

**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](#_Toc1778504284)

[2 文件拷贝 2](#_Toc1942977029)

[2.1 设计目的 2](#_Toc726692081)

[2.2 实验要求和内容 2](#_Toc756623234)

[2.3 环境及步骤 2](#_Toc1675859713)

[2.4 关键代码 4](#_Toc1204969998)

[2.5 调试记录及运行结果 6](#_Toc1457198856)

[2.6 设计感想 7](#_Toc1333650324)

[3 图形化进程并发 8](#_Toc42343758)

[3.1 实验要求和内容 8](#_Toc1182665230)

[3.2 实验过程与结果 8](#_Toc403293356)

[3.3 实验结果分析 8](#_Toc1858015722)

[3.4 心得与体会 9](#_Toc1326636341)

[4 添加系统调用 10](#_Toc695737139)

[4.1 实验要求和内容 10](#_Toc599851311)

[4.2 实验步骤 10](#_Toc772135697)

[4.3 实验过程与结果 13](#_Toc495133248)

[4.4 实验结果分析 15](#_Toc780221273)

[4.5 心得与体会 15](#_Toc1912973257)

[5 设备驱动 16](#_Toc689674388)

[5.1 设计目的和内容 16](#_Toc1755271941)

[5.2 环境及步骤 16](#_Toc532359840)

[5.3 设计实现及关键代码 16](#_Toc1303484350)

[5.5 设计感想 19](#_Toc1177709566)

[6 QT系统监控器 20](#_Toc1210693838)

[6.1 设计目的 20](#_Toc2143368777)

[6.2 设计内容 20](#_Toc2096538880)

[6.3 环境及步骤 20](#_Toc767841850)

[6.4 设计实现及关键代码 21](#_Toc943020470)

[6.6 调试记录及运行结果 27](#_Toc1267520906)

[6.7 设计感想 30](#_Toc1218644581)

[7 模拟文件系统设计 31](#_Toc574041106)

[7.1 设计目的 31](#_Toc1063014287)

[7.2 设计内容 31](#_Toc1945336662)

[7.3 环境及步骤 31](#_Toc1330664341)

[7.4 内存版本设计实现 32](#_Toc591390353)

[7.5 硬盘版本设计实现 35](#_Toc1002823012)

[7.5 实验总结 42](#_Toc640379549)

[参考文献 43](#_Toc1925040677)

[附录 44](#_Toc1045166770)

[QTTest 44](#_Toc1823044779)

[mycp 52](#_Toc180850385)

[mySystemCall 59](#_Toc755698844)

[myDevDriver 65](#_Toc1002197472)

[myLinuxMonitor 70](#_Toc876587524)

[myFileSystem 129](#_Toc1355550155)

[myFileSystem\_mem 169](#_Toc1774333169)

# 

# 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

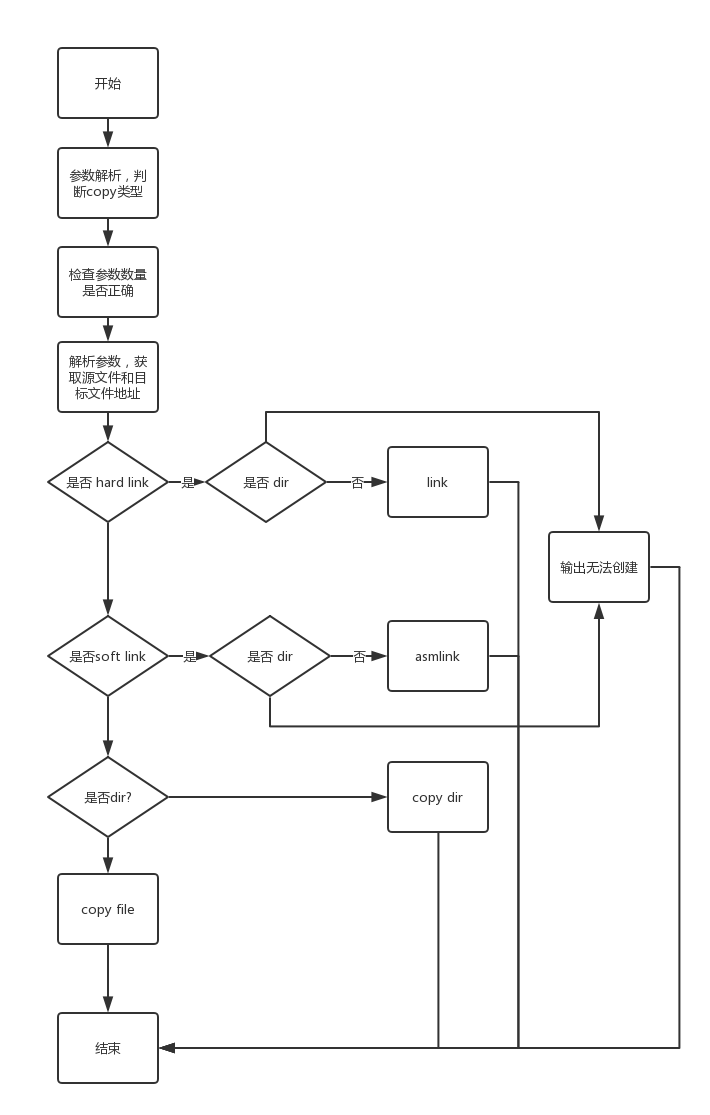


图2-1 copy函数流程图

## 2.4 关键代码

### 2.4.1 copyF2F

1. 函数原型

int copyF2F(char \*src\_file, char \*dest\_file)

