

**课程设计报告**

**课程名称： 嵌入式操作系统**

**专业班级： 物联网工程1801班**

**学 号： U201814500**

**姓 名： 王英嘉**

**指导教师： 石柯**

**报告日期： 2020年9月27日**

**计算机科学与技术学院**

目录

[1 课程设计概述 3](#_Toc941126710)

[2 熟悉和理解Linux编程环境 4](#_Toc1667268168)

[2.1 目的与要求 4](#_Toc1939570655)

[2.2 内容 4](#_Toc255589740)

[2.3 过程与结果分析 5](#_Toc311378546)

[3 添加系统调用 9](#_Toc1423088207)

[3.1 目的与要求 9](#_Toc1882506908)

[3.2 内容 9](#_Toc518686615)

[3.3 过程与结果分析 10](#_Toc387907449)

[4 添加设备驱动 21](#_Toc99098531)

[4.1 目的与要求 21](#_Toc684707970)

[4.2 内容 21](#_Toc1194607639)

[4.3 过程与结果分析 22](#_Toc1938756424)

[5 理解和分析/proc文件 27](#_Toc545885542)

[5.1 目的与要求 27](#_Toc582471222)

[5.2 内容 27](#_Toc330086341)

[5.3 过程与结果分析 28](#_Toc900029419)

[6 模拟文件系统设计 44](#_Toc815799124)

[6.1 目的与要求 44](#_Toc1412676665)

[6.2 内容 44](#_Toc347155541)

[6.3 过程与结果分析 45](#_Toc853056355)

[7 课设总结与心得 62](#_Toc1284220662)

[8 附录 63](#_Toc225700428)

[（一）熟悉和理解linux编程环境 63](#_Toc471577588)

[（二）添加系统调用 63](#_Toc1641178749)

[（三）添加设备驱动 63](#_Toc1907577162)

[（四）理解/proc文件 63](#_Toc1407874516)

[（五）模拟文件系统设计 63](#_Toc1610500458)

## 课程设计概述

表1.1实验测试环境

|  |  |
| --- | --- |
| 配 置 | 版本 |
| CPU | Intel® Core™ i7-8550U CPU @ 1.80GHz × 8 |
| OS  内核版本 | Ubuntu 18.04.5 LTS  linux-4.15.0-115-generic |

图1.1 系统调用表

嵌入式操作系统课程设计包含五个子任务，分别是熟悉和理解linux编程环境、添加系统调用、添加设备驱动、理解和分析/proc文件和模拟文件系统设计。

通过课程设计，可以进一步掌握linux操作系统的使用方法、了解linux系统内核代码结构、掌握实例操作系统的实现方法。

## 熟悉和理解Linux编程环境

### 目的与要求

熟悉和理解Linux编程环境。

### 内容

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

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

### 过程与结果分析

#### 2.3.1 文件拷贝

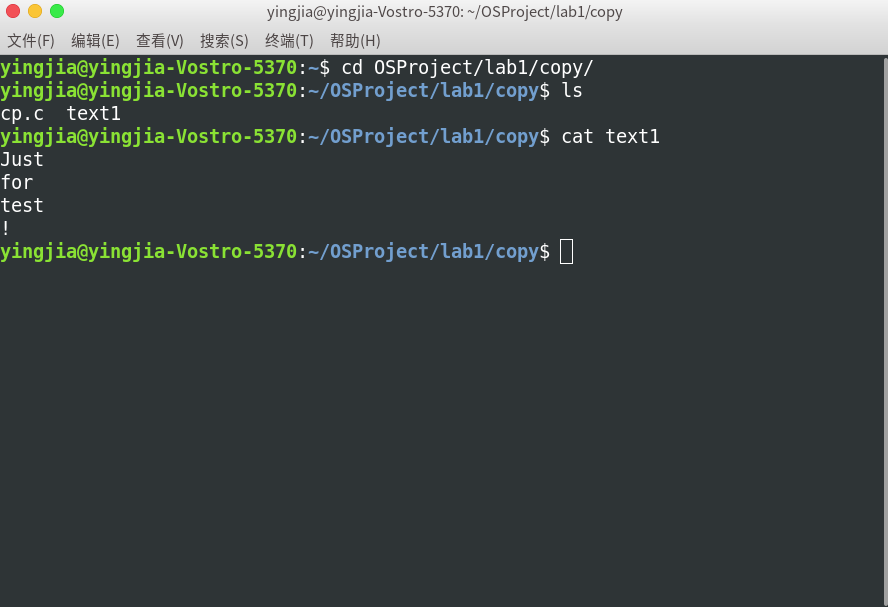


图2.1 text1内容

首先进入程序目录下，输入命令cat text1查看text1内容。

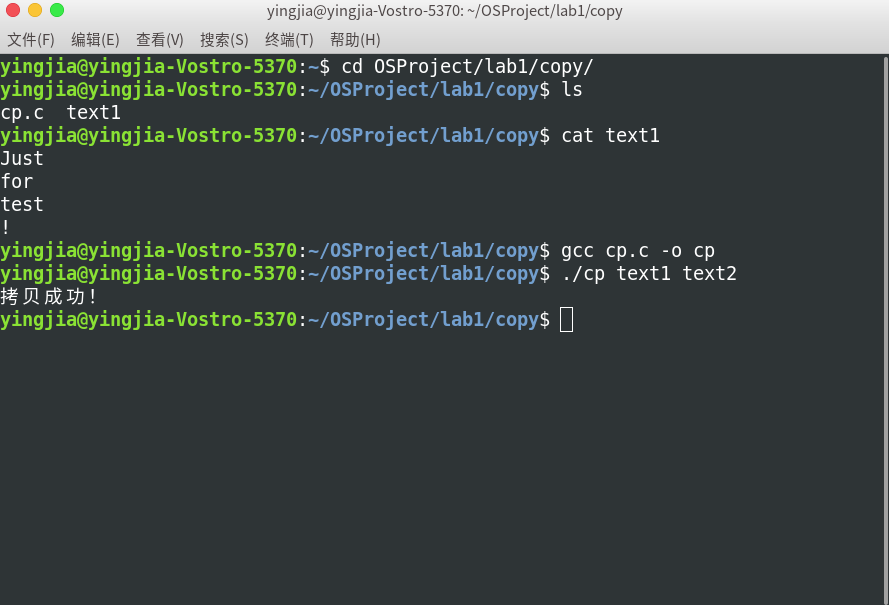


图2.2 编译并运行

之后用gcc编译c文件，并运行可执行文件，将text1中的内容拷贝到text2，若text2存在且有内容，将先清空内容再拷贝，若text2不存在，则先创建。



图2.3 text2内容

观察可知，拷贝后text2的内容与text1完全一致。

#### 2.3.2 分窗口显示并发进程的运行

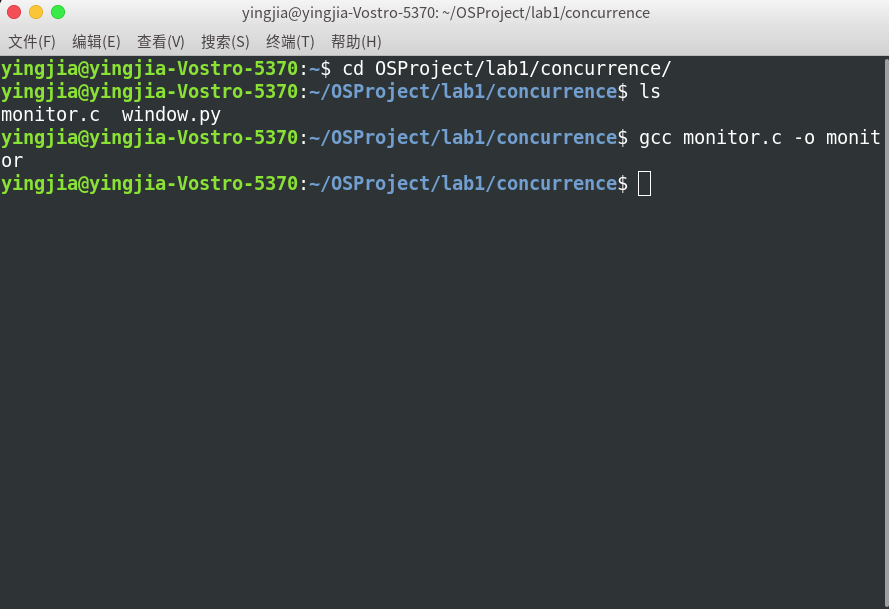


图2.4 编译monitor.c

进入到程序目录下，首先编译monitor.c文件，该文件里面包含了子进程的创建，对于每一个子进程，会通过进程IO函数popen()创建管道，调用系统命令执行window.py文件，而每个子进程由于传递给popen()的command不同，因此将在window.py内部调用不同的函数执行，也即在可视化界面中有不同的显示。

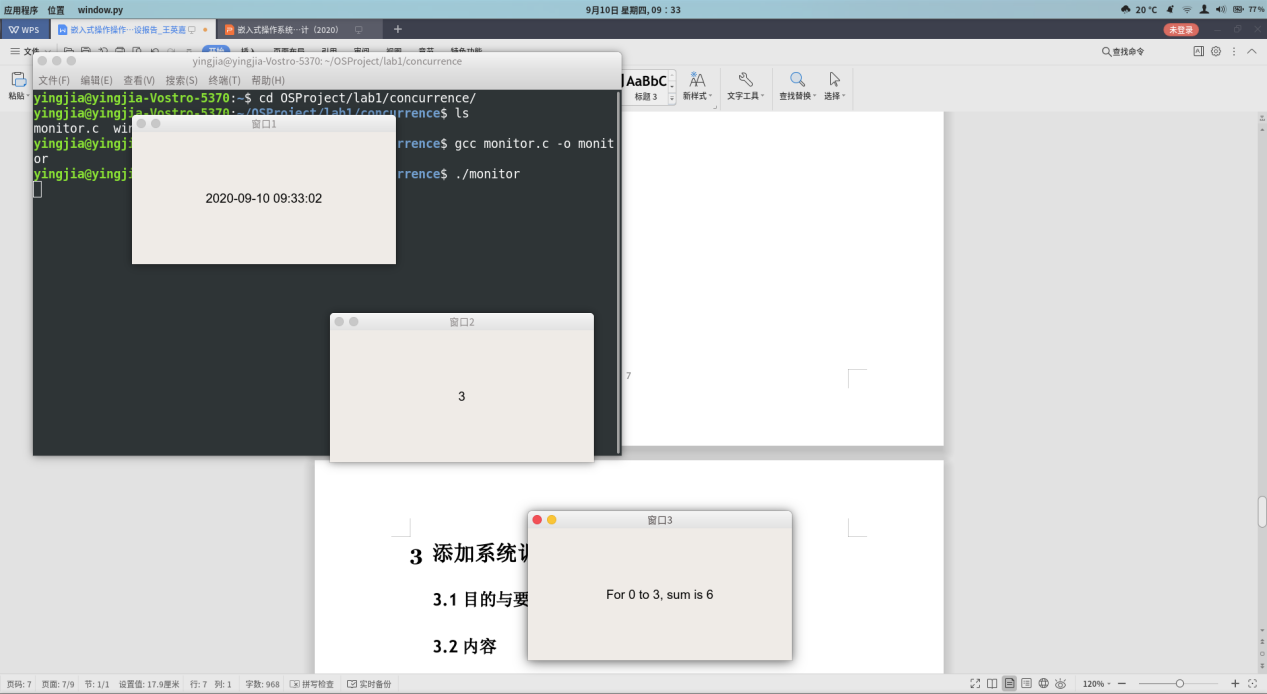


图2.5 运行过程截图

之后运行可执行文件，运行过程截图见2.5，可以看到三个窗口并发运行，每隔1秒钟刷新结果，这里可以让三个窗口一直运行下去，也可以设定一个时间阈值，当到达阈值时窗口关闭，程序自动终止。

#### 2.3.3 实验心得

在文件拷贝实验中，熟练了read和write等系统调用的操作，但要注意异常处理以及考虑text2若不存在或已经存在内容的情况。

在分窗口显示并发进程运行实验中，构建可视化界面使用了python中的PyQt5，来学习C和python的混合编程，这里使用popen函数从命令行执行python程序返回结果，也可以用Python.h头文件中的函数直接嵌入python语句执行。

值得注意的是，在窗口中，每次更新文本内容后必须要加上一行QApplication.processEvents()。对于执行密集型耗时的程序，例如本实验中的刷新页面，在文本显示处理后通过该语句可以实时刷新，防止页面锁死。

## 添加系统调用

### 目的与要求

掌握添加系统调用的方法。

### 内容

1、采用编译内核的方法，添加一个新的系统调用，实现文件拷贝功能

2、编写一个应用程序，测试新加的系统调用。

### 过程与结果分析

由于原本的linux双系统中内存容量不足以支持内核源码编译，且编译低版本linux内核出现了较多问题，最终在windows上利用vmware重新搭建了一台linux虚拟机完成了本实验。

Linux虚拟机分配了4个核心用来加速编译，具体实验测试环境如下：

表3.1实验测试环境

|  |  |
| --- | --- |
| 配 置 | 版本 |
| CPU | Intel® Core™ i7-8550U CPU @ 1.80GHz × 4 |
| OS  内核版本 | Ubuntu 16.04 LTS  linux-4.15.0-117-generic |

由表3.1可知，内核版本为linux-4.15.0-117-generic，故本实验中选择了linux-4.15.2，其清华镜像源下载地址如下：

<https://mirror.tuna.tsinghua.edu.cn/kernel/v4.x/linux-4.15.2.tar.gz>

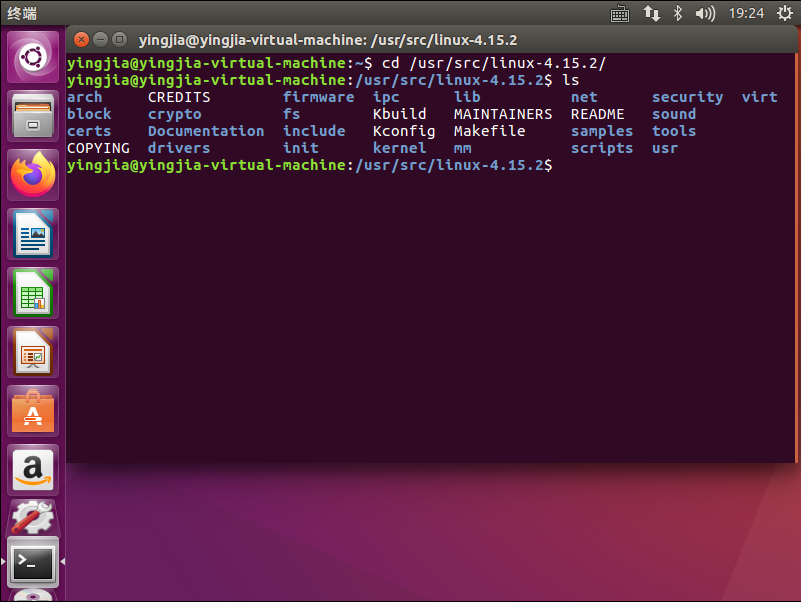


图3.1 内核源码目录

解压后内核源码目录文件见图3.1。

#### 3.3.1 添加源代码



图3.2 添加源代码

在linux-4.15.2中，系统调用服务例程定义位置为./kernel/sys.c，由图3.2可知，在文件末尾添加了一个新的程序代码sys\_mycopy，实现文件拷贝功能，函数内部与实验二中实现的拷贝函数内部大致相同。

但需要注意两点：其一，如open、write等函数应替换为与之对应的内核函数sys\_open、sys\_write等；其二，为了避免内存保护检查错误，要暂时将访问限制值设置为内核的内存访问范围，然后再修改为原来的值，对应:

mm\_segment\_t old\_fs = get\_fs();

set\_fs(KERNEL\_DS);

和

set\_fs(old\_fs);

三行代码。

#### 3.3.2连接新的系统调用

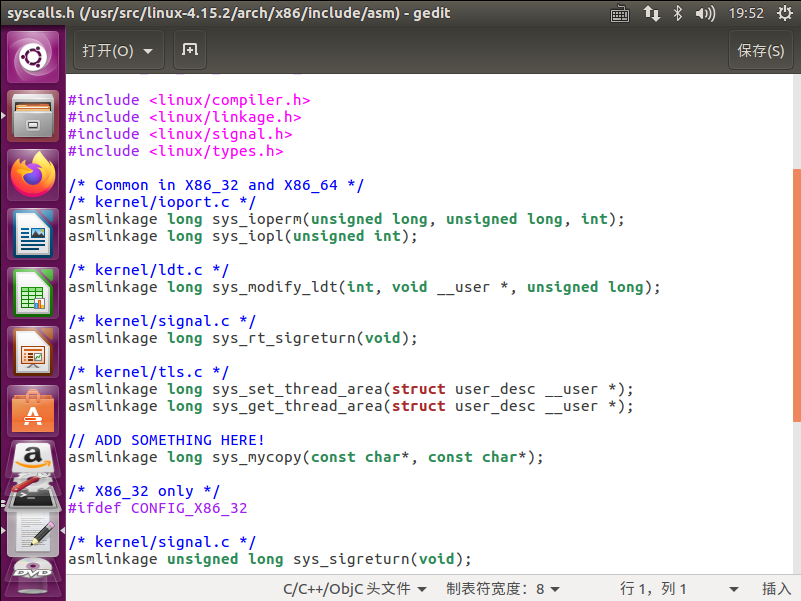


图3.3 连接新的系统调用

在linux-4.15.2中，系统调用函数声明的位为./arch/x86/include/asm/ syscalls.h，由图3.3可知，在文件中添加了一行：

asmlinkage long sys\_mycopy(const char\*, const char\*);

