);
	strcpy(tempSrc, src_dir);


	//open dir
	if ((dp = opendir(src_dir)) == NULL)
		return 1;
	else
	{
		//get dirent
		while ((dirp = readdir(dp)))
		{
			struct stat file_stat;
			if (!isdir(dirp->d_name))
			{
				//link name
				strcat(tempDest, "/");
				strcat(tempSrc, "/");
				strcat(tempDest, dirp->d_name);
				strcat(tempSrc, dirp->d_name);
				// printf("%s\n",tempDest);
				// printf("%s\n",tempSrc);
				//copy file
				copyF2F(tempSrc, tempDest);

				//recover name
				strcpy(tempDest, dest_dir);
				strcpy(tempSrc, src_dir);
			}
		}
		//close dir
```

```
            closedir(dp);
            return 0;
        }
}
/*
D:to convert the string
*/
char * strrev(char *str)
{
        int i = strlen(str) - 1, j = 0;
        char ch;
        while (i>j)
        {
            ch = str[i];
            str[i] = str[j];
            str[j] = ch;
            i--;
            j++;
        }
        return str;
}
```

mySystemCall

mySystemKernel_make

```
#!/usr/bin/bash

KERNEL_VERSION="418"

# set compile arg
kernel_num=8
# Back edited file
cp arch/x86/entry/syscalls/syscall_64.tbl ../
cp kernel/sys.c ../

# make image
# make mrproper
# make menuconfig
make bzImage –j $KERNEL_VERSION

# make modules
```

```
make modules –j $kernel_num
make modules_install –j $kernel_num

# make install
make install –j $kernel_num

# copy kernel image to boot
cp arch/x86_64/boot/bzImage /boot/vmlinuz-linux$KERNEL_VERSION

# build initramfs
mkinitcpio -p linux$KERNEL_VERSION

# update-grub
grub-mkconfig -o /boot/grub/grub.cfg
```

sys.c

```
SYSCALL_DEFINE3(mycall,long,num,char*,str,int, MAX_LENGTH)
{
    mm_segment_t old_fs =get_fs();
    set_fs(KERNEL_DS);
    printk("asdas2");
    long k_num=0;
    int i=copy_from_user(&k_num,&num,sizeof(long));
    if(i==0)
    {
        printk("copy num from user sus!");
    }
    else
    {
        printk("copy num from user fail!");
    }
    printk("num %ld\n",num);
    printk("k_num %ld\n",k_num);

    char k_str[MAX_LENGTH];
    i=copy_from_user(k_str,str,MAX_LENGTH*sizeof(char));
    if(i==0)
    {
        printk("copy str from user sus!");
    }
```

```
    else
    {
        printk("copy str from user fail!");
    }
    // printk("str %s\n",str);
    printk("k_str %s\n",k_str);
    set_fs(old_fs);
    return k_num;
}


SYSCALL_DEFINE2(mycopy, const char *, src, const char *, dst)
{
    int MAX_LENGTH=256;
    struct kstat k_buf;
    char buf[MAX_LENGTH];
    int read_fd, write_fd;
    long read_num;

    //save old fs
    mm_segment_t old_fs =get_fs();
    set_fs(KERNEL_DS);

    int i=0;
    //copy src name
    char _src[MAX_LENGTH];
    i=copy_from_user(_src,src,MAX_LENGTH*sizeof(char));
    if(i==0)
    {
        printk("copy src from user sus!");
    }
    else
    {
        printk("copy src from user fail!");
        set_fs(old_fs);
        return -1;
    }

    //copy dst name
    char _dst[MAX_LENGTH];
```

```
    i=copy_from_user(_dst,dst,MAX_LENGTH*sizeof(char));
    if(i==0)
    {
        printk("copy dst from user sus!");
    }
    else
    {
        printk("copy dst from user fail!");
        set_fs(old_fs);
        return -2;
    }


    // check the src mode
    if (vfs_stat(_src, &k_buf) != 0)
    {
        set_fs(old_fs);
        return -3;
    }

    // open src
    if ((read_fd = ksys_open(_src, O_RDONLY, S_IRUSR)) == -1)
    {
        set_fs(old_fs);
        return -3;
    }
    if ((write_fd = ksys_open(_dst, O_WRONLY | O_CREAT | O_TRUNC,
k_buf.mode)) == -1)
    {
        set_fs(old_fs);
        return -3;
    }

    // until the read_num = 0
    while(1)
    {
        read_num = ksys_read(read_fd, buf, sizeof(buf));
        if (read_num < 0)
        {
            set_fs(old_fs);
```

```
            return -4;
        }
        else if (read_num == 0)
        {
            break;
        }
        ksys_write(write_fd, buf, read_num);
    }

    ksys_close(read_fd);
    ksys_close(write_fd);

    //return to back fs
    set_fs(old_fs);
    return 0;
}
```

mycp_test.c

```
/* FileName:       mycp_test.c
  * Author:         Hover
  * E-Mail:         hover@hust.edu.cn
  * GitHub:         HoverWings
  * Description: tesy my linux call
  */
#include <stdio.h>
#include <unistd.h>

int main(int argc, char **argv)
{
    if (argc != 3)
    {
        printf("arg num error! \nusage: mycp_test <src> <dst>\n");
        return 0;
    }
    printf("src: %s\n",argv[1]);
    printf("dst: %s\n",    argv[2]);
    long ret = syscall(336, argv[1], argv[2]);
    if(ret!=0)
    {
        printf("error code :%d\n",ret);
```

```
        }
        else
        {
            printf("mycp finished!");
        }
    return 0;
}
```

myDevDriver

myDevDriver.c

```
#include "linux/kernel.h"
#include "linux/module.h"
#include "linux/fs.h"
#include "linux/init.h" // dd init and exit
#include "linux/types.h"
#include "linux/errno.h"
#include <linux/uaccess.h>
#include <linux/kdev_t.h>
/* FileName:       myDevDriver.c
  * Author:          Hover
  * E-Mail:          hover@hust.edu.cn
  * GitHub:          HoverWings
  * Description: myDevDriver
  */
#include <linux/types.h>

#define MAX_SIZE 1024

int my_open(struct inode *inode, struct file *file);
int my_release(struct inode *inode, struct file *file);
ssize_t my_read(struct file *file, char __user *user, size_t t, loff_t *f);
ssize_t my_write(struct file *file, const char __user *user, size_t t, loff_t *f);

char message[MAX_SIZE] = "---------Hover'sDriver---------";    // the message buffer
for text device
int devNum;
char* devName = "myDevDrive";

struct file_operations pStruct=
{
    open : my_open,
    release : my_release,
    read : my_read,
    write : my_write,
};

/*
static struct char_device_struct
```

```
{
    struct char_device_struct *next;
    unsigned int major;
    unsigned int baseminor;
    int minorct;
    char name[64];
    struct cdev *cdev;          // will die
} *chrdevs[CHRDEV_MAJOR_HASH_SIZE];
*/

//D:init module
//I:devNum:0 present dynamic alloc
//    devName
//    fOp_ptr
//O:init result
int init_module()
{
    int ret = register_chrdev(0,devName,&pStruct);
    if(ret < 0)
    {
        printk("regist fail!\n");
        return -1;
    }
    else
    {
        printk("myDevDrive has been registered!\n");
        devNum = ret;
        // debug information
        printk("myDevDrive's id: %d\n",ret);
        printk("usage: mknod /dev/myDevDrive c %d 0\n",devNum);
        printk("delete device\n\t usage: rm /dev/%s ",devName);
        printk("delete module\n\t usage: rmmode device ");
        return 0;
    }

}

//D: unregister module
//I: devNum, devName
void unregister_module(void)
```

```
{
	unregister_chrdev(devNum,devName);
	printk("unregister success !\n");
}

int my_open(struct inode *inode, struct file *file)
{
	printk("open myDrive OK ! \n");
	try_module_get(THIS_MODULE);
	return 0;
}


int my_release(struct inode *inode, struct file *file)
{
	atomic_set(&mod->refcnt, 1);
	printk("Device released !\n");
	module_put(THIS_MODULE);	//Reference amount minus 1
	return 0;
}

ssize_t my_read(struct file *file, char __user *user, size_t t, loff_t *f)
{
	if (copy_to_user(user ,message,sizeof(message)))
	{
		return -2;
	}
	return sizeof(message);
}

ssize_t my_write(struct file *file, const char __user *user, size_t t, loff_t *f)
{
	if(copy_from_user(message,user,sizeof(message)))
	{
		return -3;
	}
	return sizeof(message);
}
```

myDevDriver_test.c

```
#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

int main()
{
    int fd;
    char buf[MAX_SIZE];
    char get[MAX_SIZE]; // to be written

    char devName[20];
    char dir[50] = "/dev/";
    system("ls /dev/");
    printf("Please input the device's name: ");
    gets(devName);
    strcat(dir, devName);
    fd = open(dir, O_RDWR | O_NONBLOCK);
    if( fd != -1)
    {
        // get str from buf
        read( fd , buf , sizeof(buf) );
        printf( "%s\n" , buf);

          // read
        printf( "Please input a string : ");
        gets(get);
        write( fd , get , sizeof(get) );

        // read back
        read(fd, buf, sizeof(buf));
        printf("device Message : %s\n", buf);

        close(fd);
```

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

}
```

myLinuxMonitor

myLinuxMonitor.pro

```
#-------------------------------------------------
#
# Project created by QtCreator 2018-04-22T22:13:21
#
#-------------------------------------------------

QT += core gui
QT          +=charts
QT += widgets

CONFIG      += c++11

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = myLinuxMonitor
TEMPLATE = app

# The following define makes your compiler emit warnings if you use
# any feature of Qt which has been marked as deprecated (the exact warnings
# depend on your compiler). Please consult the documentation of the
# deprecated API in order to know how to port your code away from it.
DEFINES += QT_DEPRECATED_WARNINGS

# You can also make your code fail to compile if you use deprecated APIs.
# In order to do so, uncomment the following line.
# You can also select to disable deprecated APIs only up to a certain version of Qt.
#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000        # disables
all the APIs deprecated before Qt 6.0.0

#debug: LIBS+= processes.o

SOURCES += \
            main.cpp \
            mainwindow.cpp \
        filesystems.cpp \
        processes.cpp \
        progressbardelegate.cpp \
        tableview.cpp \
```

```
        tablemodel.cpp \
        netinfo.cpp

HEADERS += \
                mainwindow.h \
        filesystems.h \
        processes.h \
        progressbardelegate.h \
        tableview.h \
        tablemodel.h \
        netinfo.h

FORMS += \
                mainwindow.ui

DISTFILES +=

RESOURCES += \
        rsc.qrc
```

filesystems.h

```
/* FileName:       filesystems.h
  * Author:          Hover
  * E-Mail:          hover@hust.edu.cn
  * GitHub:          HoverWings
  * Description:FileSystem Module
  */
#ifndef FILESYSTEMS_H
#define FILESYSTEMS_H

#include <QString>
#include <QTimer>
#include <QProcess>
#include <QDebug>
#include <QObject>
#include "sys/statfs.h"
#include <stdio.h>
#include <mntent.h>
#include <string.h>
#include <sys/vfs.h>
```

```
static const unsigned long long G = 1024*1024*1024ull;
static const unsigned long long M = 1024*1024;
static const unsigned long long K = 1024;
static char str[20];


extern int device_num;
extern char mount_on_device[20];
extern QString device_info[20][6];

char* kscale(unsigned long b, unsigned long bs);
int mydf();



#endif //FILESYSTEMS_H
```

filesystems.cpp

```
/* FileName:filesystems.cpp
  * Author:Hover
  * E-Mail:hover@hust.edu.cn
  * GitHub:HoverWings
  * Description:FileSystem Module
  */
#include "filesystems.h"

int device_num;
char mount_on_device[20];
QString device_info[20][6];


char* kscale(unsigned long b, unsigned long bs)
{
	unsigned long long size = b * (unsigned long long)bs;
	if (size > G)
	{
		sprintf(str, "%0.2f GB", size/(G*1.0));
		return str;
	}
	else if (size > M)
```

```
        {
                sprintf(str, "%0.2f MB", size/(1.0*M));
                return str;
        }
        else if (size > K)
        {
                sprintf(str, "%0.2f K", size/(1.0*K));
                return str;
        }
        else
        {
                sprintf(str, "%0.2f B", size*1.0);
                return str;
        }
}

int mydf()
{
        device_num=0;
        char str[500];
        FILE* mount_table;
        struct mntent *mount_entry;
        struct statfs s;
        unsigned long blocks_used;
        unsigned blocks_percent_used;
        const char *disp_units_hdr = NULL;
        mount_table = NULL;
        mount_table = setmntent("/etc/mtab", "r");
        if (!mount_table)
        {
                fprintf(stderr, "set mount entry error/n");
                return -1;
        }
        disp_units_hdr = "          Size";
//      printf("Filesystem                      %-15sUsed Available %s Mounted on/n",
//                      disp_units_hdr, "Use%");
        while (1)
        {
                const char *device;
                const char *mount_point;
```

```
        if (mount_table)
        {
            mount_entry = getmntent(mount_table);
            if (!mount_entry)
            {
                endmntent(mount_table);
                break;
            }
        }
        else
            continue;
        device = mount_entry->mnt_fsname;
        mount_point = mount_entry->mnt_dir;
        //fprintf(stderr, "mount info: device=%s mountpoint=%s/n", device,
mount_point);
        if (statfs(mount_point, &s) != 0)
        {
            fprintf(stderr, "statfs failed!/n");
            continue;
        }
        if ((s.f_blocks > 0) || !mount_table )
        {
            blocks_used = s.f_blocks - s.f_bfree;
            blocks_percent_used = 0;
            if (blocks_used + s.f_bavail)
            {
                blocks_percent_used = (blocks_used * 100ULL+ (blocks_used +
s.f_bavail)/2) / (blocks_used + s.f_bavail);
            }
            /* GNU coreutils 6.10 skips certain mounts, try to be compatible.    */
            if (strcmp(device, "rootfs") == 0)
                continue;
            if (printf("/n%-20s" + 1, device) > 20)
                    printf("/n%-20s", "");
            char s1[20];
            char s2[20];
            char s3[20];
            strcpy(s1, kscale(s.f_blocks, s.f_bsize));
            strcpy(s2, kscale(s.f_blocks - s.f_bfree, s.f_bsize));
            strcpy(s3, kscale(s.f_bavail, s.f_bsize));
```

```
                    device_info[device_num][0]=QString(device);
                    device_info[device_num][1]=QString(s1);
                    device_info[device_num][2]=QString(s2);
                    device_info[device_num][3]=QString(s3);
//
qDebug()<<device_num<<"blocks_percent_used"<<blocks_percent_used;
                    device_info[device_num][4]=QString::number(blocks_percent_used);
                    device_info[device_num][5]=QString(mount_point);
                    device_num++;
                }

        }
        return device_num;
//        printf("%s",str);
}
```

mainwindow.h

```
/* FileName:       mainwindow.h
  * Author:          Hover
  * E-Mail:          hover@hust.edu.cn
  * GitHub:          HoverWings
  * Description: mainwindow
  */
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QDir>
#include <QDebug>
#include <QFile>
#include <QTabBar>
#include <QWidget>
#include <QTabWidget>
#include <QMessageBox>
#include <QStandardItemModel>
#include <QTableView>
//#include <Qtab>
#include <QProcess>
#include <QTableWidget>
#include <QTableWidgetItem>
#include <QPoint>
```

```
#include <QTimer>
#include <qmath.h>
#include <QGraphicsView>
#include <QChartView>
#include <QLineSeries>
#include <QScatterSeries>
#include <QValueAxis>
#include <QtCharts>
#include "processes.h"
#include "filesystems.h"
#include "progressbardelegate.h"
#include "tablemodel.h"
#include "tableview.h"
#include "netinfo.h"
namespace Ui
{
    class MainWindow;
}
class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    TableView *tv;
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();

    QChart *cpu_chart;
    QLineSeries *cpu_series;
    QChart *mem_chart;
    QLineSeries *mem_series;
    QLineSeries *swap_series;
    QChart *net_chart;
    QLineSeries *upload_series;
    QLineSeries *download_series;


    QLabel *receiving_label;
    QLabel * totalReceived_label;
    QLabel *sending_label;
```

```
        QLabel * totalsent_label;
        int maxSize = 60;
        int timeId;
        int process_timeId;
        bool set_Process=true;
        QStandardItemModel *process_model;
//        QStandardItemModel *fs_model;
        // process info
        QModelIndex focus_index;
        QString process_focus_pid="0";
        int process_focou_row=0;
        int process_focus_col=0;
        bool selected=false;

        // mem info var
        QFile memFile; //用于打开系统文件
        QString memTotal;
        QString memFree;
        QString memUsed;
        QString swapTotal;
        QString swapFree;
        QString swapUsed;
        QString MemAvailable;
        int nMemAvailable,nMemTotal, nMemFree, nMemUsed, nSwapTotal,
nSwapFree, nSwapUsed;
        QString tempStr;
        int pos;
        QPieSeries *mem_pieseries;
        QPieSeries *swap_pieseries;
        QChartView *mem_piechartView;
        QChartView *swap_piechartView;

        TableView* fs_view;
        TableModel* fs_model;
        QValueAxis *net_axisY;
        int net_max_ax;


public slots:
//        void update_resources();
```

```
        void onHeaderClicked(int _colNum);
        void searchModelandItem(QString ID);
        void rowDoubleClicked(const QModelIndex index);
private slots:
//        void on_tabWidget_currentChanged(int index);
        void on_kill_pushButton_clicked();

        void on_tabWidget_currentChanged(int index);


        void on_Process_tableView_clicked(const QModelIndex &index);

//        void on_lineEdit_editingFinished();

        void on_lineEdit_returnPressed();

private:
        int _colNum;
        Ui::MainWindow *ui;

        void setProcess(bool update_process);

        void setRescources();
        void setMem();
        void setNet();
        void setFileSystem(int device_num);
        void updateFileSystem();
        void show_procs(bool update_process);
        double getData(double time);

        void updateCPUGraph();
        void updateMemGraph();
        void updateSwapGraph();
        void updateNetGraph();



protected:
        void timerEvent(QTimerEvent *event) Q_DECL_OVERRIDE;
};
```

```
#endif // MAINWINDOW_H
```

