# 華中科技大學

# 操作系统课程设计报告


姓　　名：　　刘一龙

学　　院：　　计算机科学与技术学院

专　　业：　　计算机科学与技术

班　　级：　　计科 1409 班

学　　号：　　U201414800

指导教师：　　谢美意


| 分数 | |
|---|---|
| 教师签名 | |

2017 年　3 月　12　日

# 目　录

# 实验一

# 1 实验目的

1. 编写程序实现文件的复制功能；
2. 编写程序实现三个并发进程用三个窗口分别显示当前系统时间，当前 cpu 利用率，显示 1-1000 的累加和。

# 2 实验内容

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)。

# 3 实验设计

## 3.1 平台环境

硬件平台：Intel i5-4210U(1.70GHz), 8.00GB RAM, 500GB Hard Disk

软件平台：Ubuntu 16.04 LTS, LinuxMint 18.1

编程环境：Ubuntu SDK（Qt Creator） with FakeVim plugin, git, zsh

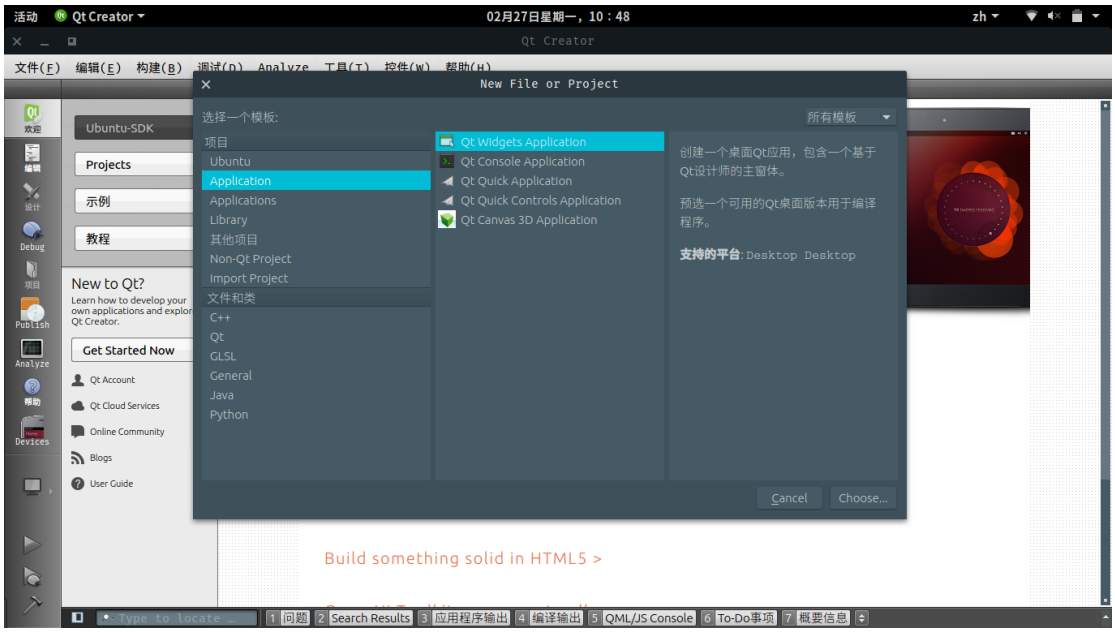


图 1 Qt Creator 创建项目图

## 3.2 方案设计

1. 第一个实验为并发进程实验。在主进程利用 fork 函数创建 2 个子进程，再利用 exec 系列函数（本次实验选用 execlp 函数）执行 writebuf 与 readbuf 二进制；
2. writebuf 是一个独立的 C 语言二进制程序，其功能为从源文件中不断读入有效字符，并送入共享内存缓冲区中，共享内存缓冲区由 semget/semat/semdt/semctl 系列函数负责控制管理；
3. readbuf 是一个独立的 C 语言二进制程序，其功能为从共享内存缓冲区中不断读入有效字符，并将其送入目标文件。其与 writebuf 程序的交互便是由共享内存缓冲区实现的。通过对同一块共享内存进行读写，即可完成文件的并行拷贝；
4. 通过 semaphore 进行两个读写进程间的同步关系，如图 2 writebuf 与 readbuf 进程同步互斥关系图所示。读写进程间除了要进行同步读写外，还要考虑读写何时结束的问题。在共享内存块内部保存当前缓冲区剩余数据的字节数，显然这个变量是一个共享变量，需要加锁以使得读进程与写进程对其进行互斥访问。利用 fread 返回值小于 0 可以得知源文件已经读取完毕，此时 writebuf 进程将结束标志置 1，并结束自身；readbuf 进程会同时检测剩余字节数与结束标志，只要当剩余字节数为 0 与结束标志为 1 同时发生时，才会结束自身，否则持续从共享内存块取数据并写入目标文件。
5. 第二个实验的主要难点在于 Qt 图形库的使用与 CPU 使用率的计算；
6. 关于 Qt 图形库的使用，主要是参考 https://www.gitbook.com/book/wizardforcel/qt-beginning/details 这本开源电子书，在其中了解到了如何使用纯净的 C++ 代码建立简单的多窗口 Qt 应用，从而实现了纯代码 UI（不含乱糟糟的自动生成的 Qt UI 描述文件）界面的构建；

7. 此次多窗口的实现选择了一个最奢侈的办法，即为每一个窗口都建立一个独立的QApplication。利用此种方法可以有效地进行并发多窗口的建立，不会出现什么奇怪的错误。对于每个窗口类而言，只需实现 3 个基本函数即可，分别为 getXXX、setText 与 updateXXX。其中 getXXX 用于计算下一次需要显示的有效值，setText 用于更新 QLabel 的文本，updateXXX 绑定到定时器 slot 上，并在其中调用 getXXX 与 setText 函数，从而实现定时更新文本的功能；
8. 从操作系统原理课上可以知道，在 Linux 内核中存在一个名为 idle 的内核级线程。它是整个系统中的第一个线程，如果没有其它进程或线程需要使用 CPU，则它持续地进行空转，直至第一个进程或线程被 CPU 调度至就绪状态，它便让出 CPU 使用权。所以，计算 CPU 时间的关键在于计算出 idle 线程占用 CPU 时间比。可以对 /proc/stat 进行 2 次取样，计算出此段时间内 idle 时间的增长占总 CPU 时间的增长的百分比，即可求出此段时间内大致的 CPU 空闲率，从而得到 CPU 使用率。其中，总 CPU 时间为 /proc/stat 文件第一行所有值的累加，idle 时间第 4 列（从 1 开始计数）值。最终效果图，如图 3 多窗口运行图(LinuxMint 18.1)所示。

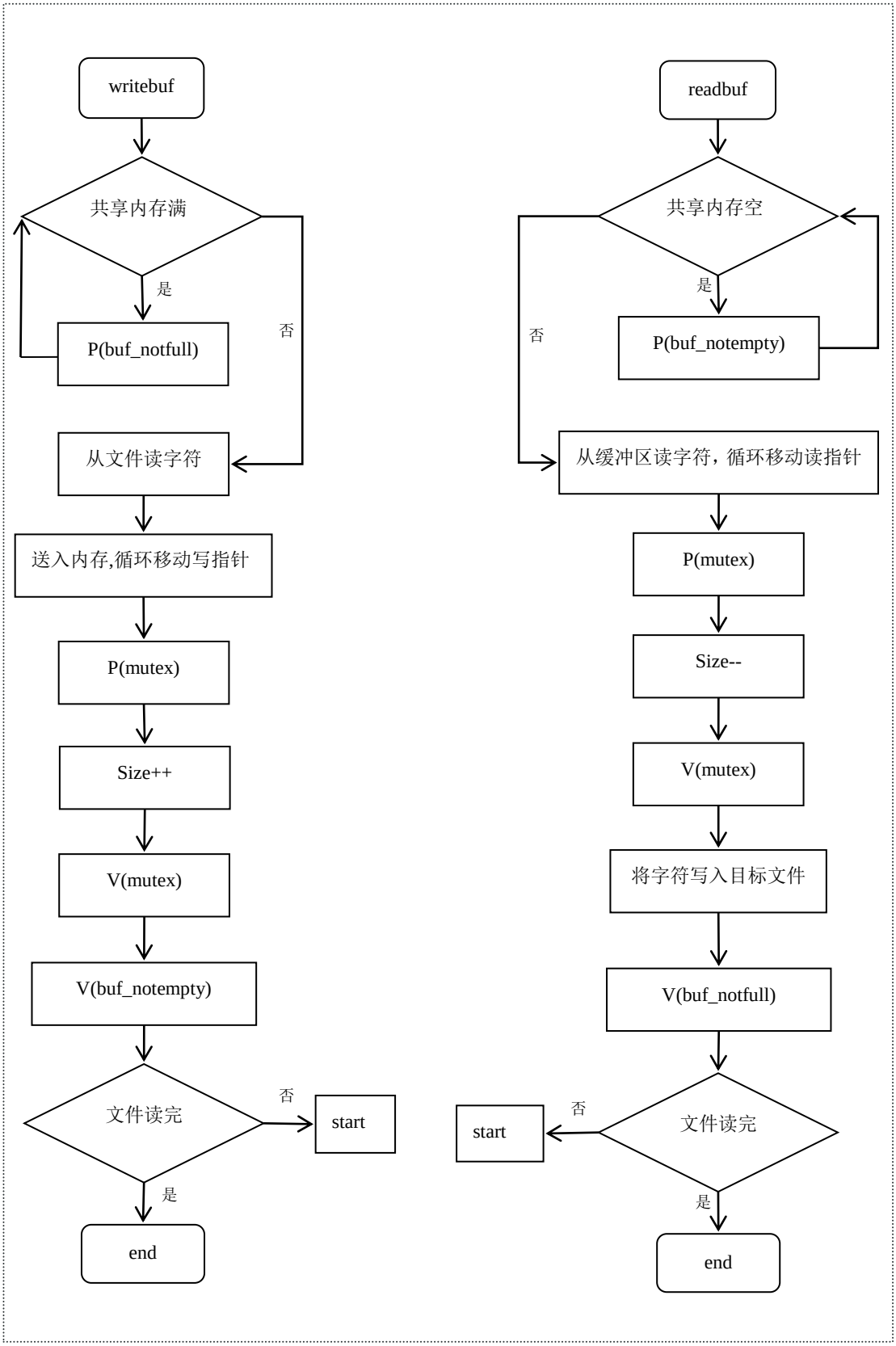


图 2 writebuf 与 readbuf 进程同步互斥关系图

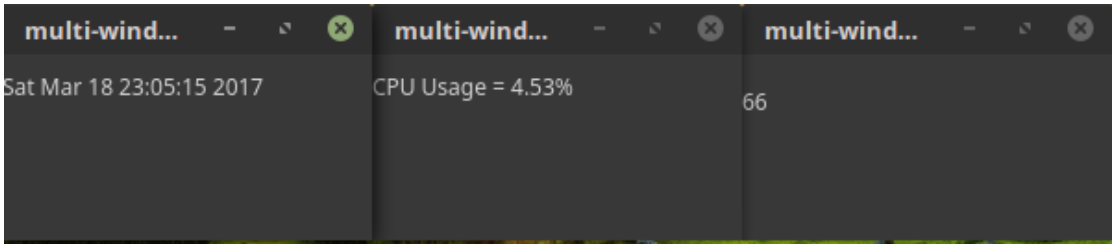


图 3 多窗口运行图(LinuxMint 18.1)