1. 函数流程

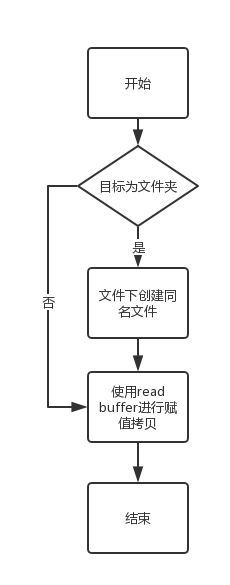


图2-2 copyF2F函数流程图

1. 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

1. 函数流程

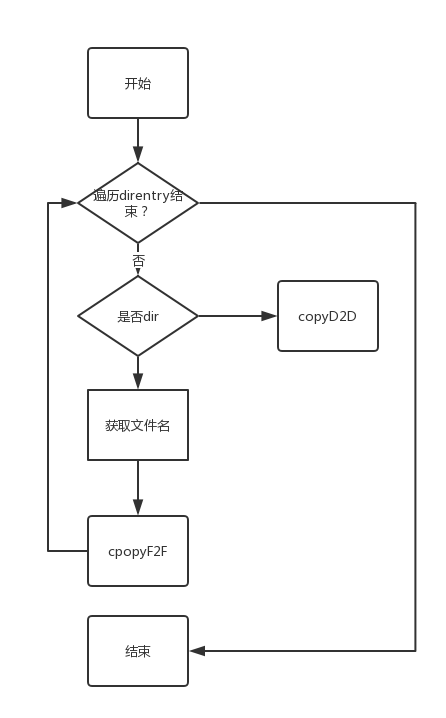


图2-3 copyD2D函数流程图

1. 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

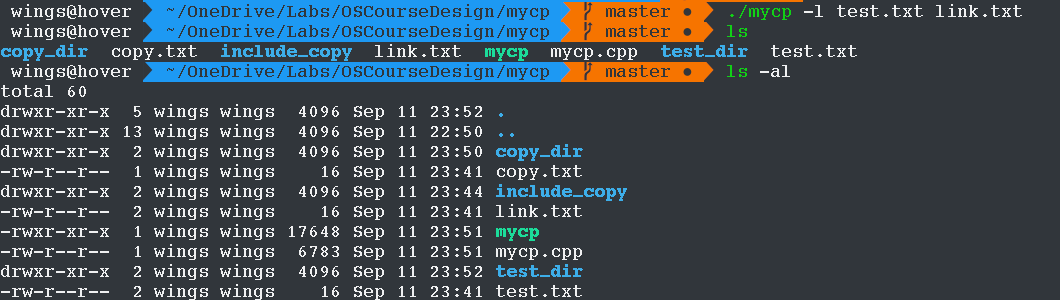


图2-6 link Test

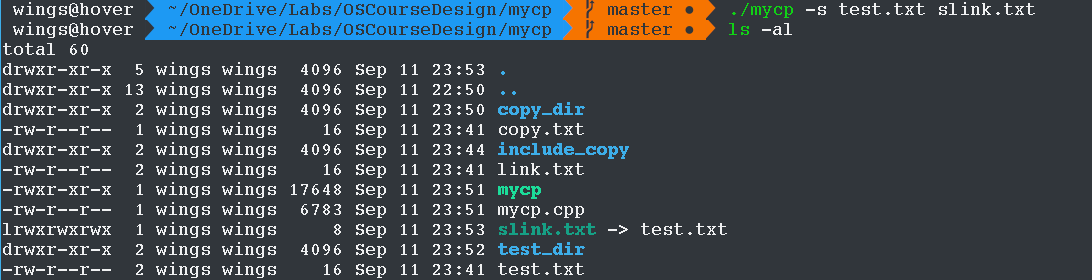


图2-7 syblink Test

## 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

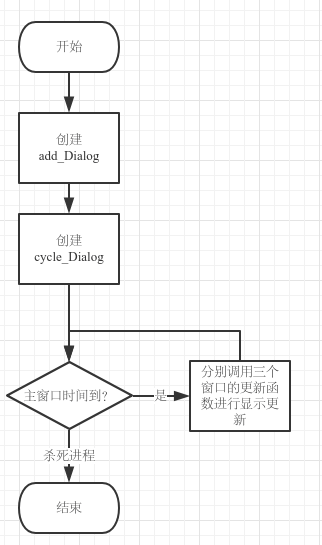


图3-1 三窗口流程图

## 3.3 实验结果分析

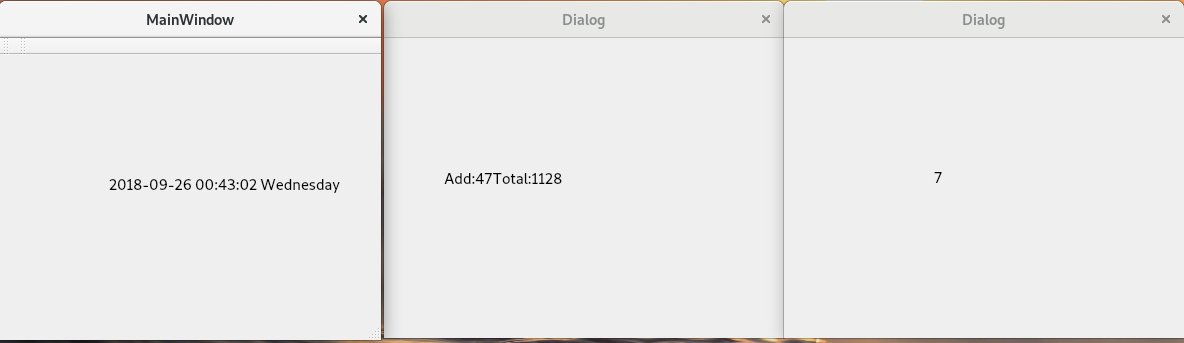


图3-2 三窗口测试

## 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。

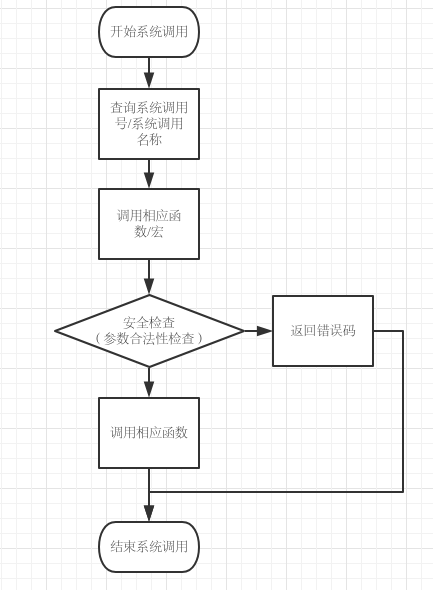


图4-1 系统调用分析

### 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。:

* 1. 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);