mainwindow.cpp

```
/* FileName:mainwindow.cpp
  * Author:Hover
  * E-Mail:hover@hust.edu.cn
  * GitHub:HoverWings
  * Description: Draw Mainwindow
  */
#include "mainwindow.h"
#include "ui_mainwindow.h"

QT_CHARTS_USE_NAMESPACE

#include "processes.h"
/*
  * us — 用户空间占用 CPU 的百分比。
  * sy — 内核空间占用 CPU 的百分比。
  * ni — 改变过优先级的进程占用 CPU 的百分比
  * id — 空闲 CPU 百分比
  * wa — IO 等待占用 CPU 的百分比
  * hi — 硬中断（Hardware IRQ）占用 CPU 的百分比
  * si — 软中断（Software Interrupts）占用 CPU 的百分比
*/

MainWindow::MainWindow(QWidget *parent) :
        QMainWindow(parent),
        ui(new Ui::MainWindow)
{
        ui->setupUi(this);
        process_model= new QStandardItemModel();
        net_init();

        setRescources();


ui->Process_tableView->setSelectionBehavior(QAbstractItemView::SelectRows);
//single line
```

```
ui->Process_tableView->setSelectionMode( QAbstractItemView::SingleSelection);//s
ingle choose
    ui->Process_tableView->setEditTriggers
( QAbstractItemView::NoEditTriggers );
    QHeaderView *hv = ui->Process_tableView->horizontalHeader();
    hv->setSortIndicatorShown(true); // set sort avaliable
    hv->sectionsClickable();              // set title clickable
    connect(hv, SIGNAL(sectionClicked(int)), this, SLOT(onHeaderClicked(int)));

    fs_view=new TableView(this);
    fs_model=fs_view->tableModel();
    fs_view->setModel(fs_model);
    fs_view->setEditTriggers ( QAbstractItemView::NoEditTriggers );
    fs_view->setSelectionBehavior(QAbstractItemView::SelectRows);     //single
line
    fs_view->setSelectionMode(QAbstractItemView::SingleSelection);    //single
choose
    QStringList fs_titleList;
    fs_titleList<<QString("Device")
                <<QString("Total")
                <<QString("Used")
                <<QString("Avaliable")
                <<QString("Persentage")
                <<QString("Directory")
                <<QString(" ");
    fs_model->setHorizontalHeader(fs_titleList);
    QVBoxLayout* layout = new QVBoxLayout();
    layout->addWidget(fs_view);
    ui->tab_3->setLayout(layout);
    connect(fs_view, SIGNAL(doubleClicked(const QModelIndex &)), this,
SLOT(rowDoubleClicked(const QModelIndex &)));

    max_procs = 0;
    delay = 1;
    iterations = -1;

    free_procs = NULL;

    num_new_procs = num_old_procs = 0;
```

```
    new_procs = old_procs = NULL;

    int device_num=mydf();
    setFileSystem(device_num);

    //set cpu
    old_procs = new_procs;
    num_old_procs = num_new_procs;
    memcpy(&old_cpu, &new_cpu, sizeof(old_cpu));
    read_procs();
    setProcess(true);
    free_old_procs();

}


void MainWindow::rowDoubleClicked(const QModelIndex index)
{
    QVector<QStringList>& data = fs_model->DataVector();
    if (index.isValid())
    {
        QString path=data[index.row()][5];
        QProcess *path_proc = new QProcess();
        path_proc->start("nautilus "+path);
    }
}

void MainWindow::onHeaderClicked(int colNum)
{
//      qDebug()<<"colNum"<<colNum;
    _colNum=colNum;
}


MainWindow::~MainWindow()
{

}

void MainWindow::show_procs(bool update_process)
```

```
{
	if(update_process)
	{
		process_model->clear();;
	}
	int i;
	struct proc_info *old_proc, *proc;
	long unsigned total_delta_time;
	struct passwd *user;
	struct group *group;
	char *user_str, user_buf[20];
	char *group_str, group_buf[20];

	for (i = 0; i < num_new_procs; i++)
	{
		if (new_procs[i])
		{
			old_proc = find_old_proc(new_procs[i]->pid, new_procs[i]->tid);
			if (old_proc)
			{
				new_procs[i]->delta_utime = new_procs[i]->utime -
old_proc->utime;
				new_procs[i]->delta_stime = new_procs[i]->stime -
old_proc->stime;
			}
			else
			{
				new_procs[i]->delta_utime = 0;
				new_procs[i]->delta_stime = 0;
			}
			new_procs[i]->delta_time = new_procs[i]->delta_utime +
new_procs[i]->delta_stime;
		}
	}

	total_delta_time = (new_cpu.utime + new_cpu.ntime + new_cpu.stime +
new_cpu.itime
						+ new_cpu.iowtime + new_cpu.irqtime +
new_cpu.sirqtime)
					- (old_cpu.utime + old_cpu.ntime + old_cpu.stime +
```

```
old_cpu.itime
                                    + old_cpu.iowtime + old_cpu.irqtime +
old_cpu.sirqtime);


    // change proc_cmp while call
    qsort(new_procs, num_new_procs, sizeof(struct proc_info *), proc_cmp);
    cpu_user= ((new_cpu.utime + new_cpu.ntime) - (old_cpu.utime +
old_cpu.ntime)) * 100    / total_delta_time;
    cpu_sys= ((new_cpu.stime ) - (old_cpu.stime)) * 100 / total_delta_time;
    if(!update_process)
    {
        return;
    }
    //Print Title
    QStringList titleList;
    if (!threads)
    {
        titleList<<QString("PID")
                <<QString("CPU%")
                <<QString("S")
                <<QString("#THR")
                <<QString("VSS")
                <<QString("RSS")
                <<QString("UID")
                <<QString("Name");
    }
    else
    {
        titleList<<QString("PID")
                    <<QString("TID")
                    <<QString("CPU%")
                    <<QString("S")
                    <<QString("VSS")
                    <<QString("RSS")
                    <<QString("UID")
                    <<QString("Thread")
                    <<QString("Proc");
    }
    if(set_Process==true)
```

```
    {
        process_model->setHorizontalHeaderLabels(titleList);
//          set_Process=false;
    }

    QList<QStandardItem*> strList;
    for (i = 0; i < num_new_procs; i++)
    {
        proc = new_procs[i];

        if (!proc || (max_procs && (i >= max_procs)))
            break;
        user    = getpwuid(proc->uid);
        group = getgrgid(proc->gid);
        if (user && user->pw_name)
        {
            user_str = user->pw_name;
        }
        else
        {
            snprintf(user_buf, 20, "%d", proc->uid);
            user_str = user_buf;
        }
        if (group && group->gr_name)
        {
            group_str = group->gr_name;
        }
        else
        {
            snprintf(group_buf, 20, "%d", proc->gid);
            group_str = group_buf;
        }
        if (!threads)
        {
            strList<<new QStandardItem(QString::number(proc->pid))
                    <<new QStandardItem(QString::number(proc->delta_time *
100 / total_delta_time))
                    <<new QStandardItem(QString::number(proc->state))
                    <<new QStandardItem(QString::number(proc->num_threads))
                    <<new QStandardItem(QString::number(proc->vss / 1024))
```

```
                        <<new QStandardItem(QString::number(proc->rss *
getpagesize() / 1024))
//                          <<new QStandardItem(proc->policy)
                        <<new QStandardItem(user_str)
                        <<new QStandardItem(proc->name[0] != 0 ? proc->name :
proc->tname);
                process_model->appendRow(strList); //在第 0 行插入一条记录
                strList.clear();
            }
            else
            {
                strList<<new QStandardItem(proc->pid)
                        <<new QStandardItem(proc->tid)
                        <<new QStandardItem(proc->delta_time * 100 /
total_delta_time)
                        <<new QStandardItem(proc->state)
                        <<new QStandardItem(proc->vss / 1024)
                        <<new QStandardItem(proc->rss * getpagesize() / 1024)
//                          <<new QStandardItem(proc->policy)
                        <<new QStandardItem(user_str)
                        <<new QStandardItem(proc->tname)
                        <<new QStandardItem(proc->name);
                process_model->appendRow(strList);
                strList.clear();
            }
        }

}

void MainWindow::setProcess(bool update_process)
{
        show_procs(update_process);
        ui->Process_tableView->setModel(process_model);
        ui->Process_tableView->setSortingEnabled(true);
        ui->Process_tableView->sortByColumn(_colNum);
        // search to focous
//        ui->Process_tableView->findChild(process_focus_pid);
        //setfocus
        if(selected)
        {
```

```
            searchModelandItem(process_focus_pid);
            ui->Process_tableView->selectRow(focus_index.row());
            //set scroll
            int maxValue = ui->Process_tableView->verticalScrollBar()->maximum();
           // 当前 SCROLLER 最大显示值
            int pageValue =
ceil(((focus_index.row())*maxValue/ui->Process_tableView->model()->rowCount()))
;

ui->Process_tableView->verticalScrollBar()->setSliderPosition(pageValue);
        }
}


void MainWindow::setRescources()
{
        //setCPU
        cpu_chart = new QChart;
        QChartView *cpu_chartView = new QChartView(cpu_chart);
        cpu_chartView->setRubberBand(QChartView::RectangleRubberBand);

        cpu_series = new QLineSeries;
        cpu_chart->addSeries(cpu_series);

        for(int i=0;i<maxSize;++i)
        {
            cpu_series->append(i,0);
        }
        cpu_series->setUseOpenGL(true);//openGl 加速
//        qDebug()<<cpu_series->useOpenGL();

        QValueAxis *cpu_axisX = new QValueAxis;
        cpu_axisX->setRange(0,maxSize);
        cpu_axisX->setLabelFormat("%g");

        QValueAxis *cpu_axisY = new QValueAxis;
        cpu_axisY->setRange(0,100);
        cpu_axisY->setTitleText("CPU Rate");

        cpu_chart->setAxisX(cpu_axisX,cpu_series);
```

```
    cpu_chart->setAxisY(cpu_axisY,cpu_series);
    cpu_chart->legend()->hide();
    cpu_chart->setTitle("CPU History");

    QVBoxLayout *layout = new QVBoxLayout();
    layout->setContentsMargins(0, 0, 0, 0);
    layout->addWidget(cpu_chartView);


    //setMemory and swap
    mem_chart = new QChart;
    QChartView *mem_chartView = new QChartView(mem_chart);
    mem_chartView->setRubberBand(QChartView::RectangleRubberBand);

    mem_series = new QLineSeries;
    mem_chart->addSeries(mem_series);

    for(int i=0;i<maxSize;++i)
    {
        mem_series->append(i,0);
    }
    mem_series->setUseOpenGL(true);
//    qDebug()<<mem_series->useOpenGL();

    QValueAxis *mem_axisX = new QValueAxis;
    mem_axisX->setRange(0,maxSize);
    mem_axisX->setLabelFormat("%g");

    QValueAxis *mem_axisY = new QValueAxis;
    mem_axisY->setRange(0,100);
    mem_axisY->setTitleText("Memory Rate");

    mem_chart->setAxisX(mem_axisX,mem_series);
    mem_chart->setAxisY(mem_axisY,mem_series);

    // SWAP

    swap_series = new QLineSeries;
    mem_chart->addSeries(swap_series);
```

```
for(int i=0;i<maxSize;++i)
{
    swap_series->append(i,0);
}
swap_series->setUseOpenGL(true);//openGl
qDebug()<<swap_series->useOpenGL();
mem_chart->setAxisX(mem_axisX,swap_series);
mem_chart->setAxisY(mem_axisY,swap_series);
mem_chart->legend()->hide();
mem_chart->setTitle("Memory History");
layout->addWidget(mem_chartView);


//add pie mem
QHBoxLayout *Hbox=new QHBoxLayout();//水平布局管理器（父管理器）；
mem_pieseries = new QPieSeries();
mem_pieseries->setHoleSize(0.35);

QChartView *mem_piechartView = new QChartView();
mem_piechartView->setRenderHint(QPainter::Antialiasing);
mem_piechartView->chart()->setTitle("Mem");
mem_piechartView->chart()->addSeries(mem_pieseries);
mem_piechartView->chart()->legend()->setAlignment(Qt::AlignBottom);
mem_piechartView->chart()->setTheme(QChart::ChartThemeBlueCerulean);
mem_piechartView->chart()->legend()->setFont(QFont("Arial", 7));
Hbox->addWidget(mem_piechartView);


swap_pieseries = new QPieSeries();
swap_pieseries->setHoleSize(0.35);
swap_piechartView = new QChartView();
swap_piechartView->setRenderHint(QPainter::Antialiasing);
swap_piechartView->chart()->setTitle("SWAP");
swap_piechartView->chart()->addSeries(swap_pieseries);
swap_piechartView->chart()->legend()->setAlignment(Qt::AlignBottom);
swap_piechartView->chart()->setTheme(QChart::ChartThemeBlueCerulean);
swap_piechartView->chart()->legend()->setFont(QFont("Arial", 7));
Hbox->addWidget(swap_piechartView);

layout->addLayout(Hbox);
```

```
//      set net
//      download
        net_chart = new QChart;
        QChartView *net_chartView = new QChartView(net_chart);
        layout->addWidget(net_chartView);
        net_chartView->setRubberBand(QChartView::RectangleRubberBand);

        download_series = new QLineSeries;
        net_chart->addSeries(download_series);

        for(int i=0;i<maxSize;++i)
        {
            download_series->append(i,0);
        }
//      download_series->setUseOpenGL(true);//openGl

        QValueAxis *net_axisX = new QValueAxis;
        net_axisX->setRange(0,maxSize);
        net_axisX->setLabelFormat("%g");

        net_axisY = new QValueAxis;
        net_axisY->setRange(0,100);
        net_axisY->setTitleText("Net Rate");

        net_chart->setAxisX(net_axisX,download_series);
//      // upload
        upload_series = new QLineSeries;
        net_chart->addSeries(upload_series);
        for(int i=0;i<maxSize;++i)
        {
            upload_series->append(i,0);
        }
        upload_series->setUseOpenGL(true);//openGl 加速
//      qDebug()<<swap_series->useOpenGL();
        net_chart->setAxisX(net_axisX,upload_series);
        net_chart->setAxisY(net_axisY,upload_series);
        net_chart->setAxisY(net_axisY,download_series);
```

```
//      net_chart->legend()->hide();
        net_chart->setTitle("Network History");


        //set net info
//      QHBoxLayout *net_Hbox=new QHBoxLayout();
        QGridLayout *gridLayout = new QGridLayout;
        QLabel* download_label = new QLabel();
        QImage *download_img = new QImage(":/icon/download.jpg");
        QImage *scaled_logo_img = new QImage();
        *scaled_logo_img=download_img->scaled(32,32,Qt::KeepAspectRatio);
        download_label->setPixmap(QPixmap::fromImage(*scaled_logo_img));
//      net_Hbox->addWidget(download_label);
        gridLayout->addWidget(download_label,0,0);
        gridLayout->setSpacing(0);
        QVBoxLayout *d_Vbox=new QVBoxLayout();
        receiving_label=new QLabel();
        receiving_label->setText("Receiving");
        totalReceived_label=new QLabel();
        totalReceived_label->setText("Total Received");
        d_Vbox->addWidget(receiving_label);
        d_Vbox->addWidget(totalReceived_label);
        gridLayout->addWidget(receiving_label,0,2,1,3);
        gridLayout->addWidget(totalReceived_label,1,2,1,3);
//      net_Hbox->addLayout(d_Vbox);


        QLabel* upload_label = new QLabel();
        QImage *upload_img = new QImage(":/icon/upload.jpg");
        *scaled_logo_img=upload_img->scaled(32,32,Qt::KeepAspectRatio);
        upload_label->setPixmap(QPixmap::fromImage(*scaled_logo_img));
        gridLayout->addWidget(upload_label,0,3);

        sending_label=new QLabel();
        sending_label->setText("Sending");
        totalsent_label=new QLabel();
        totalsent_label->setText("Total Sent");
        gridLayout->addWidget(sending_label,0,4,1,3);
        gridLayout->addWidget(totalsent_label,1,4);
```

```
    layout->addLayout(gridLayout);

//    layout->setStretchFactor(cpu_chartView, 8);
//    layout->setStretchFactor(mem_chartView, 8);
//    layout->setStretchFactor(mem_piechartView,8);
//    layout->setStretchFactor(net_chartView, 8);
//    layout->setStretchFactor(net_Hbox, 1);



    ui->tab_2->setLayout(layout);


    timeId = startTimer(1000);
    process_timeId=startTimer(10000);
}


void MainWindow::setMem()
{
    memFile.setFileName("/proc/meminfo"); //打开内存信息文件
    if (!memFile.open(QIODevice::ReadOnly) )
    {
        QMessageBox::warning(this, tr("warning"), tr("The meminfo file can not
open!"), QMessageBox::Yes);
        return ;
    }
    while (1)
    {
        tempStr = memFile.readLine();
//        qDebug()<<"mememem"<<tempStr;
        pos = tempStr.indexOf("MemTotal");
        if (pos != -1)
        {
            memTotal = tempStr.mid(pos+10, tempStr.length()-13);
            memTotal = memTotal.trimmed();
            nMemTotal = memTotal.toInt()/1024;
        }
```

```
            else if (pos = tempStr.indexOf("MemFree"), pos != -1)
            {
                memFree = tempStr.mid(pos+9, tempStr.length()-12);
                memFree = memFree.trimmed();
                nMemFree = memFree.toInt()/1024;
//                  qDebug()<<nMemFree<<"memem";
            }
            else if (pos = tempStr.indexOf("SwapTotal"), pos != -1)
            {
                swapTotal = tempStr.mid(pos+11, tempStr.length()-14);
                swapTotal = swapTotal.trimmed();
                nSwapTotal = swapTotal.toInt()/1024;
            }
            else if(pos = tempStr.indexOf("MemAvailable"), pos != -1)
            {
//                  qDebug()<<tempStr;
                MemAvailable = tempStr.mid(pos+14, tempStr.length()-17);
//                  qDebug()<<MemAvailable;
                MemAvailable = MemAvailable.trimmed();
                nMemAvailable = MemAvailable.toInt()/1024;
//                  qDebug()<<nMemAvailable<<"nMemAvailable";
            }
            else if (pos = tempStr.indexOf("SwapFree"), pos != -1)
            {
                swapFree = tempStr.mid(pos+10,tempStr.length()-13);
                swapFree = swapFree.trimmed();
                nSwapFree = swapFree.toInt()/1024;
                break;
            }
        }
        nMemUsed = nMemTotal - nMemAvailable;
        nSwapUsed = nSwapTotal - nSwapFree;
        memUsed = QString::number(nMemUsed, 10);
        swapUsed = QString::number(nSwapUsed, 10);
        memFree = QString::number(nMemFree, 10);
        memTotal = QString::number(nMemTotal, 10);
        swapFree = QString::number(nSwapFree, 10);
        swapTotal = QString::number(nSwapTotal, 10);
        memFile.close();
    }
```

```
void MainWindow::updateCPUGraph()
{
    double cpu_data=cpu_user+cpu_sys;
    QTime dataTime(QTime::currentTime());
    long int eltime = dataTime.elapsed();
    int lastpointtime = 0;
    //int size = (eltime - lastpointtime);//数据个数
    int size=1;
    if(isVisible())
    {
        //update cpu graph
        QVector<QPointF> oldPoints = cpu_series->pointsVector();//Returns the
points in the series as a vector
        QVector<QPointF> points;
        for(int i=size;i<oldPoints.count();++i)
        {
            points.append(QPointF(i-size ,oldPoints.at(i).y()));//替换数据用
        }
        qint64 sizePoints = points.count();
        points.append(QPointF(sizePoints,cpu_data));
        cpu_series->replace(points);
        lastpointtime = eltime;
    }
}


void MainWindow::updateMemGraph()
{
    double mem_data=100*(nMemTotal-nMemAvailable)/nMemTotal;
    QTime dataTime(QTime::currentTime());
    long int eltime = dataTime.elapsed();
    int lastpointtime = 0;
    //int size = (eltime - lastpointtime);//数据个数
    int size=1;
    if(isVisible())
    {
        //update cpu graph
```

```
            QVector<QPointF> oldPoints = mem_series->pointsVector();//Returns the
points in the series as a vector
            QVector<QPointF> points;
            for(int i=size;i<oldPoints.count();++i)
            {
                points.append(QPointF(i-size ,oldPoints.at(i).y()));//替换数据用
            }
            qint64 sizePoints = points.count();
            points.append(QPointF(sizePoints,mem_data));
            mem_series->replace(points);
            lastpointtime = eltime;
        }
        mem_pieseries->clear();

mem_pieseries->append("MemUsed:"+QString::number(nMemUsed/1024)+"GiB("+
QString::number(100*(float)nMemUsed/nMemTotal)+"%) of
"+QString::number(nMemTotal/1024)+"GiB",nMemUsed);
        mem_pieseries->append("", nMemAvailable);

}

void MainWindow::updateSwapGraph()
{
        double swap_data=100*nSwapUsed/nSwapTotal;
        QTime dataTime(QTime::currentTime());
        long int eltime = dataTime.elapsed();
        int lastpointtime = 0;
        //int size = (eltime - lastpointtime);//数据个数
        int size=1;
        if(isVisible())
        {
            //update cpu graph
            QVector<QPointF> oldPoints = swap_series->pointsVector();//Returns the
points in the series as a vector
            QVector<QPointF> points;
            for(int i=size;i<oldPoints.count();++i)
            {
                points.append(QPointF(i-size ,oldPoints.at(i).y()));//替换数据用
            }
```

```
            qint64 sizePoints = points.count();
            points.append(QPointF(sizePoints,swap_data));
            swap_series->replace(points);
            lastpointtime = eltime;
        }
        swap_pieseries->clear();

swap_pieseries->append("SwapUsed:"+QString::number(nSwapUsed/1024)+"GiB("+
QString::number(100*(float)nSwapUsed/nSwapTotal)+"%) of
"+QString::number(nSwapTotal/1024)+"GiB",nSwapUsed);
        swap_pieseries->append("", nSwapFree);
}


void MainWindow::updateNetGraph()
{
//      double download_data=100*nSwapUsed/nSwapTotal;
        QTime dataTime(QTime::currentTime());
        long int eltime = dataTime.elapsed();
        int lastpointtime = 0;
        //int size = (eltime - lastpointtime);//数据个数
        int size=1;
        if(isVisible())
        {
            QVector<QPointF> oldPoints = download_series->pointsVector();//Returns
the points in the series as a vector
            QVector<QPointF> points;
            for(int i=size;i<oldPoints.count();++i)
            {
                points.append(QPointF(i-size ,oldPoints.at(i).y()));//替换数据用
            }
            qint64 sizePoints = points.count();
            points.append(QPointF(sizePoints,net_receive_speed));
            download_series->replace(points);

            oldPoints.clear();
            points.clear();
            oldPoints = upload_series->pointsVector();//Returns the points in the series
as a vector
            for(int i=size;i<oldPoints.count();++i)
```

```
{
    points.append(QPointF(i-size ,oldPoints.at(i).y()));//替换数据用
}
sizePoints = points.count();
points.append(QPointF(sizePoints,net_transmit_speed));
upload_series->replace(points);

char* tempc;
tempc = (char *)malloc(100 * sizeof(char));
tempc=kscale(net_current_receive_total,1);
totalReceived_label->setText("TotalReceived:"+QString(tempc));

tempc=kscale(net_current_transmit_total,1);
totalsent_label->setText("TotalSent:"+QString(tempc));

tempc=kscale(net_receive_speed,1);
receiving_label->setText("Receiving:"+QString(tempc));

tempc=kscale(net_transmit_speed,1);
sending_label->setText("Sending:"+QString(tempc));



tempc=kscale(max_net_speed,1);
QString net_spped_max=QString(tempc);
QString temp=net_spped_max;
temp.chop(1);
float max=temp.toFloat();
if(net_spped_max.right(1).compare("B")==0)
{
    net_axisY->setRange(0,1024);
}
else if(net_spped_max.right(1).compare("K")==0)
{
    net_axisY->setRange(0,ceil(max*1.5)*1024);
}
else if(net_spped_max.right(1).compare("M")==0)
{
    net_axisY->setRange(0,ceil(max*1.5)*1024*1024);
}
```

```
//              net_chart->setAxisY(net_axisY,upload_series);
//              net_chart->setAxisY(net_axisY,download_series);
            lastpointtime = eltime;
        }
}

void MainWindow::timerEvent(QTimerEvent *event)
{
        if(event->timerId()==timeId)//定时器到时间,//模拟数据填充
        {
            //update net
            net_update();
            updateNetGraph();
            //update mem and swap info
            setMem();

            //update cpu info but not process
            old_procs = new_procs;
            num_old_procs = num_new_procs;
            memcpy(&old_cpu, &new_cpu, sizeof(old_cpu));
            read_procs();
            setProcess(false);
            free_old_procs();

            //update_graph
            updateCPUGraph();
            updateMemGraph();
            updateSwapGraph();
        }
        if(event->timerId()==process_timeId)
        {
            setMem();
            old_procs = new_procs;
            num_old_procs = num_new_procs;
            memcpy(&old_cpu, &new_cpu, sizeof(old_cpu));
//              sleep(4);
            read_procs();
            setProcess(true);
            free_old_procs();
```

```
    }
}

void MainWindow::updateFileSystem()
{
//      qDebug()<<device_info;
}

void MainWindow::setFileSystem(int device_num)
{
    //get file system info
    updateFileSystem();
//      fs_model->
//      fs_model->clear();
    QStringList strList;
//      QVector<QStringList> data;
    QVector<QStringList>& data = fs_model->DataVector();
    data.clear();
    for(int i=0;i<device_num;i++)
    {
        QString device=device_info[i][0];

        if(device.mid(0,4).compare("/dev")==0)
        {
//              qDebug()<<i
//                      <<device_info[i][0]
//                      <<device_info[i][1]
//                      <<device_info[i][2]
//                      <<device_info[i][3]
//                      <<device_info[i][4].toInt()
//                      <<device_info[i][5];
            strList<<device_info[i][0]
                   <<device_info[i][1]
                   <<device_info[i][2]
                   <<device_info[i][3]
                   <<device_info[i][4]
                   <<device_info[i][5];
            data.append(strList);
            strList.clear();
        }
```

```
    }
    fs_model->setData(data);
    emit fs_model->layoutChanged();
}


void MainWindow::on_kill_pushButton_clicked()
{
    int pid=process_focus_pid.toInt();

    QString cmd=QString("kill ") +QString::number(pid);
    system(cmd.toLatin1());
    selected=false;
    QMessageBox::warning(this, tr("kill"), QString::fromUtf8("The process has
been killed!"), QMessageBox::Yes);
}

void MainWindow::on_tabWidget_currentChanged(int index)
{
    switch (index)
    {
    case 0:
        break;
    case 1:
        break;
    case 2:
        break;
    }
}



void MainWindow::on_Process_tableView_clicked(const QModelIndex &index)
{
    selected=true;
    qDebug()<<"index"<<index;
    QModelIndex pid_index = process_model->index(index.row(),0);
    process_focus_pid = process_model->data(pid_index).toString();

}
```

```
void MainWindow::on_lineEdit_returnPressed()
{
	QString str=ui->lineEdit->text();
	qDebug()<<"search!!!!!"<<str;
	if (str.isEmpty())
		return;
	searchModelandItem(str);
	selected=true;

}

void MainWindow::searchModelandItem(QString ID)
{
	QStandardItemModel * model;
	model=process_model;
	int rows = model->rowCount();//1
	int column = model->columnCount();//1
	QModelIndex item_index;
	QVariant item_data;
//	  QModelIndex pid_index = process_model->index(2,0);
//	  QVariant asd_data = process_model->data(pid_index);
	for (int i = 0; i<rows; ++i)
	{
		for (int j = 0; j<column; ++j)
		{
			item_index = model->index(i, j);
			qDebug()<<"data"<<model->data(item_index);
			item_data=model->data(item_index);
			if (item_data== ID)
			{
				focus_index = item_index;
				return;
			}

		}
	}
}
```