可以让内核的其余部分知道该系统调用的存在。

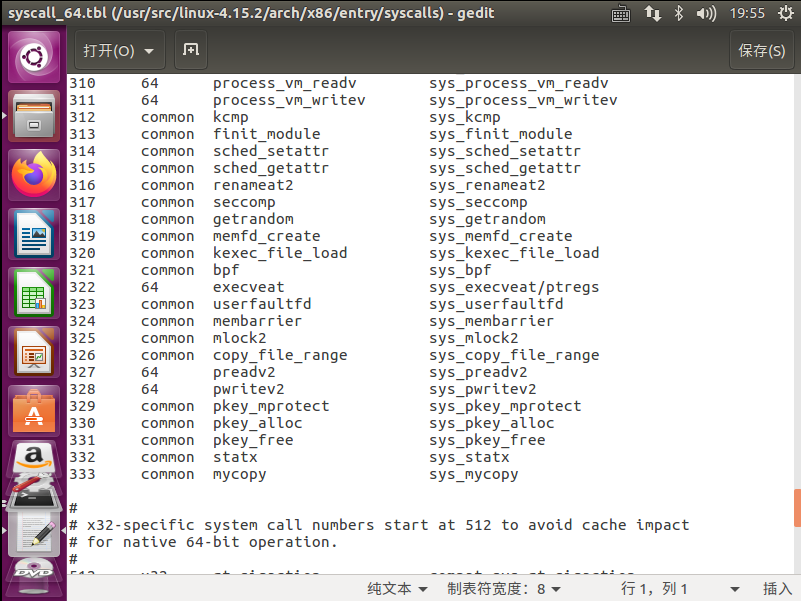


图3.4 分配系统调用号和系统调用名

系统调用表的位置为./arch/x86/entry/syscalls/syscall\_64.tbl，在该表中添加一行系统调用号为333的系统调用函数sys\_mycopy，这样syscall函数可以通过参数333检查系统调用号，查找系统调用表即可找到对应的内核函数。

#### 3.3.3 重建linux内核



图3.5 清除旧配置文件

这里首先需要清空以前的配置信息，避免编译内核时生成的文件不一致。

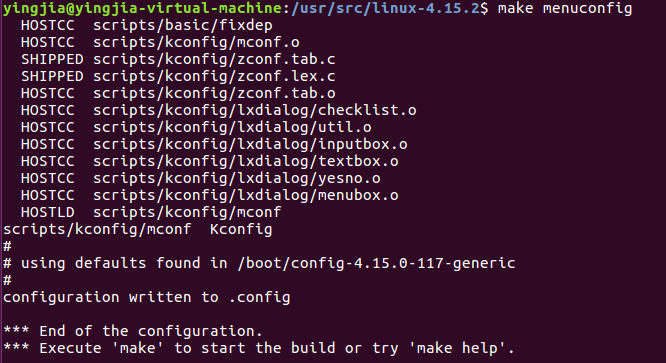


图3.6 生成内核配置文件

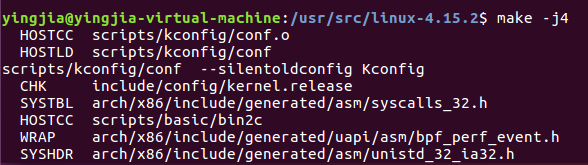


图3.7 编译内核模块

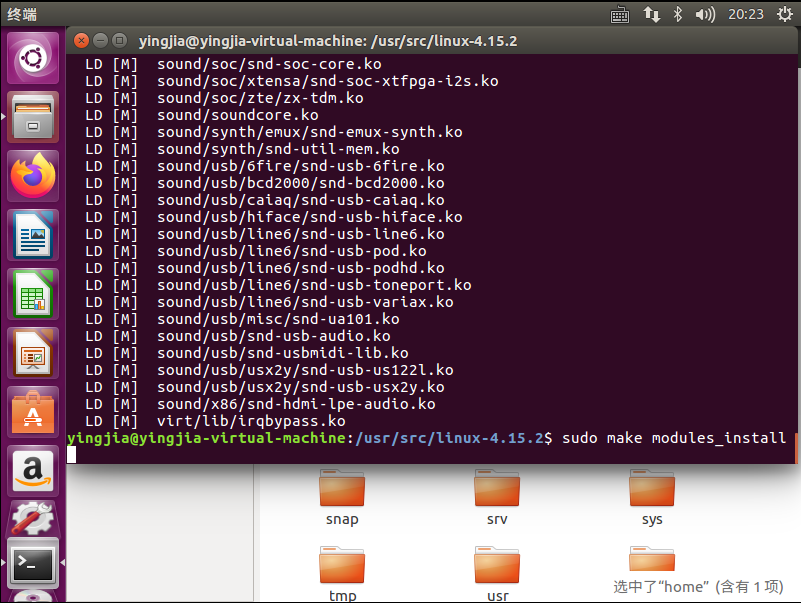


图3.8 编译内核模块结束

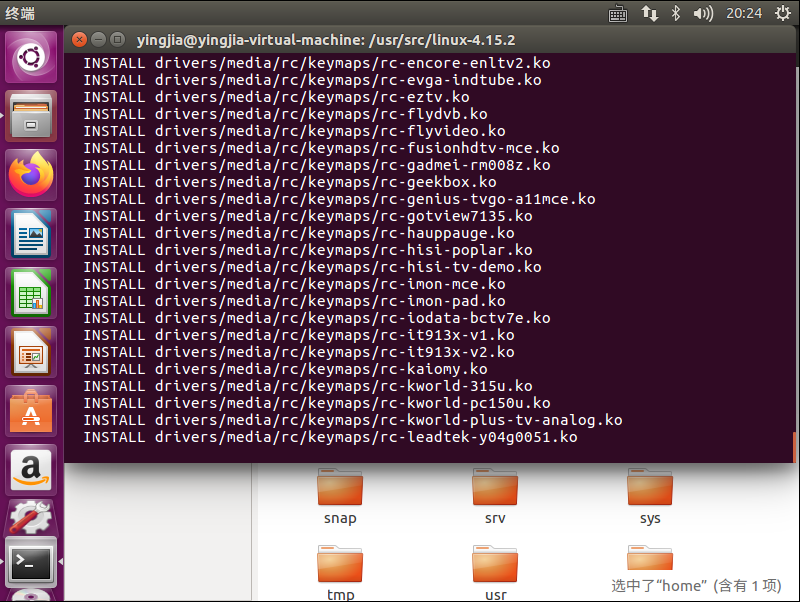


图3.9 生成安装模块

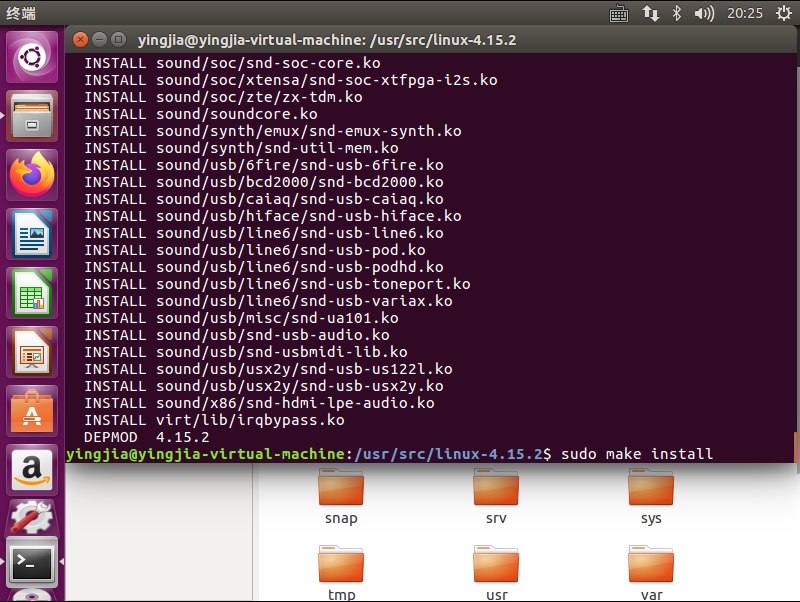


图3.10 生成安装模块结束

紧接着通过make menufig生成内核配置文件，这里采用默认配置，不作修改；通过make -j4编译内核模块，-j4表示4个线程同时进行编译，对应着分配虚拟机内核时分配了四个核心，编译过程大概有半小时左右。

编译结束后，通过make modules\_install生成安装模块，最后通过make install安装。

#### 3.3.4 结果测试

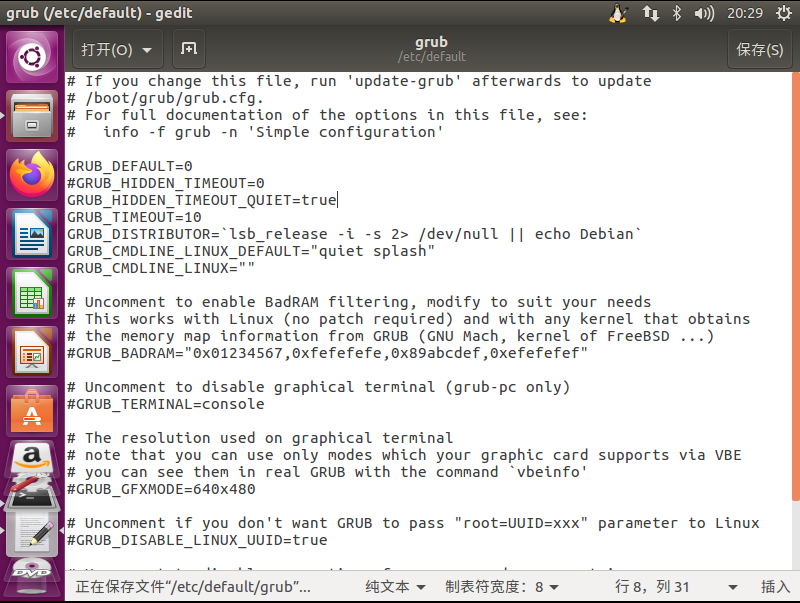


图3.11 修改grub文件

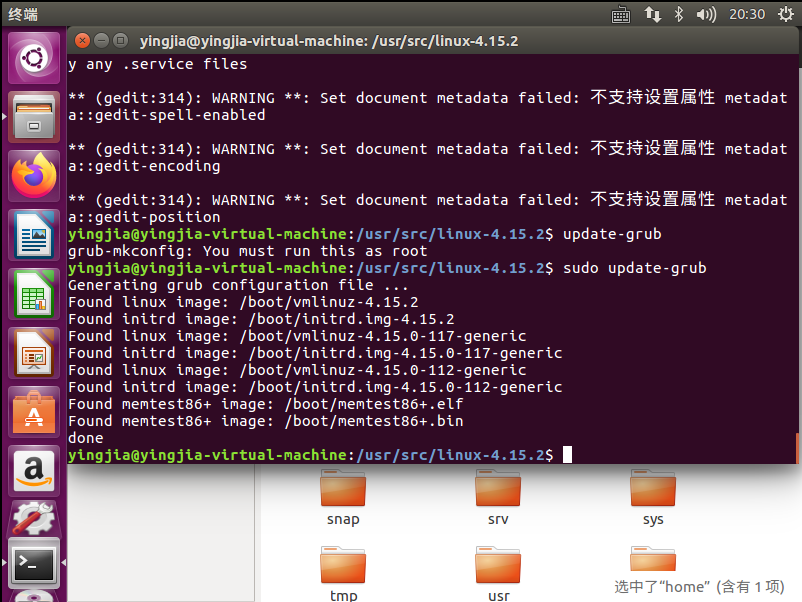


图3.12 更新grub信息

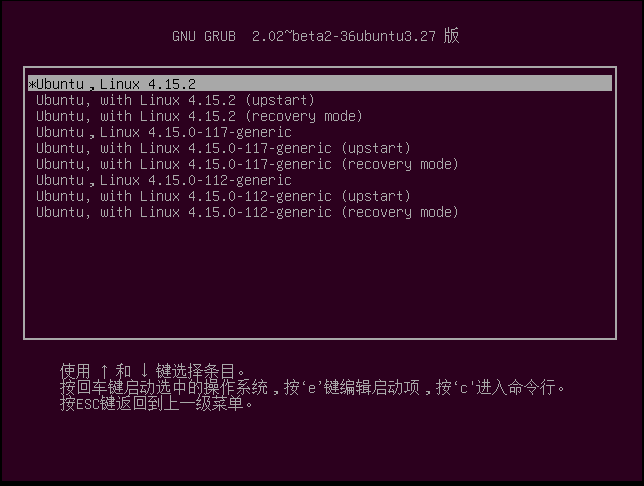


图3.13 选择新内核

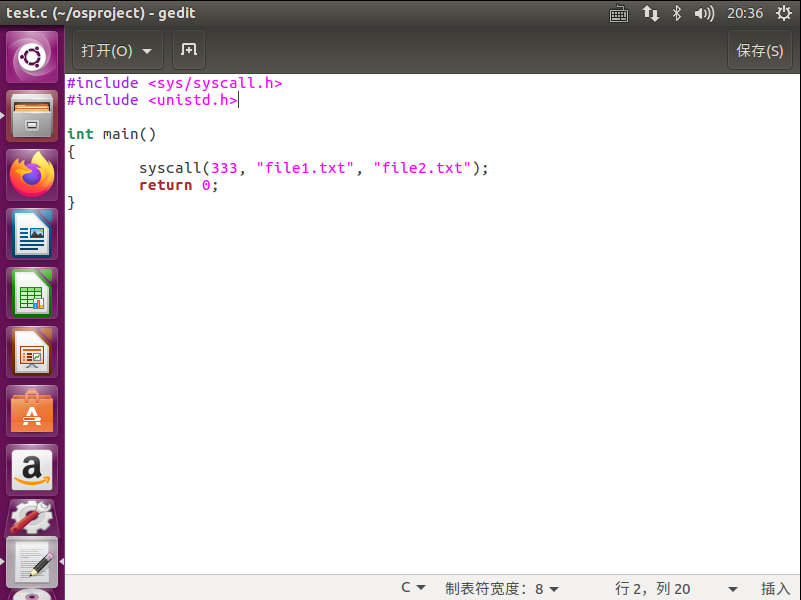


图3.14 编写测试程序

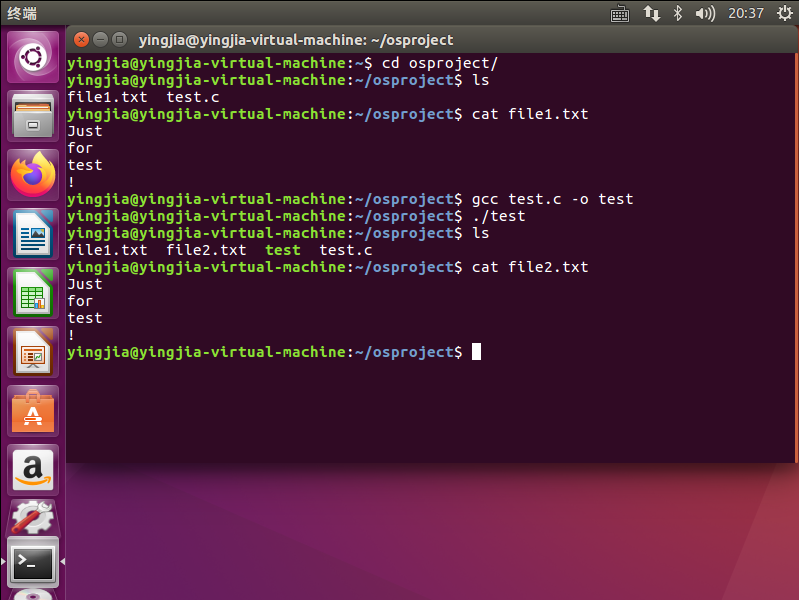


图3.15 测试功能

测试环节首先打开/etc/default/grub文件，注释掉GRUB\_HIDDEN\_TIMEOUT = 0， 之后更新grub文件信息，重启选择新内核，编写测试程序测试即可。

由图3.15可知，测试功能正常，实验成功。

#### 3.3.5 实验心得

由于之前一直在双系统下练习linux，本次实验得以在虚拟机平台下进行实验，二者相比而言：虚拟机利用虚拟化技术实现了多系统互通，方便调试，切换效率高，安装卸载都比较方便；而双系统独立性高，运行更加流畅，不用担心虚拟机下大量占用系统资源的问题，用户体验也更好一些。

实验中编译内核使用了命令make，这个命令包含了指导书中的make bzlmage编译内核映像和make modules编译内核模块，相当于make all，同时，为了提高编译速度，通过参数-j引入多线程，例如本实验中为虚拟机分配了4个核，编译命令为make -j4。要注意的是，安装的时候make install没有安装modules只安装了bzimage，因此要单独执行命令make modules\_install。

#### 3.3.6 参考资料

1、linux采用编译内核的方法增加系统功能调用

url: <https://blog.51cto.com/rangercyh/521200>

2、给linux系统增加一个系统调用

url:[https://blog.csdn.net/qq\_41175905/article/details/80529245#commentBox](https://blog.csdn.net/qq_41175905/article/details/80529245" \l "commentBox)