* 1. 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

1. 编译内核模块

Make modules

1. 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。

1. 编译内核映像

make bzImage

1. 编译内核模块

make modules

1. 生成并安装模块

make modules\_install

1. 安装新的系统

make install

1. 重启，选择新修改的内核
2. 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



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

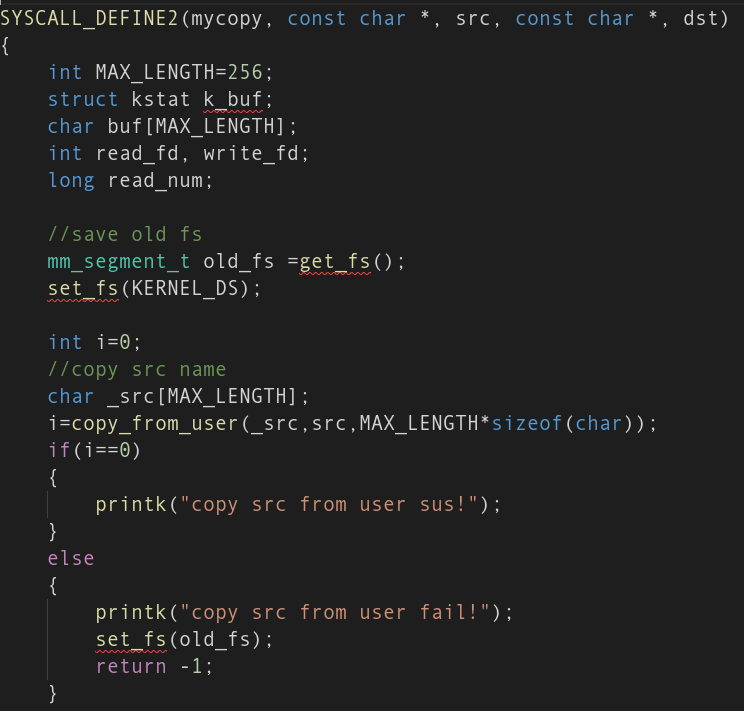


图4-2 系统调用声明截图

1. 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。



图4-3 系统调用表截图

1. 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

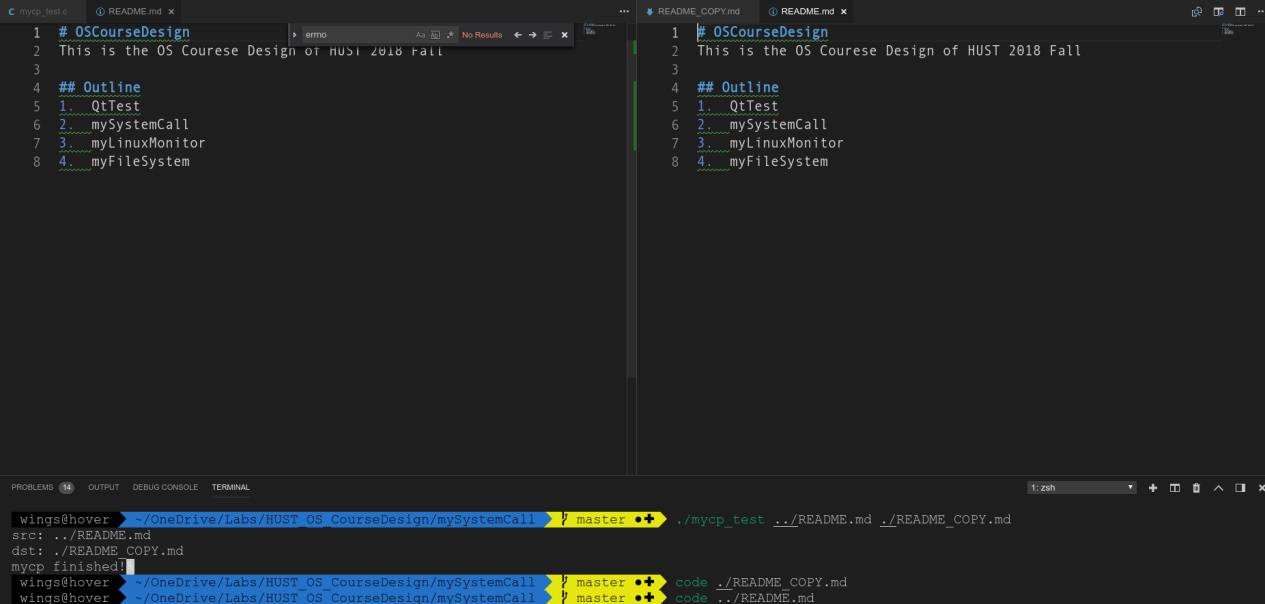


图4-4 系统调用测试

## 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

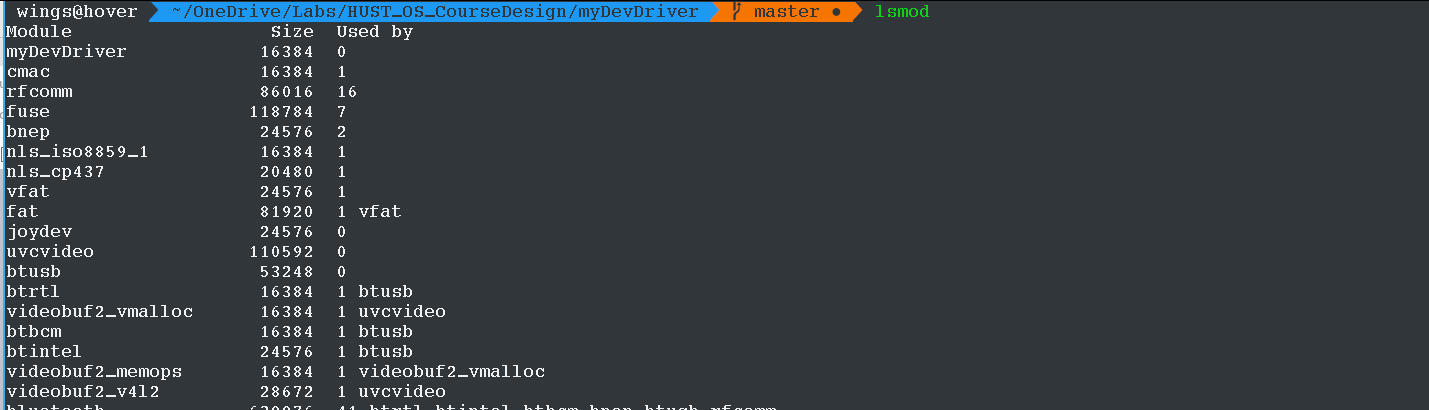
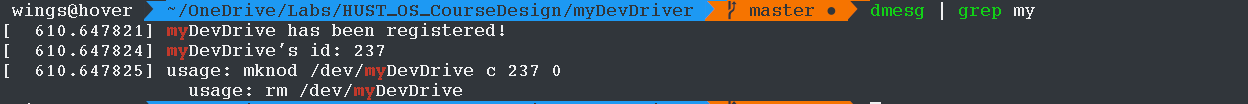