# 4 实验调试

## 4.1 实验步骤

1. 封装 semaphore API，编写主进程函数、writebuf 主函数、readbuf 主函数，进行测试；
2. 编写第一个窗口程序 TimeWindow，进行测试；
3. 以 TimeWindow 为模板编写 CPUWindow 与 SumWindow，进行测试。

## 4.2 实验调试及结果

**故障 1**：误用 0 作为 semget(semnew 封装函数) 的键值参数，创建 buf_notfull 信号灯

**故障现象**：当 writebuf 进程持续将源文件字符写入共享内存，readbuf 进程持续将共享内存字符写入目标文件，两个进程间将产生死锁，造成整个程序阻塞，结果如图 4 读写文件死锁现象(Ubuntu 16.04 LTS)所示。

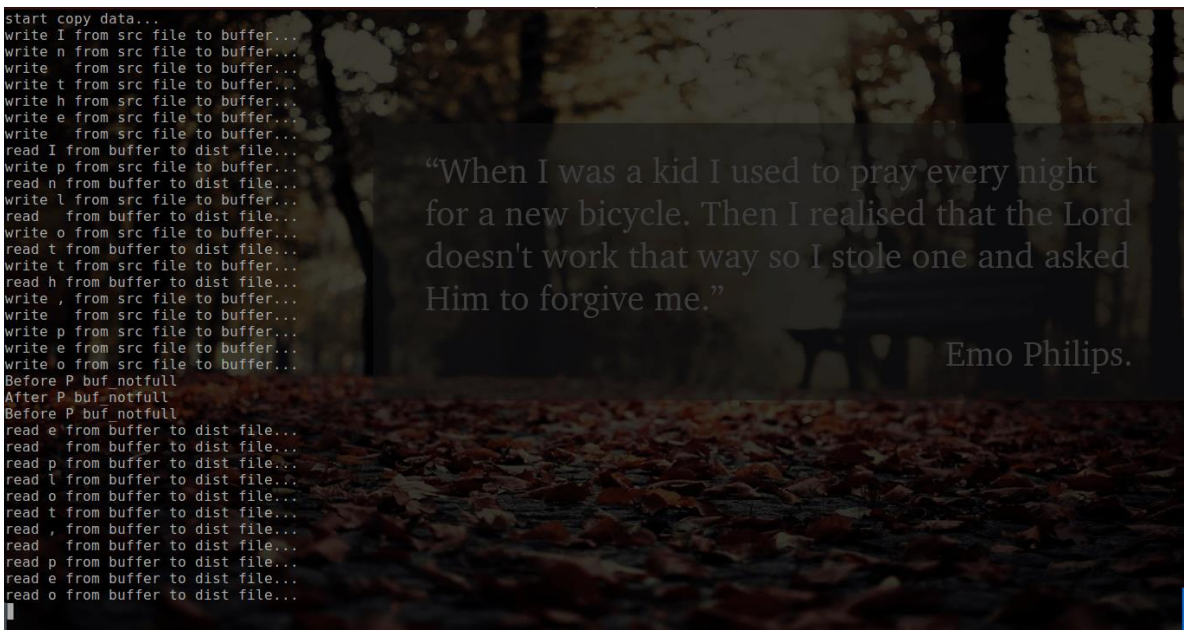


图 4 读写文件死锁现象(Ubuntu 16.04 LTS)

**原因分析**：使用 logger 的方式进行调试，在所有 P/V 操作前后都进行日志输出，发现

writebuf 进程由于 P(buf_notfull) 而阻塞， readbuf 进程由于 P(buf_notempty) 而阻塞。从进程执行过程可以看出，尽管在 writebuf 进行 P 操作后，readbuf 进程进行了多次读操作，若程序正常执行，则 writebuf 进程会因 readbuf 的读操作(V(buf_notfull)) 而进入就绪队列，停止阻塞，但事实却与之相反。故猜想，由于 P/V 操作没能成功地对指定信号灯进行 +/- 操作，使得 writebuf 进程一直阻塞，当 readbuf 将内存区字符读完后也会阻塞，从而造成死锁。

**解决方案：**打印两个信号灯的 id 后发现，用 0 作为键值创建的信号灯，无法通过有效手段再次获取其引用(即第二次进行 semget 操作时得到的是一个全新的信号灯)。将键值改为非 0 值后，死锁现象消失，程序正常运行。

**故障 2：**文件读写完成标志设置不合理

**故障现象：**当源文件为一个特殊文件时，其可能含有编码为 -1 的字符，此时不能使用 EOF 作为文件结束标志。

**解决方案：**不再利用读出的字符 ch 是否为 EOF 作为文件结束标志，而是利用 fread 返回值是否 <= 0(读取失败)作为文件结束标志，有效地处理一切特殊编码的文件。

**故障 3：**TimeWindow 时间显示不全

**故障现象：**TimeWindow 时间显示不全，如图 5 时间显示不全图(Ubuntu 16.04 LTS)所示。

**解决方案：**发现时间被截断是由于 QLabel 的宽度设置过小，其用没有自适应功能，造成文字被截断的错误。利用 label->setFixedSize(220, 50) 可以重新设定宽度，顺利解决问题，如图 6 时间显示完全图(Ubuntu 16.04 LTS)所示。

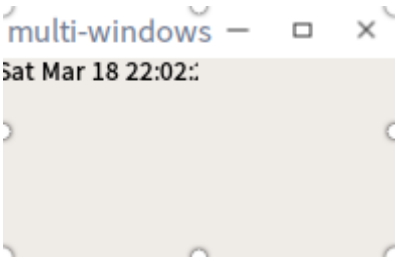


图 5 时间显示不全图(Ubuntu 16.04 LTS)

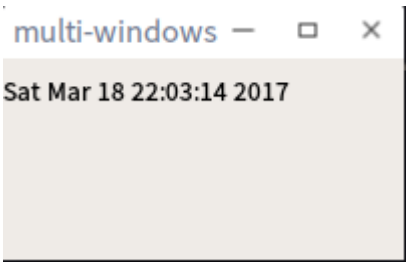


图 6 时间显示完全图(Ubuntu 16.04 LTS)

**故障 4：**多进程窗口无法正常启动

**故障现象：**当试图在不同的进程下分别启动 3 个窗口时，Qt Creator 控制台未输出任何信息，启动进度条也显示为 100 %（绿色），但窗口不显示。

**解决方案：**仔细思考便可发现，主进程里的 QApplication 与子进程里的 QWindow 不存在

任何联系，导致窗口无法正常地设置在特定的 QApplication 上。所以，为每个子进程都创建一个单独的 QApplication 即可解决问题。

## 4.3 实验心得

1. 此次实验掌握了利用 fork/exec 函数创建子进程的基本方法，掌握了 sem/shm 信号量/共享内存进行进程间通信的进本方法；

2. 此次实验掌握了 Qt 图形库中窗口的建立与标签(QLabel)组件的使用，并初次接触了/proc 伪文件系统，成功地计算了 CPU 使用率；

3. 感受到了 Qt 图形库的强大性与易用性。通过后续的资料阅读发现，Qt 不仅提供了基本的 UI 组件，还为几乎所有基本类型建立了对应的内建 Qt 类型，如 QString 对应 string，QPoint 对应二位坐标，甚至是当前时间都可以利用 QDateTime 直接得到。

# 实验二

# 1 实验目的

使用编译内核的方法添加一个实现文件复制操作的系统调用，并进行功能测试。了解 linux 系统调用的过程。

# 2 实验内容

1. 采用编译内核的方法，添加一个新的系统调用，实现文件拷贝功能
2. 编写一个应用程序，测试新加的系统调用

# 3 实验设计

## 3.1 平台环境

硬件平台：Intel i5-4210U(1.70GHz), 8.00GB RAM, 500GB Hard Disk

软件平台：CentOS 6（Linux Kernel 2.6.0）, Ubuntu 14.04 LTS（Linux Kernel 4.4.0）, Ubuntu 16.04 LTS （Linux Kernel 4.4.0/Linux Kernel 4.10.0）

编程环境：Vim 7.4, Make, GCC, git, zsh

## 3.2 方案设计

1. 第一次尝试是在 VPS 上尝试的。利用 CentOS 官方 Wiki 上的 https://wiki.centos.org/HowTos/Custom_Kernel 进行 CentOS 内核编译尝试。按照官方 Wiki 所给步骤进行内核编译（未添加系统调用），最后结果如图 7 CentOS 编译内核失败图所示。经查阅相关资料后发现，云服务提供商提供的 VPS 是无法进行内核的替换的，所以放弃的 VPS 上进行此次实验。

2. 第二次尝试是在本地机器上（Ubuntu 16.04 LTS, Linux Kernel 4.4.0）尝试的。结合 PPT，以及 Medium 上的这篇 Story Medium: Implementing a system call in Linux Kernel （需要翻墙），成功地掌握了向 Linux 内核添加基本系统调用的方法；

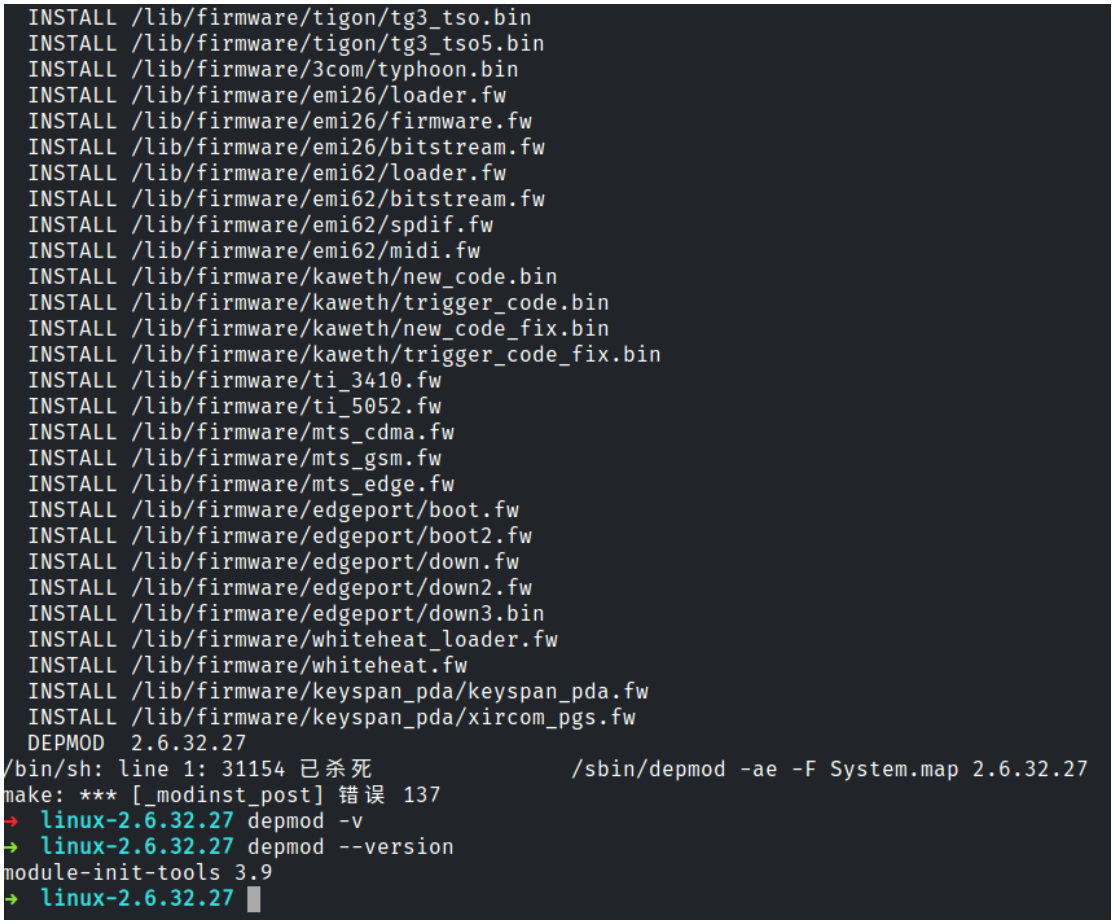


图 7 CentOS 编译内核失败图

3. 编译新内核分为 4 步。第一步，利用 wget 从 linux.org 上获取 4.10.0 版本（最新版）的内核源代码，并解压至 /usr/src。第二步，安装编译内核所需依赖，如 gcc,make,build-essential,libncurses5-dev（用于显示内核配置界面）,libssl-dev（内核中有模块依赖 openssl）等。第三步，清空旧内核残留，配置新内核的编译设置，利用 make mrproper && make clean && make menuconfig 命令即可完成。第四步，编译内核并重启，利用 make –j 4 && make modules_install –j 4 && make install –j 4 && reboot 即可完成。利用 make –j 4 可以利用多个核加快编译。