## 添加设备驱动

### 目的与要求

掌握添加设备驱动程序的方法。

### 内容

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

2、编写一个应用程序，测试添加的驱动程序。

### 过程与结果分析

#### 4.3.1 实验过程



图4.1 设备驱动程序mydev.c



图4.2 Makefile

首先编写设备驱动程序dev.c和Makefile文件。

设备驱动程序dev.c的编写采用模块方法编写函数，分别实现模块初始化、模块卸载、打开、关闭、读、写功能。

Makefile文件以课设指导书中的通用Makefile为准，并将模块的文件组成和生成的模块文件名都改为dev.o即可。

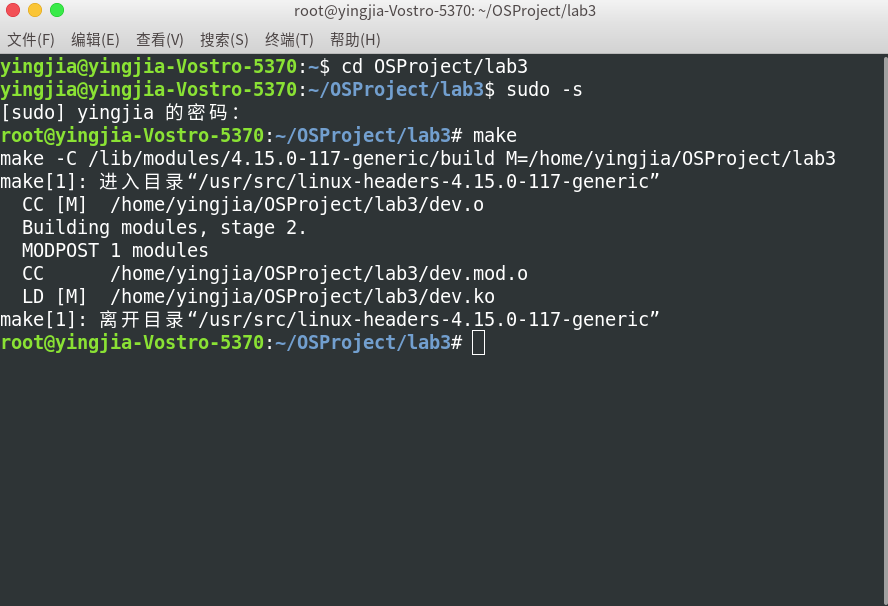


图4.3 设备驱动模块编译



图4.4 dmesg输出结果



图4.5 生成设备文件

编写完成后通过make指令进行设备驱动模块的编译，查看dmesg输出可以看到新增的设备的主设备号为239，最后通过mknod指令生成设备文件。

#### 4.3.2 结果测试



图4.6 测试文件dev\_test.c

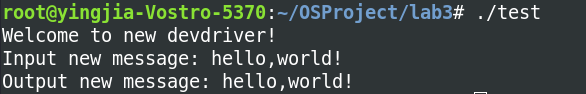


图4.7 测试效果

为了检验驱动程序安装及其功能的正常，编写测试文件dev\_test.c如图4.6，首先read出设备中保存的字符串，紧接着输入一段新字符串并write进入设备，再read出刚刚输入的字符串。

由图4.7可见，设备驱动程序安装成功，功能正常。

#### 4.3.3 实验心得

下面对本实验中出现到的一些问题进行总结：

其一，在dev.c中模块化编写init\_module和cleanup\_module时一定要保证设备引用计数的准确，即register\_chrdev和unregister\_chrdev的对应，否则会导致异常出错。

其二，在加载设备驱动模块时，出现了错误，详见4.3.3参考博客2中的问题，博客中提到：出现此问题是因为Ubuntu Kernel使用 EFI\_SECURE\_BOOT\_SIG\_ENFORCE 内核配置，这样会阻止内核载入第三方模块。因此只要关闭SECURE\_BOOT就可以顺利加载了。

#### 4.3.3 参考资料

1、linux字符设备驱动程序

url: <https://www.cnblogs.com/downey-blog/p/10500828.html>

2、关于insmod时出现ERROR：Required Key not available的解决

url: <https://blog.csdn.net/m0_38066161/article/details/81812816>

## 理解和分析/proc文件

### 目的与要求

使用GTK/QT实现一个系统监控器。

### 内容

1、了解/proc文件的特点和使用方法

2、监控系统状态，显示系统部件的使用情况

3、用图形界面监控系统状态，包括CPU和内存利用率、所有进程信息等(可自己补充、添加其他功能)

### 过程与结果分析

#### 5.3.1 实验过程

/proc文件系统是进程文件系统和内核文件系统组成的复合体，将内核数据对象化为文件形式进行存取，其包括系统信息、进程信息等等。/proc文件系统是伪文件系统，只存在于内存当中，但不占用外存空间。

本实验中实现的System Monitor系统管理器主要包括OverView总览、Process进程管理、Files文件管理、Resources资源显示四大部分，编程语言为python，可视化编写使用了PyQt5模块，以下逐一详述。

由图5.1，OverView部分主要包括Hostname(主机名)、System Version(系统版本)、CPU(CPU型号)和CPU MHz(CPU当前实际使用主频)，与/proc文件获取关系如下：

|  |  |
| --- | --- |
| Hostname | /proc/sys/kernel/hostname |
| System Version | /proc/sys/kernel/ostype |
| /proc/sys/kernel/osrelease |
| CPU | /proc/cpuinfo |
| CPU MHz | /proc/cpuinfo |

表5.1 功能与/proc文件获取关系

在选项卡下方，还设计了包括时间、一键关机按钮、CPU Usage、Memory Usage和Swap Usage的显示进度条，后三者将在Resources中详述。

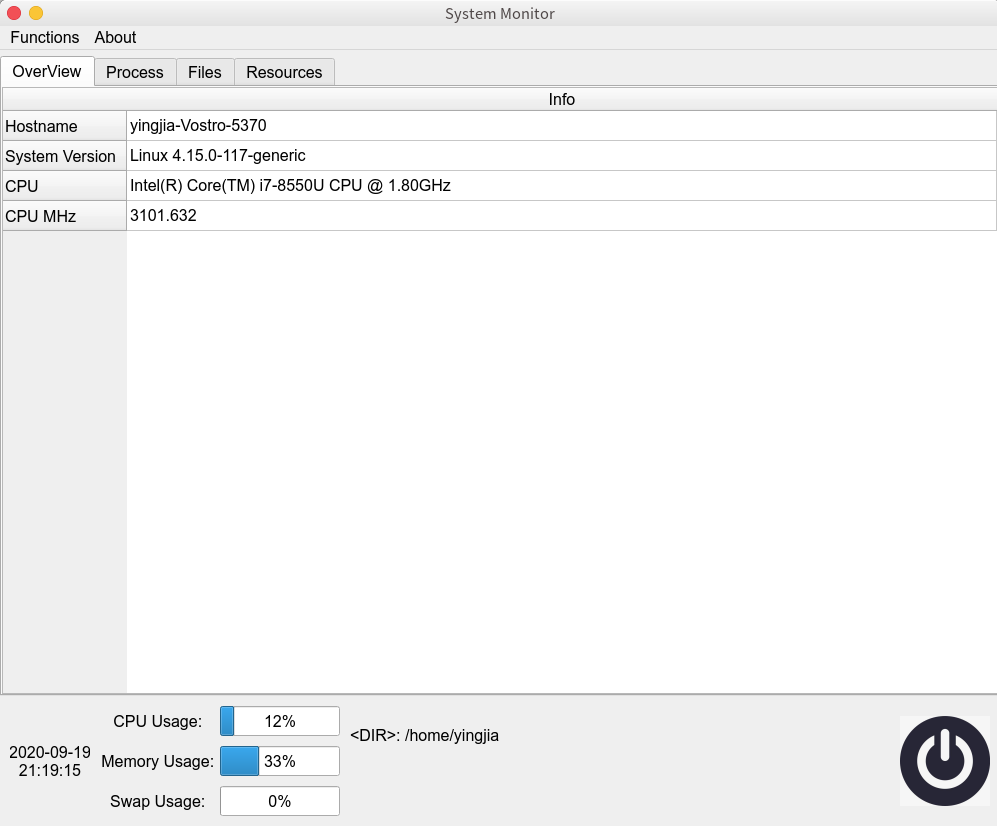


图5.1 系统管理器OverView界面

由图5.2，Process部分主要包括了系统当前全部进程的进程号PID，进程名NAME，进程状态STAT和正在使用的物理内存MEMORY，对于不同的PID，与/proc文件获取关系如下，其中$为具体PID：

|  |  |
| --- | --- |
| PID | /proc下筛选 |
| NAME | /proc/$/stat |
| STAT |
| MEMORY | /proc/$/statm |

表5.2 功能与proc文件获取关系

在Process部分还实现了查找进程功能(Find)和杀死进程功能(Kill)。

其中查找进程功能直接调用Qt中的findItems()函数，查找成功后将当前行选中，且自动操纵光标跳转到包含当前行的页面。

杀死进程功能在查找成功后调用kill -9命令杀死进程。

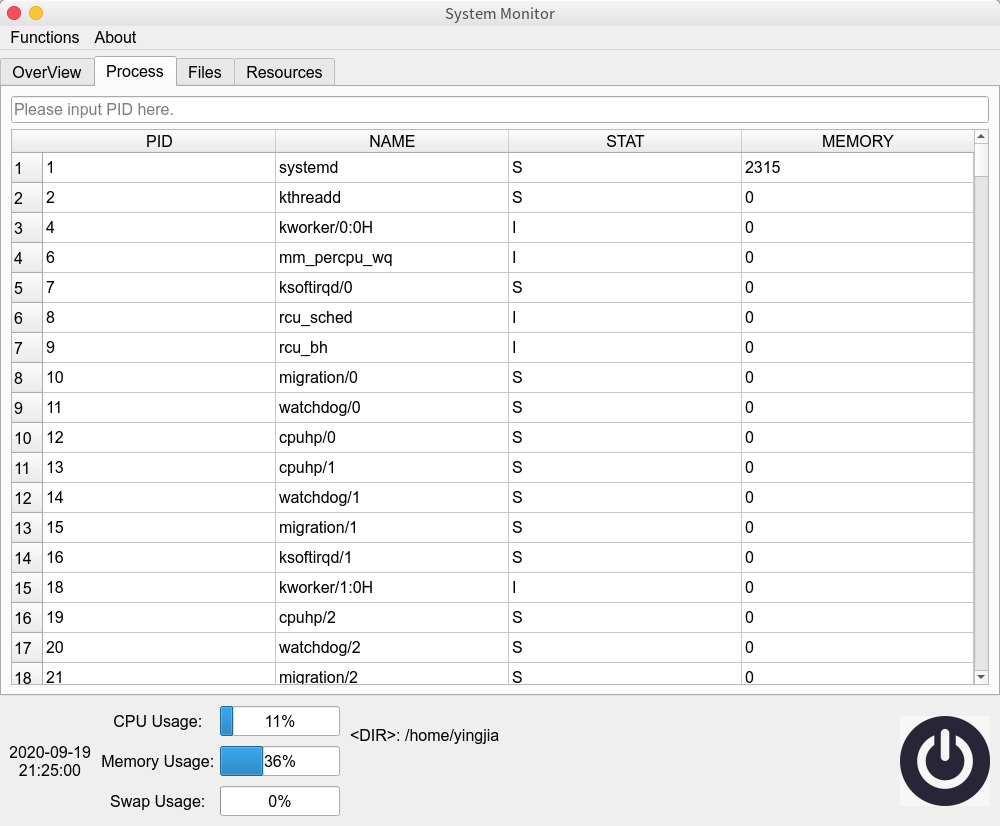


图5.2 系统管理器Process界面

由图5.3，Files部分包括了文件名name、文件大小size和是否为目录is\_dir三个属性列，这一部分没有用到/proc文件系统，设计的目的是为了方便用户在System Monitor中管理文件。

对于Files系统，这里实现了查找文件功能(Find)、计算该层文件占用总内存功能(Total Memory)、双击打开文件或进入下层目录等功能，在文件的不同层查找中，下方的DIR显示条将随时动态展示目录的变化。

其中查找文件功能直接调用Qt中的findItems()函数即可，查找成功将当前行选中，且自动操纵光标跳转到包含当前行的页面。

计算该层文件占用总内存功能开启新进程调用du -sh命令在当前目录的父目录中获取文件节点大小。

双击打开文件或进入下层目录功能首先判断是否为目录，若是目录则创建进程调用du -sh命令更新，若不是目录则调用xdg-open命令打开文件。

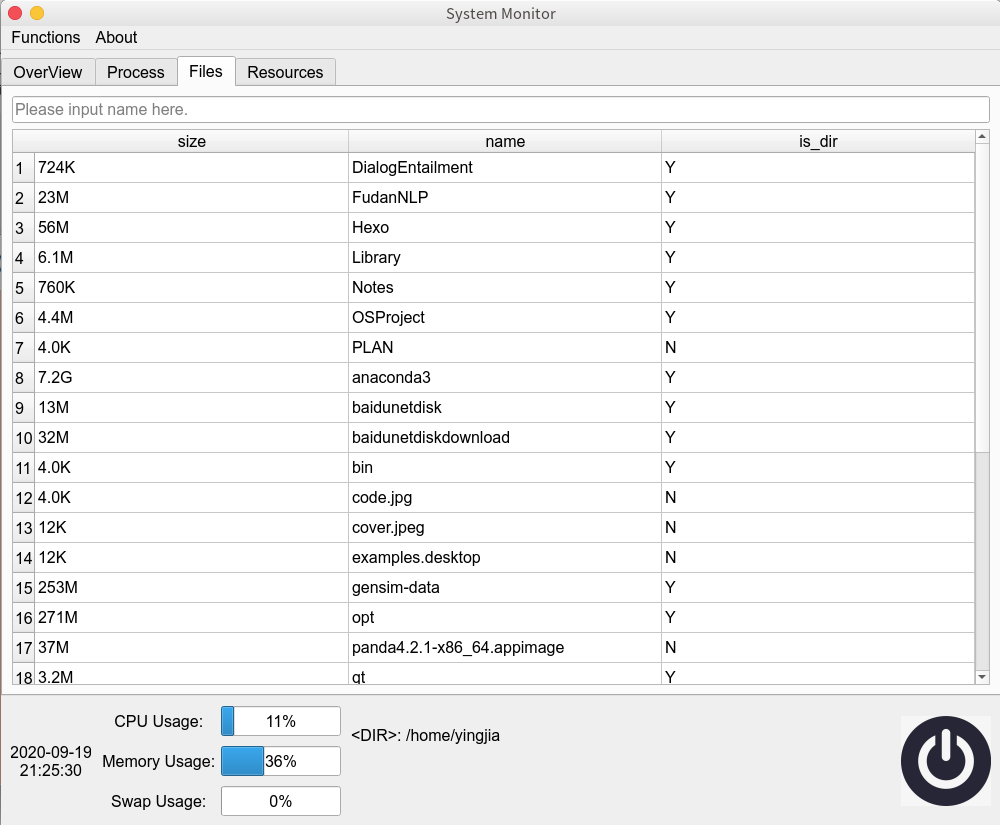


图5.3 系统管理器Files界面

由图5.4,Resources部分将CPU Usage、Memory Usage和Swap Usage三个百分比属性以饼状图和动态折线图的方式实时展现，且支持改变刷新周期或停止刷新，另外在切换到其他子系统时，也可以通过下面的百分比条体现。

CPU Usage通过os.popen()开启子进程调用vmstat -w -w命令获取结果。

Memory Usage和Swap Usage则获取了/proc/meminfo中的MemTotal、MemAvailable、SwapTotal、SwapFree四个属性经过计算获得，公式为：

Memory Usage = 1 - MemAvailable/MemTotal

Swap Usage = 1 - SwapFree/SwapTotal

这里注意到Memory Usage的计算没有用到MemFree，这是因为MemFree表示的是系统尚未使用的内存，而MemAvailable表示的是系统可用的内存，系统中有些内存虽然已被使用但是是可以被回收的，因此MemFree不能表示全部可用的内存。更深一步来说，MemFree说的是系统层面，MemAvailable说的是应用程序层面。

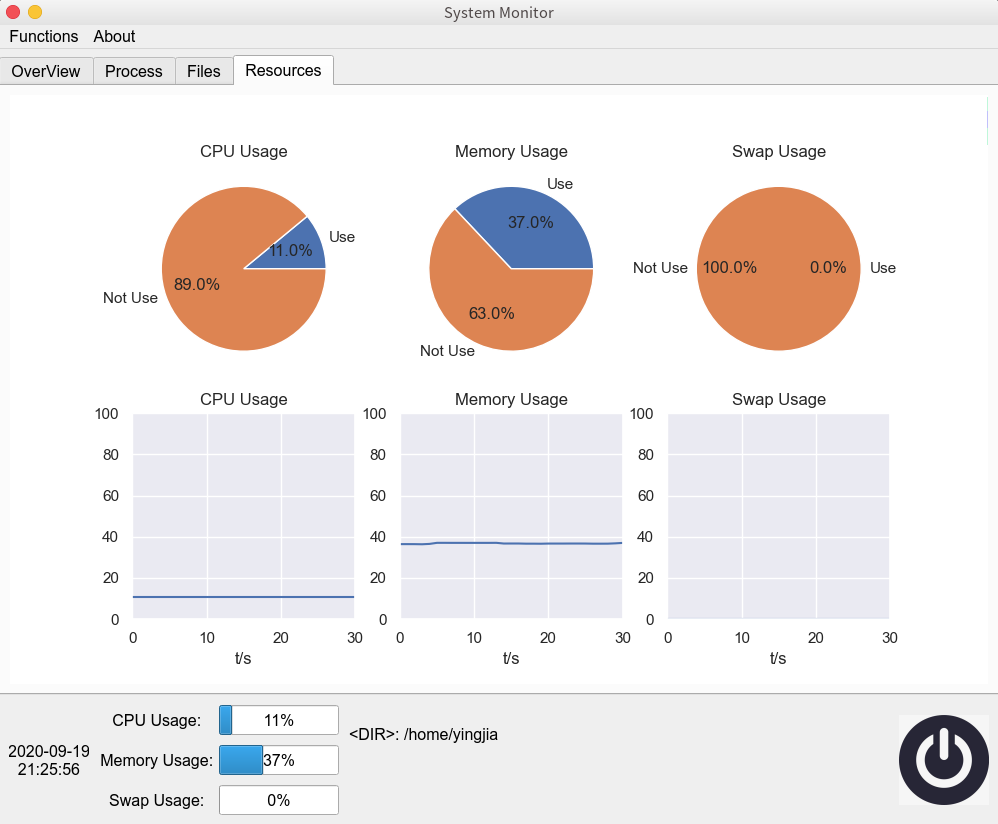


图5.4 系统管理器Resources界面

本实验的核心在于在一个定时周期T内，System Monitor需要对上述的大部分信息进行实时刷新，而如果全部在主进程内进行，频繁的读写与实时交互将共同运行在一个进程内，会造成严重的卡顿现象。