图5-1 设备驱动装载结果

图5-2 设备驱动调试信息

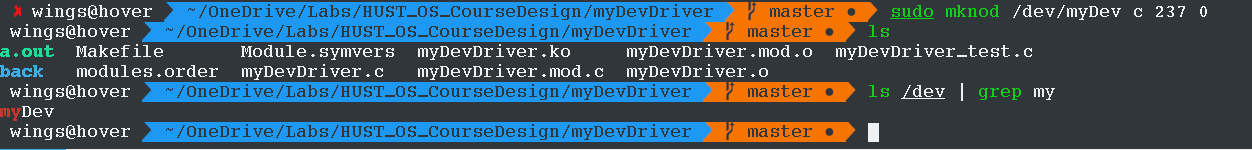


图5-3 分配设备号

### 5.3.4 设备驱动测试

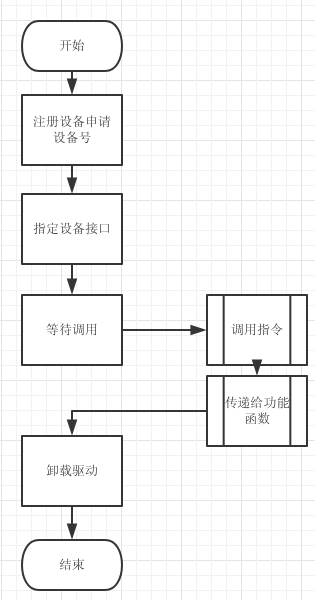


图5-4 设备驱动测试流程图

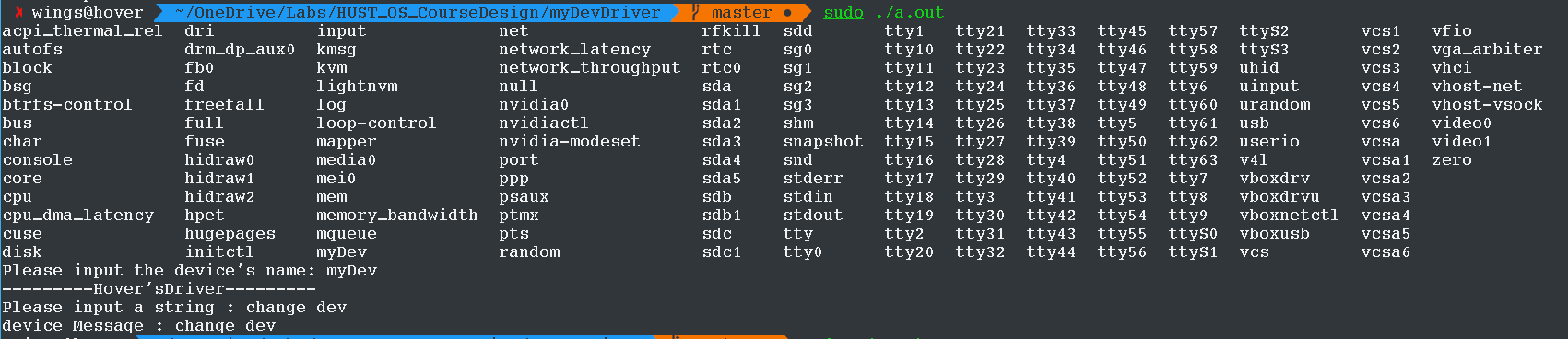


图5-5 设备驱动测试结果

## 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:
   1. 获取CPU的时间信息，利用间隔时间进行CPU利用率计算
   2. 将新的CPU信息点加入图表，进行刷新
3. Process
   1. 定时读取进程信息
   2. 维护proc内存池，进行process列表更新
   3. 进行table展示的更新
   4. 设置焦点和当前所在的页面
4. Net:
   1. 获取上一秒和当期秒当前网卡的数据包信息
   2. 进行数据负载的计算
   3. 维护chart展示，计算60s之类的峰值，进行表格的适当展示
5. Mem:
   1. 读取内存信息
   2. 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
   1. 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;

};

* 1. 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];

};

* 1. proc\_list

struct proc\_list

{

struct proc\_info \*\*array;

int size;

};

1. 更新过程

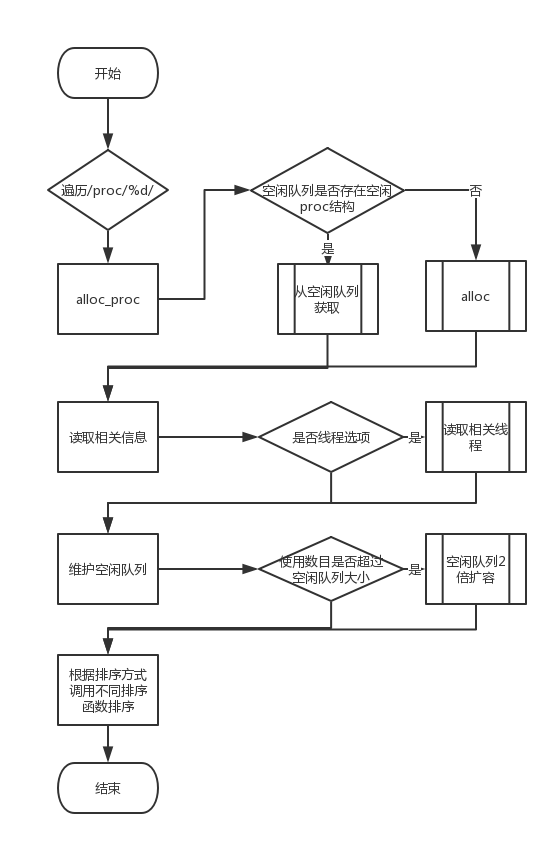


图5-1 CPU信息更新过程流程图

1. 利用率计算
   1. 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);

* 1. 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;

1. 图表绘制

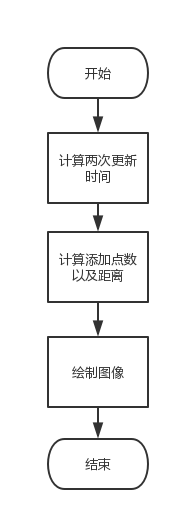


图5-2 CPU绘图过程流程图

### 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。

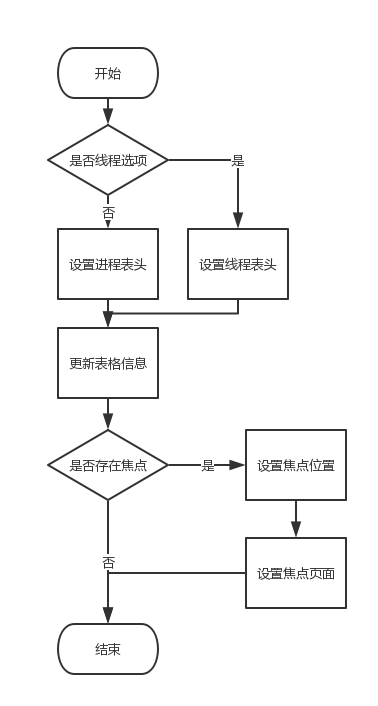


图5-3 CPU模块流程图

### 6.4.3 Memory

1. 更新内存信息
   1. 打开/proc/meminfo文件
   2. 定位不同的标识符，更新内存信息
   3. 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