netinfo.h

```
/* FileName:       netinfo.h
  * Author:          Hover
  * E-Mail:          hover@hust.edu.cn
  * GitHub:          HoverWings
  * Description: net module
  */
#ifndef NETINFO_H
#define NETINFO_H

#include <QDebug>
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <iostream>
#include <vector>
#include <algorithm>

#define NET_INFO_PATH "/proc/net/dev"

#define NET_DIFF_TIME 1

using namespace std;


extern time_t net_previous_timeStamp;
extern time_t net_current_timeStamp;
extern double net_dif_time;

extern char net_file[64];
extern int net_off;
extern int line_num;
extern FILE *net_stream;
extern char net_buffer[256];
extern char *net_line_return;
extern char net_tmp_itemName[32];
```

```
extern int net_itemReceive;
extern int net_itemTransmit;

extern int net_current_receive_total;
extern int net_previous_receive_total;
extern int net_receive_speed;

extern int net_current_transmit_total;
extern int net_previous_transmit_total;
extern int net_transmit_speed;

extern float net_averageLoad_speed;
//(net_current_receive_total - net_previous_receive_total) +
(net_current_transmit_total - net_previous_transmit_total) / 2
extern float net_result;

extern int max_net_speed;
extern vector<int> net_speed_60;


void    net_init();
void    net_update();
void    net_close();


#endif // NETINFO_H
```

netinfo.cpp

```
/* FileName:netinfo.cpp
  * Author:Hover
  * E-Mail:hover@hust.edu.cn
  * GitHub:HoverWings
  * Description: net module
  */
#include "netinfo.h"

time_t net_previous_timeStamp;
time_t net_current_timeStamp;
double net_dif_time;
```

```
char net_file[64]= {NET_INFO_PATH};
int net_off;
int line_num;
FILE *net_stream;
char net_buffer[256];
char *net_line_return;
char net_tmp_itemName[32];

int net_itemReceive;
int net_itemTransmit;

int net_current_receive_total;
int net_previous_receive_total;
int net_receive_speed;

int net_current_transmit_total;
int net_previous_transmit_total;
int net_transmit_speed;

float net_averageLoad_speed;
//(net_current_receive_total - net_previous_receive_total) +
(net_current_transmit_total - net_previous_transmit_total) / 2
float net_result;

bool first=true;
int max_net_speed;
vector<int> net_speed_60;


using namespace std;

void net_init()
{
	max_net_speed=0;
	net_line_return = "INIT";
	net_stream = fopen (net_file, "r");
	net_off = fseek(net_stream, 0, SEEK_SET);

	net_update();
```

```
    net_previous_receive_total = net_current_receive_total;
    net_previous_transmit_total = net_current_transmit_total;

    net_receive_speed = 0;
    net_transmit_speed = 0;
    net_averageLoad_speed = 0.0;
    net_previous_timeStamp = net_current_timeStamp = time(NULL);
    net_dif_time = 0;

}


void net_update()
{
    net_previous_receive_total = net_current_receive_total;
    net_previous_transmit_total = net_current_transmit_total;
    net_current_receive_total = 0;
    net_current_transmit_total = 0;


    net_off = fseek(net_stream, 0, SEEK_SET);

    line_num = 1;
    net_line_return = fgets (net_buffer, sizeof(net_buffer), net_stream);//读取第一
行
    //printf("[net_update] line %d: %s\n", line_num, net_line_return);
    line_num++;
    net_line_return = fgets (net_buffer, sizeof(net_buffer), net_stream);//读取第二
行
    //printf("[net_update] line %d: %s\n", line_num, net_line_return);

    net_itemReceive = 0;
    net_itemTransmit = 0;
    int flag = 1;
    while(flag == 1)
    {
        line_num++;
        net_line_return = fgets (net_buffer, sizeof(net_buffer), net_stream);
        net_itemReceive = 0;
```

```
            net_itemTransmit = 0;
            if(net_line_return != NULL)
            {
                sscanf( net_buffer,
                    "%s%d%d%d%d%d%d%d%d%d",
                    net_tmp_itemName,
                    &net_itemReceive,
                    &net_itemTransmit,
                    &net_itemTransmit,
                    &net_itemTransmit,
                    &net_itemTransmit,
                    &net_itemTransmit,
                    &net_itemTransmit,
                    &net_itemTransmit,
                    &net_itemTransmit);
                net_current_receive_total += net_itemReceive;
                net_current_transmit_total += net_itemTransmit;
            }
            else
            {
                flag = -1;
            }

        }
        net_receive_speed = (net_current_receive_total - net_previous_receive_total) /
NET_DIFF_TIME;
        net_transmit_speed = (net_current_transmit_total - net_previous_transmit_total)
/ NET_DIFF_TIME;
        net_averageLoad_speed = (net_receive_speed + net_transmit_speed) / 2;
        if(first)
        {
            net_receive_speed=0;
            net_transmit_speed=0;
            first=false;
        }

        net_dif_time = (double)(net_current_timeStamp - net_previous_timeStamp);
        net_current_timeStamp = time(NULL);
        if( (net_dif_time) >= 60 )
        {
```

```
            net_previous_timeStamp = net_current_timeStamp;
            auto max=max_element(net_speed_60.begin(),net_speed_60.end());
            max_net_speed=*max;
            net_speed_60.clear();
        }
        else
        {
            if(net_receive_speed>max_net_speed)
            {
                max_net_speed=net_receive_speed;
                net_previous_timeStamp = net_current_timeStamp;
                net_speed_60.clear();
            }
            if(net_transmit_speed>max_net_speed)
            {
                max_net_speed=net_transmit_speed;
                net_previous_timeStamp = net_current_timeStamp;
                net_speed_60.clear();
            }
            net_speed_60.insert(net_speed_60.begin(),net_receive_speed);
            net_speed_60.insert(net_speed_60.begin(),net_transmit_speed);
        }
}


void net_close()
{
        fclose(net_stream);         //关闭文件 net_stream
}
```

processes.h

```
/* FileName:       process.h
  * Author:          Hover
  * E-Mail:          hover@hust.edu.cn
  * GitHub:          HoverWings
  * Description: process module
  *                     get process info and maintain the proc struct
  */
#ifndef PROCESSES_H
#define PROCESSES_H
```

```
#include <ctype.h>
#include <dirent.h>
#include <grp.h>
#include <pwd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <unistd.h>
#include <QVector>
#include <QString>
#include <iostream>

using namespace std;
/*
us: user cpu time (or) % CPU time spent in user space
sy: system cpu time (or) % CPU time spent in kernel space
ni: user nice cpu time (or) % CPU time spent on low priority processes
id: idle cpu time (or) % CPU time spent idle
wa: io wait cpu time (or) % CPU time spent in wait (on disk)
hi: hardware irq (or) % CPU time spent servicing/handling hardware interrupts
si: software irq (or) % CPU time spent servicing/handling software interrupts
st: steal time - - % CPU time in involuntary wait by virtual cpu while hypervisor is
servicing another processor (or) % CPU time stolen from a virtual machine
*/
/*
  * utime: user
  * stime: system time
  * ntime: nice time, the time for modefiy the priority of cpu
  * itime: idle time
  * iowtime: io waiting time
  * irqtime: interuption time
  * sirqtime: soft interuption time
*/
struct cpu_info
{
        long unsigned utime, ntime, stime, itime;
        long unsigned iowtime, irqtime, sirqtime;
};
```

```
#define PROC_NAME_LEN 64
#define THREAD_NAME_LEN 32

struct proc_info
{
	struct proc_info *next;
	pid_t pid;
	pid_t tid;
	uid_t uid;
	gid_t gid;
	char name[PROC_NAME_LEN];
	char tname[THREAD_NAME_LEN];
	char state;
	long unsigned utime;
	long unsigned stime;
	long unsigned delta_utime;
	long unsigned delta_stime;
	long unsigned delta_time;
	long vss;
	long rss;
	int num_threads;
	char policy[32];
};

struct proc_list
{
	struct proc_info **array;
	int size;
};

#define die(...) { fprintf(stderr, __VA_ARGS__); exit(EXIT_FAILURE); }

#define INIT_PROCS 50
#define THREAD_MULT 8


#define MAX_LINE 256
extern QVector<QVector<QString>> process_table;
extern QVector<QString> process_vec;
```

```
extern QVector<QString> process_title;
extern double cpu_user;
extern double cpu_sys;
extern struct proc_info **old_procs, **new_procs;
extern int num_old_procs, num_new_procs;
extern struct proc_info *free_procs;
extern int num_used_procs, num_free_procs;
extern int max_procs, delay, iterations, threads;
extern struct cpu_info old_cpu, new_cpu;

struct proc_info *alloc_proc(void);
void free_proc(struct proc_info *proc);
void read_procs(void);
int read_stat(char *filename, struct proc_info *proc);
void read_policy(int pid, struct proc_info *proc);
void add_proc(int proc_num, struct proc_info *proc);
int read_cmdline(char *filename, struct proc_info *proc);
int read_status(char *filename, struct proc_info *proc);
void print_procs(void);
struct proc_info *find_old_proc(pid_t pid, pid_t tid);
void free_old_procs(void);

extern int (*proc_cmp)(const void *a, const void *b);
//extern int (*a)(const void *a, const void *b);

int proc_cpu_cmp(const void *a, const void *b);
int proc_vss_cmp(const void *a, const void *b);
int proc_rss_cmp(const void *a, const void *b);
int proc_thr_cmp(const void *a, const void *b);
int numcmp(long long a, long long b);
void usage(char *cmd);

#endif // PROCESSES_H
```