为了解决这一问题，主进程将完全用于交互，且另创建多个线程，分别用于读和写，具体来说：

首先，创建信号灯，对每一个共享资源x，如时间、进程等，要设置两个信号灯，分别为x\_empty、令初值为1，x\_full、令初值为0。

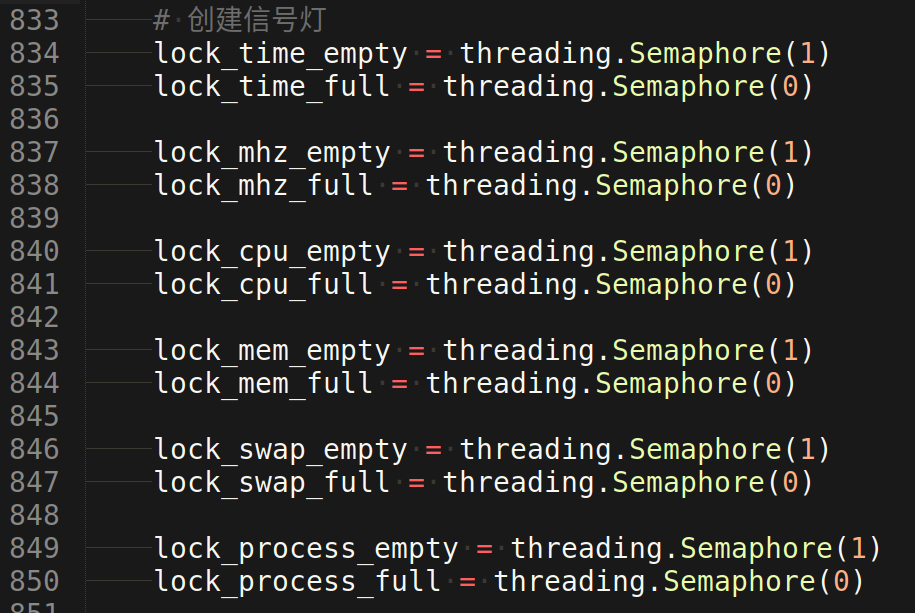


图5.5 创建信号灯代码

对于读线程，使用QTimer进行计时，每隔一个刷新周期，将触发信号，先对x\_empty使用P操作，然后读取信息，再对x\_full使用V操作。

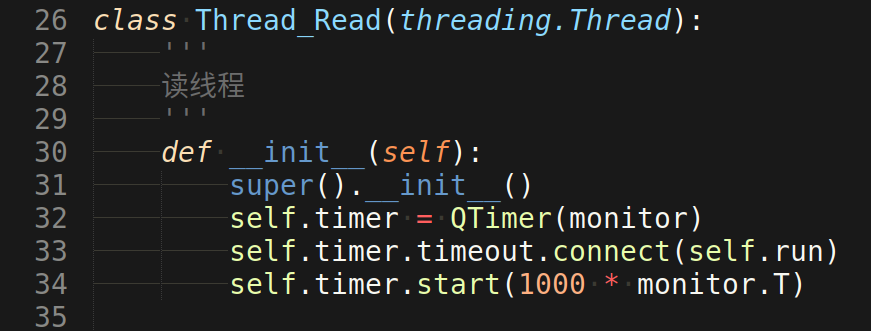


图5.6 创建定时器QTimer代码

对于写线程，不额外计时，而通过信号灯进行判断何时可以开始写数据，写线程运行后，将一直尝试对x\_full进行P操作，可知只有读线程读取完成时，即对x\_full使用完V操作时，写线程才可以开始写。

通过这种方式，写线程不必等待读线程全部读取完再写，因为这样会造成较大的时间延迟，并且读写顺序得到了保证，从而减少了卡顿，使系统管理器运行得更加流畅。

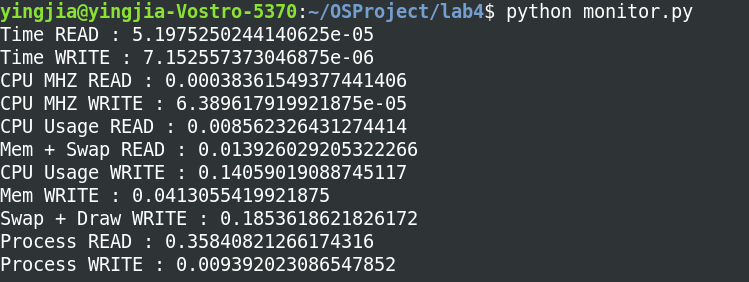


图5.7 读写各部分耗时输出

图5.7是对于双线程，即读写各一个线程时各部分内容读写分别耗时的输出，单位为秒，可知读写过程耗时显著最长的是Process部分的读过程。

对此进行的进一步优化方案是：将Process的读过程单独创建一个新线程Thread\_Read\_Process，与另一个读线程Thread\_Read和写线程Thread\_Write同步运行，将Process的读过程分离开来，这样进一步减少了延迟。

#### 5.3.2 结果测试与分析

系统管理器的核心功能在于其能够一个固定的刷新周期T控制下流畅地更新管理器中各部分信息的值，且支持同时与用户交互。

以下分别对各部分功能进行测试：

测试OverView界面和下部的组件，图5.8和5.9是运行时前后两个时刻的截屏，可知，时间、Memory Usage和CPU MHz三个值发生了变化。

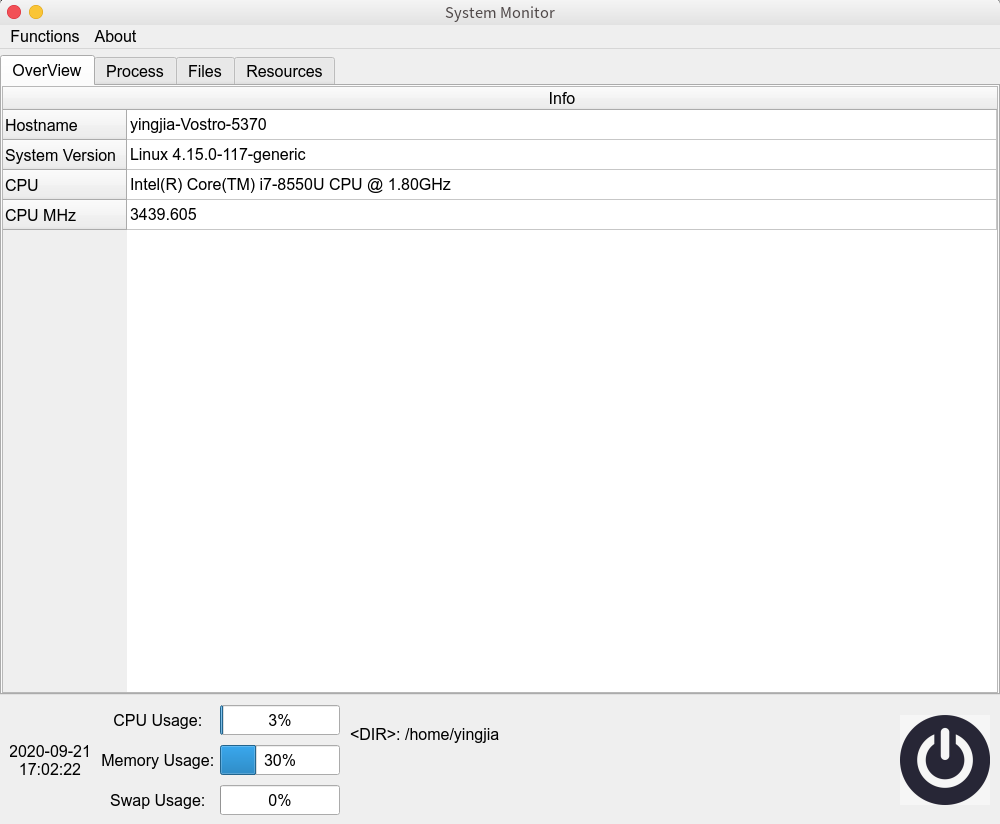


图5.8 OverView界面运行截屏



图5.9 OverView界面运行截屏

测试Process界面，在图5.10中进程信息的基础上，打开chrome浏览器，进程信息更新效果见图5.11，可以看到多个和chrome浏览器有关的进程开启运行，且下方Memory Usage也从31%增长到34%。

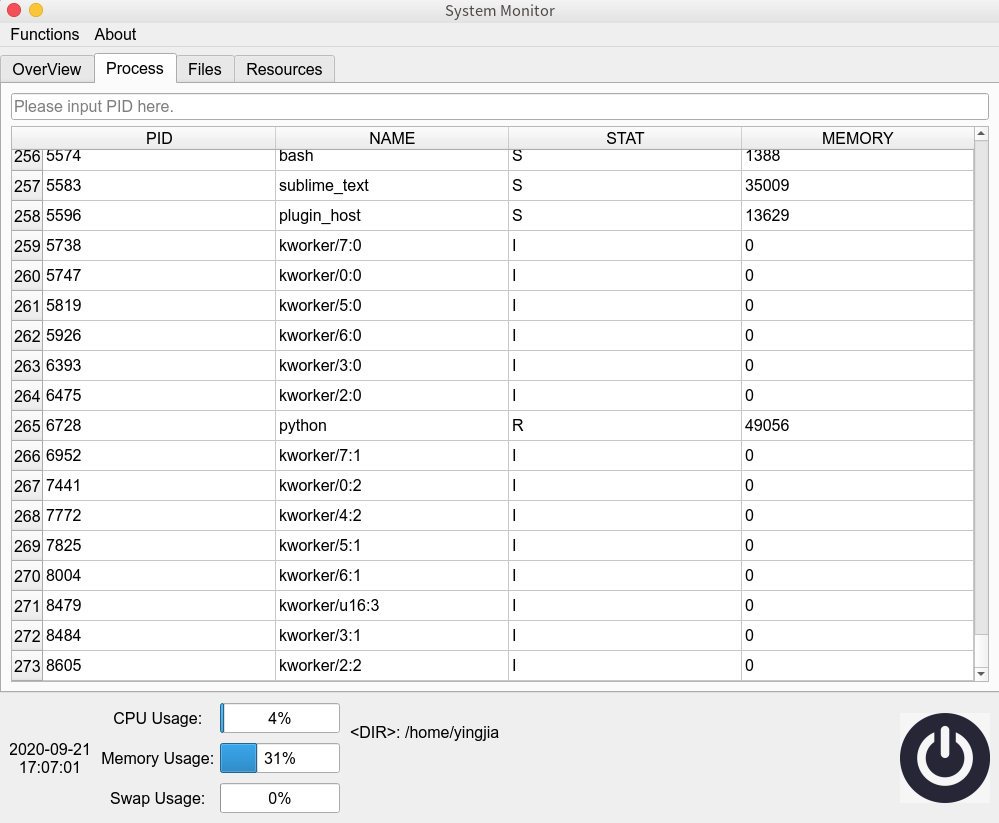


图5.10 Process界面在浏览器打开前

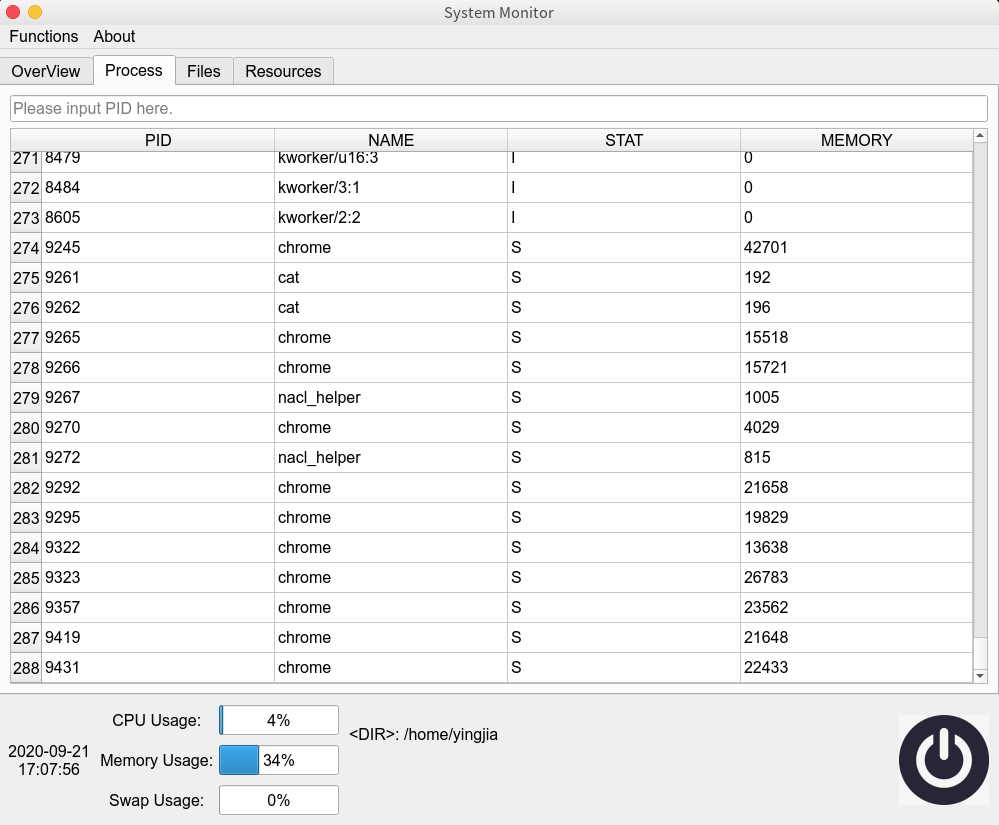


图5.11 Process界面在浏览器打开后

此时在上方输入框内输入11135，测试查找进程功能，由图5.12可知，查找成功。在上方输入框输入11118，测试杀死进程功能，由图5.13可知，杀死成功，且11118对应的进程是chrome里有关番茄工作法组件的部分。

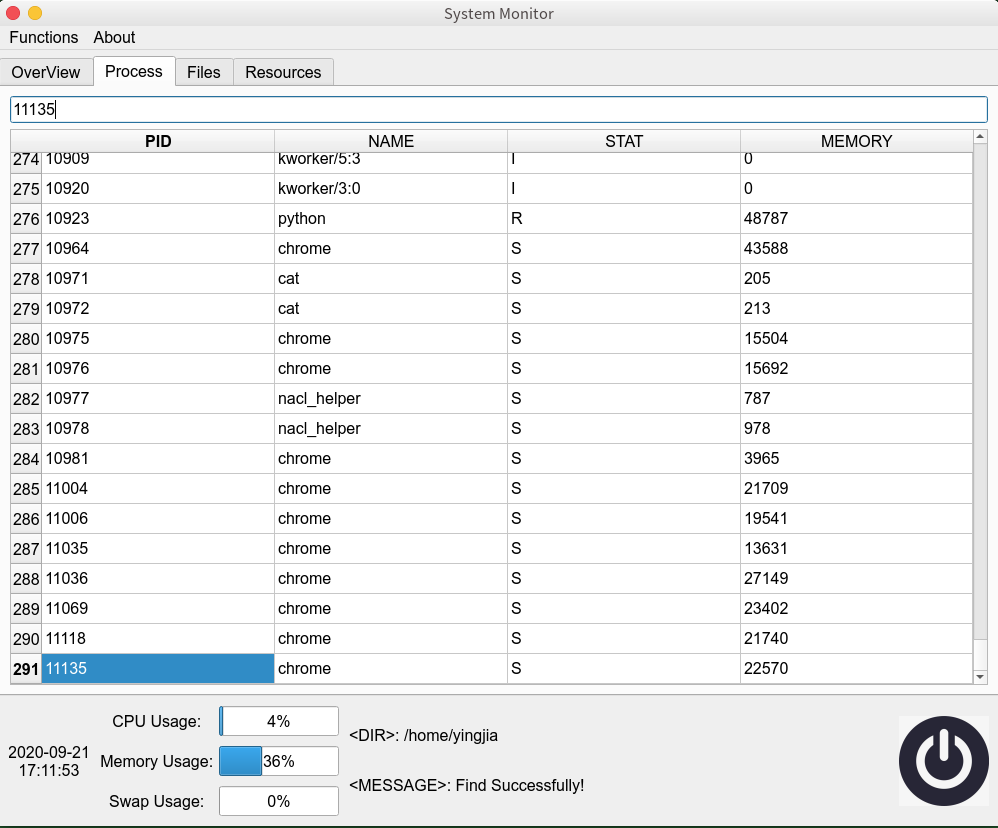


图5.12 Process界面测试查找进程功能

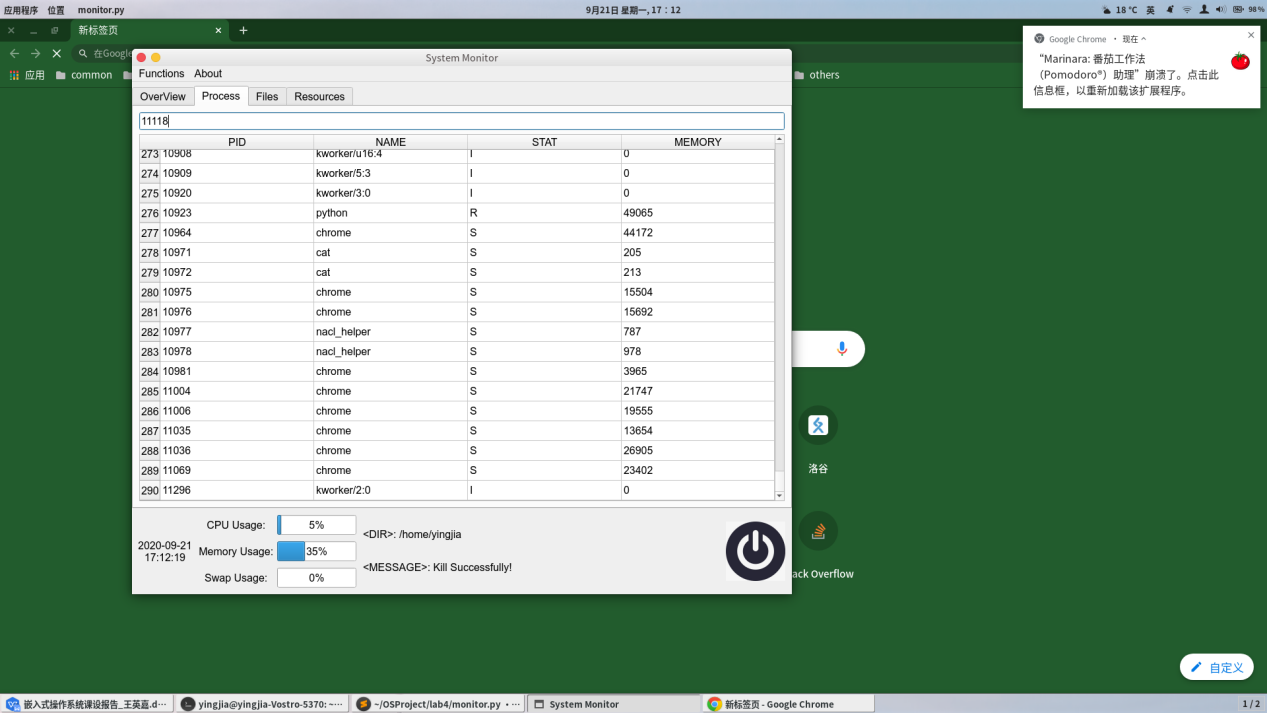


图5.13 Process界面测试杀死进程功能

测试Files界面，由图5.14展示的是/home/yingjia下的文件及文件夹清单，此时在输入框输入OSProject，查找此文件或文件夹，由图5.15可知，查找成功。