4. 添加系统调用分为 3 步。第一步，向 arch/x86/entry/syscalls/syscall_64.tbl 中添加中断号如 548 common dragoncopy sys_dragoncopy。第二步，向 include/linux/syscalls.h 添加中断处理函数原型 asmlinkage long sys_dragoncopy(const char *src, const char *dst)。第三步，在一个独立的 dragoncopy.c 里编写内核文件拷贝函数，并修改主 Makefile 中的 core-y 宏的值，讲 dragoncopy.c 所在的子目录加入至 core-y 宏，这样可以使得 dragoncopy.c 被编译链接至内核中；

5. 编写内核文件拷贝系统调用需要注意 2 点。第一，不能再使用 C 语言库函数，必须使用如 printk/filp_open/filp_close/vfs_read/vfs_write 等内核函数实现文件的相关操作。第二，必须利用 set_fs 函数将堆栈段临时地切换为内核堆栈，在进行完内存操作后，再恢复堆栈段，这样才可实现正常的存取拷贝。

# 4 实验调试

## 4.1 实验步骤

1. 在此不再赘述 CentOS 上编译内核的步骤，完全地按照官方 Wiki 所给步骤进行操作的，但无法成功地替换新内核，基本步骤为登录 VPS、安装依赖、获取源码、编译内核、安装内核，如图 8、图 9、图 10、图 11 所示。其中在安装内核时，发生了如图 7 所示的错误，意识到 VPS 无法更换内核后，放弃此种尝试；

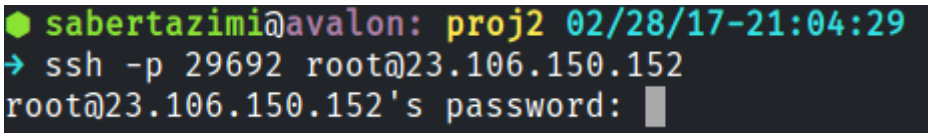


图 8 登录 VPS

```
➜  ~ yum groupinstall "Development Tools"
已加载插件：fastestmirror
设置组进程
Loading mirror speeds from cached hostfile
 * base: mirror.hostduplex.com
 * extras: centos.mirrors.hoobly.com
 * updates: mirror.supremebytes.com
包 flex-2.5.35-9.el6.i686 已安装并且是最新版本
包 gcc-4.4.7-17.el6.i686 已安装并且是最新版本
包 redhat-rpm-config-9.0.3-51.el6.centos.noarch 已安装并且是最新版本
包 rpm-build-4.8.0-55.el6.i686 已安装并且是最新版本
包 1:make-3.81-23.el6.i686 已安装并且是最新版本
包 patch-2.6-6.el6.i686 已安装并且是最新版本
包 1:pkgconfig-0.23-9.1.el6.i686 已安装并且是最新版本
包 gettext-0.17-18.el6.i686 已安装并且是最新版本
包 automake-1.11.1-4.el6.noarch 已安装并且是最新版本
包 bison-2.4.1-5.el6.i686 已安装并且是最新版本
包 libtool-2.2.6-15.5.el6.i686 已安装并且是最新版本
包 autoconf-2.63-5.1.el6.noarch 已安装并且是最新版本
包 gcc-c++-4.4.7-17.el6.i686 已安装并且是最新版本
包 binutils-2.20.51.0.2-5.44.el6.i686 已安装并且是最新版本
包 patchutils-0.3.1-3.1.el6.i686 已安装并且是最新版本
包 byacc-1.9.20070509-7.el6.i686 已安装并且是最新版本
包 indent-2.2.10-7.el6.i686 已安装并且是最新版本
包 systemtap-2.9-4.el6.i686 已安装并且是最新版本
包 diffstat-1.51-2.el6.i686 已安装并且是最新版本
包 elfutils-0.164-2.el6.i686 已安装并且是最新版本
包 cvs-1.11.23-16.el6.i686 已安装并且是最新版本
包 rcs-5.7-37.el6.i686 已安装并且是最新版本
包 subversion-1.6.11-15.el6_7.i686 已安装并且是最新版本
包 gcc-gfortran-4.4.7-17.el6.i686 已安装并且是最新版本
包 1:doxygen-1.6.1-6.el6.i686 已安装并且是最新版本
```

图 9 安装依赖

```
95% [=============================================================================>     ] 78,125,904  1.39M/s eta(英国中部时
95% [=============================================================================>     ] 78,457,680  1.27M/s eta(英国中部时
96% [=============================================================================>     ] 78,797,648  1.22M/s eta(英国中部时
96% [=============================================================================>     ] 78,994,256  1.26M/s eta(英国中部时
96% [=============================================================================>     ] 79,031,120  1.11M/s eta(英国中部时
96% [=============================================================================>     ] 79,207,248  1.16M/s eta(英国中部时
96% [=============================================================================>     ] 79,342,416  1.13M/s eta(英国中部时
96% [=============================================================================>     ] 79,391,568  1.12M/s eta(英国中部时
96% [=============================================================================>     ] 79,457,104  1.09M/s eta(英国中部时
97% [=============================================================================>     ] 79,543,120  1.04M/s eta(英国中部时
97% [=============================================================================>     ] 79,633,232   989K/s eta(英国中部时
97% [=============================================================================>     ] 79,706,960   927K/s eta(英国中部时
97% [=============================================================================>     ] 79,829,840   857K/s eta(英国中部时
97% [=============================================================================>     ] 79,936,336   669K/s eta(英国中部时
97% [=============================================================================>     ] 80,063,312   693K/s eta(英国中部时
97% [=============================================================================>     ] 80,100,176   577K/s eta(英国中部时
97% [=============================================================================>     ] 80,214,864   486K/s eta(英国中部时
97% [=============================================================================>     ] 80,276,304   404K/s eta(英国中部时
98% [=============================================================================>     ] 80,407,376   393K/s eta(英国中部时
98% [=============================================================================>     ] 80,509,776   382K/s eta(英国中部时
98% [=============================================================================>     ] 80,677,712   426K/s eta(英国中部时
98% [=============================================================================>     ] 80,833,360   414K/s eta(英国中部时
98% [=============================================================================>     ] 80,939,856   417K/s eta(英国中部时
98% [=============================================================================>     ] 81,050,448   432K/s eta(英国中部时
99% [=============================================================================>     ] 81,263,440   476K/s eta(英国中部时
99% [=============================================================================>     ] 81,410,896   484K/s eta(英国中部时
99% [=============================================================================>     ] 81,509,200   493K/s eta(英国中部时
99% [=============================================================================>     ] 81,648,464   503K/s eta(英国中部时
99% [=============================================================================>     ] 81,791,824   509K/s eta(英国中部时
99% [=============================================================================>     ] 81,910,608   515K/s eta(英国中部时
100%[=============================================================================>] 81,978,970   511K/s   in 2m 11s

2017-02-28 10:47:57 (612 KB/s) - 已保存 “linux-2.6.32.27.tar.gz” [81978970/81978970])

➜ src wget https://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.32.27.tar.gz
```

图 10 获取内核源码

```
[sabertazimi@localhost SPECS]$ rpmbuild --without kabichk -bb --target=`uname -m` kernel.spec 2> build-err.log | tee build-out.log
Building target platforms: i686
Building for target i686
Executing(%prep): /bin/sh -e /var/tmp/rpm-tmp.C43InU
###
### Now generating a PGP key pair to be used for signing modules.
###
### If this takes a long time, you might wish to run rngd in the background to
### keep the supply of entropy topped up.  It needs to be run as root, and
### should use a hardware random number generator if one is available, eg:
###
###     rngd -r /dev/hwrandom
###
### If one isn't available, the pseudo-random number generator can be used:
###
###     rngd -r /dev/urandom
###
###
### Key pair generated.
###
Executing(%build): /bin/sh -e /var/tmp/rpm-tmp.IkNDpA
BUILDING A KERNEL FOR i686...
USING ARCH=i386
d465be0fa14544aac6d0ca442ef1064e1ce5170f
058772dbfae097a5ce3cc0b4c0e175b1186a1a99
77351c6b949656f2fefceb6e6f9eb770b6f40afe
f272ad7f0a4f77b9c9647ec0161d6887de470c97
3c027b8877d3d3e82a02152a97016bd7aec5679b
3c027b8877d3d3e82a02152a97016bd7aec5679b
Kernel: arch/x86/boot/bzImage is ready  (#1)
```

图 11 编译内核

2. 开始在本机上进行内核编译与系统调用添加的尝试，首先防止由于编译内核造成系统崩溃（实体机上只有 Ubuntu 16.04 LTS 单系统），先在虚拟机（Ubuntu 14.04 LTS）里进行试验；

3. 在本机上安装一个 Ubuntu 14.04 LTS 的虚拟机（VirtualBox），如图 12 所示。安装完成后，查看系统内核版本号，如图 13 所示。

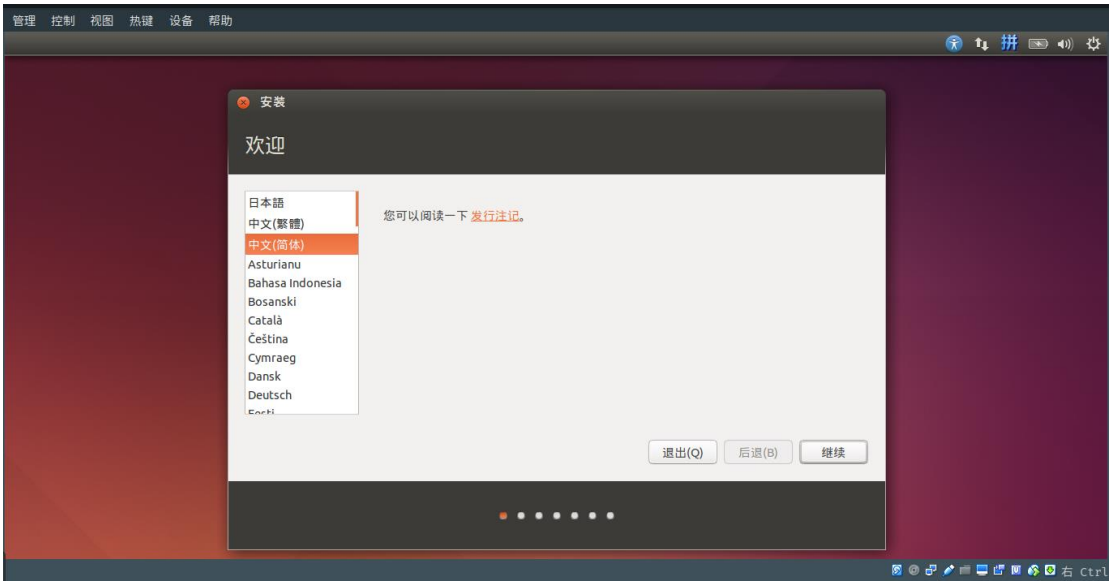


图 12 创建 VirtualBox 虚拟机

```
sabertazimi@kernelmake:~$ cat /etc/lsb-release
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=14.04
DISTRIB_CODENAME=trusty
DISTRIB_DESCRIPTION="Ubuntu 14.04.5 LTS"
sabertazimi@kernelmake:~$ uname -a
Linux kernelmake 4.4.0-31-generic #50~14.04.1-Ubuntu SMP Wed Jul 13 01:07:32 UTC 2016 x86_64 x86_64 x86_64 GNU/Linux
sabertazimi@kernelmake:~$
```