processes.cpp

```
/* FileName:       process.cpp
  * Author:         Hover
  * E-Mail:         hover@hust.edu.cn
  * GitHub:         HoverWings
  * Description: process module
```

```
 *                    get process info and maintain the proc struct
 */
#include "processes.h"
/*
 * us — the precentage of time in user space
 * sy — the precentage of time in kernel space
 * ni — the precentage of process whose pri were updated
 * id — the precentage of idle cpu
 * wa — the precentage of I/O waiting
 * hi — the precentage of Hardware IRQ
 * si — the precentage of Software Interrupts
*/
QVector<QVector<QString>> process_table;
QVector<QString> process_vec;
QVector<QString> process_title;
double cpu_user;
double cpu_sys;
struct proc_info **old_procs, **new_procs;
int num_old_procs, num_new_procs;
struct proc_info *free_procs;
int num_used_procs, num_free_procs;

int max_procs, delay, iterations, threads;

struct cpu_info old_cpu, new_cpu;


int (*proc_cmp)(const void *a, const void *b)=&proc_cpu_cmp;
 struct proc_info *alloc_proc(void)
{
	struct proc_info *proc;

	if (free_procs)
	{
		proc = free_procs;
		free_procs = free_procs->next;
		num_free_procs--;
	}
	else
	{
```

```
		proc = (proc_info*)malloc(sizeof(*proc));
		if (!proc) die("Could not allocate struct process_info.\n");
	}

	num_used_procs++;

	return proc;
}

void free_proc(struct proc_info *proc)
{
	proc->next = free_procs;
	free_procs = proc;

	num_used_procs--;
	num_free_procs++;
}

#define MAX_LINE 256

void read_procs(void)
{
	DIR *proc_dir, *task_dir;
	struct dirent *pid_dir, *tid_dir;
	char filename[64];
	FILE *file;
	int proc_num;
	struct proc_info *proc;
	pid_t pid, tid;

	int i;

	proc_dir = opendir("/proc");
	if (!proc_dir) die("Could not open /proc.\n");

	new_procs = (proc_info**)calloc(INIT_PROCS * (threads ? THREAD_MULT :
1), sizeof(struct proc_info *));
	num_new_procs = INIT_PROCS * (threads ? THREAD_MULT : 1);

	file = fopen("/proc/stat", "r");
```

```
	if (!file) die("Could not open /proc/stat.\n");
	fscanf(file, "cpu  %lu %lu %lu %lu %lu %lu %lu", &new_cpu.utime,
&new_cpu.ntime, &new_cpu.stime,&new_cpu.itime, &new_cpu.iowtime,
&new_cpu.irqtime, &new_cpu.sirqtime);
	fclose(file);
	proc_num = 0;
	while ((pid_dir = readdir(proc_dir)))
	{
		if (!isdigit(pid_dir->d_name[0]))
			continue;

		pid = atoi(pid_dir->d_name);

		struct proc_info cur_proc;

		if (!threads)
		{
			proc = alloc_proc();

			proc->pid = proc->tid = pid;

			sprintf(filename, "/proc/%d/stat", pid);
			read_stat(filename, proc);

			sprintf(filename, "/proc/%d/cmdline", pid);
			read_cmdline(filename, proc);

			sprintf(filename, "/proc/%d/status", pid);
			read_status(filename, proc);

//                      read_policy(pid, proc);

			proc->num_threads = 0;
		}
		else
		{
			sprintf(filename, "/proc/%d/cmdline", pid);
			read_cmdline(filename, &cur_proc);

			sprintf(filename, "/proc/%d/status", pid);
```

```
        read_status(filename, &cur_proc);

        proc = NULL;
    }

    sprintf(filename, "/proc/%d/task", pid);
    task_dir = opendir(filename);
    if (!task_dir) continue;

    while ((tid_dir = readdir(task_dir)))
    {
        if (!isdigit(tid_dir->d_name[0]))
            continue;

        if (threads)
        {
            tid = atoi(tid_dir->d_name);

            proc = alloc_proc();

            proc->pid = pid; proc->tid = tid;

            sprintf(filename, "/proc/%d/task/%d/stat", pid, tid);
            read_stat(filename, proc);

//                  read_policy(tid, proc);

            strcpy(proc->name, cur_proc.name);
            proc->uid = cur_proc.uid;
            proc->gid = cur_proc.gid;

            add_proc(proc_num++, proc);
        }
        else
        {
            proc->num_threads++;
        }
    }

    closedir(task_dir);
```

```
		if (!threads)
			add_proc(proc_num++, proc);
	}

	for (i = proc_num; i < num_new_procs; i++)
	{
		new_procs[i] = NULL;
	}


	closedir(proc_dir);
}

int read_stat(char *filename, struct proc_info *proc)
{
	FILE *file;
	char buf[MAX_LINE], *open_paren, *close_paren;
	//	int res, idx;

	file = fopen(filename, "r");
	if (!file) return 1;
	fgets(buf, MAX_LINE, file);
	fclose(file);

	/* Split at first '(' and last ')' to get process name. */
	open_paren = strchr(buf, '(');
	close_paren = strrchr(buf, ')');
	if (!open_paren || !close_paren) return 1;

	*open_paren = *close_paren = '\0';
	strncpy(proc->tname, open_paren + 1, THREAD_NAME_LEN);
	proc->tname[THREAD_NAME_LEN-1] = 0;

	/* Scan rest of string. */
	sscanf(close_paren + 1, " %c %*d %*d %*d %*d %*d %*d %*d %*d %*d %*d
"
				"%lu %lu %*d %*d %*d %*d %*d %*d %*d %lu %ld",
				&proc->state, &proc->utime, &proc->stime, &proc->vss,
&proc->rss);
```

```
	return 0;
}

void add_proc(int proc_num, struct proc_info *proc)
{
	int i;

	if (proc_num >= num_new_procs)
	{
		new_procs = (proc_info**)realloc(new_procs, 2 * num_new_procs *
sizeof(struct proc_info *));
		if (!new_procs) die("Could not expand procs array.\n");
		for (i = num_new_procs; i < 2 * num_new_procs; i++)
			new_procs[i] = NULL;
		num_new_procs = 2 * num_new_procs;
	}
	new_procs[proc_num] = proc;
}

 int read_cmdline(char *filename, struct proc_info *proc)
 {
	FILE *file;
	char line[MAX_LINE];

	line[0] = '\0';
	file = fopen(filename, "r");
	if (!file) return 1;
	fgets(line, MAX_LINE, file);
	fclose(file);
	if (strlen(line) > 0) {
		strncpy(proc->name, line, PROC_NAME_LEN);
		proc->name[PROC_NAME_LEN-1] = 0;
	} else
		proc->name[0] = 0;
	return 0;
}


 int read_status(char *filename, struct proc_info *proc)
```

```
 {
        FILE *file;
        char line[MAX_LINE];
        unsigned int uid, gid;

        file = fopen(filename, "r");
        if (!file) return 1;
        while (fgets(line, MAX_LINE, file)) {
                sscanf(line, "Uid: %u", &uid);
                sscanf(line, "Gid: %u", &gid);
        }
        fclose(file);
        proc->uid = uid; proc->gid = gid;
        return 0;
}


struct proc_info *find_old_proc(pid_t pid, pid_t tid)
{
        int i;
        for (i = 0; i < num_old_procs; i++)
                if (old_procs[i] && (old_procs[i]->pid == pid) && (old_procs[i]->tid ==
tid))
                        return old_procs[i];

        return NULL;
}

void free_old_procs(void)
{
        int i;

        for (i = 0; i < num_old_procs; i++)
                if (old_procs[i])
                        free_proc(old_procs[i]);

        free(old_procs);
}

 int proc_cpu_cmp(const void *a, const void *b)
```

```
{
    struct proc_info *pa, *pb;

    pa = *((struct proc_info **)a); pb = *((struct proc_info **)b);

    if (!pa && !pb) return 0;
    if (!pa) return 1;
    if (!pb) return -1;

    return -numcmp(pa->delta_time, pb->delta_time);
}

int proc_vss_cmp(const void *a, const void *b)
{
    struct proc_info *pa, *pb;

    pa = *((struct proc_info **)a); pb = *((struct proc_info **)b);

    if (!pa && !pb) return 0;
    if (!pa) return 1;
    if (!pb) return -1;

    return -numcmp(pa->vss, pb->vss);
}

int proc_rss_cmp(const void *a, const void *b)
{
    struct proc_info *pa, *pb;

    pa = *((struct proc_info **)a); pb = *((struct proc_info **)b);

    if (!pa && !pb) return 0;
    if (!pa) return 1;
    if (!pb) return -1;

    return -numcmp(pa->rss, pb->rss);
}

int proc_thr_cmp(const void *a, const void *b)
{
```

```
    struct proc_info *pa, *pb;

    pa = *((struct proc_info **)a); pb = *((struct proc_info **)b);

    if (!pa && !pb) return 0;
    if (!pa) return 1;
    if (!pb) return -1;

    return -numcmp(pa->num_threads, pb->num_threads);
}

 int numcmp(long long a, long long b)
{
    if (a < b) return -1;
    if (a > b) return 1;
    return 0;
}
```

progressbardelegate.h

```
/* FileName:       progressbardelegate.h
  * Author:          Hover
  * E-Mail:          hover@hust.edu.cn
  * GitHub:          HoverWings
  * Description: inherit the QItemDelegate and overload partial fun of the module
  *                      to draw the process bar in the QTableView
  */
#ifndef PROGRESSBARDELEGATE_H
#define PROGRESSBARDELEGATE_H

#include <QItemDelegate>

class ProgressBarDelegate : public QItemDelegate
{
    Q_OBJECT
public:
    explicit ProgressBarDelegate(QObject *parent = 0);
    void paint(QPainter *painter, const QStyleOptionViewItem &option, const
QModelIndex &index) const;

signals:
```

```
public slots:



};

#endif // PROGRESSBARDELEGATE_H
```

progressbardelegate.cpp

```
/* FileName:       progressbardelegate.cpp
  * Author:          Hover
  * E-Mail:          hover@hust.edu.cn
  * GitHub:          HoverWings
  * Description: inherit the QItemDelegate and overload partial fun of the module
  *                        to draw the process bar in the QTableView
  */
#include "progressbardelegate.h"

#include <QPainter>
#include <QApplication>

ProgressBarDelegate::ProgressBarDelegate(QObject *parent) :
        QItemDelegate(parent)
{
}


void ProgressBarDelegate::paint(QPainter *painter, const QStyleOptionViewItem
&option, const QModelIndex &index) const
{
        if(index.column() == 6)
        {
//              int value=70;
                int value =
index.model()->data(index.model()->index(index.row(),4)).toInt();
                QStyleOptionProgressBarV2 progressBarOption;
                progressBarOption.rect = option.rect.adjusted(4, 4, -4, -4);
                progressBarOption.minimum = 0;
                progressBarOption.maximum = 100;
```

```
            progressBarOption.textAlignment = Qt::AlignRight;
            progressBarOption.textVisible = true;
            progressBarOption.progress = value;
            progressBarOption.text = tr("%1%").arg(progressBarOption.progress);

            painter->save();
            if (option.state & QStyle::State_Selected)
            {
                painter->fillRect(option.rect, option.palette.highlight());
                            painter->setBrush(option.palette.highlightedText());
            }
                        QApplication::style()->drawControl(QStyle::CE_ProgressBar,
&progressBarOption, painter);

            painter->restore();

        } else {
            return QItemDelegate::paint (painter, option, index);
        }
}
```

tablemodel.h

```
/* FileName:       tablemodel.h
  * Author:          Hover
  * E-Mail:          hover@hust.edu.cn
  * GitHub:          HoverWings
  * Description: inherit the QItemDelegate and overload partial fun of the module
  *                    to draw the process bar in the QTableView
  *                    setData and maintain the data in this class which is convient for
  *                    update the data and draw processbar dynamically
  */
#ifndef TABLEMODEL_H
#define TABLEMODEL_H

#include <QAbstractTableModel>

class TableModel : public QAbstractTableModel
{
    Q_OBJECT
public:
```

```
    explicit TableModel(QObject *parent = 0);
    int rowCount(const QModelIndex &parent) const;
    int columnCount(const QModelIndex &parent) const;
    QVariant data(const QModelIndex &index, int role) const;
    Qt::ItemFlags flags(const QModelIndex &index) const;
    void setHorizontalHeader(const QStringList& headers);
    QVariant headerData(int section, Qt::Orientation orientation, int role) const;
    void setData(const QVector<QStringList>& data);
    QVector<QStringList>& DataVector() {return m_data;}
    ~TableModel(void);

signals:

public slots:

private:
    QStringList m_HorizontalHeader;
    QVector<QStringList> m_data;

};

#endif // TABLEMODEL_H
```

tablemodel.cpp

```
/* FileName:       tablemodel.cpp
  * Author:         Hover
  * E-Mail:         hover@hust.edu.cn
  * GitHub:         HoverWings
  * Description: inherit the QItemDelegate and overload partial fun of the module
  *                  to draw the process bar in the QTableView
  *                  setData and maintain the data in this class which is convient for
  *                  update the data and draw processbar dynamically
  */
#include "tablemodel.h"

TableModel::TableModel(QObject *parent) :
    QAbstractTableModel(parent)
{
}
```

```
TableModel::~TableModel()
{

}


int TableModel::rowCount(const QModelIndex &parent) const
{
	return m_data.size();
}

int TableModel::columnCount(const QModelIndex &parent) const
{
	return m_HorizontalHeader.count();
}

QVariant TableModel::data(const QModelIndex &index, int role) const
{
	if (!index.isValid())
		return QVariant();
	if (role == Qt::DisplayRole) {
		int ncol = index.column();
		int nrow =    index.row();
		QStringList values = m_data.at(nrow);
		if (values.size() > ncol)
			return values.at(ncol);
		else
		return QVariant();
	}
	return QVariant();
}

Qt::ItemFlags TableModel::flags(const QModelIndex &index) const
{
	if (!index.isValid())
		return Qt::NoItemFlags;
	Qt::ItemFlags flag = QAbstractItemModel::flags(index);
	// flag|=Qt::ItemIsEditable
	return flag;
}
```

```
void TableModel::setHorizontalHeader(const QStringList &headers)
{
	m_HorizontalHeader =	headers;
}


QVariant TableModel::headerData(int section, Qt::Orientation orientation, int role)
const
{
	if (role == Qt::DisplayRole && orientation == Qt::Horizontal) {
		return m_HorizontalHeader.at(section);
	}
	return QAbstractTableModel::headerData(section, orientation, role);
}

void TableModel::setData(const QVector<QStringList> &data)
{
	m_data = data;
}
```

tableview.h

```
/* FileName:	tableview.h
  * Author:		Hover
  * E-Mail:		hover@hust.edu.cn
  * GitHub:		HoverWings
  * Description: tableview overload
  */
#ifndef TABLEVIEW_H
#define TABLEVIEW_H

#include <QTableView>

class TableModel;
class ProgressBarDelegate;

class TableView : public QTableView
{
	Q_OBJECT
public:
```

```
    explicit TableView(QWidget *parent = 0);
    TableModel* tableModel() {return m_model;}

    ~TableView();

signals:

public slots:

private:
    void iniData();

private:
    TableModel *m_model;
    ProgressBarDelegate *m_progressBarDelegate;

};

#endif // TABLEVIEW_H
```

tableview.cpp

```
/* FileName:     tableview.cpp
  * Author:       Hover
  * E-Mail:       hover@hust.edu.cn
  * GitHub:       HoverWings
  * Description: tableview overload
  */
#include "tableview.h"

#include "tablemodel.h"
#include "progressbardelegate.h"

TableView::TableView(QWidget *parent) :
    QTableView(parent)
{
    iniData();
}

TableView::~TableView()
{
```

```
    delete m_model;
}

void TableView::iniData()
{
    m_model = new TableModel();
    this->setModel(m_model);
    m_progressBarDelegate = new ProgressBarDelegate(this);
    this->setItemDelegate(m_progressBarDelegate);
}
```

mainwindow.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>MainWindow</class>
 <widget class="QMainWindow" name="MainWindow">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>1013</width>
    <height>1006</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>MainWindow</string>
  </property>
  <widget class="QWidget" name="centralWidget">
   <widget class="QTabWidget" name="tabWidget">
    <property name="geometry">
     <rect>
      <x>10</x>
      <y>10</y>
      <width>971</width>
      <height>921</height>
     </rect>
    </property>
    <property name="tabPosition">
     <enum>QTabWidget::North</enum>
    </property>
    <property name="tabShape">
```

```
  <enum>QTabWidget::Rounded</enum>
</property>
<property name="currentIndex">
  <number>0</number>
</property>
<property name="elideMode">
  <enum>Qt::ElideMiddle</enum>
</property>
<property name="tabsClosable">
  <bool>false</bool>
</property>
<property name="movable">
  <bool>true</bool>
</property>
<property name="tabBarAutoHide">
  <bool>false</bool>
</property>
<widget class="QWidget" name="tab">
  <attribute name="title">
    <string>Process</string>
  </attribute>
  <widget class="QLineEdit" name="lineEdit">
    <property name="geometry">
      <rect>
        <x>250</x>
        <y>10</y>
        <width>411</width>
        <height>27</height>
      </rect>
    </property>
  </widget>
  <widget class="QPushButton" name="kill_pushButton">
    <property name="geometry">
      <rect>
        <x>790</x>
        <y>600</y>
        <width>88</width>
        <height>27</height>
      </rect>
    </property>
```

```
    <property name="text">
      <string>kill</string>
    </property>
  </widget>
  <widget class="QTableView" name="Process_tableView">
    <property name="geometry">
      <rect>
        <x>30</x>
        <y>50</y>
        <width>851</width>
        <height>511</height>
      </rect>
    </property>
  </widget>
</widget>
<widget class="QWidget" name="tab_2">
  <attribute name="title">
    <string>Resourses</string>
  </attribute>
  <widget class="QWidget" name="graph_widget" native="true">
    <property name="geometry">
      <rect>
        <x>150</x>
        <y>80</y>
        <width>120</width>
        <height>80</height>
      </rect>
    </property>
    <widget class="QGraphicsView" name="graphicsView">
      <property name="geometry">
        <rect>
          <x>-100</x>
          <y>-40</y>
          <width>851</width>
          <height>571</height>
        </rect>
      </property>
    </widget>
  </widget>
</widget>
```

```
        <widget class="QWidget" name="tab_3">
          <attribute name="title">
            <string>FileSystems</string>
          </attribute>
        </widget>
      </widget>
    </widget>
    <widget class="QMenuBar" name="menuBar">
      <property name="geometry">
        <rect>
          <x>0</x>
          <y>0</y>
          <width>1013</width>
          <height>24</height>
        </rect>
      </property>
    </widget>
    <widget class="QToolBar" name="mainToolBar">
      <attribute name="toolBarArea">
        <enum>TopToolBarArea</enum>
      </attribute>
      <attribute name="toolBarBreak">
        <bool>false</bool>
      </attribute>
    </widget>
    <widget class="QStatusBar" name="statusBar"/>
  </widget>
  <layoutdefault spacing="6" margin="11"/>
  <resources/>
  <connections/>
</ui>
```

myFileSystem

Buffer

```
/* FileName:       buffer.hpp
  * Author:         Hover
  * E-Mail:         hover@hust.edu.cn
  * GitHub:         HoverWings
  * Description: buffer
  */
#ifndef BUFFER_H
#define BUFFER_H

#include <iostream>
#include <cstring>
#include <vector>
#include <fstream>
#include "myfs_macro.h"
#include "assert.h"
#include "inode.hpp"


using namespace std;

struct BufferNode
{
	char buffer[SEC_SIZE + 1];	//read in 1 sec
	int pri;
	int sec_num;
	BufferNode operator = (const BufferNode& b)
	{
		memcpy(buffer, b.buffer, SEC_SIZE + 1);
		pri = b.pri;
		sec_num = b.sec_num;
	}
	BufferNode()
	{
		memset(buffer, 0, SEC_SIZE);
		pri = 0;
		sec_num = 0;
	}
	void init(int _sec_num)
```

```
    {
        pri = 1;
        sec_num = _sec_num;
    }
    void update(const BufferNode& b)
    {
        memcpy(buffer, b.buffer, SEC_SIZE + 1);
        pri = b.pri + 1;
        sec_num = b.sec_num;
    }
};

class Buffer
{
    public:
    int buffer_size;
    Buffer();
    ~Buffer();
    bool write_disk(const BufferNode& node);
    bool read_disk(int sec_num, BufferNode& node);
    void all_write_to_disk();

    private:

    bool real_disk_write(const BufferNode& node);
    bool real_disk_read(int sec_num, BufferNode& node);
    int has_sec(int sec_number);
    int is_full();


    vector<BufferNode> cache;
    fstream disk;
};

#endif
```

```
/* FileName:       buffer.cpp
  * Author:          Hover
  * E-Mail:          hover@hust.edu.cn
  * GitHub:          HoverWings
  * Description: buffer
  */
#include "Buffer.hpp"

using    namespace std;

Buffer::Buffer()
{
        disk.open(DISK, std::fstream::in | std::fstream::out | std::fstream::binary);
        if(disk.is_open())
                cout << "File has opened" << endl;
        else
                cout << "File Not Open" << endl;
}