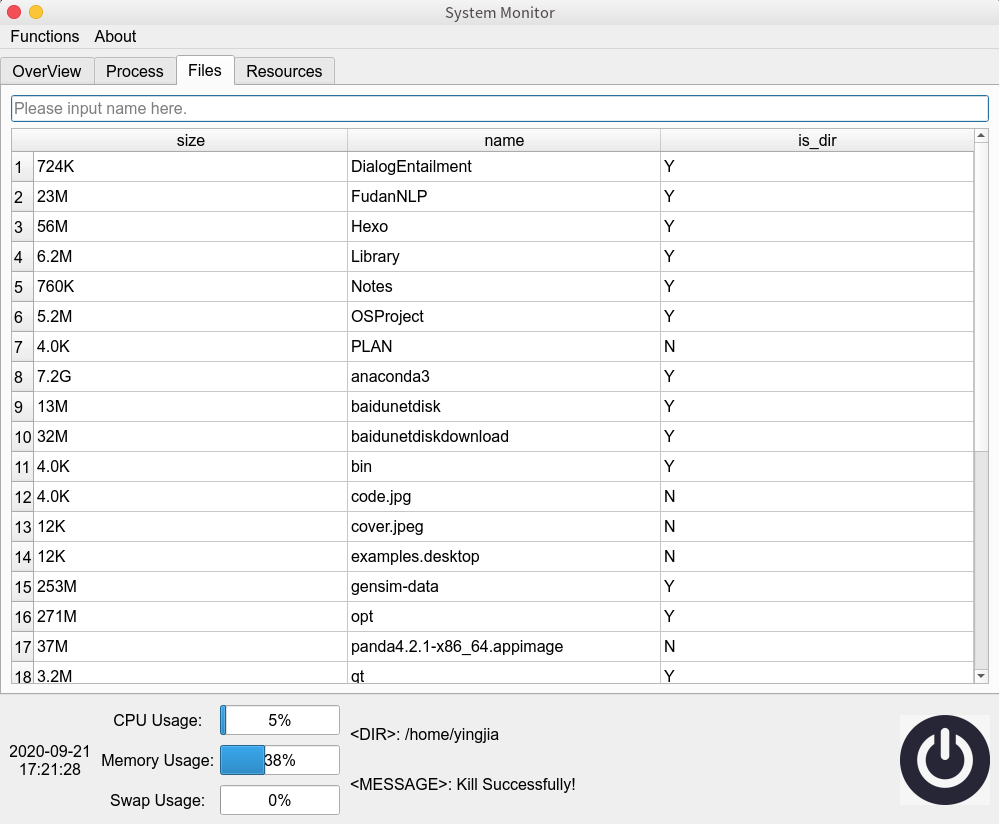


图5.14 Files界面

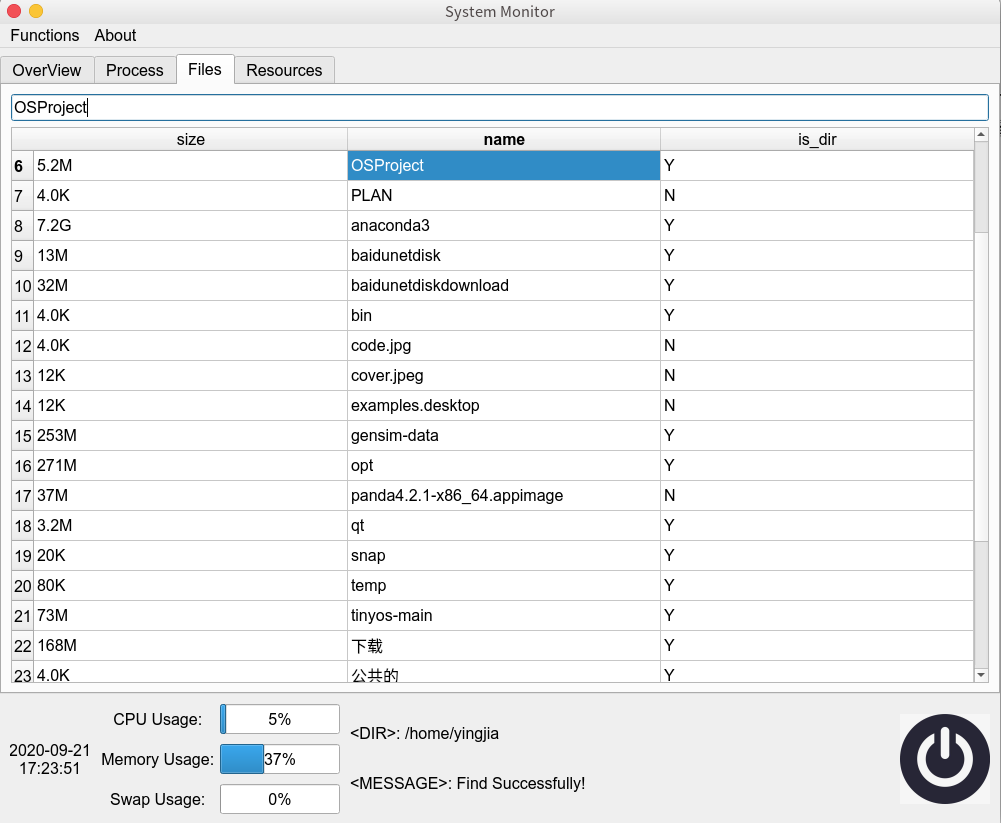


图5.15 Files界面测试查找文件（夹）功能

然后双击OSProject，进入目录下，由图5.16可知，目录发生了更新，此时测试计算当前层占用总内存功能，由图5.17下方提示信息与图5.15对比，功能正常。



图5.16 Files界面测试进入目录

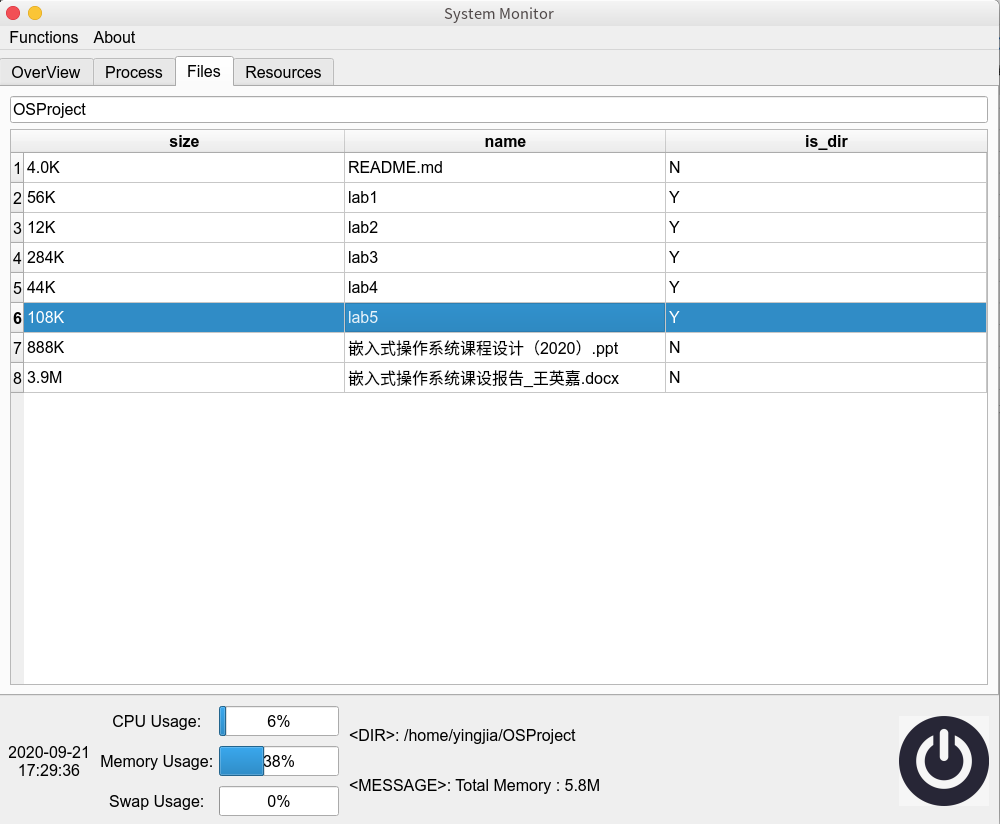


图5.17 Files界面测试计算当前层占用总内存功能

测试返回上一层功能，输入Ctrl+U，由图5.18可知，功能正常；测试打开文件功能，双击图片cover.jpeg，图片打开成功，功能正常。

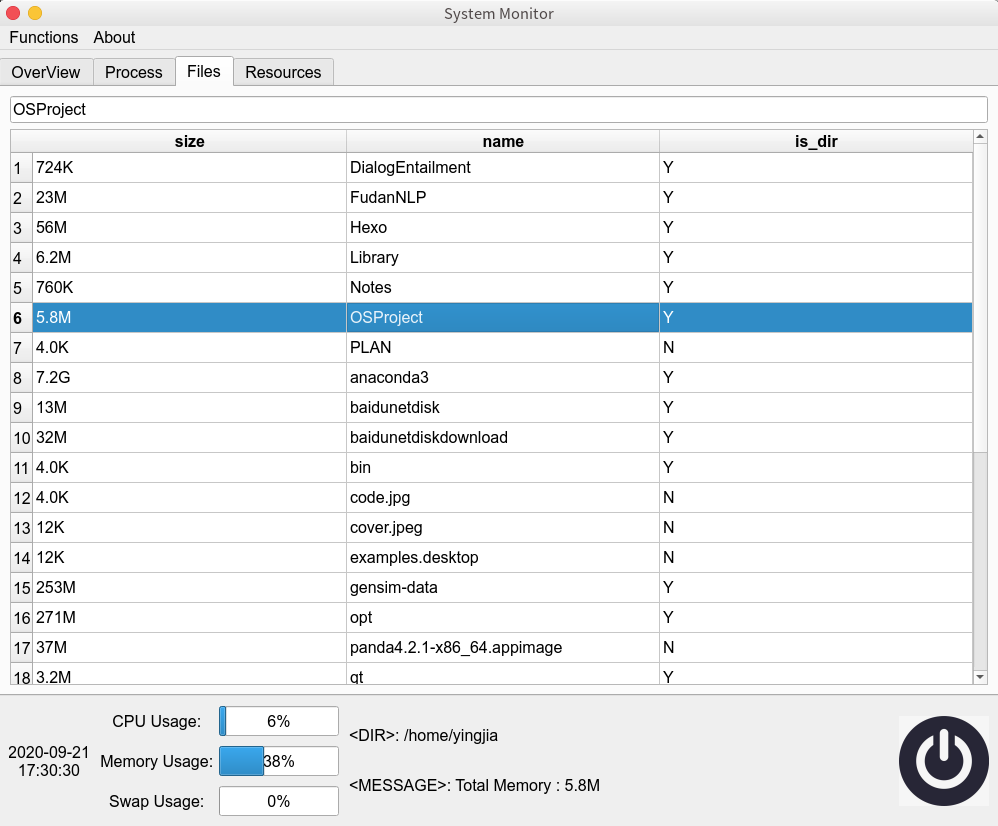


图5.18 Files界面测试返回上一层功能



图5.19 Files界面测试打开文件

测试Resources界面，图5.20为运行某时刻Resources的状态，可以看到三个百分比通过饼状图和折线图进行展示，这时我们尝试迅速打开chrome中多个页面，观察效果如图5.21，可以看到Memory Usage有了小幅增长，折线图里更是清晰地反映出了变化。

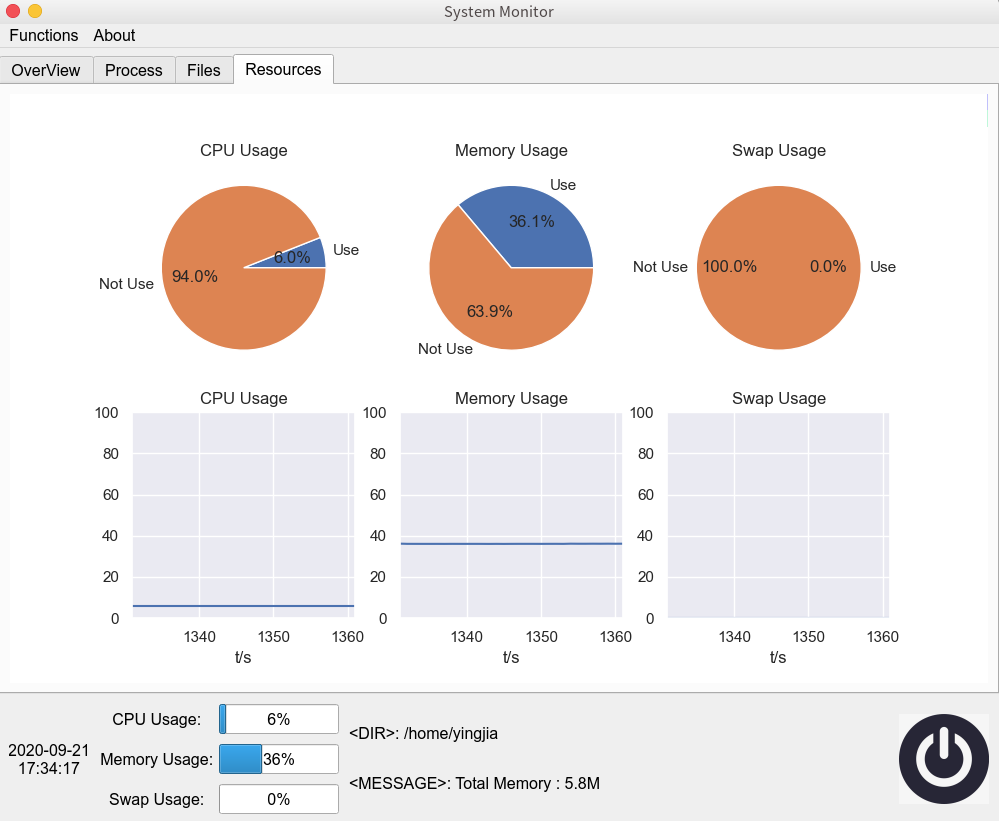


图5.20 Resources界面浏览器打开前

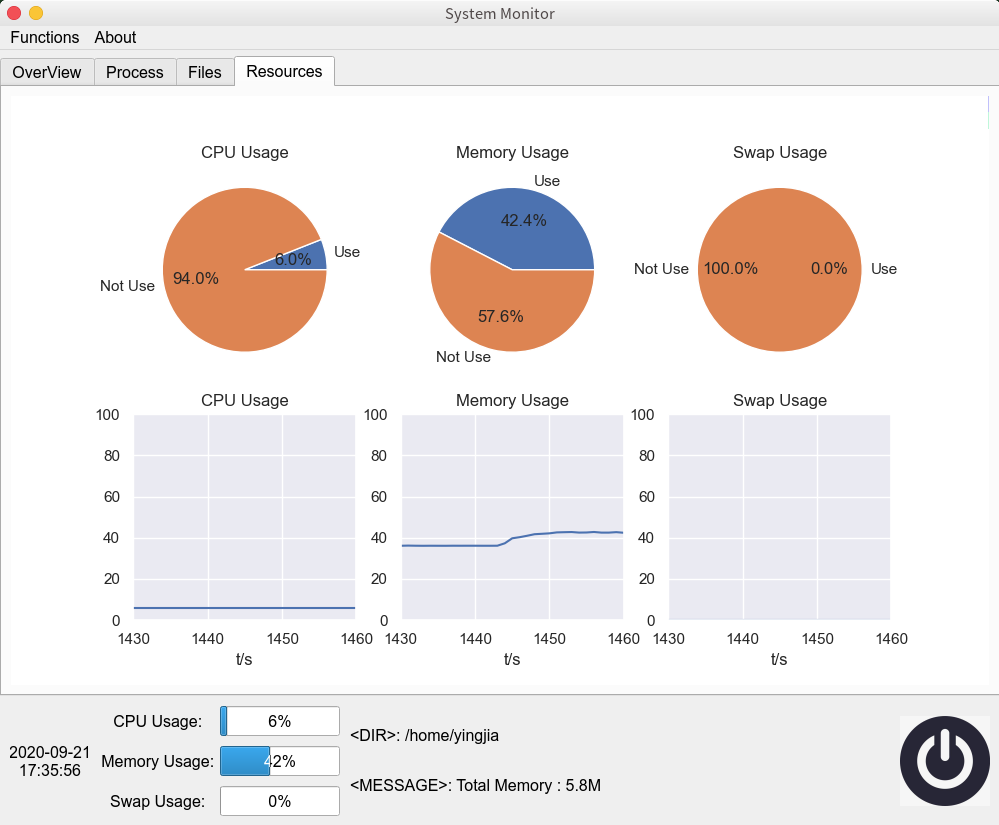


图5.21 Resources界面浏览器打开后

最后，更新刷新周期(Change Refresh Cycle)和停止刷新功能(Stop Refresh)经过测试也正常，由于以图片形式不方便控制间隔进行展示，故从略。