图 13 查看内核版本

4. 按照前述方案设计，将基本的操作写成 shell 脚本，如图 14 所示，方便简化内核编译流程（实际上，为了方便地截图，并未直接运行这个脚本，而是一步步地进行内核编译）；

```
#! /bin/bash
#
# kernel.sh
# Copyright (C) 2017 sabertazimi <sabertazimi@avalon>
#
# Distributed under terms of the MIT license.
#

# exec under /usr/src

# su root
# wget https://www.kernel.org/pub/linux/kernel/v4.x/linux-4.4.31.tar.gz
# tar -zxvf linux-4.4.31.tar.gz
# cd linux-4.4.31/

apt-get install libncurses5-dev
make mrproper             # 清除旧的编译内容
make menuconfig           # 编译新内核的配置文件
make bzImage              # 生成新的内核镜像
make modules              # 编译模块
make modules_install      # 安装模块
make install              # 安装内核

# update-grub             # 更新引导文件
# reboot                  # 重新启动

```

图 14 内核编译脚本

5. 获取内核源码，如图 15、图 16 所示；

```
root@kernelmake:/usr/src# wget https://www.kernel.org/pub/linux/kernel/v4.x/linux-4.4.31.tar.gz && tar -zxvf linux-4.4.31.tar.gz
--2017-03-01 17:31:50--  https://www.kernel.org/pub/linux/kernel/v4.x/linux-4.4.31.tar.gz
正在解析主机 www.kernel.org (www.kernel.org)... 198.145.20.140, 149.20.4.69, 199.204.44.194, ...
正在连接 www.kernel.org (www.kernel.org)|198.145.20.140|:443... 已连接。
已发出 HTTP 请求，正在等待回应... 200 OK
长度： 132979654 (127M) [application/x-gzip]
正在保存至: “linux-4.4.31.tar.gz”

 9% [========>                                                                ] 12,201,733   758KB/s  估时 3m 15ss
```

图 15 wget 下载内核源码

```
root@kernelmake:/usr/src/linux-4.4.31# ls
arch    COPYING  Documentation  fs       ipc      kernel     MAINTAINERS  net            samples   sound  virt
block   CREDITS  drivers        include  Kbuild   kernel.sh  Makefile     README         scripts   tools
certs   crypto   firmware       init     Kconfig  lib        mm           REPORTING-BUGS  security  usr
root@kernelmake:/usr/src/linux-4.4.31#
```

图 16 成功获取内核源码

6. 开始添加系统调用。首先，如图 17 所示，修改 syscall_64.tbl，添加系统调用号；然后，如图 18 所示，修改 syscalls.h，增加系统调用函数原型；最后，如图 19、图 20 所示，增加子目录 dragoncopy，在子目录下实现内核拷贝函数，再修改 Makefile 文件，使得 dragoncopy.o 被链接至内核镜像中。至此，即可完成系统调用的添加；

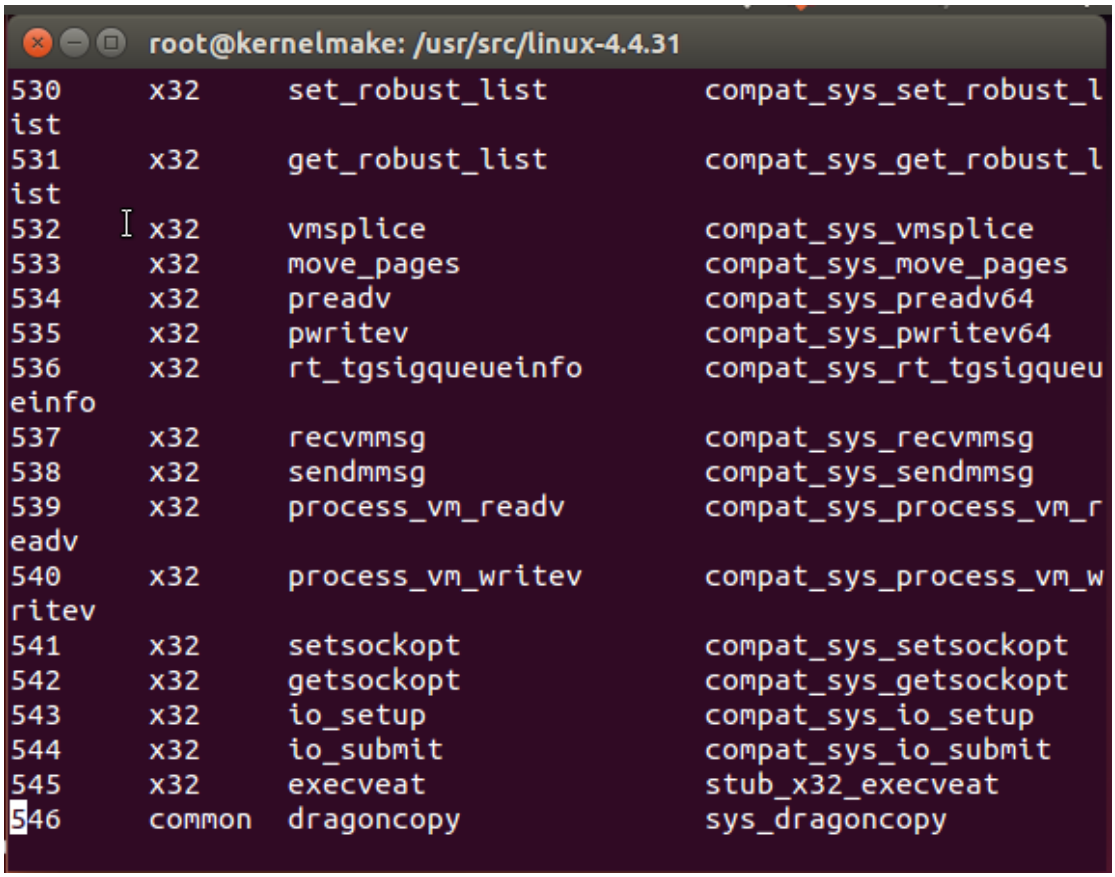
root@kernelmake: /usr/src/linux-4.4.31
530 x32 set_robust_list compat_sys_set_robust_list
531 x32 get_robust_list compat_sys_get_robust_list
532 x32 vmsplice compat_sys_vmsplice
533 x32 move_pages compat_sys_move_pages
534 x32 preadv compat_sys_preadv64
535 x32 pwritev compat_sys_pwritev64
536 x32 rt_tgsigqueueinfo compat_sys_rt_tgsigqueueinfo
537 x32 recvmmsg compat_sys_recvmmsg
538 x32 sendmmsg compat_sys_sendmmsg
539 x32 process_vm_readv compat_sys_process_vm_readv
540 x32 process_vm_writev compat_sys_process_vm_writev
541 x32 setsockopt compat_sys_setsockopt
542 x32 getsockopt compat_sys_getsockopt
543 x32 io_setup compat_sys_io_setup
544 x32 io_submit compat_sys_io_submit
545 x32 execveat stub_x32_execveat
546 common dragoncopy sys_dragoncopy