Buffer::~Buffer()
{

        disk.close();
}


bool Buffer::write_disk(const BufferNode& node)
{
        assert(node.sec_num >= 0 && node.sec_num < MAX_SEC);
        int i;
        i = has_sec(node.sec_num);
        if(i >= 0)
        {
//              cout << "write disk: update buffer" << endl;
                cache[i].update(node);
                return true;
        }

        i = is_full();
        if(i >= 0)
```

```
        {
//              cout << "write disk:buffer full replace buffer " << i << endl;
            real_disk_write(cache[i]);
            cache.erase(cache.begin() + i);
        }
        cache.push_back(node);
//      cout << "write disk: write to buffer" << endl;
        return true;
}



// D: read in than buffer, the new pir will be 5
//      if the node is in buffer, then pir+=1
bool Buffer::read_disk(int sec_num, BufferNode& node){
        assert(sec_num >= 0 && sec_num < MAX_SEC);

        int i;
        i = has_sec(sec_num);

        if(i >= 0)
        {
//              cout << "read disk: the sec is in buffer " <<sec_num<< endl;
            node.update(cache[i]);
            return true;
        }

        i = is_full();
        if(i >= 0)
        {
//              cout << "read disk: buffer ， replace buffer " << i    << endl;
            real_disk_write(cache[i]);
            cache.erase(cache.begin() + i);
            real_disk_read(sec_num, node);
            node.init(sec_num);
            cache.push_back(node);
        }
        else {
            real_disk_read(sec_num, node);
            node.init(sec_num);
```

```
            cache.push_back(node);
//              cout << "read disk: buffer available ， write to buffer " << endl;
        }
        return true;
}

// read write
bool Buffer::real_disk_write(const BufferNode& node)
{
        assert(node.sec_num >= 0 && node.sec_num < MAX_SEC);
//        cout << "read disk write " << node.sec_num << "num sec" << endl;
        disk.seekg(node.sec_num * SEC_SIZE, disk.beg);
        disk.write(node.buffer, SEC_SIZE);
        return true;
}

// read read
bool Buffer::real_disk_read(int sec_num, BufferNode& node)
{
        assert(sec_num >= 0 && sec_num < MAX_SEC);
//        cout << "real disk read read " << sec_num << "sec" << endl;
        disk.seekg(sec_num * SEC_SIZE, disk.beg);
        disk.read(node.buffer, SEC_SIZE);
        node.buffer[SEC_SIZE] = '\0';
        node.sec_num = sec_num;
        return true;
}


int Buffer::has_sec(int sec_number)
{
        for(int i = 0; i < cache.size(); i++)
        {
            if(cache[i].sec_num == sec_number)
                return i;
        }
        return -1;
}

// return lowest sec
```

```
int Buffer::is_full()
{
	if(cache.size() == 15)
	{
		int min = 9999, min_i = 0;
		for(int i = 0; i < cache.size(); i++)
		{
			if(cache[i].pri < min)
			{
				min = cache[i].pri;
				min_i = i;
			}
		}
		return min_i;
	}
	else
	{
		return -1;
	}
}

void Buffer::all_write_to_disk()
{
	for(int i = 0; i < cache.size(); i++)
	{
		real_disk_write(cache[i]);
	}
}
```

Direntry

```
/* FileName:       direntry.hpp
  * Author:         Hover
  * E-Mail:         hover@hust.edu.cn
  * GitHub:         HoverWings
  * Description: direntry
  */
#ifndef DIRENTRY_H
#define DIRENTRY_H
#include "Buffer.hpp"
#include "assert.h"
```

```
// 32 Bytes
struct sector_dir_entry
{
	char name[28];
	int inode_num;
	void init(const char* _name, int _num);
	sector_dir_entry();
	sector_dir_entry operator = (const sector_dir_entry& dir);

	bool operator == (const sector_dir_entry& dir);
	void clone(const sector_dir_entry& dir);
};

// 512 Bytes. the final link to the next
class sector_dir
{
public:
	sector_dir();
	char dir_name[28];
	bool write_back_to_disk(Buffer& buffer, int sec_num);

	sector_dir operator = (const sector_dir& sec_dir);

	sector_dir_entry dirs[16];

	bool read_dir_from_disk(Buffer& buffer, int sec_num);
	bool isroot();
};

// 512 Bytes   !
class sector_file
{
public:
	char data[VALID_DATA_LENGTH];
	int next;

	sector_file();
	sector_file operator = (const sector_file& sec_file);
	bool read_dir_from_disk(Buffer& buffer, int sec_num);
	bool write_back_to_disk(Buffer& buffer, int sec_num);
```

```
};

#endif
```

Inode

```
/* FileName:       inode.hpp
  * Author:          Hover
  * E-Mail:          hover@hust.edu.cn
  * GitHub:          HoverWings
  * Description: inode
  */
#ifndef INODE_H
#define INODE_H
#include "myfs_macro.h"
#include "Buffer.hpp"
#include <iostream>
#include "assert.h"
#include "superblock.hpp"


// compensate to 32 Bytes
class Inode
{
friend class Buffer;
private:
        int _sec_beg;      //    link by ptr
        int _sec_num;      // total sec num
        char _compensate[12];

public:
        bool _is_file;
        int _file_size; // Byte
        int _inode_num;
        Inode();
        class Buffer *buffer;
        int mode;
        time_t creat_time;
        time_t modify_time;
        Inode(int node_num, bool _is_file, int file_size, int sec_begin);
```

```
    int get_inode_num();

    // true->file; false->dir
    bool get_type();

    int get_file_size();

    int get_sec_beg();

    int get_sec_num();

    void set_inode_num(int num);

    int get_inode_sec_num();

    bool read_inode_from_disk(int inode_num,Buffer &buffer);

    bool write_inode_back_to_disk(Buffer &buffer);

    Inode operator = (const Inode& b)
    {

    }
};

#endif
```

```
/* FileName:       inode.cpp
  * Author:         Hover
  * E-Mail:         hover@hust.edu.cn
  * GitHub:         HoverWings
  * Description: inode
  */
#include "inode.hpp"

using namespace std;
extern Buffer buffer;


Inode::Inode()
{
        _inode_num = 0;
        _is_file = false;
//        _file_size = 0;
        _sec_beg = 0;
        _sec_num = 0;
        memset(_compensate, 0, 12);
}

Inode::Inode(int node_num, bool _is_file, int file_size, int sec_begin)
{
        _inode_num = node_num;
        _is_file = _is_file;
        _file_size = file_size;
        _sec_beg = sec_begin;
        _sec_num = (file_size) / sizeof(VALID_DATA_LENGTH) + 1;
        cout << "create new inode" << node_num ;
        cout << " ， begin sec： " << sec_begin << endl;
}

int Inode::get_inode_num()
{
        return _inode_num;
}

// true->file; false->dir
bool Inode::get_type()
```

```
{
	return _is_file;
}

int Inode::get_file_size()
{
	return _file_size;
}

int Inode::get_sec_beg()
{
	return _sec_beg;
}

int Inode::get_sec_num()
{
	_sec_num = (_file_size) / VALID_DATA_LENGTH + 1;
	return _sec_num;
}

int Inode::get_inode_sec_num()
{
//	 return INODE_BEGIN / SEC_SIZE + _inode_num / sizeof(Inode);
	return INODE_BEGIN / SEC_SIZE + _inode_num;
}

void Inode::set_inode_num(int num)
{
	_inode_num = num;
}


bool Inode::read_inode_from_disk(int inode_num,Buffer &buffer)
{
	assert(inode_num >= 0 && inode_num < INODE_NUM);
	set_inode_num(inode_num);
	int sec_num = get_inode_sec_num();
	int num_in_sec = inode_num % 16;
	BufferNode buffer_node;
```

```
    buffer.read_disk(sec_num, buffer_node);
    memcpy(this, buffer_node.buffer + num_in_sec * sizeof(Inode), sizeof(Inode));

    return true;
}


bool Inode::write_inode_back_to_disk(Buffer &buffer)
{
    int sec_num = get_inode_sec_num();
    int num_in_sec = _inode_num % 16;
    BufferNode buffer_node;

    buffer.read_disk(sec_num, buffer_node);
    memcpy(buffer_node.buffer + num_in_sec * sizeof(Inode), this, sizeof(Inode));
    cout << "inode write back , inode num" << _inode_num << ", sec num: " <<
sec_num << endl;
    buffer.write_disk(buffer_node);
    return true;
}
```

myfs_macro.h

```
/* FileName:       myfs_macro.h
  * Author:         Hover
  * E-Mail:         hover@hust.edu.cn
  * GitHub:         HoverWings
  * Description: the macro need by many classes
  */
#ifndef MYFS_MACRO_H
#define MYFS_MACRO_H

#define SEC_SIZE 1024
#define INODE_NUM 1024
#define BLOCK_NUM 1024
#define DISK    "disk.img"
#define IMG "/tmp/myfs_temp"
#define SUPER_BEGIN 0
#define INODE_BEGIN sizeof(superblock)
#define VALID_DATA_LENGTH (SEC_SIZE-sizeof(int))
#define BLOCK_BEGIN (sizeof(superblock) + sizeof(Inode) * INODE_NUM)
```

```
#define MAX_SEC      ((BLOCK_BEGIN + BLOCK_NUM * SEC_SIZE) /
SEC_SIZE )


#endif //MYFS_MACRO_H
```

myfs.hpp

```
/* FileName:       myfs.hpp
  * Author:          Hover
  * E-Mail:          hover@hust.edu.cn
  * GitHub:          HoverWings
  * Description: myfs operation
  */
#include "myfs_macro.h"
#include "superblock.hpp"
#include "inode.hpp"
#include "Buffer.hpp"
#include "direntry.hpp"
#include <sstream>
#include <cstring>
#include <vector>
class myFS
{
public:
        void myshell();

        void run();

        // construct
        myFS();

        void vim(vector<string> args);
      void open(vector<string> args);
      void read(vector<string> args);
//        void write(vector<string> args);
//        void seek(vector<string> args);
//        void close(vector<string> args);
        void mkdir(vector<string> args);
        void rmdir(vector<string> args);
        void cd(vector<string> args);
//        void link(vector<string> args);
```

```
//      void unlink(vector<string> args);
//      void stat(vector<string> args);
        void ls(vector<string> args);
        void touch(vector<string> args);
        void cat(vector<string> args);
        void cp(vector<string> args);
//      void tree(vector<string> args);
//      void import(vector<string> args);
        void printpwd(vector<string> args);

        string getpwd(vector<string> args);

//      void FS_export(vector<string> args);
        string PRMPT = "sh> ";


        superblock sp;
        Inode cur_dir_node;
        sector_dir cur_dir;
private:
        Buffer my_cache;
        int get_dir_index(int inode_num);
        bool format_file_system();
        bool del_inode(Inode& node, sector_dir& del_dir);

        bool move_in(int ionde_num,string file_name);
//      bool move_out(string name);
        bool move_out(int inode_num);

        bool init_file_system();

        int is_existed_file(string filename);



};
/* FileName:      myfs.cpp
  * Author:        Hover
  * E-Mail:        hover@hust.edu.cn
  * GitHub:        HoverWings
```

```
 * Description: myfs operation
 */
#include "myfs.hpp"
using    namespace std;

// less than opt required
#define ops_at_least(x)                                                    \
    if (static_cast<int>(args.size()) < x+1) {                    \
        cerr << args[0] << ": missing operand" << endl;          \
        return;                                                            \
    }

// more than opt required
#define ops_less_than(x)                                                   \
    if (static_cast<int>(args.size()) > x+1) {                    \
        cerr << args[0] << ": too many operands" << endl;      \
        return;                                                            \
    }\


// construct
myFS::myFS()
{
        cout << endl << "****************** Hover's FileSystem
******************" << endl;
        this->sp.myfs=this;


//        cout<<sp.cur_dir_num<<" "<<sp.cur_dir_node_num;
//
        cur_dir_node.read_inode_from_disk(sp.cur_dir_node_num,my_cache);
//        cout<<cur_dir_node.get_sec_beg()<<" "<<cur_dir_node.get_inode_num();

        sector_dir root_sec_dir;
        root_sec_dir.read_dir_from_disk(my_cache,cur_dir_node.get_sec_beg());
        cur_dir=root_sec_dir;
        cur_dir.write_back_to_disk(my_cache,sp.cur_dir_num);

//        format_file_system();
```

```
}

void myFS::myshell()
{
	string cmd;
	myFS * fs=this;
	vector<string> args;
	string token;
	PRMPT=getpwd(args);
//	  cout<<fs->getpwd(args);
	PRMPT+=">";
	cout << PRMPT;
	while (getline(cin, cmd))
	{
//		  PRMPT=fs->getpwd(args);
//		  PRMPT+=">";
		args.clear();
		istringstream iss(cmd);
		while (iss >> token) { args.push_back(token); }
		if (args.size() == 0)
		{
			cout << PRMPT;
			continue;
		}
		if (args[0] == "ls")
		{
			ls(args);
		}
		else if (args[0] == "touch")
		{
			touch(args);
		}
		else if (args[0] == "cd")
		{
			cur_dir_node.write_inode_back_to_disk(my_cache);
			cd(args);
		}
		else if (args[0] == "mkdir")
		{
			mkdir(args);
```

```
}
else if (args[0] == "rmdir")
{
    rmdir(args);
}
else if (args[0] == "print")
{
    sp.print_block_bitmap();
    cout<<endl;
    sp.print_inode_bitmap();
}
else if (args[0] == "cat")
{
    cat(args);
}
else if (args[0] == "format")
{
    format_file_system();
}
else if (args[0] == "pwd")
{
    cout<<"pwd:";
    printpwd(args);
}
else if (args[0] == "move_in")
{
    string file_name=args[1];
    const char *name = file_name.c_str();
    int inode_num=is_existed_file(file_name);
    move_in(inode_num,file_name);
}
else if (args[0] == "move_out")
{
    string file_name=args[1];
    const char *name = file_name.c_str();
    int inode_num=is_existed_file(file_name);
    move_out(inode_num);
}
else if (args[0] == "vim")
{
```

```
                vim(args);
            }
            else if (args[0] == "exit")
            {
                cerr<<cur_dir_node.get_sec_beg()<<"
"<<cur_dir_node.get_inode_num()<<endl;
                cur_dir_node.read_inode_from_disk(cur_dir_node.get_inode_num(),
my_cache);
                cur_dir_node.write_inode_back_to_disk(my_cache);
                fs->sp.write_to_disk();
                fs->my_cache.all_write_to_disk();
                return;
            }
            else
            {
                cerr<< "comman not found"<<endl;
            }
            PRMPT=fs->getpwd(args);
            PRMPT+=">";
            cout<<PRMPT;
        }

}


namespace strtool
{
    string trim(const string& str)
    {
        string::size_type pos = str.find_first_not_of(' ');
        if (pos == string::npos)
        {
            return str;
        }
        string::size_type pos2 = str.find_last_not_of(' ');
        if (pos2 != string::npos)
        {
            return str.substr(pos, pos2 - pos + 1);
        }
        return str.substr(pos);
```

```
}

int split(const string& str, vector<string>& ret_, string sep = ",")
{
	if (str.empty())
	{
		return 0;
	}

	string tmp;
	string::size_type pos_begin = str.find_first_not_of(sep);
	string::size_type comma_pos = 0;

	while (pos_begin != string::npos)
	{
		comma_pos = str.find(sep, pos_begin);
		if (comma_pos != string::npos)
		{
			tmp = str.substr(pos_begin, comma_pos - pos_begin);
			pos_begin = comma_pos + sep.length();
		}
		else
		{
			tmp = str.substr(pos_begin);
			pos_begin = comma_pos;
		}

		if (!tmp.empty())
		{
			ret_.push_back(tmp);
			tmp.clear();
		}
	}
	return 0;
}

string replace(const string& str, const string& src, const string& dest)
{
	string ret;
```

```
        string::size_type pos_begin = 0;
        string::size_type pos       = str.find(src);
        while (pos != string::npos)
        {
            cout <<"replacexxx:" << pos_begin <<" " << pos <<"\n";
            ret.append(str.data() + pos_begin, pos - pos_begin);
            ret += dest;
            pos_begin = pos + 1;
            pos       = str.find(src, pos_begin);
        }
        if (pos_begin < str.length())
        {
            ret.append(str.begin() + pos_begin, str.end());
        }
        return ret;
    }
}


void myFS::ls(vector<string> args)
{
    for(int i = 2; i < 15; i++)
    {
        cout << cur_dir.dirs[i].name << " ";
    }
    cout << endl;
}


//Begin
void myFS::run()
{
    myshell();
    return;
}


bool myFS::del_inode(Inode& node, sector_dir& del_dir)
{
    cout << "delete inode, inode num" << node.get_inode_num() << endl;
```

```
    if(node.get_type())
    {
        for(int i = 2; i < 15; i++)
        {
            if(del_dir.dirs[i].inode_num == node.get_inode_num())
            {
                cout << "delate inode，delete sector" << endl;
                memset(&del_dir.dirs[i], 0, sizeof(sector_dir_entry));
                del_dir.write_back_to_disk(my_cache, node.get_sec_beg());
                break;
            }
        }


        sp.recv_sec(node.get_sec_beg() - BLOCK_BEGIN / 512);
        sp.recv_inode(node.get_inode_num());
    }
    else {
        // dir
        for(int i = 0; i < 15; i++) {
            if(node.get_inode_num() == del_dir.dirs[i].inode_num) {
                cout << "delate inode，delete sector" << endl;
                memset(&del_dir.dirs[i], 0, sizeof(sector_dir_entry));
                del_dir.write_back_to_disk(my_cache, node.get_sec_beg());
                break;
            }
        }
        sp.recv_sec(node.get_sec_beg() - BLOCK_BEGIN / 512);
        sp.recv_inode(node.get_inode_num());
        Inode new_node;
        new_node = node;
        sector_dir new_dir;
        new_dir = del_dir;

        new_dir.read_dir_from_disk(my_cache, new_node.get_sec_beg());
        // 4. delete every files and directories recursively
        for(int i = 2; i < 15; i++) {
            if(new_dir.dirs[i].inode_num != 0) {
                new_node.read_inode_from_disk(new_dir.dirs[i].inode_num,
my_cache);
```

```
                del_inode(new_node, new_dir);
            }
        }
    }
}


bool myFS::move_in(int ionde_num,string file_name)
{
    const char *name = file_name.c_str();
    // get file size, compute needed block number, allocate block
    ifstream is(IMG);
    if(is)
    {
        is.seekg(0, is.end);
        int length = is.tellg();
        cout << "size of the file:" << length << " bytes" << endl;

        // 2. compute needed blocks
        int needed_block = length / VALID_DATA_LENGTH;
        if(length % VALID_DATA_LENGTH != 0)
            needed_block++;
        int left = length % VALID_DATA_LENGTH;
        cout << endl << "last node contain " << ((left == 0) ?
VALID_DATA_LENGTH : left) << "bytes of data" << endl;
        cout << "need " << needed_block << " blocks to store data" << endl;
        int flag = false;
        Inode now_file_inode;
        if(ionde_num==-1)
        {
            Inode new_file_inode(sp.get_new_inode(), true, length,
sp.get_new_sec());
            new_file_inode._is_file= true;
            new_file_inode.write_inode_back_to_disk(my_cache);
            ionde_num=new_file_inode.get_inode_num();

            cout << "img inode info: #inode: " << new_file_inode.get_inode_num()
<< endl;
            cout << "file length " << new_file_inode.get_file_size() << endl;
            cout << " #sector begin: " << new_file_inode.get_sec_beg() << endl;
```