#### 5.3.3 实验心得

Process的读取耗时较长(大约在0.4s左右)，现阶段已经给出另创建一个线程读取Process的优化方式，由于C的速度显著快于python，进一步可以用C重写这部分代码，动态链接编译成so文件，用ctypes.CDLL模块进行读取获取结果，以此进一步缩短延迟时间。

一开始关于定时是使用了time.sleep()函数，但是该函数计时过程中会阻塞后续代码段的运行，从而导致写上的数据不是精确的目标时间的数据。为此改用了QTimer计时器，该计时器的计时是非常准确的，可以理解为触发的函数以新进程的方式执行，不会影响其下一轮计时。

关于Resource中的饼状图和动态折线图的绘制，一开始使用了PyQt5的QChart模块，用QChartView封装显示，但出现了一些更新异常的问题，并且每次修改值都需要重新生成QChart和QChartView，极为不便，后续换用了matplotlib中的FigureCanvas作图(继承自QWidget)，FigureCanvas中调用matplotlib.pyplot中的模块正常绘制即可，但仍然需要更新。

#### 5.3.4 参考资料

1、Qt官方文档

Url: <https://doc.qt.io/>

2、PyQt5图形界面编程

Url: <https://zhuanlan.zhihu.com/p/48373518>

1. 在PyQt5嵌入matplotlib画的图

Url: <https://blog.csdn.net/qq_39105012/article/details/88584124>

以及一些关于proc文件系统介绍的博客。

## 模拟文件系统设计

### 目的与要求

设计并实现一个模拟的文件系统。

### 内容

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

### 过程与结果分析

#### 6.3.1 总体设计

文件系统是操作系统中负责管理和存取文件信息的软件机构，它由负责文件操作和管理的程序模块、所需的数据结构（如目录表、文件控制块、存储分配表）以及访问文件的一组操作所组成，其主要功能是在逻辑文件和物理文件之中建立映射关系，实现二者的相互转换。

本实验中实现的文件系统是以一个大文件作为磁盘来进行模拟的。在系统中共设计了20万个磁盘块，每个磁盘块最多允许存放512个字符，保存后文件总大小约为102.6MB。

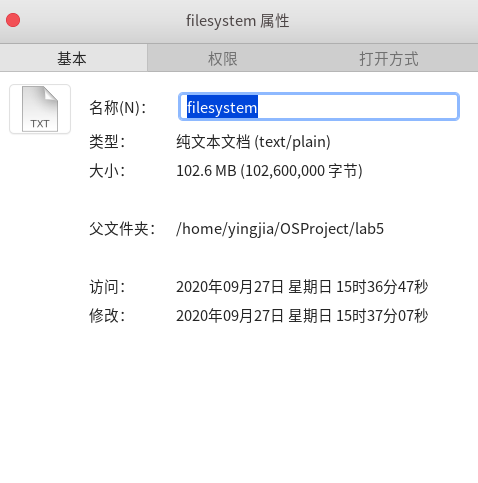


图6.1 文件系统大小

文件的物理结构方面，设计并支持一级间接索引结构，即对于每一个文件实例的内容分配了唯一的一个磁盘块，当文件内容大小小于等于512个字符时，该磁盘块即作为直接索引，保存文件内容信息；当文件内容大小大于512个字符且小于等于65536个字符时，该磁盘块作为文件索引表，由于磁盘块数总计为20万个，用32进制进行编码可以保存为四位字符，因此这个索引表可以保存512/4=128个磁盘块号，文件内容最大容量即为128\*512=65536个字符。

文件存储空间管理方面，全部块保存在一个列表中，但是每一个块的地址都保存在了字典中，因此通过字典可以迅速索引到某个块的位置。空闲块列表也仅仅保存了空闲块的ID，所以分配与回收过程开销很小。

文件与目录类设计方面，对可读写的“文件”和不可读写的“目录”单独进行了封装。如图6.2所示，每一个文件类实例包含的信息包括文件名、文件ID、最后修改时间、创建者ID、文件索引表长度、文件索引表（块）ID、占用最后一个块的长度。

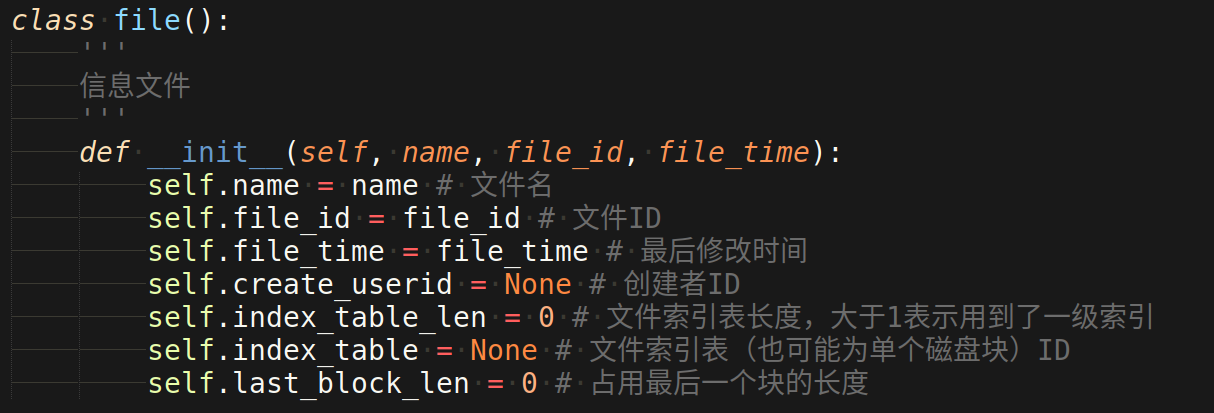


图6.2 文件类

而如图6.3所示，每一个目录类实例只需要包含目录名、目录ID和子节点ID数组。

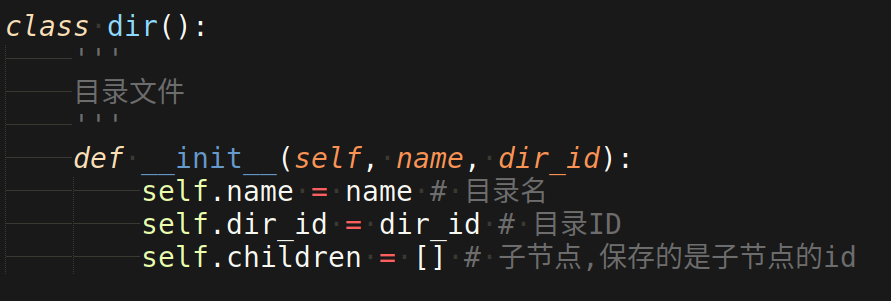


图6.3 目录类

文件目录结构采用多级文件目录设计，构成树型层次结构，以root为根节点进行延伸，但使用了一个字典来保存全部文件（目录）的地址，因此查找某个结点时不需要进行树的遍历查找，只需要从字典中直接获取即可。

值得注意的是，全局文件和目录的ID不允许冲突，即不可能存在一个文件或目录的ID与另一个文件或目录的ID相等，这是因为对于每个文件和目录，还分配了唯一的一个块用来保存文件或目录的全部属性信息，在文件或目录ID和块ID间需要建立一对一的映射关系。但是全局文件或目录名不需要唯一，只需要保证某一个目录下的名唯一即可，这也是使用树型结构的优势之一。

文件用户方面，支持了多用户，用户分为管理员用户admin和普通用户两类，全体用户共享一个文件系统，但对于某一个文件来说，不同的用户可能存在不同的权限。用户可以增加或删除，也可以查看全部用户的账号和密码，但只允许管理员用户admin进行这三种操作。

文件权限方面进行了一定的简化，将权限分类只分为自己(self)和他人(others)，创建时默认自己可读写，他人的权限需要手动指定，创建后可以使用chmod对权限进行修改，但chmod指令只允许当前文件的创建者和管理员用户admin使用，其他用户使用会被拒绝操作。

文件系统存储方面，主要通过写文件的方式进行存储，前4个块保存了系统的一些重要信息，文件及目录均不可用。其中，第一个块存储MEMORY\_MAX和BLOCKS\_MAX，这两个值表示文件系统的块最大容量和最大块数，标志了读取的文件系统是否与程序相兼容。第二个块存储了全部用户的账号密码信息，第三个块存储了全部文件和目录与对应属性结点磁盘块的信息，第四个块存储了全部的文件与对应权限的信息。写文件以字符为单位逐个写入，以换行标志一个磁盘块内存的结束(这里出现了一些问题，将在实验心得中提到），因此行数应为BLOCKS\_MAX行，每一行有MEMORY\_MAX个字符。

#### 6.3.2 功能介绍

文件系统程序流程图如图6.4所示。

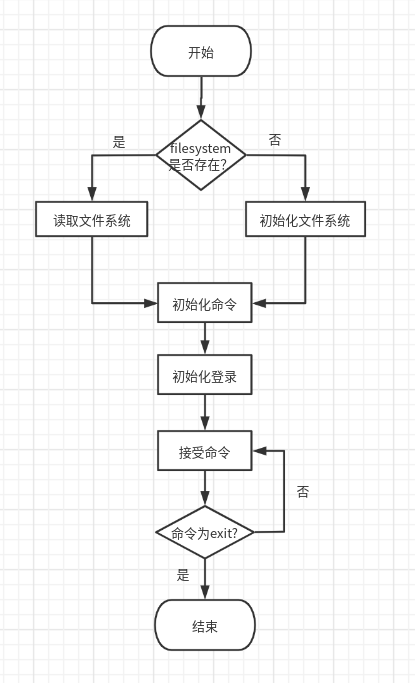


图6.4 程序流程图

在文件系统中，实现了20个功能，其命令与简要的功能说明的对应关系如下：

|  |  |  |
| --- | --- | --- |
| 序号 | 命令 | 功能 |
| 1 | cd [dir] | 切换路径 |
| 2 | ls | 展示当前路径下全部文件（夹）名 |
| 3 | clear | 清空界面 |
| 4 | mkdir [dir\_name] | 创建目录 |
| 5 | rmdir [dir\_name] | 删除目录 |
| 6 | create [file\_name][access] | 创建文件 |
| 7 | rm [file\_name] | 删除文件 |
| 8 | read [file\_name] | 读文件 |
| 9 | write [file\_name] | 写文件 |
| 10 | cp [file\_name] [dir] | 复制文件 |
| 11 | mv [file\_name] [dir] | 移动文件 |
| 12 | chmod [file\_name] [access] [s(elf)/o(thers)] | 更改文件权限(admin/create\_user) |
| 13 | aduser [user\_name] [password] | 增加用户(admin) |
| 14 | rmuser [user\_name] | 删除用户(admin) |
| 15 | lsuser | 展示全部用户(admin) |
| 16 | info | 显示系统信息 |
| 17 | save | 保存文件系统 |
| 18 | logout | 注销 |
| 19 | exit | 退出 |
| 20 | help | 帮助 |

表6.1 命令与功能对应表

#### 6.3.3 结果测试与分析

测试环节主要分为三部分，第一部分进行基本功能测试，第二部分进行异常处理测试，第三部分进行存储与恢复功能测试。

基本功能测试主要对表6.1中的功能进行测试，异常处理测试主要对输入一些典型的不合法的指令后系统如何应答进行测试，存储与恢复功能的测试主要测试系统在保存并退出后再次读取是否完全恢复到之前的状态。

**6.3.3.1 基本功能测试**

下面开始基本功能测试：

打开文件系统，由于是第一次打开，文件系统会初始化磁盘，进而输入账号密码，输入无误后登录成功，页面会清屏。

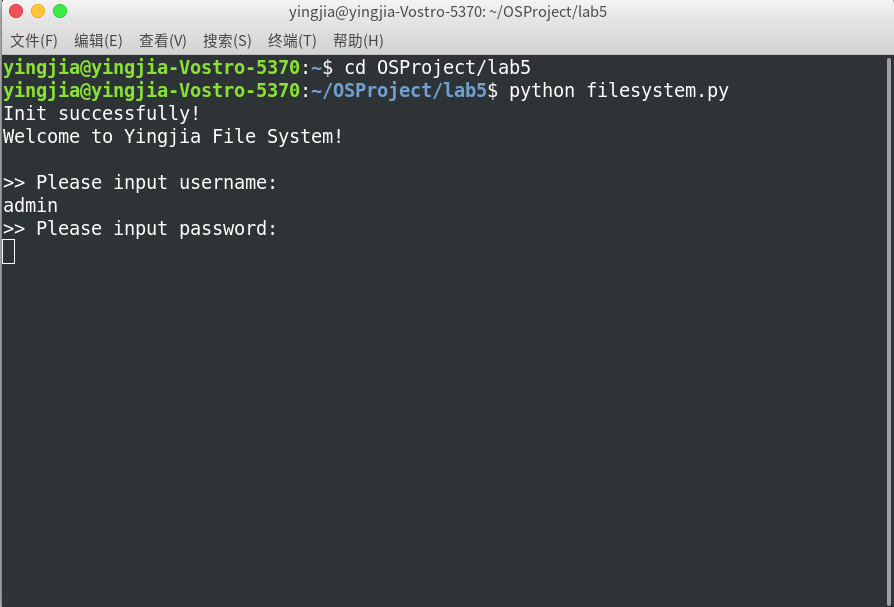


图6.5 基本功能测试截图一

#### 

图6.6 基本功能测试截图二

进入系统后，首先输入info命令，查看系统内存占用情况，由图可知，当前用户数量为2，分别是管理员用户admin和普通用户yingjia；文件和目录数量为1，即根节点root；磁盘块总块数为200000，可用块数为199995，这是因为有4个块不可用，且有1个块保存了根节点root的属性信息。

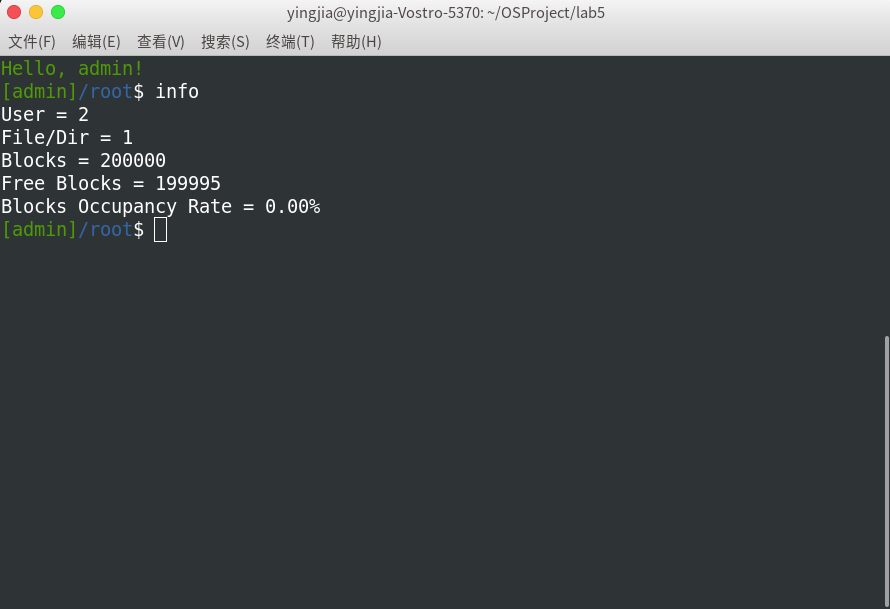


图6.7 基本功能测试截图三

接下来，通过mkdir创建目录d，再通过cd进入目录，进入后通过create创建文件f，且允许其他用户读写，输入ls命令，可以看到文件f已经位于目录d下。

#### 

图6.8 基本功能测试截图四

之后，我们先通过read命令查看一下文件f的内容，显然为空，再通过write命令写入两行内容，以Ctrl+D结束，最后通过read命令查看，刚刚输入的内容被成功写入。