1. 图表绘制

绘制内存曲线图和饼图

1. 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

1. 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

1. 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

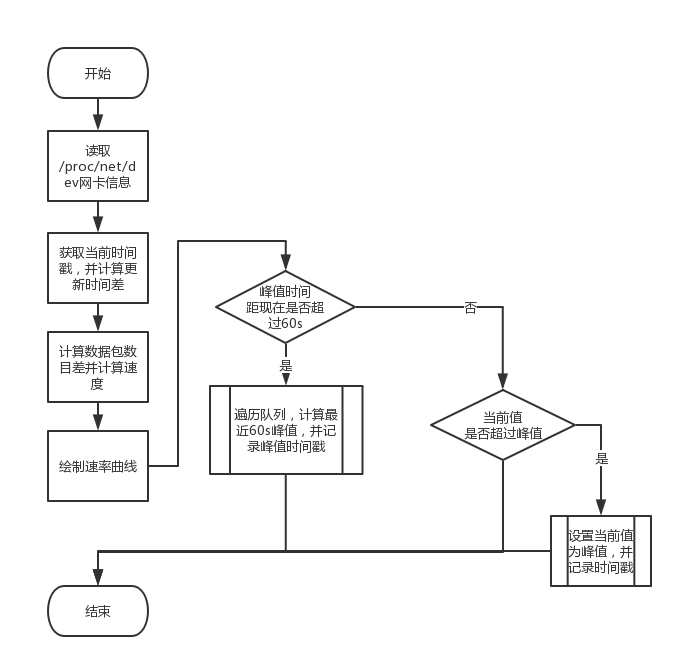


图5-4 Net模块流程图

### 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
   1. 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
   2. blocks\_percent\_used=blocks\_used/disktotal;
3. 调用绘图绘制利用率曲线图