```
            now_file_inode.read_inode_from_disk(ionde_num,my_cache);
            // 3. add new entry in current directory

            for(int i = 2; i < 15; i++)
            {
                if(cur_dir.dirs[i].inode_num == -1)
                {
                    cur_dir.dirs[i].init(name, ionde_num);
                    flag = true;
                    break;
                }
            }
        }
        else
        {
            now_file_inode.read_inode_from_disk(ionde_num,my_cache);
            now_file_inode._file_size=length;
        }
        if(flag)
        {
            cur_dir.write_back_to_disk(my_cache, cur_dir_node.get_sec_beg());//
write back now
        }
        cerr<<now_file_inode._file_size<<"!!!!!!!file size"<<endl;
        // 4. store data into file system
        is.seekg(0, is.beg);
        char buffer[VALID_DATA_LENGTH];
        sector_file img_sectors[needed_block];
        int sec_numbers[needed_block];
        sec_numbers[0] = now_file_inode.get_sec_beg();
        for(int i = 0; i < needed_block - 1; i++)
        {
            is.read(buffer, VALID_DATA_LENGTH);
            sec_numbers[i+1] = sp.get_new_sec();
            memcpy(img_sectors[i].data, buffer, VALID_DATA_LENGTH);
            img_sectors[i].next = sec_numbers[i+1];
            cout << "#next data sector:" << img_sectors[i].next << endl;
        }
        if(left == 0)
        {
```

```
                is.read(buffer, VALID_DATA_LENGTH);
                memcpy(img_sectors[needed_block - 1].data, buffer,
VALID_DATA_LENGTH);
                img_sectors[needed_block - 1].next = -1;
            }
            else
            {
                is.read(buffer, left);
                memcpy(img_sectors[needed_block - 1].data, buffer, left);
                img_sectors[needed_block - 1].next = -1;
            }

            cout << "File pointer location" << is.tellg() << endl;
            cout << "file sectors info" << endl;
            cout << now_file_inode.get_sec_beg();
            for(int i = 0; i <=needed_block; i++)
            {
//                  cout << " -> " << img_sectors[i];
            }
            cout << endl;
            for(int i = 0; i <needed_block; i++)
            {
                img_sectors[i].write_back_to_disk(my_cache, sec_numbers[i]);
            }
            now_file_inode.write_inode_back_to_disk(my_cache);
            is.close();
        }

}

bool myFS::move_out(int inode_num)
{

        if(inode_num == -1)
        {
            cerr << "file do not exist" << endl;
            return false;
        }
        Inode file_node;
        file_node.read_inode_from_disk(inode_num, my_cache);
```

```
    cout << "file info: #inode " << file_node.get_inode_num() << endl;
    cout << "file length: " << file_node.get_file_size() << endl;
    cout << "sec number: " << file_node.get_sec_num() << endl;
    cout << "sec_begin: " << file_node.get_sec_beg() << endl << endl;

    // get data from my file system
    sector_file data_sec;
    data_sec.read_dir_from_disk(my_cache, file_node.get_sec_beg());


    fstream os(IMG,     fstream::in | fstream::out |ios::trunc);
    char buffer[VALID_DATA_LENGTH];
    int next_sec = -1, left = file_node.get_file_size() % VALID_DATA_LENGTH;
    if(os)
    {
        for(int i = 0; i < file_node.get_sec_num() ; i++)
        {
            if(i != file_node.get_sec_num() - 1 || left == 0)
            {
                next_sec = data_sec.next;
                memcpy(buffer, data_sec.data, VALID_DATA_LENGTH);
                os.write(buffer, VALID_DATA_LENGTH);
                data_sec.read_dir_from_disk(my_cache, next_sec);
            }
            else
            {
                memcpy(buffer, data_sec.data, left);
                os.write(buffer, left);
            }
            cout << "size of new file:" << os.tellg() << endl;
        }
        os.close();
    }
    return true;
}

void myFS::cat(vector<string> args)
{
    ops_at_least(1);
```

```
string file_name=args[1];
const char *name = file_name.c_str();
int inode_num=is_existed_file(file_name);
if(inode_num == -1)
{
	cerr << "file do not exist" << endl;
	return;
}
Inode file_node;
file_node.read_inode_from_disk(inode_num, my_cache);
if((bool)file_node._is_file== false)
{
	cerr << "can not cat dir" << endl;
	return;
}

cout << "file info: #inode " << file_node.get_inode_num() << endl;
cout << "file length: " << file_node.get_file_size() << endl;
cout << "sec number: " << file_node.get_sec_num() << endl;
cout << "sec_begin: " << file_node.get_sec_beg() << endl << endl;

sector_file data_sec;
data_sec.read_dir_from_disk(my_cache, file_node.get_sec_beg());

char buffer[VALID_DATA_LENGTH];
int next_sec = -1, left = file_node.get_file_size() % VALID_DATA_LENGTH;
string out_str;
for(int i = 0; i < file_node.get_sec_num() ; i++)
{
	if(i != file_node.get_sec_num() - 1 || left == 0)
	{
		next_sec = data_sec.next;
		memcpy(buffer, data_sec.data, VALID_DATA_LENGTH);
		out_str+=buffer;
		data_sec.read_dir_from_disk(my_cache, next_sec);
	}
	else
	{
		memcpy(buffer, data_sec.data, left);
		out_str+=buffer;
```

```
            }

        }
        cout<<out_str.substr(0,file_node.get_file_size())<<endl;
        return;
}


string myFS::getpwd(vector<string> args)
{

        sector_dir back_dir=cur_dir;
        Inode back_inode=cur_dir_node;
        string path;
        string dir_name;
        if(cur_dir.isroot())
        {
            path="/";
//            cout<<path;
            return path;
        }
        path=cur_dir.dir_name;
        dir_name=cur_dir.dir_name;
        sector_dir* now=&cur_dir;
        while(!now->isroot())
        {
            int dir_inode_num=cur_dir.dirs[1].inode_num;
            cur_dir_node.read_inode_from_disk(dir_inode_num, my_cache);
            cur_dir.read_dir_from_disk(my_cache, cur_dir_node.get_sec_beg());
            dir_name=cur_dir.dir_name;
            path=dir_name+"/"+path;
            now=&cur_dir;
        }
//      cout<<path<<endl;
        cur_dir_node=back_inode;
        cur_dir=back_dir;
        return path;
}

void myFS::printpwd(vector<string> args)
```

```
{
	ops_less_than(0);
	string pwd;
	pwd=getpwd(args);
	cout<<"pwd!!!!"<<pwd<<endl;
}

// format
bool myFS::format_file_system()
{
	sp.init();
	sp.format_disk();
	Inode root_node(sp.get_new_inode(), false, 0, sp.get_new_sec());
	Inode bin_node(sp.get_new_inode(), false, 0, sp.get_new_sec());
	Inode etc_node(sp.get_new_inode(), false, 0, sp.get_new_sec());
	Inode home_node(sp.get_new_inode(), false, 0, sp.get_new_sec());
	Inode dev_node(sp.get_new_inode(), false, 0, sp.get_new_sec());
	Inode tangrui_node(sp.get_new_inode(), false, 0, sp.get_new_sec());


	root_node.write_inode_back_to_disk(my_cache);
	bin_node.write_inode_back_to_disk(my_cache);
	etc_node.write_inode_back_to_disk(my_cache);
	home_node.write_inode_back_to_disk(my_cache);
	dev_node.write_inode_back_to_disk(my_cache);
	tangrui_node.write_inode_back_to_disk(my_cache);


	sector_dir root_sec_dir;
	strcpy(root_sec_dir.dir_name,"root");
	root_sec_dir.dirs[0].init(".", 0);
	root_sec_dir.dirs[1].init("..", 0);
	root_sec_dir.dirs[2].init("bin", bin_node.get_inode_num());
	root_sec_dir.dirs[3].init("etc", etc_node.get_inode_num());
	root_sec_dir.dirs[4].init("home", home_node.get_inode_num());
	root_sec_dir.dirs[5].init("dev", dev_node.get_inode_num());

	sector_dir bin_sec_dir;
	strcpy(bin_sec_dir.dir_name,"bin");
	bin_sec_dir.dirs[0].init(".", bin_node.get_inode_num());
```

```
    bin_sec_dir.dirs[1].init("..", root_node.get_inode_num());

    sector_dir etc_sec_dir;
    strcpy(etc_sec_dir.dir_name,"etc");
    etc_sec_dir.dirs[0].init(".", etc_node.get_inode_num());
    etc_sec_dir.dirs[1].init("..", root_node.get_inode_num());


    sector_dir dev_sec_dir;
    strcpy(dev_sec_dir.dir_name,"dev");
    dev_sec_dir.dirs[0].init(".",    dev_node.get_inode_num());
    dev_sec_dir.dirs[1].init("..", root_node.get_inode_num());


    root_sec_dir.write_back_to_disk(my_cache, root_node.get_sec_beg());
    bin_sec_dir.write_back_to_disk(my_cache, bin_node.get_sec_beg());
    etc_sec_dir.write_back_to_disk(my_cache, etc_node.get_sec_beg());
//    home_sec_dir.write_back_to_disk(my_cache, home_node.get_sec_beg());
    dev_sec_dir.write_back_to_disk(my_cache, dev_node.get_sec_beg());

    cur_dir.read_dir_from_disk(my_cache,root_node.get_sec_beg());
    cur_dir_node.read_inode_from_disk(0,my_cache);
//    cur_dir = root_sec_dir;
//    cur_dir_node = root_node;
    return true;
}

// D:mkdir dir
void myFS::mkdir(vector<string> args)
{
    string file_name=args[1];
    const char *name = file_name.c_str();
    // create inode
    Inode new_dir_inode(sp.get_new_inode(), false, 0, sp.get_new_sec());
    cout<<"mkdir inode num num"<<new_dir_inode.get_inode_num();
//    cout<<"mkdir inode num num"<<new_dir_inode.get_inode_num();
    // write back to disk
    new_dir_inode.write_inode_back_to_disk(my_cache);

    // mkdir entry
```

```
        sector_dir new_sec_dir;
        strcpy(new_sec_dir.dir_name,name);
        new_sec_dir.dirs[0].init(".", new_dir_inode.get_inode_num());
        new_sec_dir.dirs[1].init("..", cur_dir_node.get_inode_num());
        new_sec_dir.write_back_to_disk(my_cache, new_dir_inode.get_sec_beg());

        // add dir to parent dir
        int flag = false;
        for(int i = 2; i < 15; i++)
        {
            if(cur_dir.dirs[i].inode_num == -1)
            {
                cur_dir.dirs[i].init(name, new_dir_inode.get_inode_num());
                flag = true;
                break;
            }
        }
        if(flag)
        {
            cur_dir.write_back_to_disk(my_cache, cur_dir_node.get_sec_beg());
        }
        return;
}

// D: touch file
void myFS::touch(vector<string> args)
{
        string file_name=args[1];
        const char *name = file_name.c_str();
        cout << "touch file" << endl;
        // create inode
        Inode new_file_inode(sp.get_new_inode(), true, 1, sp.get_new_sec());
        new_file_inode._is_file= true;
        new_file_inode.write_inode_back_to_disk(my_cache);

//      sector_file new_sec_file;
//      new_sec_file.write_back_to_disk(my_cache, new_file_inode.get_sec_beg());

        // add inode to dir
        int flag = false;
```

```
    for(int i = 2; i < 15; i++)
    {
        if(cur_dir.dirs[i].inode_num == -1)
        {
            cur_dir.dirs[i].init(name, new_file_inode.get_inode_num());
            flag = true;
            break;
        }
    }
    if(flag)
    {
        cur_dir.write_back_to_disk(my_cache, cur_dir_node.get_sec_beg());
    }
    return;
}


void myFS::rmdir(vector<string> args)
{
    string file_name=args[1];
    const char *name = file_name.c_str();
    int del_inode_num = -1;
    for(int i = 0; i < 15; i++)
    {
        if(strncmp(name, cur_dir.dirs[i].name, strlen(name)) == 0)
        {
            del_inode_num = cur_dir.dirs[i].inode_num;
            cout << "inode num of the dir is ： " << del_inode_num << endl;
            break;
        }
    }
    if(del_inode_num == -1)
    {
        cerr << "dir not exist" << endl;
    }


    Inode del_node;
    del_node.read_inode_from_disk(del_inode_num, my_cache);
```

```
	del_inode(del_node, cur_dir);
	cur_dir.write_back_to_disk(my_cache, cur_dir_node.get_sec_beg());

}

int myFS::is_existed_file(string file_name)
{
	const char *name = file_name.c_str();
	for(int i = 0; i < 15; i++)
	{
		if(strncmp(name, cur_dir.dirs[i].name, strlen(name)) == 0)
		{
			int inode_num = cur_dir.dirs[i].inode_num;
			return inode_num;

		}
	}
	return -1;
}

int myFS::get_dir_index(int inode_num)
{
	for(int i = 0; i < 15; i++)
	{
		if(inode_num==cur_dir.dirs[i].inode_num)
		{
			return i;
		}
	}
	return -1;
}


void myFS::vim(vector<string> args)
{
	string file_name=args[1];
	const char *name = file_name.c_str();
	int inode_num= is_existed_file(file_name);
	if(inode_num!=-1)
```

```
        {
            Inode file_node;
            file_node.read_inode_from_disk(inode_num, my_cache);
            if(!file_node._is_file)
            {
                cerr << "can not cat dir" << endl;
                return;
            }
            move_out(inode_num);
        }

//      string tstr="vim /tmp/myfs_temp";
        const char *vim_cmd="vim /tmp/myfs_temp";
        int return_val=system(vim_cmd);
        cout<<return_val<<"return_val"<<endl;
        if(return_val==0)
        {
            cout<<"move_in";
            move_in(inode_num,file_name);
        }
        return;
}


void myFS::cp(vector<string> args)
{
        string src_str=args[1];
        const char *src = src_str.c_str();
        string dest_str=args[1];
        const char *dest = src_str.c_str();
        int src_inode_num=is_existed_file(src_str);
        int dest_inode_num=is_existed_file(src_str);
        if(src_inode_num=-1||dest_inode_num==-1)
        {
            cerr<<"file not exist!";
        }
        Inode src_inode;
        Inode dest_inode;
        src_inode.read_inode_from_disk(src_inode_num,my_cache);
        dest_inode.read_inode_from_disk(dest_inode_num,my_cache);
```

```
    int src_index=get_dir_index(src_inode_num);

}


void myFS::cd(vector<string> args)
{
    string file_name=args[1];
    const char *name = file_name.c_str();
    // get subdir inode
    int dir_inode_num = -1;
    for(int i = 0; i < 15; i++)
    {
        if(strncmp(name, cur_dir.dirs[i].name, strlen(name)) == 0)
        {
            dir_inode_num = cur_dir.dirs[i].inode_num;
            cout << "inode num:" << dir_inode_num << endl;
            break;
        }
    }
    if(dir_inode_num == -1)
    {
        cerr << "cd: no such file or directory:" <<file_name << endl;
        return;
    }

    // find inode by inode num
    cur_dir_node.read_inode_from_disk(dir_inode_num, my_cache);

    // read info by inode info
    cur_dir.read_dir_from_disk(my_cache, cur_dir_node.get_sec_beg());
}
```

SuperBlcok

```
/* FileName:     superblock.hpp
 * Author:       Hover
 * E-Mail:       hover@hust.edu.cn
 * GitHub:       HoverWings
 * Description: superblock
 */
#ifndef SUPERBLOCK_H
```

```
#define SUPERBLOCK_H
#include <fstream>
#include <string.h>
#include <vector>
#include <assert.h>
#include <iostream>
#include <cstring>
#include "myfs_macro.h"
#include "inode.hpp"
//#include "myfs.hpp"


using namespace std;
class superblock
{
private:
	bool inode_bitmap[INODE_NUM];
	bool block_bitmap[BLOCK_NUM];
	fstream disk;



public:
	int cur_dir_node_num; // inode num
	int cur_dir_num;		// dir block num
	class myFS* myfs;

	int remain_inode();

	int remain_sec();

	int get_new_inode();

	int get_new_sec();

	bool recv_inode(int inode_num);

	bool recv_sec(int sec_num);

	superblock();
```

```
    ~superblock();

    bool init();
    void format_disk();
    void print_inode_bitmap();
    void print_block_bitmap();

    void read_from_disk();
    void write_to_disk();
//      fstream disk;

};

#endif
/* FileName:       superblock.cpp
  * Author:         Hover
  * E-Mail:         hover@hust.edu.cn
  * GitHub:         HoverWings
  * Description: superblock
  */
#include "superblock.hpp"
#include "myfs.hpp"
superblock::superblock()
{
    disk.open(DISK, std::fstream::in | std::fstream::out | std::fstream::binary);
    read_from_disk();
//      memset(inode_bitmap, 0, sizeof(inode_bitmap));
//      memset(block_bitmap, 0, sizeof(block_bitmap));
}

void superblock::format_disk()
{
    disk.seekg(BLOCK_BEGIN);
    const vector<char>zeroes(SEC_SIZE, 0);
    for (uint i = 0; i < BLOCK_NUM; ++i)
    {
        disk.write(zeroes.data(), SEC_SIZE);
    }
    disk.seekg(SUPER_BEGIN);
}
```

```
superblock::~superblock()
{
	write_to_disk();
}


void superblock::print_inode_bitmap()
{
	for(int i = 0; i < INODE_NUM; i++)
	{
		printf("%d",inode_bitmap[i]);
	}
};

void superblock::print_block_bitmap()
{
	for(int i = 0; i < INODE_NUM; i++)
	{
		printf("%d",block_bitmap[i]);
	}
};


int superblock::remain_inode()
{
	int count = 0;
	for(int i = 0; i < INODE_NUM; i++)
		if(!inode_bitmap[i])
			count++;
	return count;
}

int superblock::remain_sec()
{
	int count = 0;
	for(int i = 0; i < INODE_NUM; i++)
		if(!block_bitmap[i])
			count++;
	return count;
```

```
}


int superblock::get_new_inode()
{
	for(int i = 0; i < INODE_NUM; i++)
	{
		if(!inode_bitmap[i])
		{
			inode_bitmap[i] = true;
			return i;
		}
	}
	return -1;
}

int superblock::get_new_sec()
{
	for(int i = 0; i < BLOCK_NUM; i++)
		if(!block_bitmap[i])
		{
			block_bitmap[i] = true;
			return i + INODE_BEGIN / SEC_SIZE + (INODE_NUM *
sizeof(Inode)) / SEC_SIZE;
		}
	return -1;
}

bool superblock::recv_inode(int inode_num)
{
	assert(inode_num >= 0 && inode_num < INODE_NUM);

	inode_bitmap[inode_num] = false;
	return true;
}

bool superblock::recv_sec(int sec_num)
{
	// assert(sec_num >= 0 && sec_num < BLOCK_NUM);
	block_bitmap[sec_num] = false;
```