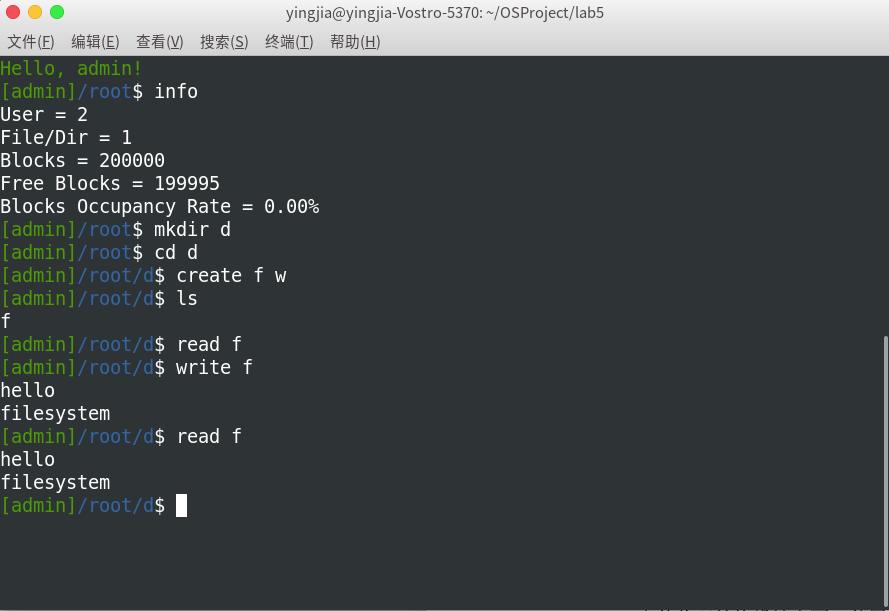


图6.9 基本功能测试截图五

这时再输入info指令，可以发现File/Dir的数目变成了3，这是因为创建了目录d和文件f的缘故；而空闲块的数目减少了3，这是因为目录d和文件f的属性块各需要一个块，文件写入的内容也需要一个块。

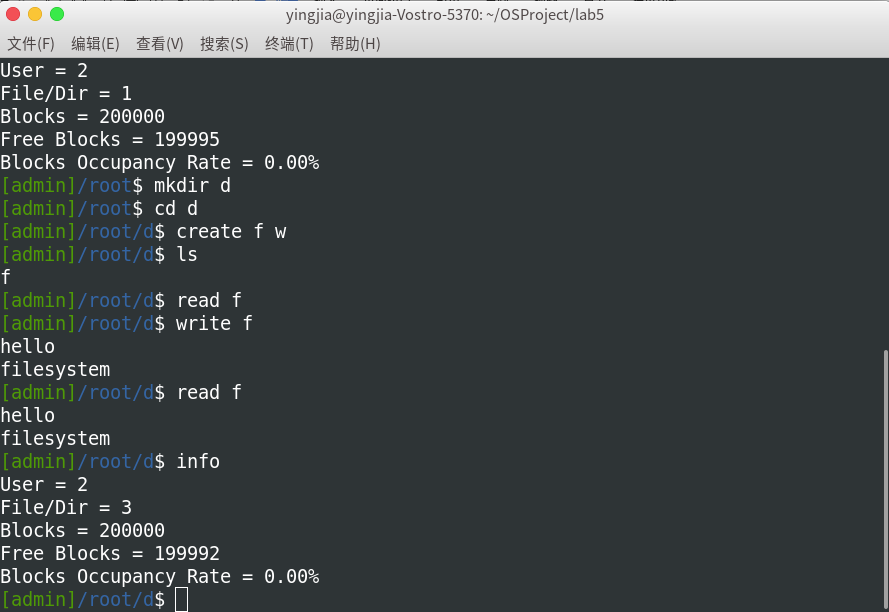


图6.10 基本功能测试截图六

之后，我们通过rm命令删除f，观察info的变化，再创建回来，返回根目录，递归删除目录d，再观察info变化。由图6.11中info的结果可知，rmdir成功删除了目录d和文件f，此时文件系统恢复到了初始化后时候的样子。

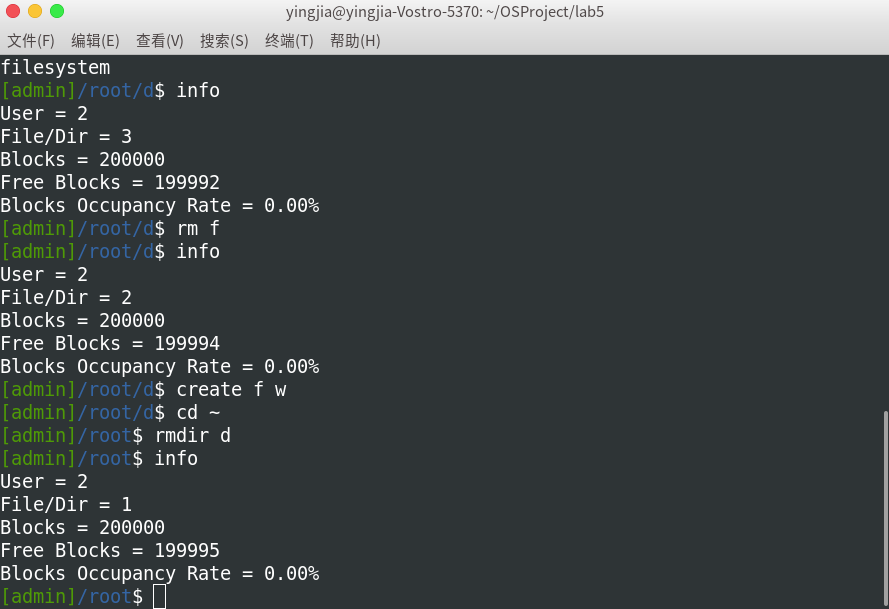


图6.11 基本功能测试截图七

我们尝试重新创建目录d，并进入该目录，再创建文件f。这时我们通过mv命令移动文件f到根目录下。由图6.12可知，原目录下文件已不存在，但根目录下同时出现了文件f和目录d，这说明mv操作成功。

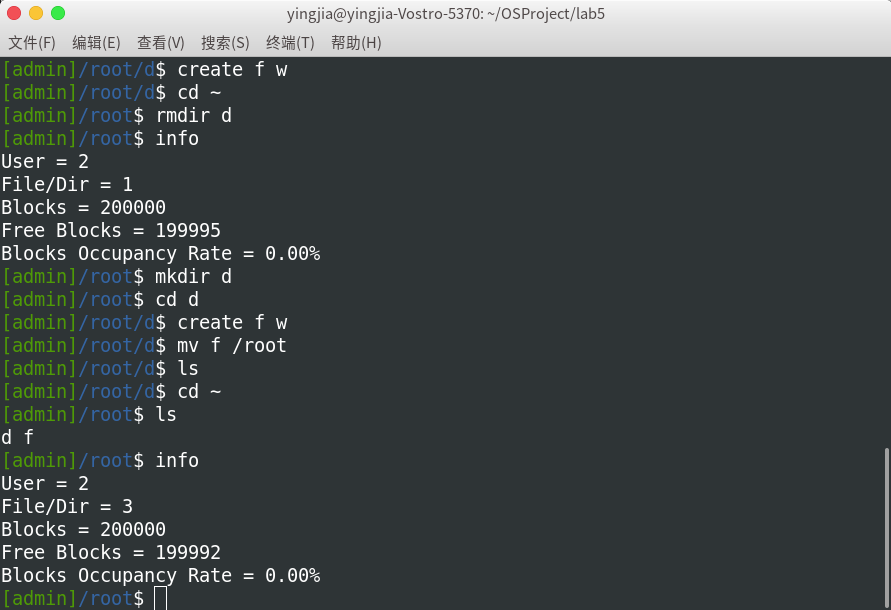


图6.12基本功能测试截图八

现在，我们通过cp命令将文件f重新拷贝进目录d，此时根目录下仍然有f，目录d下也有f，通过图6.13中info的结果可以看出，空闲块数量减少了2个，这也与预想相一致，并说明文件拷贝采用的是重新申请块复制文件的策略，新文件与原文件的属性块、数据块都不相同。

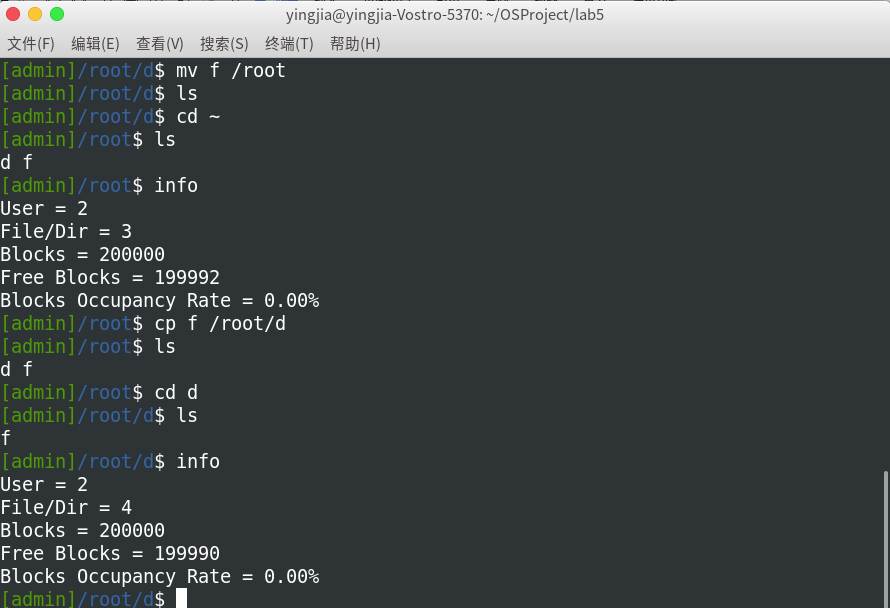


图6.13基本功能测试截图九

我们对根目录下的文件f写入两行内容，用chmod改变其他用户的权限为r，即可读但不可写。

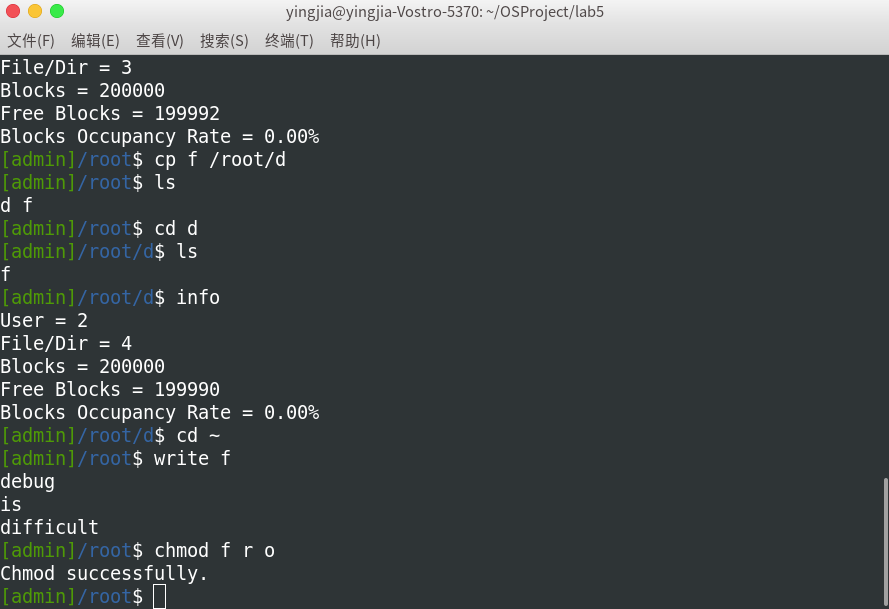


图6.14基本功能测试截图十

这时我们通过使用aduser命令创建用户test，密码为123，输入lsuser可以看到用户创建成功，并通过logout注销后登录该新创建的用户。



图6.15基本功能测试截图十一

登录成功后，尝试进行读写操作，由图6.16可知，read指令合法，但write被拒绝，这是因为刚刚chmod后，文件对非管理员的其他用户只允许读操作了。

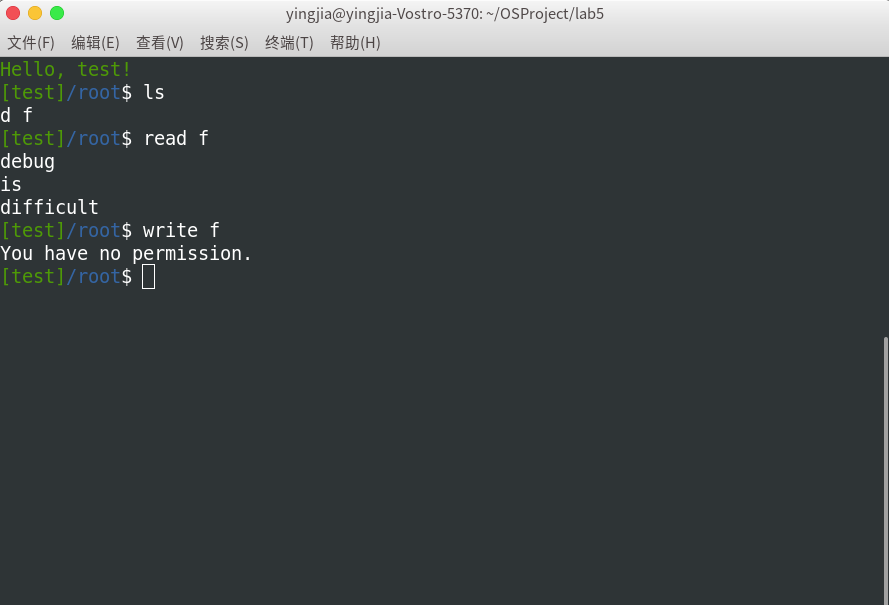


图6.16基本功能测试截图十二

切换到管理员用户并通过rmuser删除用户yingjia，输入lsuser测试是否成功删除。



图6.17基本功能测试截图十三

测试clear命令进行清屏，help命令输出文件系统操作指令的清单，exit命令先保存再退出文件系统。通过清单可以看出，全部20个功能都已经经过了测试。

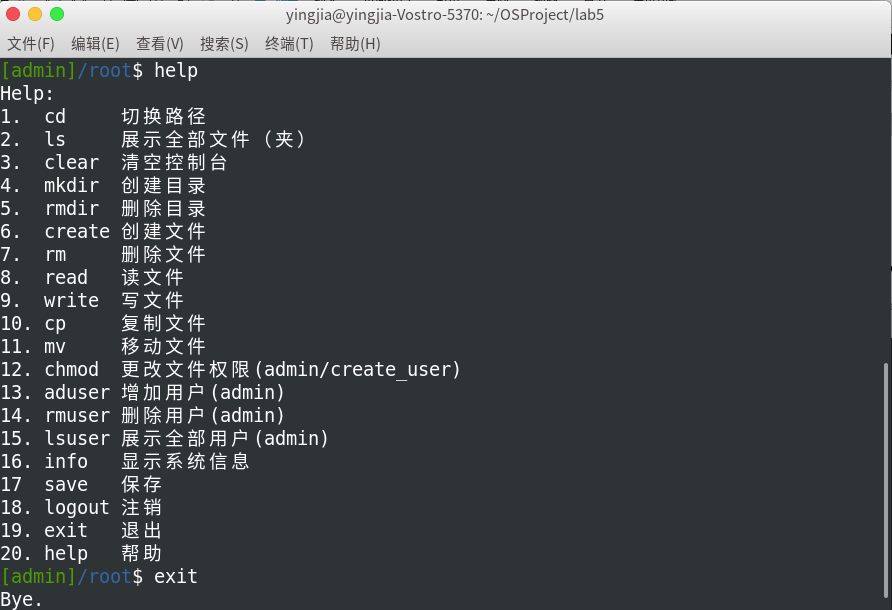


图6.18基本功能测试截图十四

最后，我们更改MEMORY\_MAX的值为40，以测试一级间接索引功能。

首先初始化系统，创建文件t，写入80个字符，输入info指令观察系统占用情况。由图6.19可知，磁盘块数量减少了4，这是因为文件索引表占用了一个块，一级索引结点占用了两个块，再加上一个文件属性块，所以info的结果无误。