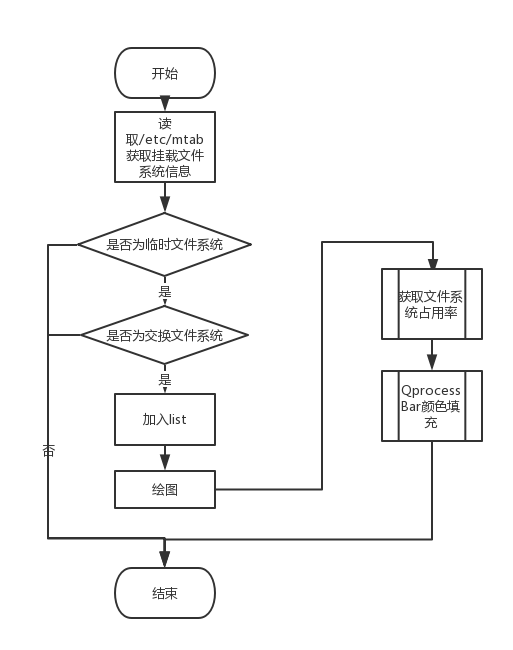


图5-5 FileSystem模块流程图

## 6.6 调试记录及运行结果

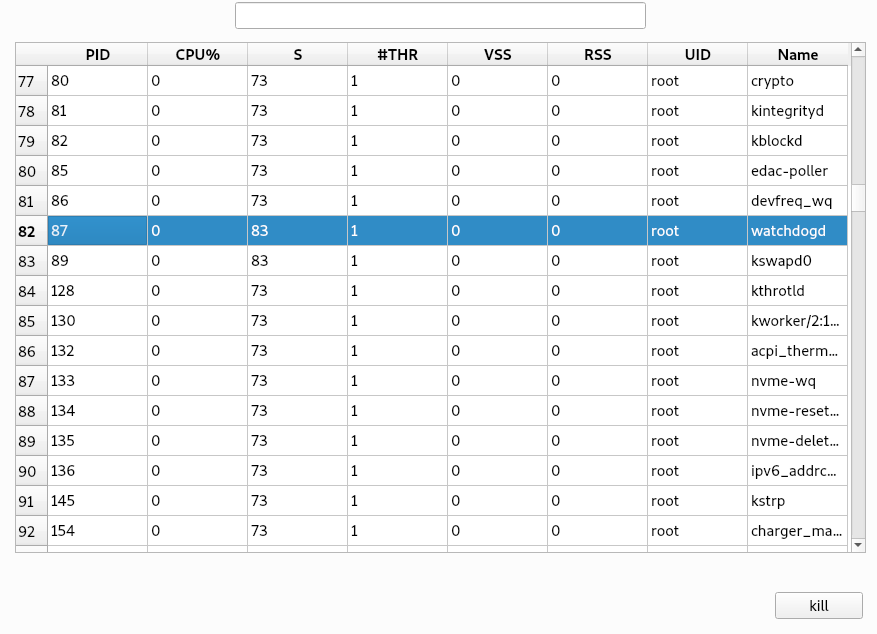


图5-6 Process模块排序测试图

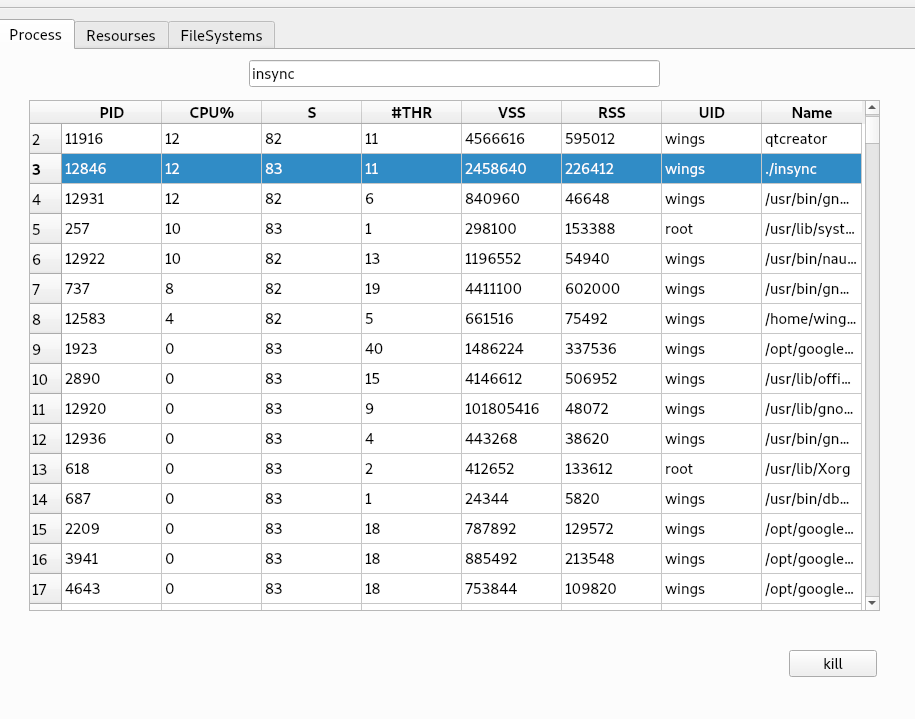


图5-7 Process模块搜索测试图

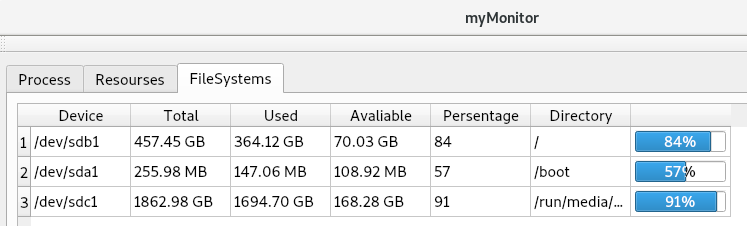


图5-7 FileSystem模块外接设备测试图

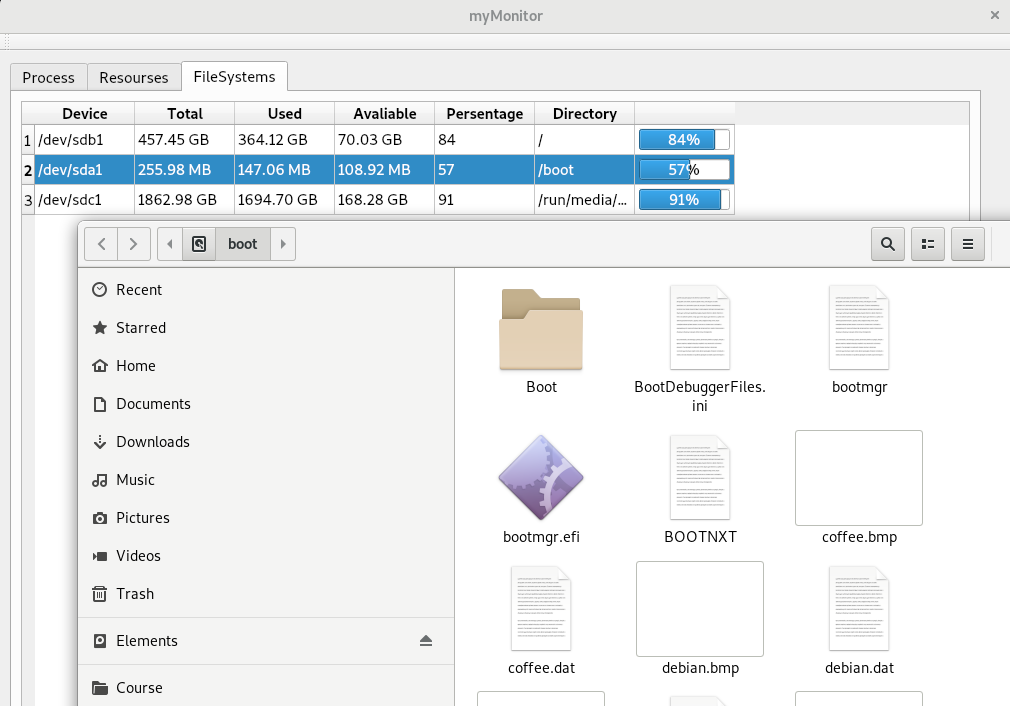


图5-7 FileSystem模块双击打开测试图



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

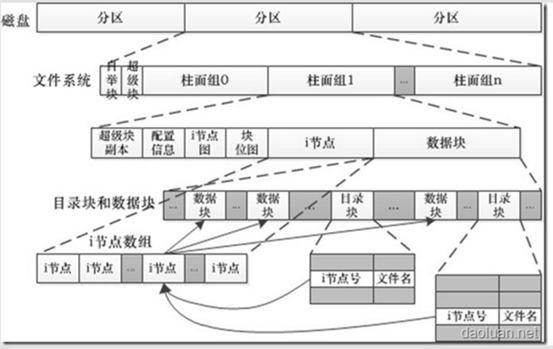


图7-1 系统结构图

## 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

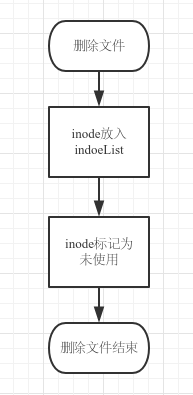


图7-2 删除文件操作

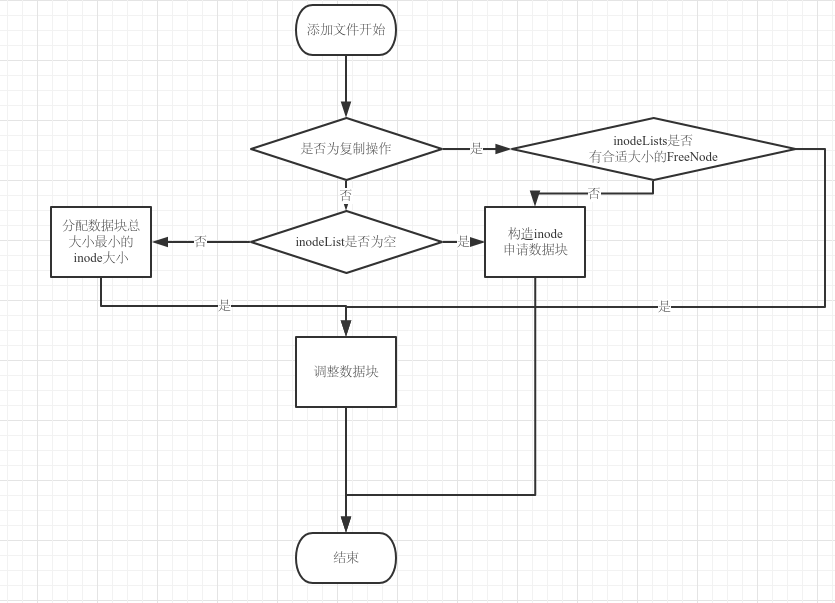


图7-3 添加文件操作

1. 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。

1. 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。

1. 关键成员
   1. 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);

* 1. 目录入口

weak\_ptr<DirEntry> parent; // .