```
	return true;
}

bool superblock::init()
{
	memset(inode_bitmap, 0, INODE_NUM);
	memset(block_bitmap, 0, sizeof(block_bitmap));

	return true;
}

void superblock::read_from_disk()
{
	disk.seekg(SUPER_BEGIN);
//	  if(disk.is_open())
//	  {
//		cout<<"read open sus!";
//	  }f
//	  int i=123;
//	  int j=789;
//	  disk.seekg(SUPER_BEGIN);
//	  disk.write((const char*)&i, sizeof(int) * 1);
//	  disk.write((const char*)&j, sizeof(int) * 1);
	disk.seekg(SUPER_BEGIN);
//	  disk>>cur_dir_node_num>>cur_dir_num;
	disk.read((char*)&cur_dir_node_num, sizeof(int));
	disk.read((char*)&cur_dir_num, sizeof(int));
	disk.read((char*)inode_bitmap, sizeof(bool) * INODE_NUM);
	disk.read((char*)block_bitmap, sizeof(bool) * BLOCK_NUM);
//	  cout<<cur_dir_node_num<<cur_dir_num;
}

void superblock::write_to_disk()
{
	cur_dir_node_num=myfs->cur_dir_node.get_inode_num();
	cur_dir_num=myfs->cur_dir_node.get_sec_beg();
//	  cout<<cur_dir_node_num<<cur_dir_num;
	if(disk.is_open())
	{
		cout<<"wriet open sus!";
```

```
    }
    disk.seekg(SUPER_BEGIN);
//    disk<<cur_dir_node_num<<cur_dir_num;
    disk.write((const char*)&cur_dir_node_num, sizeof(int));
    disk.write((const char*)&cur_dir_num, sizeof(int));
    disk.write((const char*)inode_bitmap, sizeof(bool) * INODE_NUM);
    disk.write((const char*)block_bitmap, sizeof(bool) * BLOCK_NUM);
    disk.close();

}
```

CMakeLists.txt

```
#cmake verson
cmake_minimum_required(VERSION 3.2)

#project name
PROJECT(myfs)

#head file path
INCLUDE_DIRECTORIES(include)

#source directory
AUX_SOURCE_DIRECTORY(src DIR_SRCS)

SET(src_total ${DIR_SRCS} src/superblock.cpp)

#add executable file，
ADD_EXECUTABLE(${PROJECT_NAME} ${src_total})

find_package(Boost REQUIRED COMPONENTS serialization)

if(NOT Boost_FOUND)
    message("Not found Boost")
endif()

include_directories(${Boost_INCLUDE_DIRS})
message("${Boost_INCLUDE_DIRS}")
message("${Boost_LIBRARIES}")
target_link_libraries(${PROJECT_NAME} ${Boost_LIBRARIES})
```

myFileSystem_mem

DirEntry

```
/* FileName:       direntry.hpp
  * Author:          Hover
  * E-Mail:          hover@hust.edu.cn
  * GitHub:          HoverWings
  * Description: direntry
  */
#ifndef _DIRENTRY_H_
#define _DIRENTRY_H_

#include <list>
#include <memory>
#include <string>
#include <sys/types.h>
#include "freenode.hpp"
#include "inode.hpp"

using namespace std;

enum EntryType { file, dir };

class DirEntry: public enable_shared_from_this<DirEntry>
{
        public:
        DirEntry();
        static shared_ptr<DirEntry> make_de_dir(const string name,
                                                                const
shared_ptr<DirEntry> parent);
        static shared_ptr<DirEntry> make_de_file(const string name,
                                                                const
shared_ptr<DirEntry> parent,
                                                                const shared_ptr<Inode>
&inode=nullptr);
        uint block_size;
        EntryType type;                         //file or dir
        string name;
        weak_ptr<DirEntry> parent; // .
        weak_ptr<DirEntry> self;      // ..
```

```
    shared_ptr<Inode> inode;      // file
    list<shared_ptr<DirEntry>> contents; // dir entry
    bool is_locked;                                  // lock

    shared_ptr<DirEntry> find_child(const string name) const;
    shared_ptr<DirEntry> add_dir(const string name);
    shared_ptr<DirEntry> add_file(const string name);
};

#endif /* _DIRENTRY_H_ */

/* FileName:       direntry.cpp
  * Author:          Hover
  * E-Mail:          hover@hust.edu.cn
  * GitHub:          HoverWings
  * Description: direntry
  */
#include "../include/direntry.hpp"
#include <algorithm>
#include <sstream>
#include <vector>

using std::find_if;
using std::istringstream;
using std::make_shared;
using std::shared_ptr;
using std::string;
using std::vector;
using std::weak_ptr;

//init DirEntry
DirEntry::DirEntry()
{
    is_locked = false;
}

//D: make Dir
shared_ptr<DirEntry> DirEntry::make_de_dir(const string name,const
shared_ptr<DirEntry> parent)
{
```

```
    auto sp = make_shared<DirEntry>(DirEntry()); // de ptr
    if (parent == nullptr)    // .=..
    {
        sp->parent = sp;
    }
    else
    {
        sp->parent = parent;
    }
    sp->type = dir;
    sp->self = sp;
    sp->name = name;
    sp->inode = nullptr;
    return sp;
}

//D: make file
shared_ptr<DirEntry> DirEntry::make_de_file(const string name,
                                                        const shared_ptr<DirEntry>
parent,
                                                        const shared_ptr<Inode>
&inode)
{
    auto sp = make_shared<DirEntry>(DirEntry());
    if (parent == nullptr)
    {
        sp->parent = sp;
    }
    else
    {
        sp->parent = parent;
    }
    sp->type = file;
    sp->self = sp;
    sp->name = name;
    sp->inode = inode;
    return sp;
}

//D: find child(cd)
```

```
shared_ptr<DirEntry> DirEntry::find_child(const string name) const
{
	// handle . and ..
	if (name == "..")
	{
		return parent.lock();
	}
	else if (name == ".")
	{
		return self.lock();
	}

	// search through contents and return ptr if found, otherwise nullptr for traveling
all the ptr auto
	auto named = [&] (const shared_ptr<DirEntry> de) {return de->name ==
name;};
	auto it = find_if(begin(contents), end(contents), named);
	if (it == end(contents))
	{
		return nullptr;
	}
	return *it;
}

// wrap make dir
shared_ptr<DirEntry> DirEntry::add_dir(const string name)
{
	auto new_dir = make_de_dir(name, self.lock());
	contents.push_back(new_dir);
	return new_dir;
}

//wrap make file
shared_ptr<DirEntry> DirEntry::add_file(const string name)
{
	auto new_file = make_de_file(name, self.lock(), make_shared<Inode>());
	contents.push_back(new_file);
	return new_file;
}
```

FreeNode

```
/* FileName:       freenode.hpp
  * Author:          Hover
  * E-Mail:          hover@hust.edu.cn
  * GitHub:          HoverWings
  * Description: freenode
  */
#ifndef _FREENODE_H_
#define _FREENODE_H_

#include <list>
#include <sys/types.h>

class FreeNode
{
      public:
      uint block_num;        //free node num
      uint pos;              //start pos
      FreeNode(uint block_num, uint pos): block_num(block_num),pos(pos) {}
};

#endif /* _FREENODE_H_ */
```

Inode

```
/* FileName:       inode.hpp
  * Author:          Hover
  * E-Mail:          hover@hust.edu.cn
  * GitHub:          HoverWings
  * Description: inode
  */
#ifndef _INODE_H_
#define _INODE_H_

#include <sys/types.h>
#include <list>
#include <memory>
#include <vector>
#include <string>
```

```
#include "freenode.hpp"
#include "macro.h"
using namespace std;



class Inode
{
	public:
	long create_time;
	uint inode_num;
	uint sec_num;

	uint size;
	uint blocks_used;
	static uint block_size;
	static list<FreeNode> *free_list; // freenode list


	// use unique_ptr to ensure the alloc err , if false then recollect
	vector<uint> d_blocks;
	unique_ptr<std::vector<std::vector<uint>>> i_blocks; // i_blocks

	Inode();
	~Inode();
};

#endif /* _INODE_H_ */

/* FileName:	inode.cpp
  * Author:		Hover
  * E-Mail:		hover@hust.edu.cn
  * GitHub:		HoverWings
  * Description: inode
  */
#include "../include/inode.hpp"
#include <algorithm>
#include <list>
#include <vector>
```

```
using std::list;
using std::shared_ptr;
using std::sort;
using std::vector;

uint Inode::block_size = 0;
list<FreeNode> * Inode::free_list = nullptr;

Inode::Inode(): size(0), blocks_used(0), i_blocks(new vector<vector<uint>>())
{
	inode_num= static_cast<uint>(inode_total);
	inode_total++;
}

Inode::~Inode()
{
	if (blocks_used == 0)
	{
		return;
	}
	else
	{
		if(blocks_used == 1)
		{
			free_list->emplace_front(block_size, d_blocks[0]);
		}
	}

	vector<uint> blocks;

	for (uint block : d_blocks)
	{
		blocks.push_back(block);
	}

	for (auto vec: *i_blocks)
	{
		for (uint block: vec)
		{
			blocks.push_back(block);
```

```
        }
    }

    sort(begin(blocks), end(blocks));

    uint start = blocks.front();
    uint last = start;
    uint size = block_size;
    blocks.erase(begin(blocks));
    for (uint block : blocks)
    {
        if (block - last != block_size)
        {
            free_list->emplace_back(size, start);
            start= block;
            last = start;
            size = 0;
        }
        else
        {
            last = block;
            size += block_size;
        }
    }
    free_list->emplace_front(size, start);
}
```

Marco.h

```
/* FileName:       MACRO.H
  * Author:         Hover
  * E-Mail:         hover@hust.edu.cn
  * GitHub:         HoverWings
  * Description: MACRO
  */

#ifndef MYFS_MACRO_H
#define MYFS_MACRO_H

#include <fstream>
```

```
//#include ""
#include "stdint.h"
using namespace std;
#define SEC_SIZE 512
#define INODE_NUM 1024
#define BLOCK_NUM 1024

//fstream disk_file;
#define DISK    "disk.img"
#define SUPER_BEGIN 0
#define INODE_BEGIN SEC_SIZE*10
#define BLOCK_BEGIN INODE_BEGIN+SEC_SIZE*BLOCK_NUM
#define MAX_SEC    ((BLOCK_BEGIN + BLOCK_NUM * SEC_SIZE) /
SEC_SIZE )

static int inode_total=0;

#endif //MYFS_MACRO_H
```

Myfs

```
/* FileName:       myfs.hpp
  * Author:         Hover
  * E-Mail:         hover@hust.edu.cn
  * GitHub:         HoverWings
  * Description: myfs
  */
#ifndef _MYFS_H_
#define _MYFS_H_


//#include<boost/serialization/vector.hpp>
#include <fstream>
#include <list>
#include <map>
#include <string>
#include <vector>
#include "macro.h"
#include "inode.hpp"
#include "direntry.hpp"
```

```
#include "freenode.hpp"
#include "superblock.hpp"
#include "Buffer.hpp"


using namespace std;


class myFS
{

    enum Mode {R, W, RW}; // open file mode

    // Mode
    struct Descriptor
    {
        Mode mode;
        uint byte_pos;
        weak_ptr<Inode> inode;
        weak_ptr<DirEntry> from;
        uint fd;
    };
    bool getMode(Mode *mode, string mode_s);

    //return path
    struct PathRet
    {
        bool invalid_path = false;
        string final_name;
        shared_ptr<DirEntry> parent_node;
        shared_ptr<DirEntry> final_node;
    };

    // can be seen as superblock
    string filename;        //disk file name

    uint block_size;
    uint direct_blocks;
    uint block_num;
```

```
// DirEntry root;
list<FreeNode>free_list;
shared_ptr<DirEntry> root_dir;
shared_ptr<DirEntry> pwd;
map<uint, Descriptor> open_files;    //save open file uint and desp
uint next_descriptor = 0;

void init_disk(const string& filename);
unique_ptr<PathRet> parse_path(string path_str) const;
bool basic_open(Descriptor *d, vector <string> args);
unique_ptr<string> basic_read(Descriptor &desc, const uint size);
uint basic_write(Descriptor &desc, const string data);
bool basic_close(uint fd);

public:
myFS(string& filename);
~myFS();


fstream disk_file;


class SuperBlock* p_sp;

// current dir
Inode cur_dir_node; // current dir inode
DirEntry cur_dir;      //current dir

// cache
Buffer fs_cache;

void open(vector<string> args);
void read(vector<string> args);
void write(vector<string> args);
void seek(vector<string> args);
void close(vector<string> args);
void mkdir(vector<string> args);
void rmdir(vector<string> args);
void cd(vector<string> args);
void link(vector<string> args);
```

```
    void unlink(vector<string> args);
    void stat(vector<string> args);
    void ls(vector<string> args);
    void cat(vector<string> args);
    void cp(vector<string> args);
    void tree(vector<string> args);
    void import(vector<string> args);
    void printwd(vector<string> args);

    string getpwd(vector<string> args);

    void FS_export(vector<string> args);


   friend class boost::serialization::access;
    template<class Archive>
    void save(Archive & ar, const unsigned int version) const
    {
        ar &filename;
        ar &blocks_num;
        ar &block_size;
        ar &direct_blocks;
    }
    template<class Archive>
    void load(Archive & ar, const unsigned int version)
    {
        ar &filename;
        ar &blocks_num;
        ar &block_size;
        ar &direct_blocks;
    }

};

#endif /* _MYFS_H_ */


#include "myfs.hpp"
#include <cmath>
#include <iostream>
```

```
#include <iomanip>
#include <list>
#include <memory>
#include <sstream>    //istringstream
#include <string>
#include <vector>
#include <deque>
#include <assert.h>
//#include "../include/direntry.hpp"
//#include "../include/inode.hpp"
//#include "../include/freenode.hpp"

using namespace std;

// less than opt required
#define ops_at_least(x)                                                    \
    if (static_cast<int>(args.size()) < x+1) {                    \
        cerr << args[0] << ": missing operand" << endl;          \
        return;                                                            \
    }

// more than opt required
#define ops_less_than(x)                                                   \
    if (static_cast<int>(args.size()) > x+1) {                    \
        cerr << args[0] << ": too many operands" << endl;      \
        return;                                                            \
    }\


// check exact opt arg nums
#define ops_exactly(x)                                          \
    ops_at_least(x);                                            \
    ops_less_than(x);

//// Constructor
//myFS::myFS(const string& filename,
//                    const uint fs_size,
//                    const uint block_size,
//                    const uint direct_blocks):
//                filename(filename),
```

```
//              block_size(block_size),
//              direct_blocks(direct_blocks),
//              block_num(ceil(static_cast<double>(fs_size) / block_size))
//{
//      // init inode
//      Inode::block_size = block_size;
//      Inode::free_list = &free_list;
//
//
//      root_dir = DirEntry::make_de_dir("root", nullptr);
//
//      // start at root dir/ set pwd
//      pwd = root_dir;
//
//      // init disk
//      init_disk(filename);
//      free_list.emplace_back(block_num, 0);
//}

// Constructor
myFS::myFS(string& filename)
{
    block_num=BLOCK_NUM;

    // init inode
    Inode::block_size = block_size;
    Inode::free_list = &free_list;

    class SuperBlock *p_sb=new SuperBlock();
    p_sb->myfs=this;
    p_sb->write_back_to_disk();

    root_dir = DirEntry::make_de_dir("root", nullptr); // make dir de

    // start at root dir/ set pwd
    pwd = root_dir;

    // init disk
    init_disk(filename);
    free_list.emplace_back(block_num, 0); // make freenode
```

```
}

myFS::~myFS()
{
	disk_file.close();
	remove(filename.c_str());
}


void myFS::init_disk(const string& filename)
{
	// write disk with 0, prevent some dirty data
	const vector<char>zeroes(block_num, 0);

	disk_file.open(filename,
				fstream::in |
				fstream::out |
				fstream::binary |
				fstream::trunc);

	for (uint i = 0; i < block_num; ++i)
	{
		disk_file.write(zeroes.data(), block_size);
	}
}

//D: parase path Layer by layer to the last node
//I: path string
//O: pathret ptr
unique_ptr<myFS::PathRet> myFS::parse_path(string path_str) const
{
	unique_ptr<PathRet> ret(new PathRet);

	// check if path is relative or absolute
	ret->final_node = pwd;

	// pwd==root
	if (path_str[0] =='/')
	{
		path_str.erase(0,1);
```

```
        ret->final_node = root_dir;
    }

    // initialize data structure
    ret->final_name = ret->final_node->name;
    ret->parent_node = ret->final_node->parent.lock();

    // tokenize the string, redirector
    istringstream is(path_str);
    string token;
    vector<string> path_tokens;
    while (getline(is, token, '/'))
    {
        path_tokens.push_back(token);
    }

    // walk the path updating pointers
    for (auto &node_name : path_tokens)
    {
        // something other than the last entry was not found
        if (ret->final_node == nullptr)
        {
            ret->invalid_path = true;
            return ret;
        }
        ret->parent_node = ret->final_node;
        ret->final_node = ret->final_node->find_child(node_name);
        ret->final_name = node_name;
    }

    return ret;
}

//TODO: change this
bool myFS::getMode(Mode *mode, string mode_s)
{
    if (mode_s == "w")
    {
        *mode = W;
    }
```

```
        else if(mode_s == "r")
        {
        *mode = R;
        }
        else if (mode_s == "rw")
        {
        *mode = RW;
        }
        else
        {
                return false;
        }
        return true;
}


bool myFS::basic_open(Descriptor *d, vector <string> args)
{
        assert(args.size() == 3);

        Mode mode;
        auto path = parse_path(args[1]);                        //path ret
        auto node = path->final_node;                           //final node
        auto parent = path->parent_node;                        //parent node
        bool known_mode = getMode(&mode, args[2]);

        if (path->invalid_path == true)
        {
                cerr << args[0] << ": error: Invalid path: " << args[1] << endl;
        }
        else if(!known_mode)
        {
                cerr << args[0] << ": error: Unknown mode: " << args[2] << endl;
        }
        else if (node == nullptr && (mode == R || mode == RW))
        {
                cerr << args[0] << ": error: " << args[1] << " does not exist." << endl;
        }
        else if (node != nullptr && node->type == dir)
        {
```

```
            cerr << args[0] << ": error: Cannot open a directory." << endl;
        }
        else if (node != nullptr && node->is_locked)
        {
            cerr << args[0] << ": error: " << args[1] << " is already open." << endl;
        }
        else
        {
            //create the file if the file not exist
            if(node == nullptr)
            {
                node = parent->add_file(path->final_name);
            }
            // get a descriptor
            uint fd = next_descriptor++;
            node->is_locked = true;
            *d = Descriptor{mode, 0, node->inode, node, fd};
            open_files[fd] = *d;            //save open file descriptor
            return true;
        }
        return false;
}

// wrap basic open
void myFS::open(vector<string> args)
{
        ops_exactly(2);
        Descriptor desc;
        if (basic_open(&desc, args))
        {
            cout << "SUCCESS: fd=" << desc.fd << endl;
        }
}

//D: wrap basic_read safely, read a file
//I: read filename
//O: S/F
void myFS::read(vector<string> args)
{
        ops_exactly(2);
```