图 17 添加系统调用号

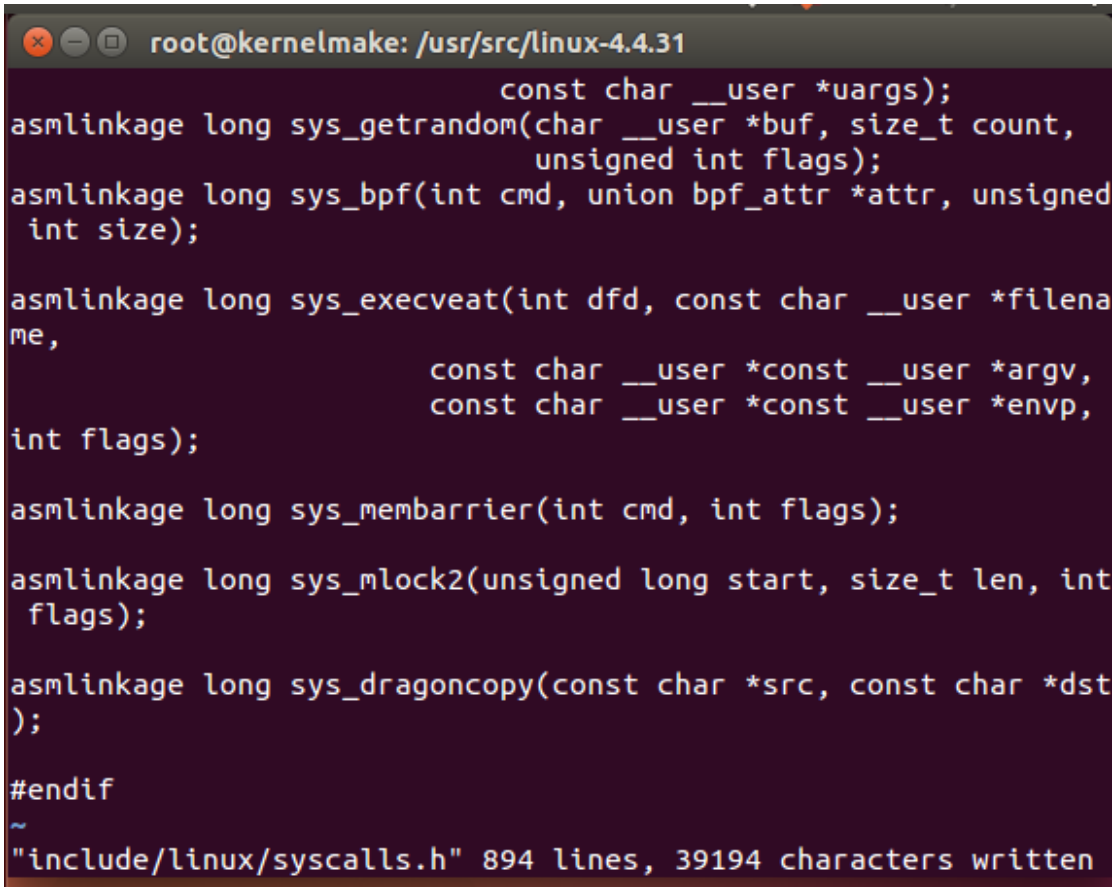


图 18 添加系统调用函数原型

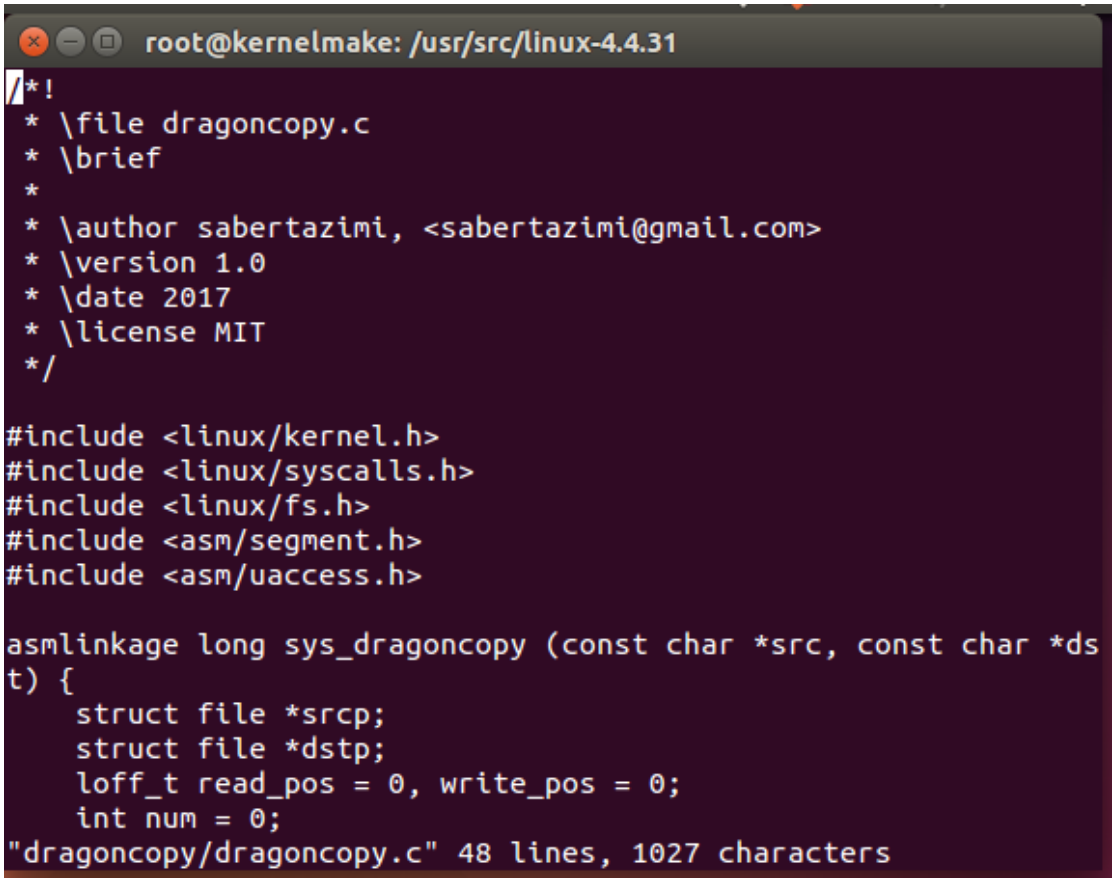


图 19 实现内核拷贝函数

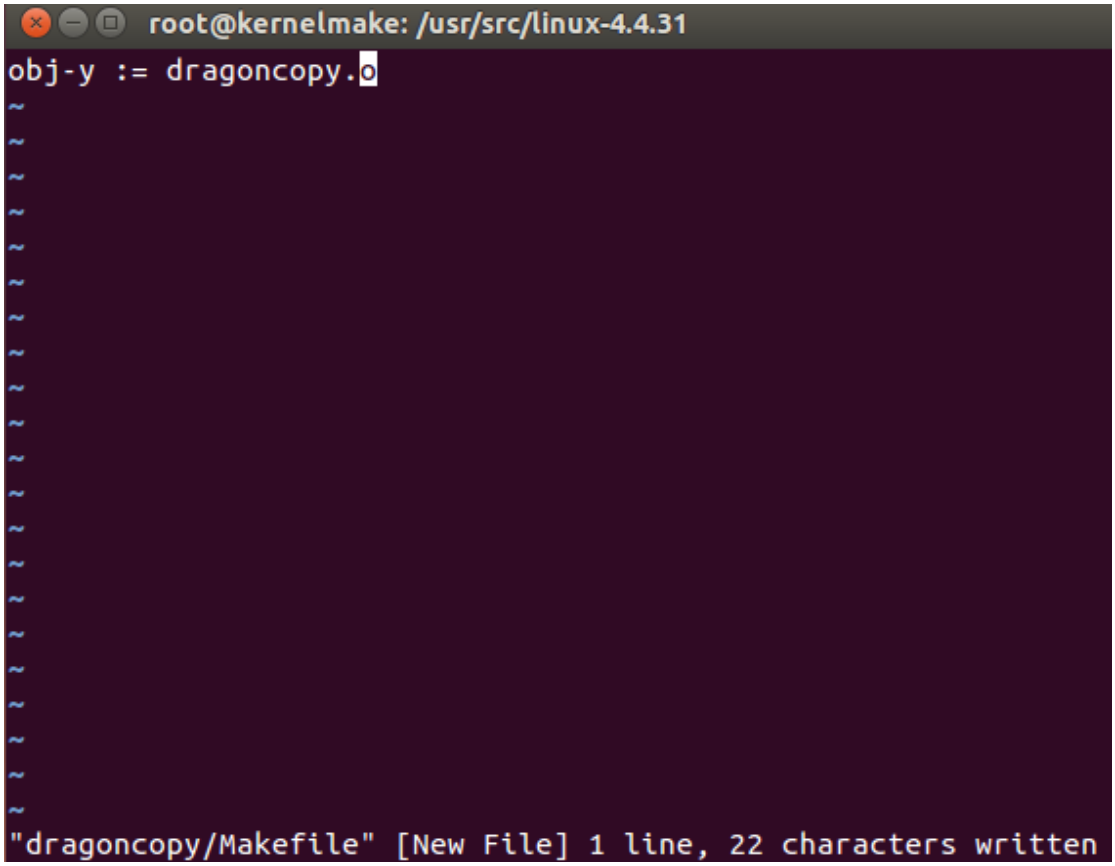


图 20 修改 Makefile 文件

7. 开始编译内核。键入 make config，进行内核编译选项的配置，如图 21 所示；然后键入 make，开始编译内核，如图 22 所示；键入 make modules_install && make install，开始安装内核，如图 23 所示；最后，更新 grub 引导文件，并修改 grub 配置文件，使得开机能够选择新内核，如图 24、图 25 所示。至此，即可完成内核的编译与安装；