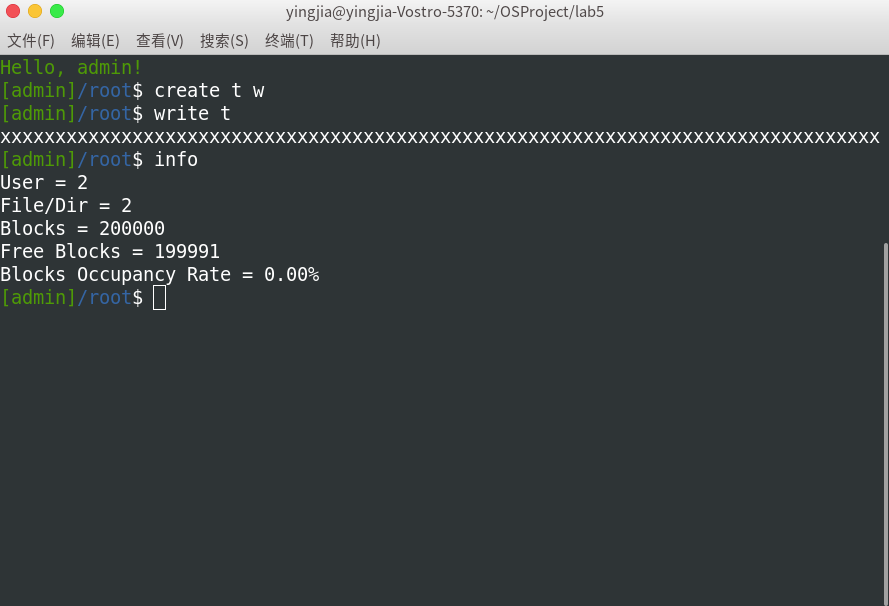


图6.19基本功能测试截图十五

然后读取文件t，观察结果，发现内容被成功写入。

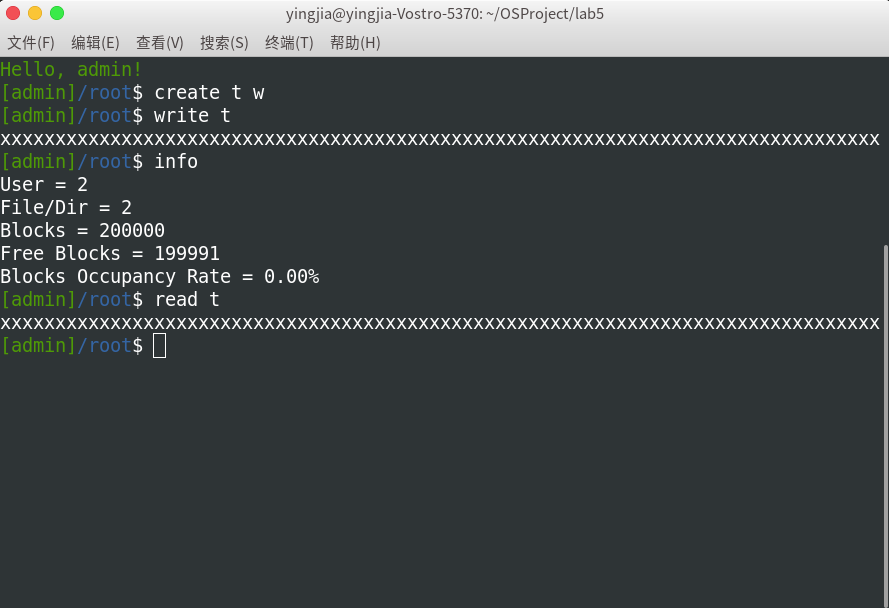


图6.20基本功能测试截图十六

重新写入30个字符x，观察read和info结果，可知由索引表到索引块切换成功。

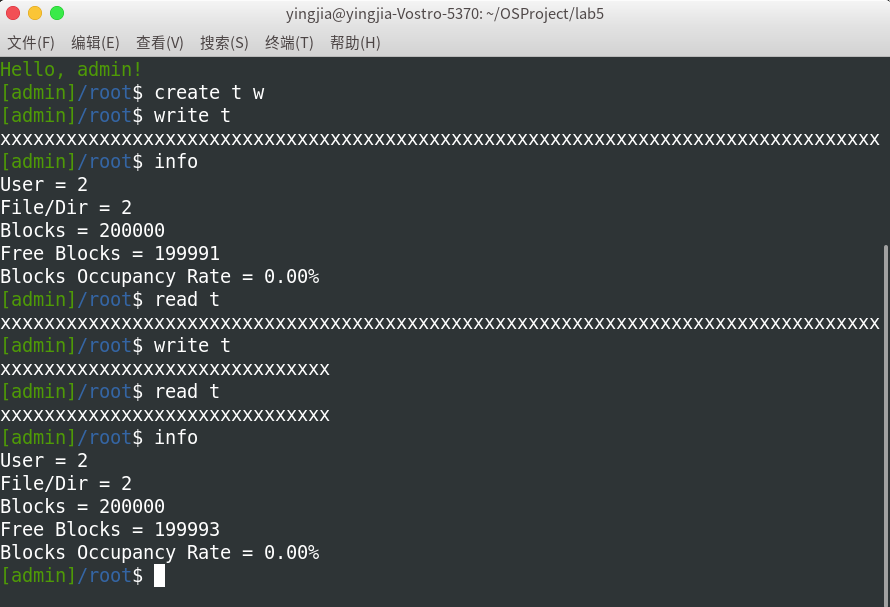


图6.21基本功能测试截图十七

再重新写入60个字符x，观察read和info结果，可知从索引块到索引表切换也成功。

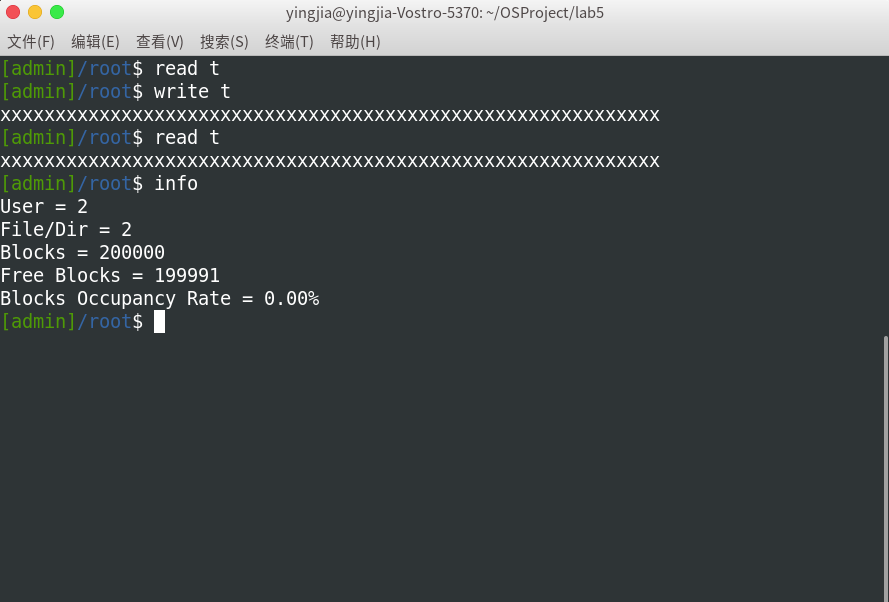


图6.22基本功能测试截图十八

**6.3.3.2 异常处理测试**

下面进行异常处理测试。

登录环节，若输入账号密码错误，将提示重新输入，当连续三次输入失败时，程序自动退出。

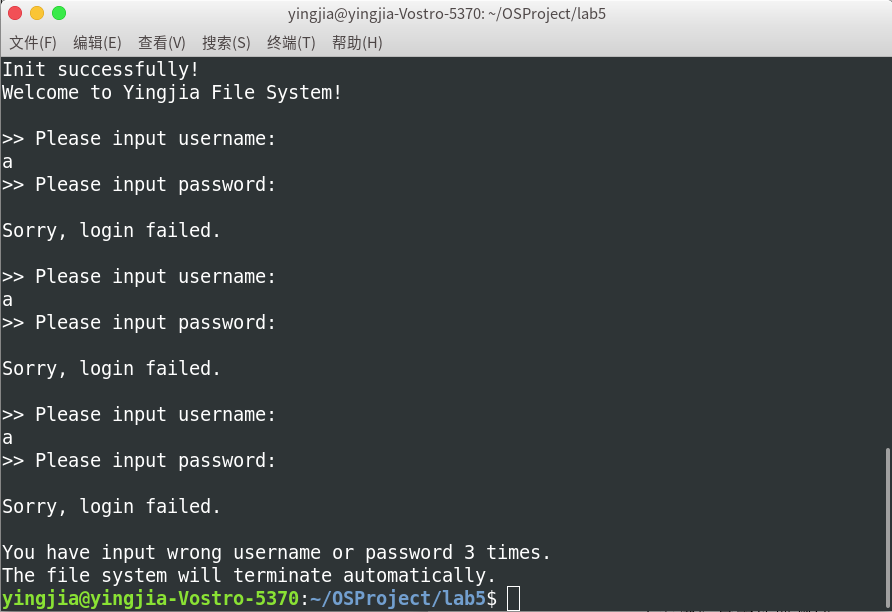


图6.23异常处理测试截图一

输入命令时，不同的命令有不同的格式控制，以create为例，create的标准输入格式为create [file\_name] [access]，其中access表示允许其他用户的权限，其中w表示可读写，r表示可写，\_表示既不可读也不可写。

下图分别进行了三种错误输入测试，由图6.24可知，程序对不同的错误可以进行分辨，并给出了适当的错误提示。



图6.24异常处理测试截图二

再举一例，首先创建目录d，并想通过cd命令进入目录d，但错误地输入了cd e，这时程序发现没有这个目录，于是提示No directory。

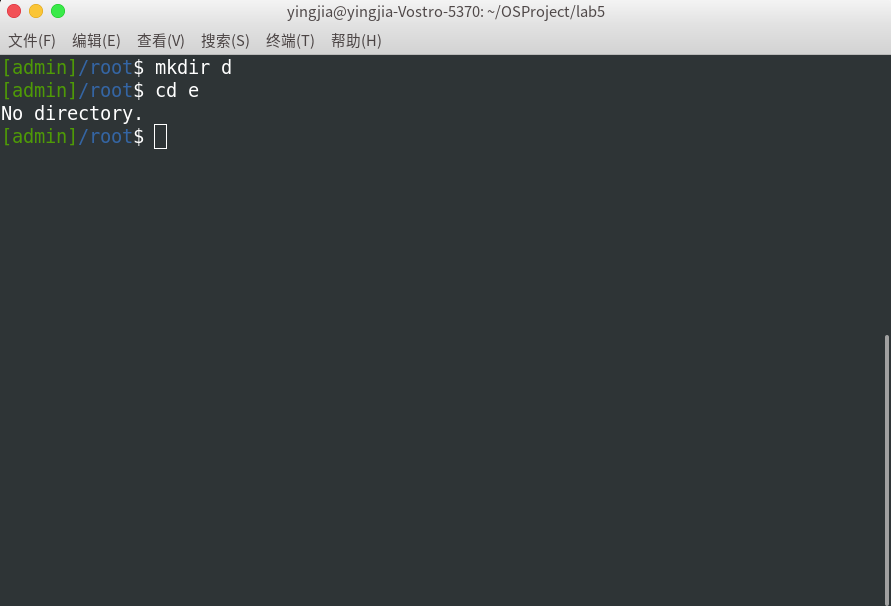


图6.25异常处理测试截图三

接下来注销，登录用户yingjia，尝试创建用户、删除用户和遍历用户三个操作，由图6.26可知，这三个操作都没有权限，这体现了文件系统对安全的控制。

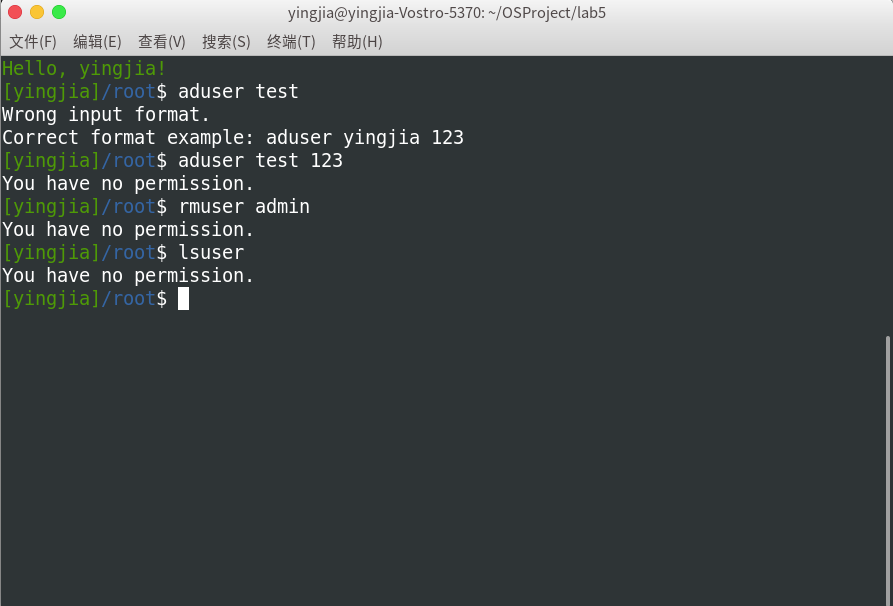


图6.26异常处理测试截图四

文件权限方面在基本功能测试中已经有了简要的测试，此处略。

**6.3.3.3 存储与恢复功能测试**

下面进行存储功能测试。

首先先创建一些目录和文件，文件内写入一些内容，如下。

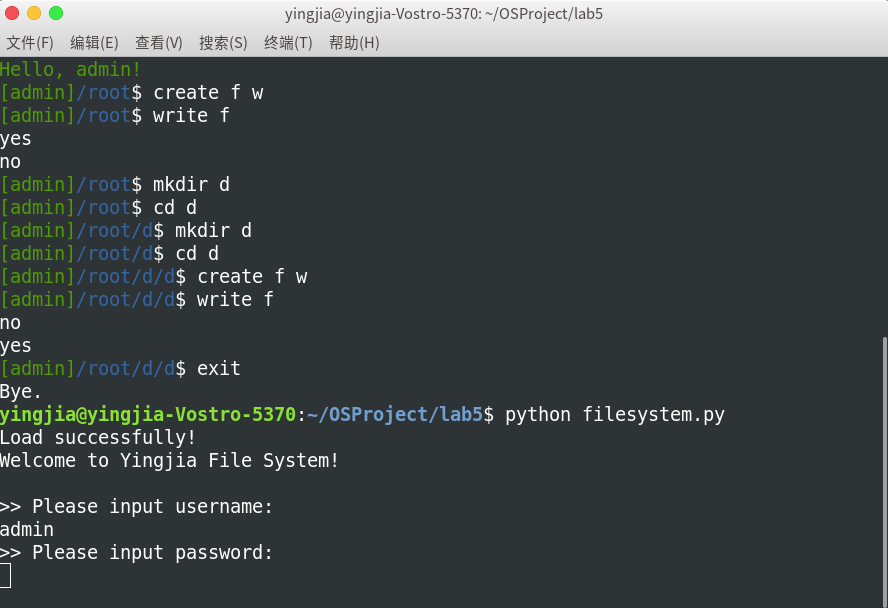


图6.27 存储与恢复功能测试截图一

之后退出，重新登录，进行测试，由图6.27与图6.28对比可知，存储与恢复功能正常，数据都得到了完好的恢复。

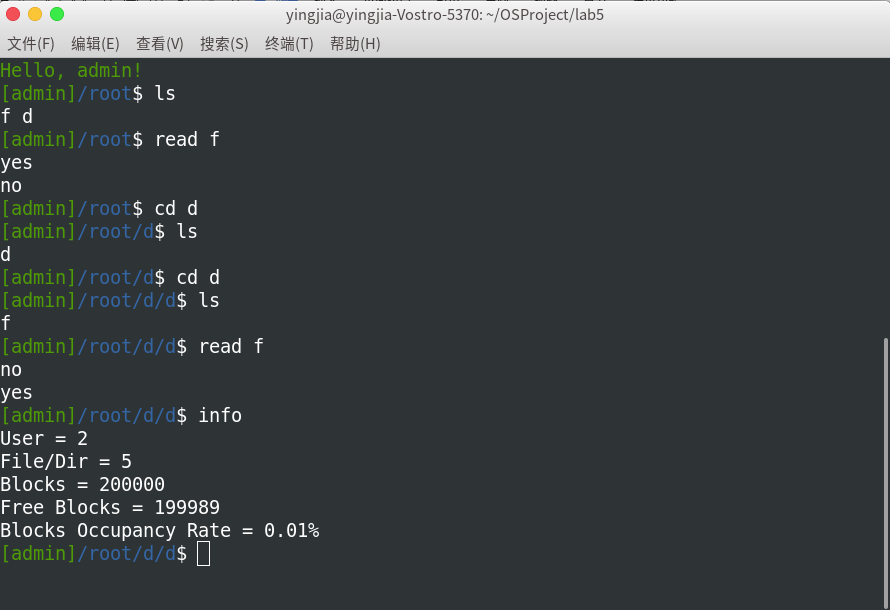


图6.28存储与恢复功能测试截图二

#### 6.3.4 实验心得

物理结构设计方面，使用索引结构既能有效地利用存储空间，又能方便存取，索引的层数越多，可支持的文件最大容量越大，但同时也提高了设计的复杂性。

存储方面，将每一个磁盘块的数据作为一行写入文件，但如果文件内容本身包含换行符，存储的数据格式就会错乱，目前采用的解决方案是将文件内容中的\n转化为$，读取时再转化回来，但这也说明了这种方案的局限性在于操作复杂，以及$字符将不允许出现在文本中。

并且，进一步可以考虑每次不全盘拷贝全部数据到内存，而只取占用了的内存块中的数据，至于哪些内存块被占用了可以补充写在前4个超级块中，这将减少大量IO时间。

交互方面，系统要注意异常判断和提示信息的完备性，对不同的错误进行捕获后提示出适当的信息对用户进行提示。

在这次实验编程中，由于一开始不注意结构化思想的运用，很多相似的代码块没有封装成一个函数进行测试并使用，所以在修改和调试过程中产生了很大的麻烦，浪费了很多时间。

#### 6.3.5 参考资料

1、《计算机操作系统》（第3版）第9章 文件系统，庞丽萍、阳富民著

## 课设总结与心得

关于子任务安排，前面的许多环节都为后续做了铺垫，例如实验一中的文本复制经过部分修改后可以用到实验二添加系统调用中，并发进程的设计也为系统管理器中读写线程分离提供了思路。

关于编程语言，课程设计中这次基本以python为主要编程语言，但实际上并未使难度减少许多，反而出现了许多其他问题。相比之下，高级语言提供更为丰富而灵活的方法，使开发时间减少，但同时也摒弃了代码的效率，并且许多内部细节是封装好的，用户很难进行干预，如在系统管理器饼状图和折线图的设计中，对于简单的图形，无论是用QChart模块还是matplotlib模块实际上都不如gtk简洁高效，并且用pyqt5写的图形界面如果没有双线程处理，运行十分卡顿。以后进行项目开发时，也要充分分析各种编程语言哪种更为合适，在开发速度和运行效率间充分考量。

在课设中学习了众多的博客与教材上的相关知识，融合自己的想法并实现到了实验中，极大地提高了对操作系统方面知识的理解与运用，对linux内核源码也有了进一步的了解。

最后感谢石柯老师对我课设进行过程中的耐心指导。

## 附录

（一）熟悉和理解linux编程环境

cp.c 文件拷贝

monitor.c 分窗口显示并发进程运行中子进程创建

window.py 分窗口显示并发进程运行中可视化设计及进程功能

### （二）添加系统调用

mycopy.c 添加到内核的源程序

test.c 测试程序

### （三）添加设备驱动

dev.c 设备驱动程序

dev\_test.c 测试程序

Makefile 编译脚本

### （四）理解/proc文件

monitor.py 系统管理器运行程序

### 模拟文件系统设计

filesystem.py 文件系统运行程序

config.py 配置参数及类定义

filesystem 文件系统