weak\_ptr<DirEntry> self; // ..

list<shared\_ptr<DirEntry>> contents; // dir entry

* 1. 锁变量

bool is\_locked; // lock

* 1. 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。

1. 关键成员
   1. 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;

};

* 1. 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;

};

* 1. 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);

1. 操作函数

所有操作函数接口包括：

文件操作：open,read,write,seek.close,

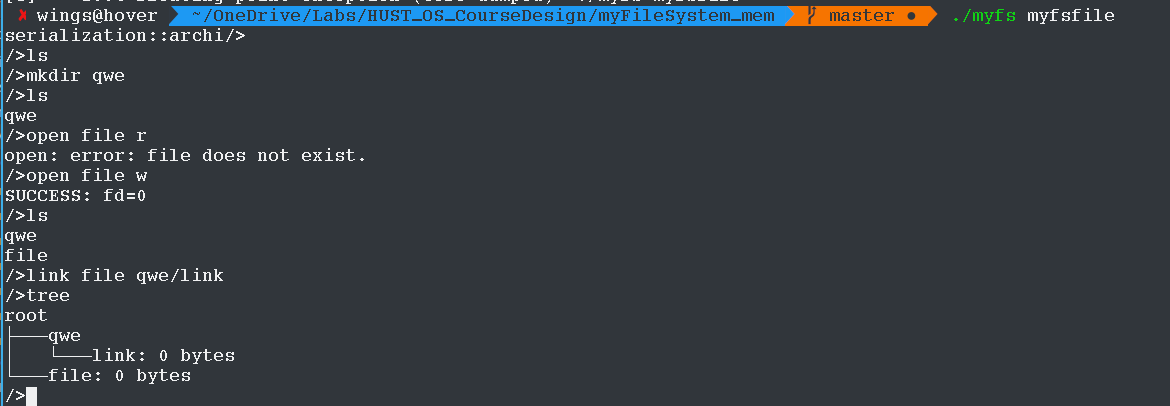
文件夹操作：mkdir,rmdir

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

1. 关键成员
   1. 函数接口

bool write\_disk(const BufferNode& node);

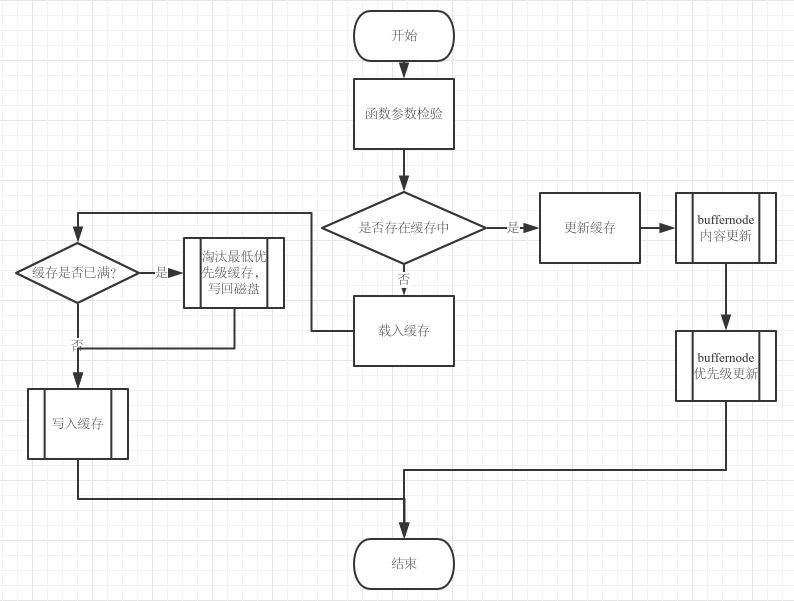


图7-5 写磁盘操作流程图

bool read\_disk(int sec\_num, BufferNode& node);

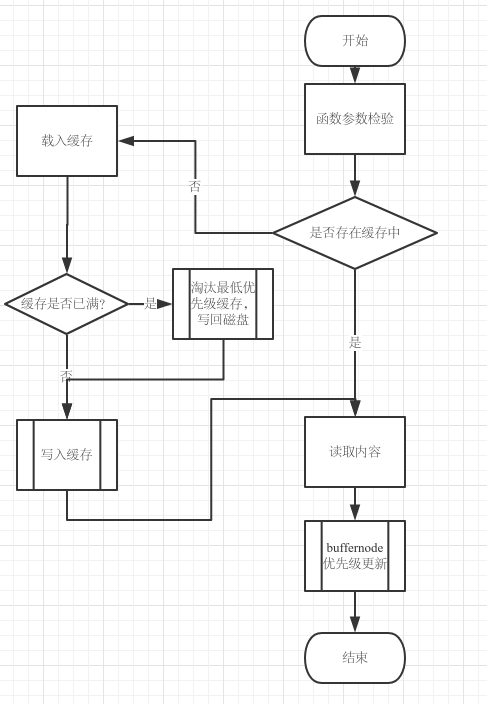


图7-6 读磁盘操作流程图

void all\_write\_to\_disk();

* 1. 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();

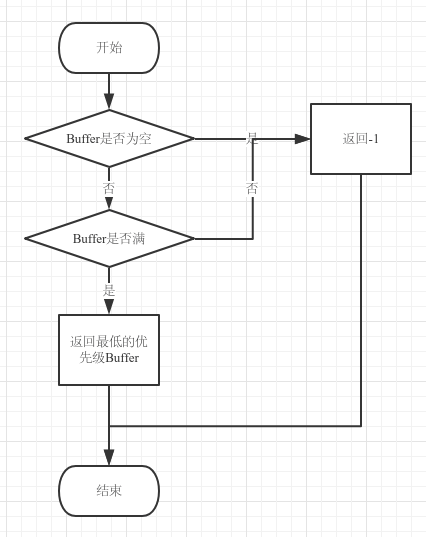


图7-7 查找最低优先级缓冲流程图

### 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

1. 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;