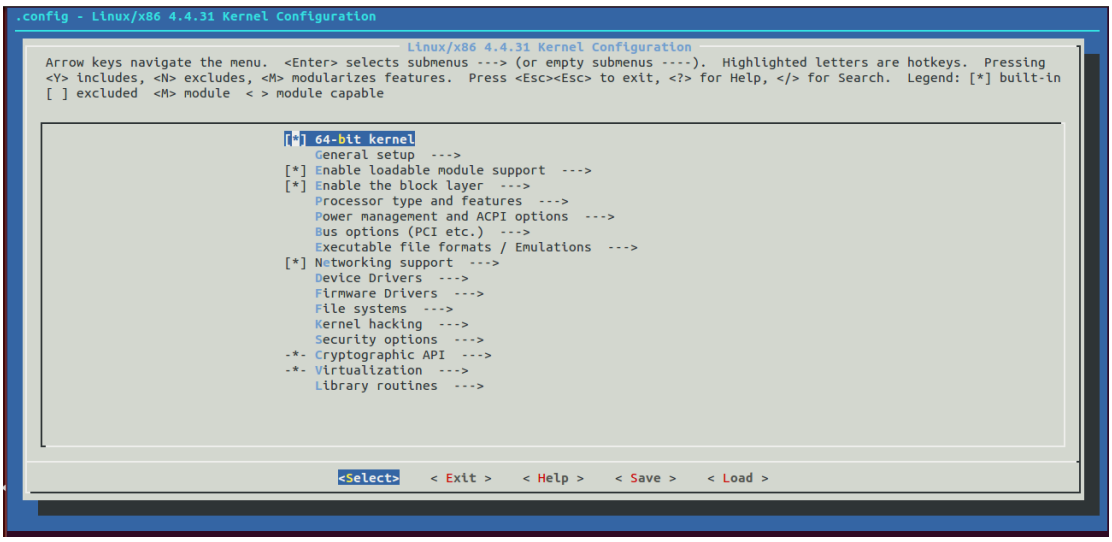


图 21 配置内核编译选项

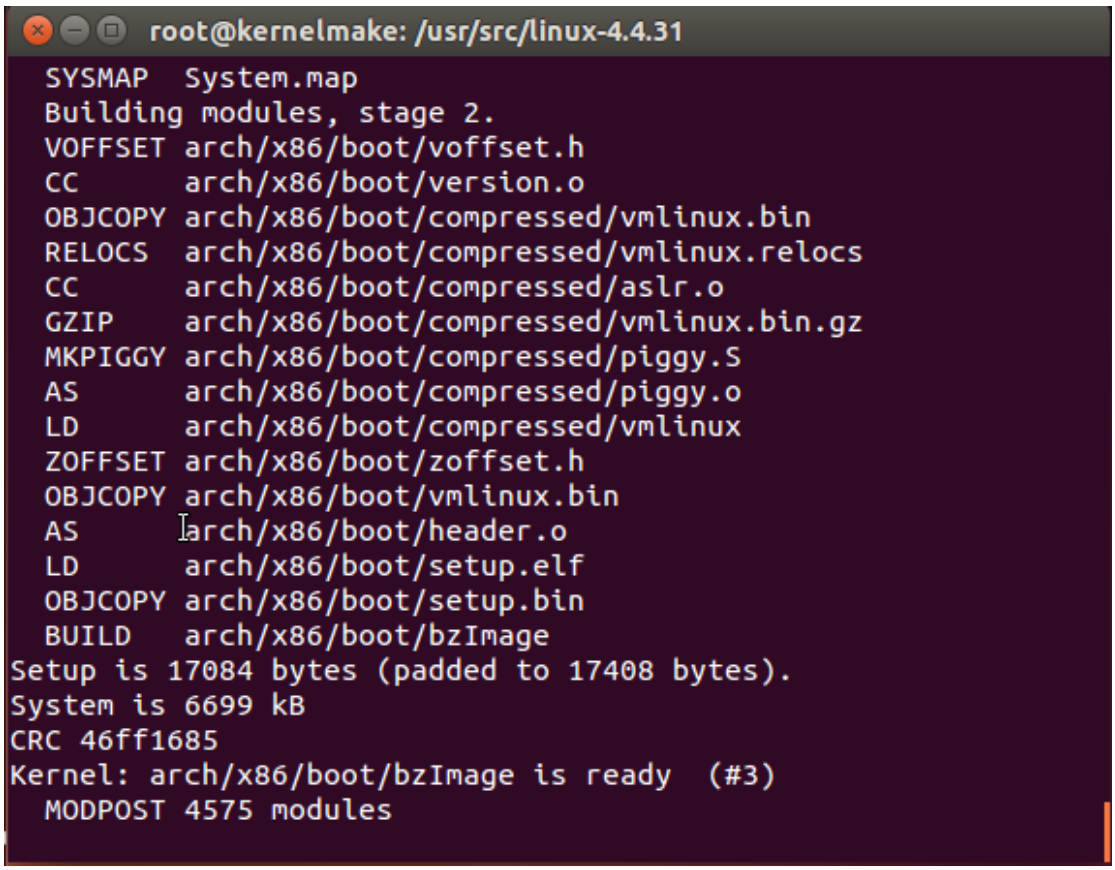


图 22 编译内核

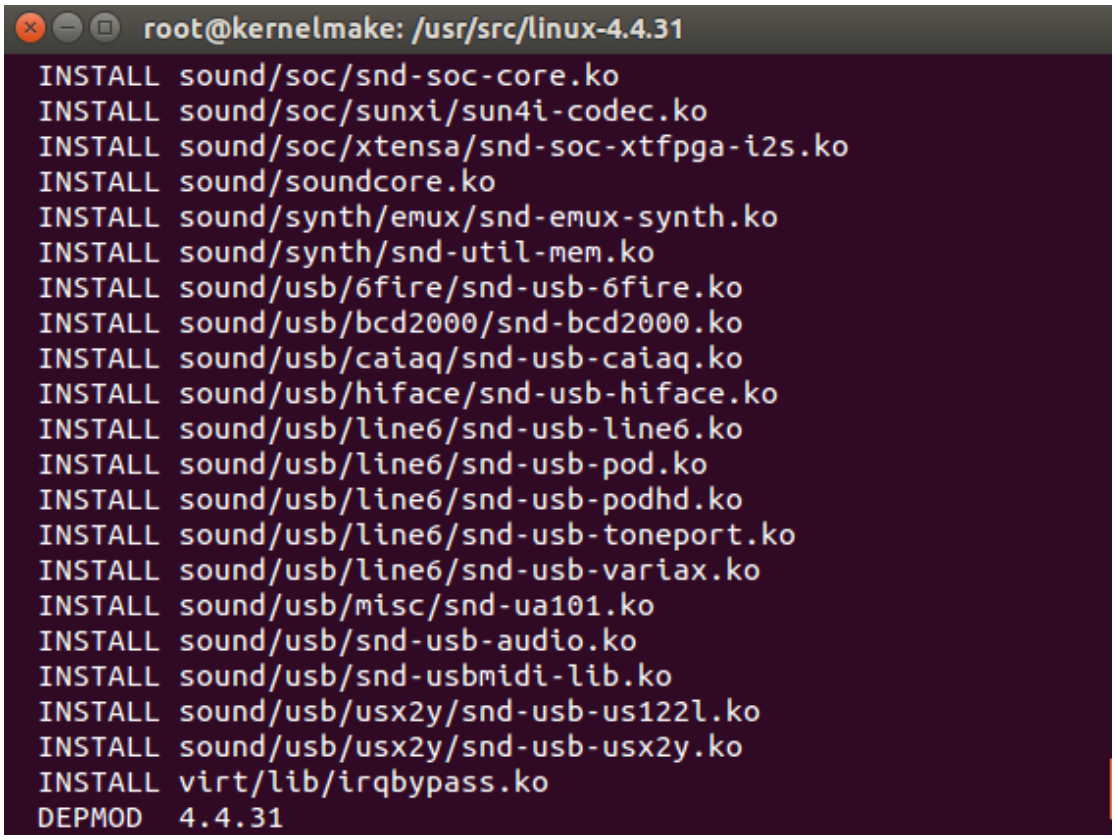


图 23 安装内核

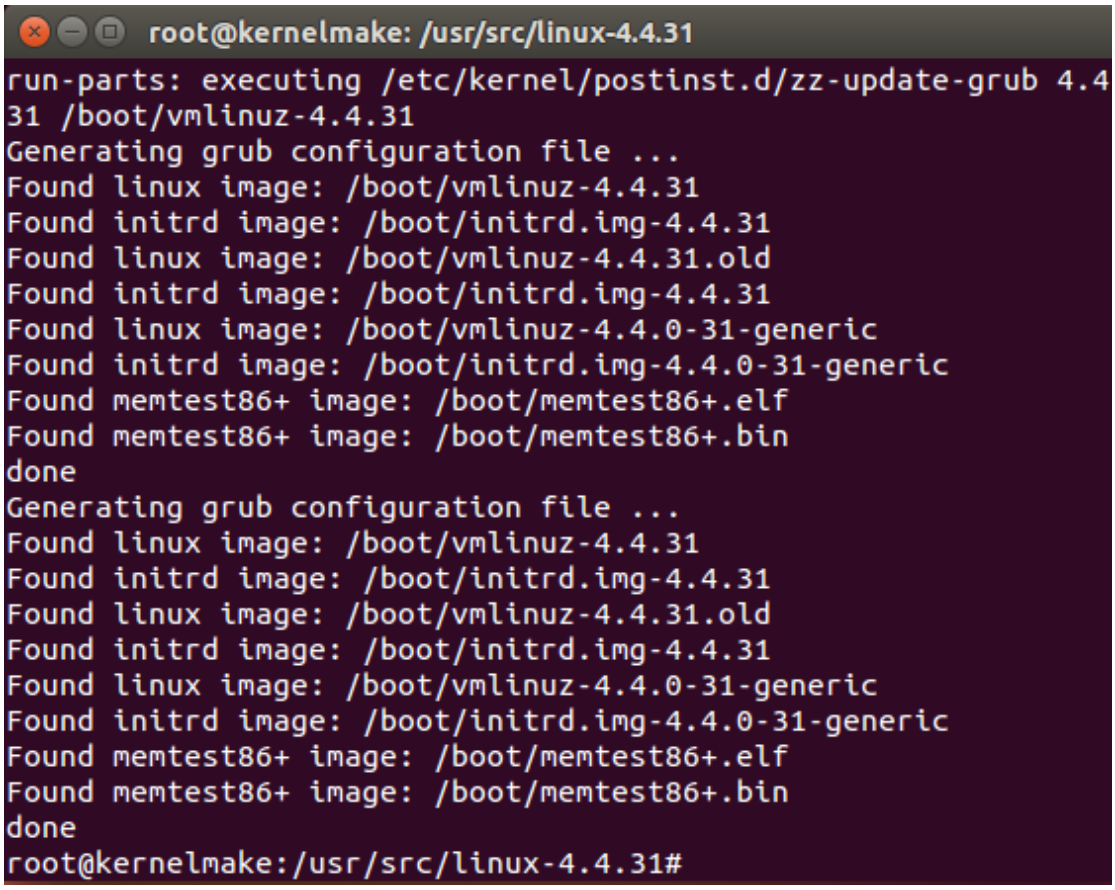


图 24 更新 grub 引导文件

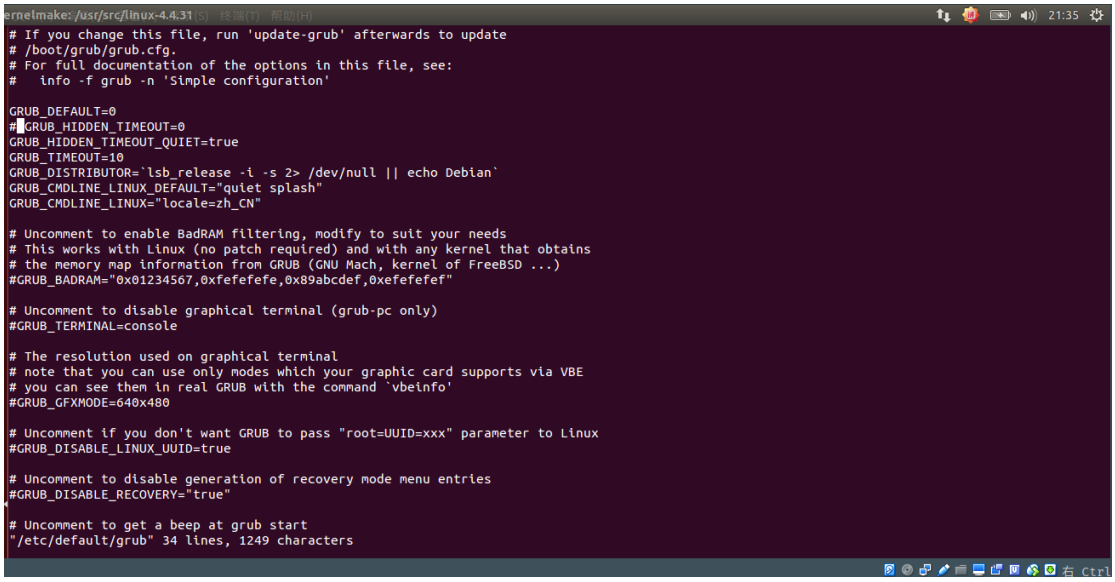


图 25 修改 grub 配置文件

8. 重启虚拟机，编写系统调用测试函数，如图 27 所示。编写 Makefile，如图 26 所示，开始进行测试。键入 make test，得到如图 28 所示结果，证明内核文件拷贝调用添加成功。再查看内核版本，如图 28 所示，内核确实进行了更新。至此，基本完成了实验二的所有任务；