```
    uint fd;
    if ( !(istringstream(args[1]) >> fd))
    {
        cerr << "read: error: Unknown descriptor." << endl;
        return;
    }
    auto desc_it = open_files.find(fd);
    if (desc_it == open_files.end())        // last is empty
    {
        cerr << "read: error: File descriptor not open." << endl;
        return;
    }
    auto &desc = desc_it->second; //map value
    if(desc.mode != R && desc.mode != RW)
    {
        cerr << "read: error: " << args[1] << " not open for read." << endl;
        return;
    }

    uint size;
    if (!(istringstream(args[2]) >> size))
    {
        cerr << "read: error: Invalid read size." << endl;
    }
    else if (size + desc.byte_pos > desc.inode.lock()->size) // Out of read ava zone
    {
        cerr << "read: error: Read goes beyond file end." << endl;
    }
    else
    {
        auto data = basic_read(desc, size);
        cout << *data << endl;
    }
}

//I: desc, read size
//O: dara ptr
unique_ptr<string> myFS::basic_read(Descriptor &desc, const uint size)
{
```

```
    //for constrite the size, using char instead of string
    char *data = new char[size];
    char *data_p = data;
    uint &pos = desc.byte_pos;
    uint bytes_to_read = size;
    auto inode = desc.inode.lock();

    uint dbytes = direct_blocks * block_size;      //
    while (bytes_to_read > 0)
    {
        uint read_size = min(bytes_to_read, block_size - pos % block_size); //
prevent reading out of range
        uint read_src;
        if (pos < dbytes)
        {
            read_src = inode->d_blocks[pos / block_size] + pos % block_size;
        }
        else
        {
            uint i = (pos - dbytes) / (direct_blocks * block_size);
            uint j = (pos - dbytes) / block_size % direct_blocks;
            read_src = inode->i_blocks->at(i)[j] + pos % block_size;
        }
        disk_file.seekp(read_src);
        disk_file.read(data_p, read_size);
        pos += read_size;
        data_p += read_size;
        bytes_to_read -= read_size;
    }
    return unique_ptr<string>(new string(data, size));
}

//D: wrap write safely
//I: write filename
//O: S/F
void myFS::write(vector<string> args)
{
    ops_exactly(2);

    uint fd;
```

```
    uint max_size = block_size * (direct_blocks + direct_blocks * direct_blocks);
    if ( !(istringstream(args[1]) >> fd))      // desc error
    {
        cerr << "write: error: Unknown descriptor." << endl;
    }
    else
    {
        auto desc = open_files.find(fd);
        if (desc == open_files.end())      //final not open
        {
            cerr << "write: error: File descriptor not open." << endl;
        }
        else if (desc->second.mode != W && desc->second.mode != RW)
        {
            cerr << "write: error: " << args[1] << " not open for write." << endl;
        }
        else if (desc->second.byte_pos + args[2].size() > max_size)
        {
            cerr << "write: error: File to large for inode." << endl;
        }
        else if (!basic_write(desc->second, args[2])) // ONLY reason!
        {
            cerr << "write: error: Insufficient disk space." << endl;
        }
    }
}

uint myFS::basic_write(Descriptor &desc, const string data)
{
    const char *bytes = data.c_str();
    uint &pos = desc.byte_pos;
    uint bytes_to_write = data.size();                    // expected to write
    uint bytes_written = 0;                                   // already write
    auto inode = desc.inode.lock();
    uint &file_size = inode->size;
    uint &file_blocks_used = inode->blocks_used;
    uint new_size = max(file_size, pos + bytes_to_write);
    uint new_blocks_used = ceil(static_cast<double>(new_size)/block_size);
    uint blocks_needed = new_blocks_used - file_blocks_used;
    uint dbytes = direct_blocks * block_size;
```

```
    // expand the inode to indirect blocks if needed
    if (blocks_needed && blocks_needed + file_blocks_used > 2)
    {
        // expand inode vec
        uint ivec_used = (ceil(min(file_blocks_used - 2, 0U) /
static_cast<float>(direct_blocks)));
        uint ivec_new = (ceil((new_blocks_used - 2) /
static_cast<float>(direct_blocks)));
        while (ivec_used < ivec_new)
        {
            inode->i_blocks->push_back(vector<uint>());
            ivec_used++;
        }
    }

    // find space
    vector<pair<uint, uint>> free_chunks;
    auto fl_it = begin(free_list);
    while (blocks_needed > 0) // can be more effecient
    {
        if (fl_it == end(free_list))
        {
            // 0 return because ran out of free space, find no space
            return 0;
        }
        if (fl_it->block_num > blocks_needed)    // chunk big enough to hold the
rest of our write
        {
            free_chunks.push_back(make_pair(fl_it->pos, blocks_needed));
            fl_it->pos += blocks_needed * block_size;
            fl_it->block_num -= blocks_needed;
            break;
        }
        // a chunk, but will fill it and need more, then find another chunk
        free_chunks.push_back((make_pair(fl_it->pos, fl_it->block_num)));
        blocks_needed -= fl_it->block_num;
        auto used_entry = fl_it++;
        free_list.erase(used_entry);
    }
```

```
    // allocate blocks
    for (auto fc_it : free_chunks)
    {
        uint block_pos = fc_it.first;
        uint block_num = fc_it.second;
        for (uint k = 0; k < block_num; ++k, ++file_blocks_used, block_pos +=
block_size)
        {
            if (file_blocks_used < direct_blocks)
            {
                inode->d_blocks.push_back(block_pos);
            }
            else
            {
                uint i = ((file_blocks_used - direct_blocks) / direct_blocks);
                inode->i_blocks->at(i).push_back(block_pos);
            }
        }
    }

    // actually write our blocks
    while (bytes_to_write > 0)
    {
        uint write_size = min(block_size - pos % block_size, bytes_to_write);
        uint write_dest = 0;
        if (pos < dbytes)
        {
            write_dest = inode->d_blocks[pos / block_size] + pos % block_size;
        }
        else
        {
            uint i = (pos - dbytes) / (direct_blocks * block_size);
            uint j = (pos - dbytes) / block_size % direct_blocks;
            write_dest = inode->i_blocks->at(i)[j] + pos % block_size;
        }
        disk_file.seekp(write_dest);
        disk_file.write(bytes + bytes_written, write_size);
        bytes_written += write_size;
        bytes_to_write -= write_size;
```

```
            pos += write_size;
        }

        disk_file.flush();
        file_size = new_size;
        return bytes_written;
}

//D: change the pos of desc
//I: seek
void myFS::seek(vector<string> args)
{
        ops_exactly(2);
        uint fd;
        if ( !(istringstream(args[1]) >> fd))
        {
            cerr << "seek: error: Unknown descriptor." << endl;
            return;
        }
        auto desc_it = open_files.find(fd);
        if (desc_it == open_files.end())
        {
            cerr << "seek: error: File descriptor not open." << endl;
            return;
        }
        auto &desc = desc_it->second;
        uint pos;
        if (!(istringstream(args[2]) >> pos))
        {
            cerr << "seek: error: Invalid position." << endl;
        }
        else if (pos > desc.inode.lock()->size)
        {
            cerr << "seek: error: Position outside file." << endl;
        }
        else
        {
            desc.byte_pos = pos;
        }
}
```

```
bool myFS::basic_close(uint fd)
{
	auto kv = open_files.find(fd);
	if(kv == open_files.end())	// file do not open
	{
		return false;
	}
	else
	{
		kv->second.from.lock()->is_locked = false;
		open_files.erase(fd);
	}
	return true;
}

//TODO
//D: wrap basic_close
//I:
void myFS::close(vector<string> args)
{
	ops_exactly(1);
	uint fd;

	if (! (istringstream (args[1]) >> fd))
	{
		cerr << "close: error: File descriptor not recognized" << endl;
	}
	else
	{
		if (!basic_close(fd))
		{
			cerr << "close: error: File descriptor not open" << endl;
		}
		else
		{
			cout << "closed " << fd << endl;
		}
	}
```

```
}

//D: mkdir, can not recrusive
void myFS::mkdir(vector<string> args)
{
        ops_at_least(1);
    /* add each new directory one at a time */
        for (uint i = 1; i < args.size(); i++)
        {
                auto path = parse_path(args[i]);        // final inode and constuct inode
                auto node = path->final_node;
                auto dirname = path->final_name;
                auto parent = path->parent_node;

                if (path->invalid_path)
                {
                    cerr << "mkdir: error: Invalid path: " << args[i] << endl;
                    return;
                }
                else if (node == root_dir)
                {
                    cerr << "mkdir: error: Cannot recreate root." << endl;
                    return;
                }
                else if (node != nullptr)
                {
                    cerr << "mkdir: error: " << args[i] << " already exists." << endl;
                    continue;
                }

                /* actually add the directory */
                parent->add_dir(dirname);
        }
}

//D: rm -r dir
//O: S/F
void myFS::rmdir(vector<string> args)
{
        ops_at_least(1);
```

```
    for (uint i = 1; i < args.size(); i++)
    {
        auto path = parse_path(args[i]);
        auto node = path->final_node;
        auto parent = path->parent_node;

        if (node == nullptr)
        {
            cerr << "rmdir: error: Invalid path: " << args[i] << endl;
        }
        else if (node == root_dir)
        {
            cerr << "rmdir: error: Cannot remove root." << endl;
        }
        else if (node == pwd)
        {
            cerr << "rmdir: error: Cannot remove working directory." << endl;
        }
        else if (node->contents.size() > 0)
        {
            cerr << "rmdir: error: Directory not empty." << endl;
        }
        else if (node->type != dir)
        {
            cerr << "rmdir: error: " << node->name << " must be directory." <<
endl;
        }
        else
        {
            parent->contents.remove(node);
        }
    }
}

//D: print pwd
void myFS::printwd(vector<string> args)
{
    ops_exactly(0);
```

```
    if (pwd == root_dir)
    {
        cout << "/" << endl;
        return;
    }

    auto wd = pwd;
    deque<string> plist;
    while (wd != root_dir)
    {
        plist.push_front(wd->name);
        wd = wd->parent.lock();
    }

    for (auto dirname : plist)
    {
        cout << "/" << dirname;
    }
    cout << endl;
}


std::string myFS::getpwd(vector<string> args)
{
    // ops_exactly(0);
    std::string str;
    if (pwd == root_dir)
    {
        str="/";
        // cout << "/" << endl;
        return str;
    }

    auto wd = pwd;
    deque<string> plist;
    while (wd != root_dir)
    {
        plist.push_front(wd->name);
        wd = wd->parent.lock();
    }
```

```
	// str+="/";
	for (auto dirname : plist)
	{
		str+="/";
		str+=dirname;
	}
	return str;
	// cout << endl;

}

//D: change dir
//I: cd dir
void myFS::cd(vector<string> args)
{
	ops_exactly(1);

	auto path = parse_path(args[1]);
	auto node = path->final_node;

	if (node == nullptr)
	{
		cerr << "cd: error: Invalid path: " << args[1] << endl;
	}
	else if (node->type != dir)
	{
		cerr << "cd: error: " << args[1] << " must be a directory." << endl;
	}
	else
	{
		pwd = node;
	}
}

void myFS::link(vector<string> args)
{
	ops_exactly(2);

	auto src_path = parse_path(args[1]);
	auto src = src_path->final_node;
```

```
	auto src_parent = src_path->parent_node;
	auto dest_path = parse_path(args[2]);
	auto dest = dest_path->final_node;
	auto dest_parent = dest_path->parent_node;
	auto dest_name = dest_path->final_name;

	if (src == nullptr)
	{
		cerr << "link: error: Cannot find " << args[1] << endl;
	}
	else if (dest != nullptr)
	{
		cerr << "link: error: " << args[2] << " already exists." << endl;
	}
	else if (src->type != file)
	{
		cerr << "link: error: " << args[1] << " must be a file." << endl;
	}
	else if (src_parent == dest_parent)
	{
		cerr << "link: error: src and dest must be in different directories." << endl;
	}
	else
	{
		auto new_file = DirEntry::make_de_file(dest_name, dest_parent,
src->inode);
		dest_parent->contents.push_back(new_file);
	}
}

void myFS::unlink(vector<string> args)
{
	ops_exactly(1);

	auto path = parse_path(args[1]);
	auto node = path->final_node;
	auto parent = path->parent_node;

	if (node == nullptr)
	{
```

```
            cerr << "unlink: error: File not found." << endl;
        }
        else if (node->type != file)
        {
            cerr << "unlink: error: " << args[1] << " must be a file." << endl;
        }
        else if (node->is_locked)
        {
            cerr << "unlink: error: " << args[1] << " is open." << endl;
        }
        else
        {
            parent->contents.remove(node);
        }
}

void myFS::stat(vector<string> args)
{
        ops_at_least(1);

        for (uint i = 1; i < args.size(); i++)
        {
            auto path = parse_path(args[i]);
            auto node = path->final_node;

            if (node == nullptr)
            {
                cerr << "stat: error: " << args[i] << " not found." << endl;
            }
            else
            {
                cout << "    File: " << node->name << endl;
                if (node->type == file)
                {
                    cout << "    Type: file" << endl;
                    cout << " Inode: " << node->inode.get() << endl;
                    cout << " Links: " << node->inode.use_count() << endl;
                    cout << "    Size: " << node->inode->size << endl;
                    cout << "Blocks: " << node->inode->blocks_used << endl;
                }
```

```
                else if(node->type == dir)
                {
                        cout << "    Type: directory" << endl;
                }
            }
        }
}

void myFS::ls(vector<string> args)
{
        ops_exactly(0);
        for (auto dir : pwd->contents)
        {
            cout << dir->name << endl;
        }
}

void myFS::cat(vector<string> args)
{
        ops_at_least(1);

        for(uint i = 1; i < args.size(); i++)
        {
            Descriptor desc;
            if(!basic_open(&desc, vector<string> {args[0], args[i], "r"}))
            {
            /* failed to open */
            continue;
            }

            auto size = desc.inode.lock()->size;
            read(vector<string>{args[0], std::to_string(desc.fd), std::to_string(size)});
            basic_close(desc.fd);
        }
}

void myFS::cp(vector<string> args)
{
        ops_exactly(2);
```

```
    Descriptor src, dest;
    if(basic_open(&src, vector<string> {args[0], args[1], "r"}))
    {
        if(!basic_open(&dest, vector<string> {args[0], args[2], "w"}))
        {
            basic_close(src.fd);
        }
        else
        {
            auto data = basic_read(src, src.inode.lock()->size);
            if (!basic_write(dest, *data))
            {
                cerr << args[0] << ": error: out of free space or file too large"<<
endl;
            }
            basic_close(src.fd);
            basic_close(dest.fd);
        }
    }
}

void tree_helper(shared_ptr<DirEntry> directory, string indent)
{
    auto cont = directory->contents;
    if (directory->type == file)
    {
        cout << directory->name << ": " << directory->inode->size << " bytes" <<
endl;
    }
    else
    {
        cout << directory->name << endl;
    }
    if (cont.size() == 0) return;

    if (cont.size() >= 2)
    {
        auto last = *(cont.rbegin());
        for (auto entry = cont.begin(); *entry != last; entry++)
        {
```

```
                cout << indent << " ├────";
                tree_helper(*entry, indent + " │      ");
            }
        }

        cout << indent << " └────";
        tree_helper(*(cont.rbegin()), indent + "       ");
}

void myFS::tree(vector<string> args)
{
    ops_exactly(0);

    tree_helper(pwd, "");
}

//D: file only import
void myFS::import(vector<string> args)
{
//      ops_exactly(2);
        Descriptor desc;
        fstream in(args[1]);
        if(!in.is_open())
        {
            cerr << args[0] << ": error: Unable to open " << args[1] << endl;
            return;
        }
        if (basic_open(&desc, vector<string>{args[0], args[2], "w"}))
        {
            string data((istreambuf_iterator<char>(in)), istreambuf_iterator<char>());
            if (!basic_write(desc, data))
            {
                cerr << args[0] << ": error: out of free space or file too large"<< endl;
            }
            basic_close(desc.fd);
        }
}

//D: file only export
void myFS::FS_export(vector<string> args)
```

```
{
    ops_exactly(2);

    Descriptor desc;
    ofstream out(args[2], ofstream::binary);
    if (!out.is_open())
    {
        cerr << args[0] << ": error: Unable to open " << args[2] << endl;
        return;
    }

    if (basic_open(&desc, vector<string>{args[0], args[1], "r"}))
    {
        unique_ptr<string> data = basic_read(desc, desc.inode.lock()->size);
        out << *data;
        basic_close(desc.fd);
    }
}
```

SuperBlock

```
/* FileName:     superblock.hpp
  * Author:       Hover
  * E-Mail:       hover@hust.edu.cn
  * GitHub:       HoverWings
  * Description: superblock
  */
#ifndef _SUPERBLOCK_H_
#define _SUPERBLOCK_H_
#include "inode.hpp"
#include <fstream>
#include <list>
#include <map>
#include <string>
#include <vector>
#include <cstring>
#include "macro.h"
#include "assert.h"
#include "myfs.hpp"

#include <boost/archive/text_oarchive.hpp>
```

```
#include <boost/archive/text_iarchive.hpp>

#include <boost/archive/binary_iarchive.hpp>
#include <boost/archive/binary_oarchive.hpp>


class myFS;
class SuperBlock
{
	private:
		bool inode_bitmap[INODE_NUM];
		bool block_bitmap[BLOCK_NUM];

	public:
//		uint node_num;		// now inode num
//		uint direct_blocks;
//		uint blocks_num;
//		uint block_size;
		int q;
		class myFS* myfs;
		SuperBlock();
		bool write_back_to_disk();

		bool read_from_disk();

		// template<class Archive>
		// void serialize(Archive & ar, const unsigned int version)
		// {
		//	ar& inode_bitmap;
		//	ar& block_bitmap;
		// }
};

#endif
/* FileName:	superblock.cpp
 * Author:		Hover
 * E-Mail:		hover@hust.edu.cn
 * GitHub:		HoverWings
 * Description: superblock
 */
```

```
#include "superblock.hpp"



using namespace boost;
//Inode::Inode():
SuperBlock::SuperBlock()
{

	static uint node_num;		// now inode num
	memset(inode_bitmap, 0, sizeof(inode_bitmap));
	memset(block_bitmap, 0, sizeof(block_bitmap));
}


bool SuperBlock::write_back_to_disk()
{
	stringstream binary_sstream;
	boost::archive::binary_oarchive binary_oa(binary_sstream);
	binary_oa<<block_bitmap;
	binary_oa<<inode_bitmap;
	cout<<binary_sstream.str();
//	binary_sstream<<binary_sstream;
}
```

CMakeLists.txt

```
#cmake verson
cmake_minimum_required(VERSION 3.2)

#project name
PROJECT(myfs)

#head file path
INCLUDE_DIRECTORIES(
		include
)

#source directory
AUX_SOURCE_DIRECTORY(src DIR_SRCS)
```

```
SET(src_total
        ${DIR_SRCS}
        include/macro.h)

#add executable file，
ADD_EXECUTABLE(${PROJECT_NAME} ${src_total})

find_package(Boost REQUIRED COMPONENTS
        # regex
        serialization
        )
if(NOT Boost_FOUND)
    message("Not found Boost")
endif()

include_directories(${Boost_INCLUDE_DIRS})
message("${Boost_INCLUDE_DIRS}")
message("${Boost_LIBRARIES}")

target_link_libraries(${PROJECT_NAME} ${Boost_LIBRARIES})
```