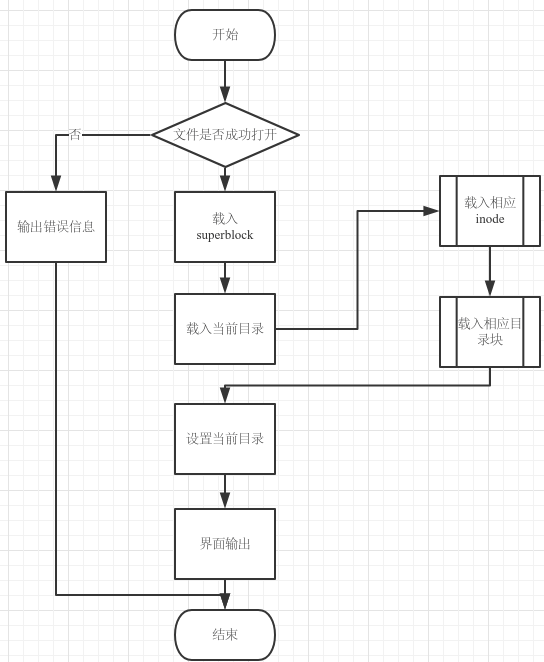


图7-8 系统初始化流程图

### 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。

1. 关键成员

Vim:使用vim编辑器编辑文件：

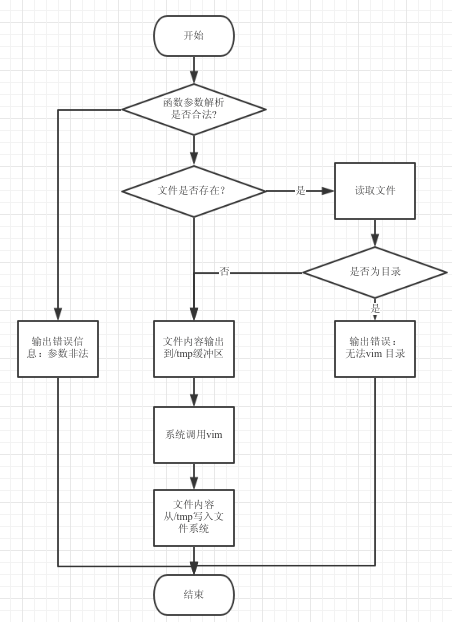


图7-9 vim系统调用流程图

### 7.5.4 系统测试

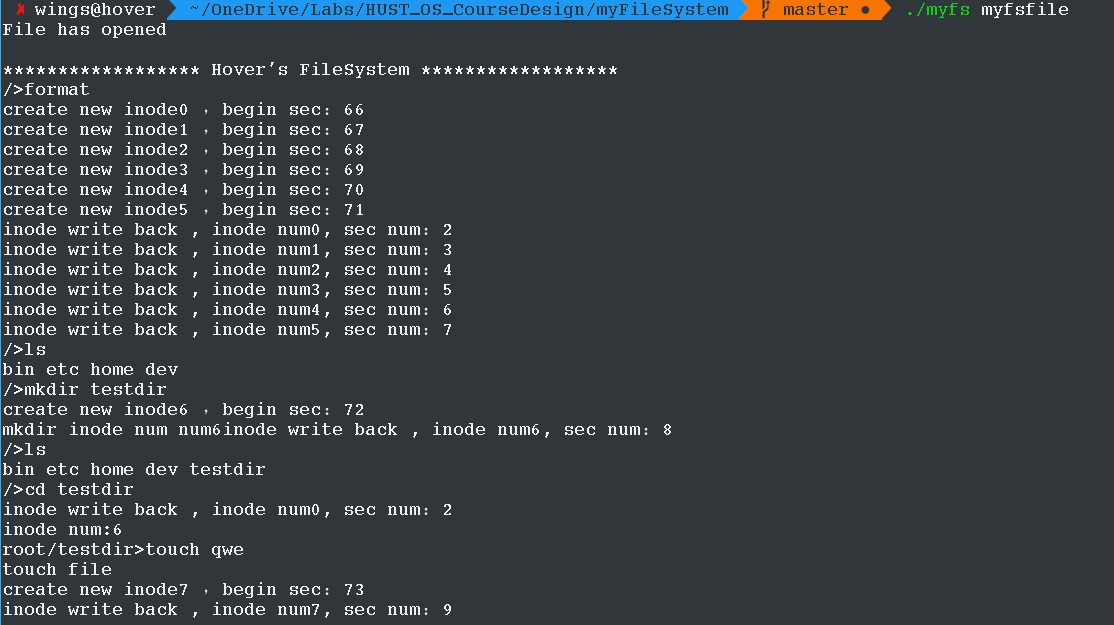


图7-10 目录函数测试

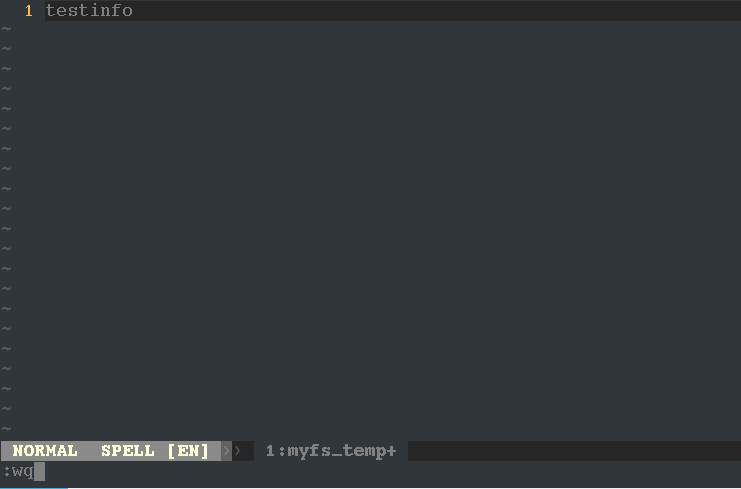


图7-11 vim系统调用测试1

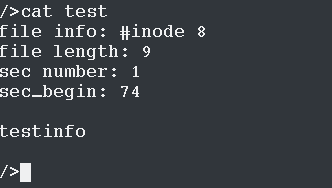


图7-12 vim系统调用测试2

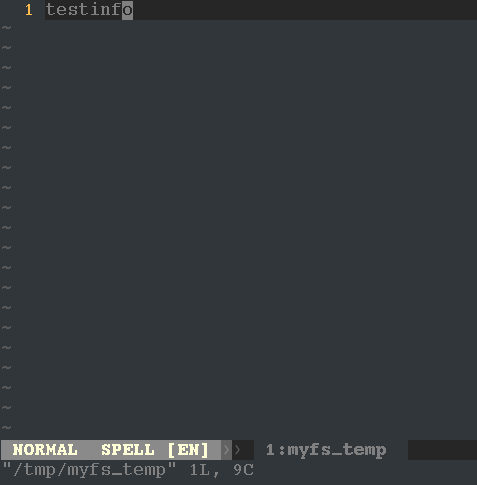


图7-13 vim系统调用测试3

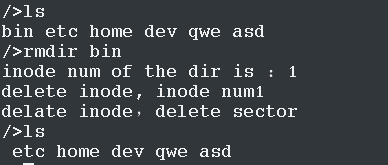


图7-14 rm测试

## 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/>

1. 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(total));

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);//single 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})