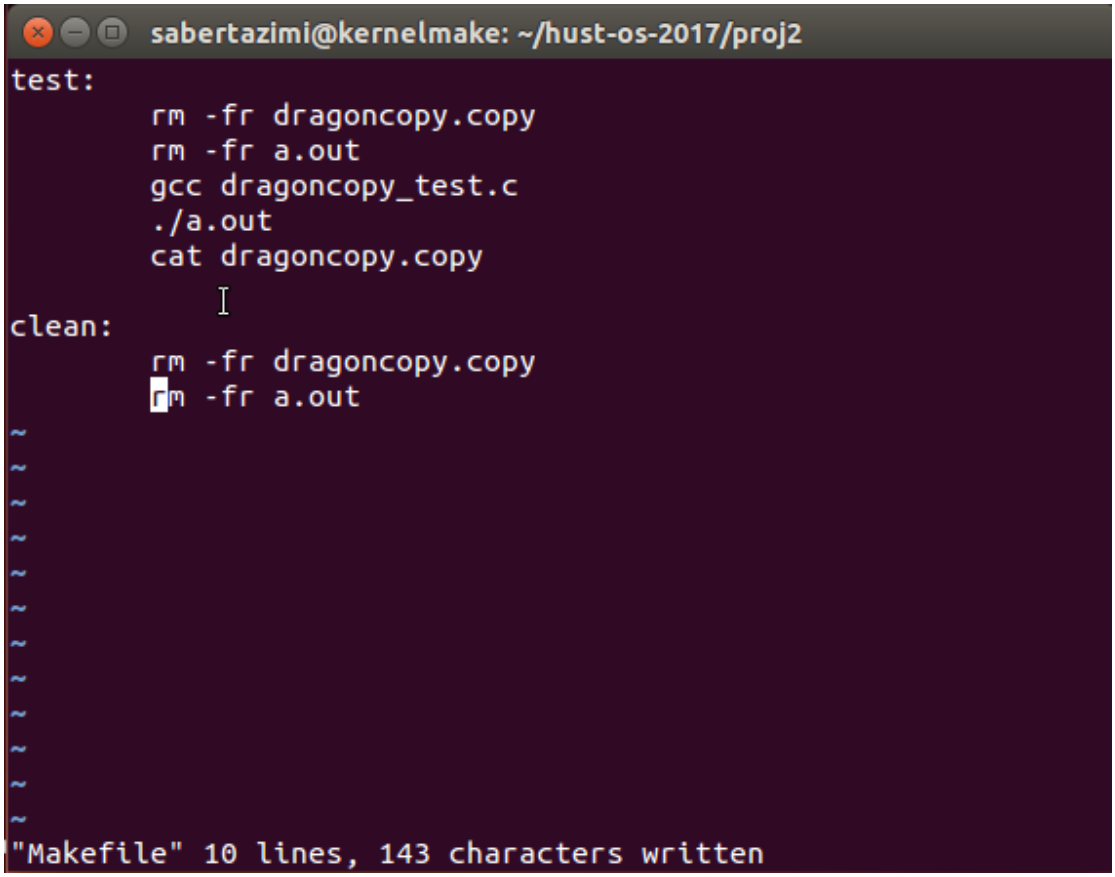


图 26 测试用 Makefile 文件

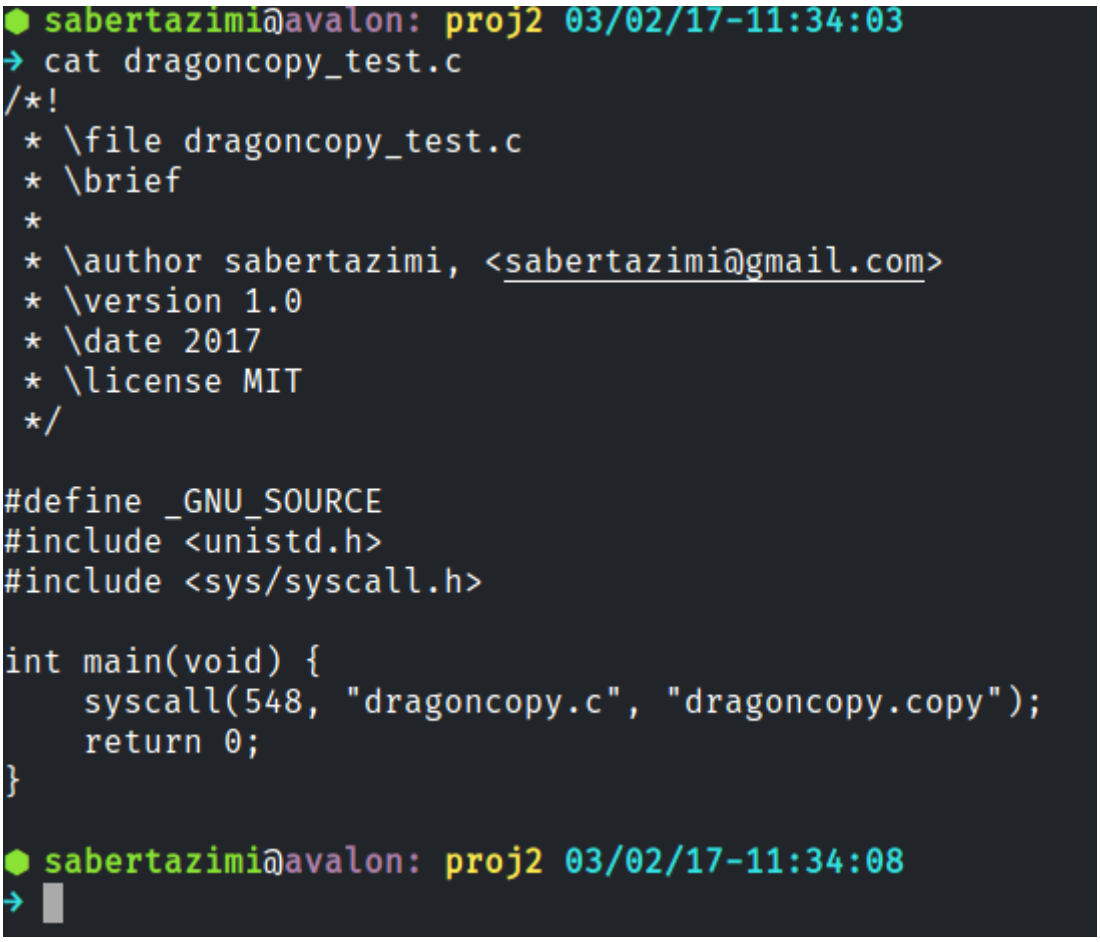


图 27 系统调用测试函数

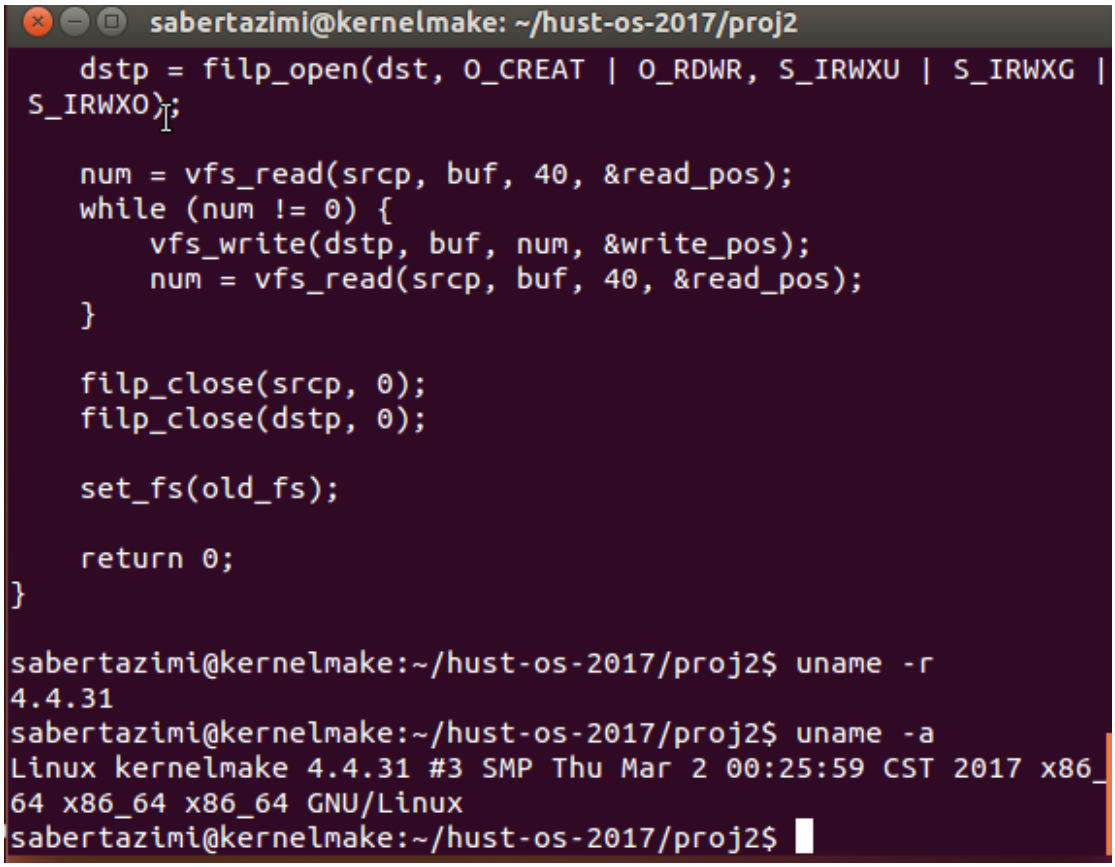


图 28 系统调用测试结果

9. 最后，在实体机上（Ubuntu 16.04 LTS）利用之前写好的内核编译脚本，快速地进行内核编译与替换，如图 29 所示。操作完成后重启计算机，结果如图 30 所示，成功地为实体机换上了最新的内核，同时也将系统调用加入至了新内核。

```
HOSTCC  scripts/mod/mk_elfconfig
MKELF   scripts/mod/elfconfig.h
HOSTCC  scripts/mod/file2alias.o
HOSTCC  scripts/mod/modpost.o
HOSTCC  scripts/mod/sumversion.o
HOSTLD  scripts/mod/modpost
HOSTCC  scripts/selinux/mdp/mdp
HOSTCC  scripts/kallsyms
HOSTCC  scripts/pnmtologo
HOSTCC  scripts/conmakehash
CC      init/main.o
CHK     include/linux/compile.h
UPD     include/linux/compile.h
CC      init/version.o
CC      init/do_mounts.o
CC      init/do_mounts_rd.o
CC      init/do_mounts_initrd.o
CC      init/do_mounts_md.o
LD      init/mounts.o
CC      init/initramfs.o
CC      init/calibrate.o
LD      init/built-in.o
HOSTCC  usr/gen_init_cpio
GEN     usr/initramfs_data.cpio
AS      usr/initramfs_data.o
LD      usr/built-in.o
LD      arch/x86/crypto/built-in.o
CC      arch/x86/kernel/process_32.o
CC      arch/x86/kernel/signal.o
AS      arch/x86/kernel/entry_32.o
CC      arch/x86/kernel/traps.o
CC      arch/x86/kernel/irq.o
CC      arch/x86/kernel/irq_32.o
CC      arch/x86/kernel/dumpstack_32.o
```

图 29 实体机上编译内核

```
● sabertazimi@avalon: proj2 03/02/17-11:34:49
→ ls
a.out  dragoncopy.c  dragoncopy.copy  dragoncopy_test.c  kernel.sh  Makefile
● sabertazimi@avalon: proj2 03/02/17-11:34:51
→ uname -a
Linux avalon 4.10.0 #1 SMP Thu Mar 2 10:14:06 CST 2017 x86_64 x86_64 x86_64 GNU/Linux
● sabertazimi@avalon: proj2 03/02/17-11:34:56
→ uname -r
4.10.0
● sabertazimi@avalon: proj2 03/02/17-11:35:26
→
```

图 30 实体机上成功换核

## 4.2 实验调试及结果

**故障：**依赖缺失

**故障现象：**当编译内核时，提示缺少 ncurses 与 openssl 相关依赖。

**解决方案：**键入 sudo apt install libncurses5-dev libssl-dev，安装相关依赖即可。前一个库用于显示图形配置界面（make config），后一个用于编译内核中的 openssl 依赖。

## 4.3 实验心得

1. 若手中没有任何资料，此次实验的难度相当大。第一，由于缺少编译内核的经验，内核编译过程中遇到一些问题不知道如何解决；第二，添加系统调用难度更大。其一，没有充分资料的情况下，无法找到合适的地方添加系统调用，其二，无法编写出能够正常使用的内核拷贝函数；

2. 在查询相关资料后，进行吸收消化，使得此次实验变得顺畅了许多。编译内核按部就班，系统调用添加位置也成功找到。唯一需要注意的便是内核拷贝函数的实现。第一，不可使用平时用过的文件操作函数，必须寻找替代的内核文件操作函数；第二，要想拷贝函数正常运行，必须临时切换内核堆栈段，一开始仅仅是按照 ppt 所给提示加上去了，并没有去想为什么要添加切换堆栈段的代码。经过仔细思考，意识到切换堆栈段是十分必要的。

# 实验三

# 1 实验目的

采用模块法添加一个字符设备驱动程序，并编写应用程序进行测试，从而掌握添加设备驱动程序的方法。

# 2 实验内容

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

# 3 实验设计

## 3.1 平台环境

硬件平台：Intel i5-4210U(1.70GHz), 8.00GB RAM, 500GB Hard Disk
软件平台：Ubuntu 16.04 LTS（编码平台），ElementaryOS （测试平台）
编程环境：Vim 7.4, Make, GCC, git, zsh

## 3.2 方案设计

1. 编写的设备驱动程序，需要实现设备的打开关闭以及读写功能，读写操作的函数功能仿照文件的 read 与 write 函数实现；
2. 通过在内存中开辟一个缓冲区来模拟设备，通过向缓冲区读取或写入信息模拟设备的读取与写入；
3. 通过 struct file_operations 结构定义设备的各种操作函数；
4. 编写各种操作函数时，需要注意函数的参数不能随意定义，应与函数原型相符；
5. 通过模块化的方式完成设备驱动的添加，在模块的初始化阶段完成设备的注册操作；
6. 编写驱动测试程序，进行相关测试。

# 4 实验调试

## 4.1 实验步骤

1. 编写字符设备驱动的 4 个基本接口，open/release/read/write，使其具有基本的读写功能；

2. 利用 register 函数以及模块函数，将字符设备驱动以模块的形式插入内核；

3. 按照 PPT 编写 Makefile，并将一系列 cp/insmod/mknod/rmmod 也写成脚本的形式加入 Makefile，已到达简化工作流的目的，如图 31 所示；

4. 键入 make，利用内核源码编译出可用内核模块 dragondev.ko，如图 32 所示；

5. 插入 dragondev 模块，查看其主、从设备号，如图 33 所示；利用设备号创建设备结点，如图 34 所示；

6. 键入 sudo make test，编译驱动测试文件，进行内核模块的测试，如图 35 所示，可以看到字符设备正常工作。

```
DEVICE_NAME=dragondev

ifneq ($(KERNELRELEASE),)
        obj-m := $(DEVICE_NAME).o
else
        KERNELDIR ?= /lib/modules/$(shell uname -r)/build
        PWD := $(shell pwd)
default:
        sudo rm -fr /usr/src/linux-headers-$(shell uname -r)/drivers/misc/$(DEVICE_NAME).c
        sudo cp -fr $(DEVICE_NAME).c /usr/src/linux-headers-$(shell uname -r)/drivers/misc
        $(MAKE) -C $(KERNELDIR) M=$(PWD) modules
endif

obj-m += $(DEVICE_NAME).o

install:
        sudo insmod $(DEVICE_NAME).ko

showdev:
        cat /proc/devices | grep $(DEVICE_NAME)
        ls -alh /dev/$(DEVICE_NAME)

showmsg:
        dmesg -c | grep Dragon

# mknod:
# mknod /dev/$(DEVICE_NAME) c $(DEVICE_NUM) 0

uninstall:
        sudo rm -fr /dev/$(DEVICE_NAME)
        sudo rmmod $(DEVICE_NAME)

clean:
        sudo rm -fr /dev/$(DEVICE_NAME)
        rm -fr *.o *.ko .*.cmd *.mod.c .tmp_versions modules.order Module.symvers

test:
        gcc -Wall -Wextra -o $(DEVICE_NAME)_test $(DEVICE_NAME)_test.c
        sudo ./$(DEVICE_NAME)_test

# vim:ft=make
#
sabertazimi@avalon:~/Work/Source/hust-os-2017/proj3$
```

图 31 字符设备驱动相关操作


图 32 生成设备模块

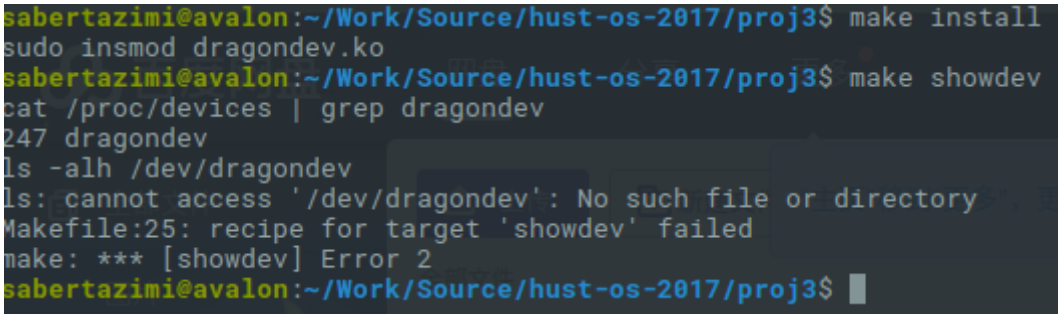


图 33 插入设备

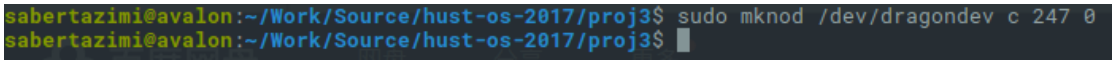


图 34 创建设备结点

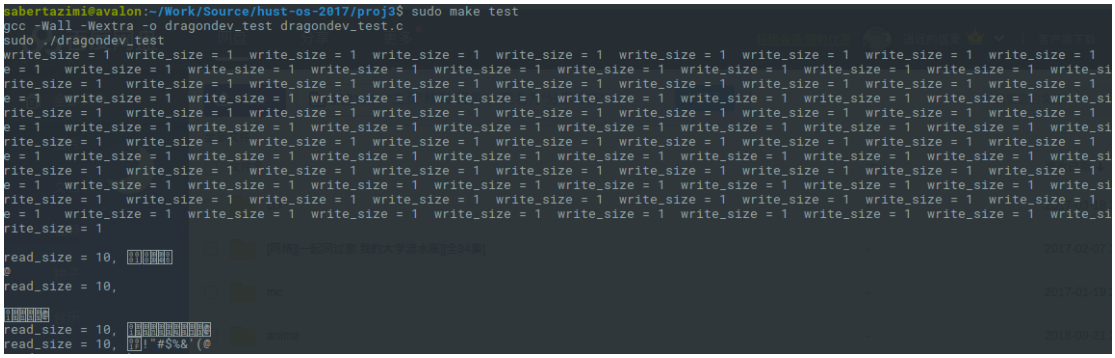


图 35 字符设备驱动测试结果

## 4.2 实验调试及结果

**故障 1：**读写操作异常

**故障现象：**当文件写操作执行完毕后，再次执行写操作，会将上次写的内容覆盖，即未在上次写操作移动的读写指针后继续写操作，而是从文件开始位置执行了写操作。

**解决方案：**编写程序时疏忽，在调用 copy_to_user 与 copy_from_user 函数时，没有在模拟的缓冲区的开始地址上加上读写指针的偏移量，导致虽然在操作时移动了读写指针，但没有使用移动后的值，每次操作仍是从缓冲区的开始位置进行操作。在调用 copy_to_user 与 copy_from_user 函数时，传入的参数不为缓冲区的开始位置 buf，而是加上读写指针偏移量的新地址，即可解决问题。

**故障 2：**驱动测试程序运行异常

**故障现象：**键入 ./dragodev_test 运行测试程序时，控制台输出 “permission denied”，如图 36

所示。


图 36 测试程序错误

**解决方案：**发现是由于 insmod/mknod 都是由 root 身份创建的，所以运行相关测试程序时，需要对设备结点进行打开、关闭、读写操作时也需 root 权限。故键入 sudo ./dragondev_test 即可正常运行测试程序，结果如图 35 所示。

## 4.3 实验心得

1. 通过此次实验，基本掌握了 Linux Module API 的使用方法，利用模块化的方式，不用修改内核源码即可完成一些内核级的工作。回想起实验二中，编译内核过程中，许多设备的驱动程序便是以模块形式加入内核中的；
2. 通过此次实验，基本掌握了简单字符设备驱动的编写，了解了 file_operations 接口定义。

# 实验四

# 1 实验目的

使用图形界面实现一个显示总的 cpu 与内存使用率，以及各个进程信息的窗口程序。理解/proc 文件的特点，并且了解 cpu 与内存利用率的计算方法。

# 2 实验内容

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

# 3 实验设计

## 3.1 平台环境

硬件平台：Intel i5-4210U(1.70GHz), 8.00GB RAM, 500GB Hard Disk
软件平台：Ubuntu 16.04 LTS（编码平台），ElementaryOS （测试平台）
编程环境：Nodejs, Electron, NPM, Atom Editor, git, zsh

## 3.2 方案设计

1. 总体 cpu 利用率的计算方法，仍然采用实验一中计算 idle 线程时间占比的方法：在/proc/stat 文件中存储了 cpu 的状态信息，如图 37 所示，通过读取文件第一行的数据，即可计算 cpu 的利用率，由于文件中存储的是从开始到当前时刻，总的 cpu 时间，所以需要统计两次 cpu 时间，相减，即为当前时间段内的 cpu 时间，cpu 利用率的计算公式为：(Total2-total1-(idle2-idle1))/(total2-total1)*100%；
2. 总体内存利用率的计算方法：在/proc/meminfo 文件中存储了内存的状态信息，如图 38 所示，从中读出 memtotal 项内从即为总的物理内存，而 memavail 为空闲内存；
3. 所有进程的进程号的获取：通过遍历/proc 中的所有目录项，目录名为数字的即为进程号；

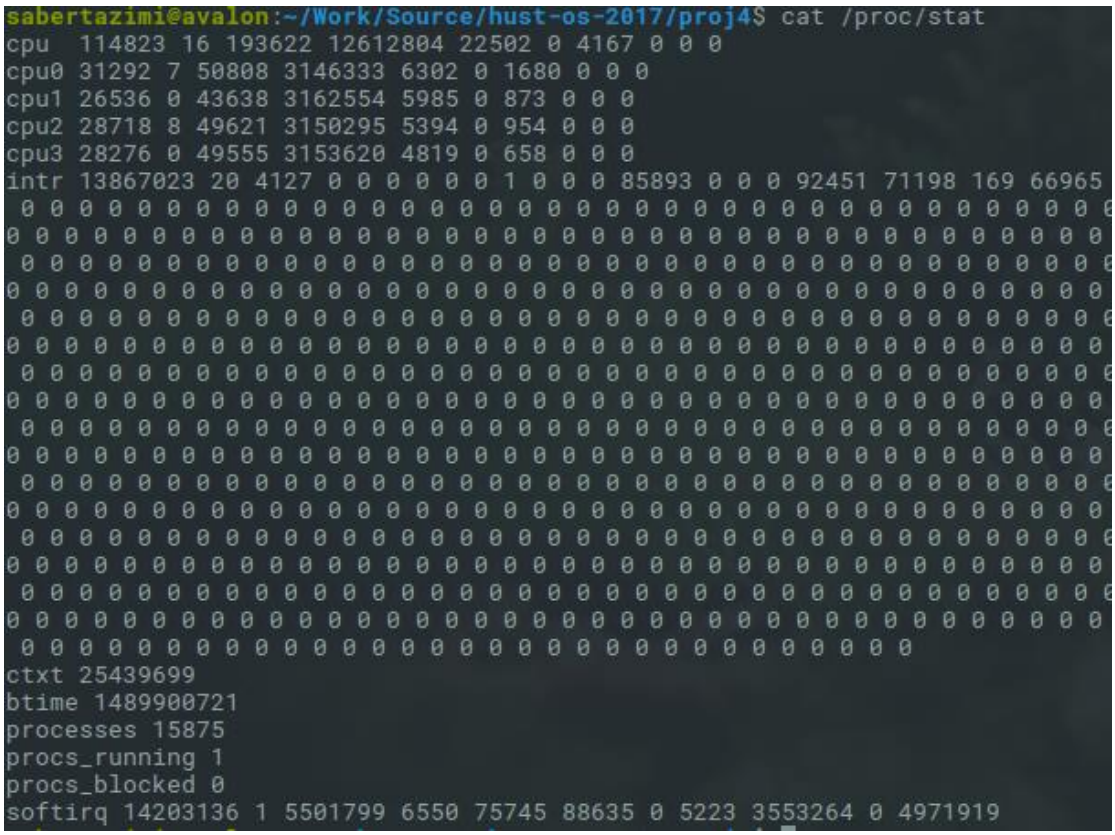


图 37 /proc/stat

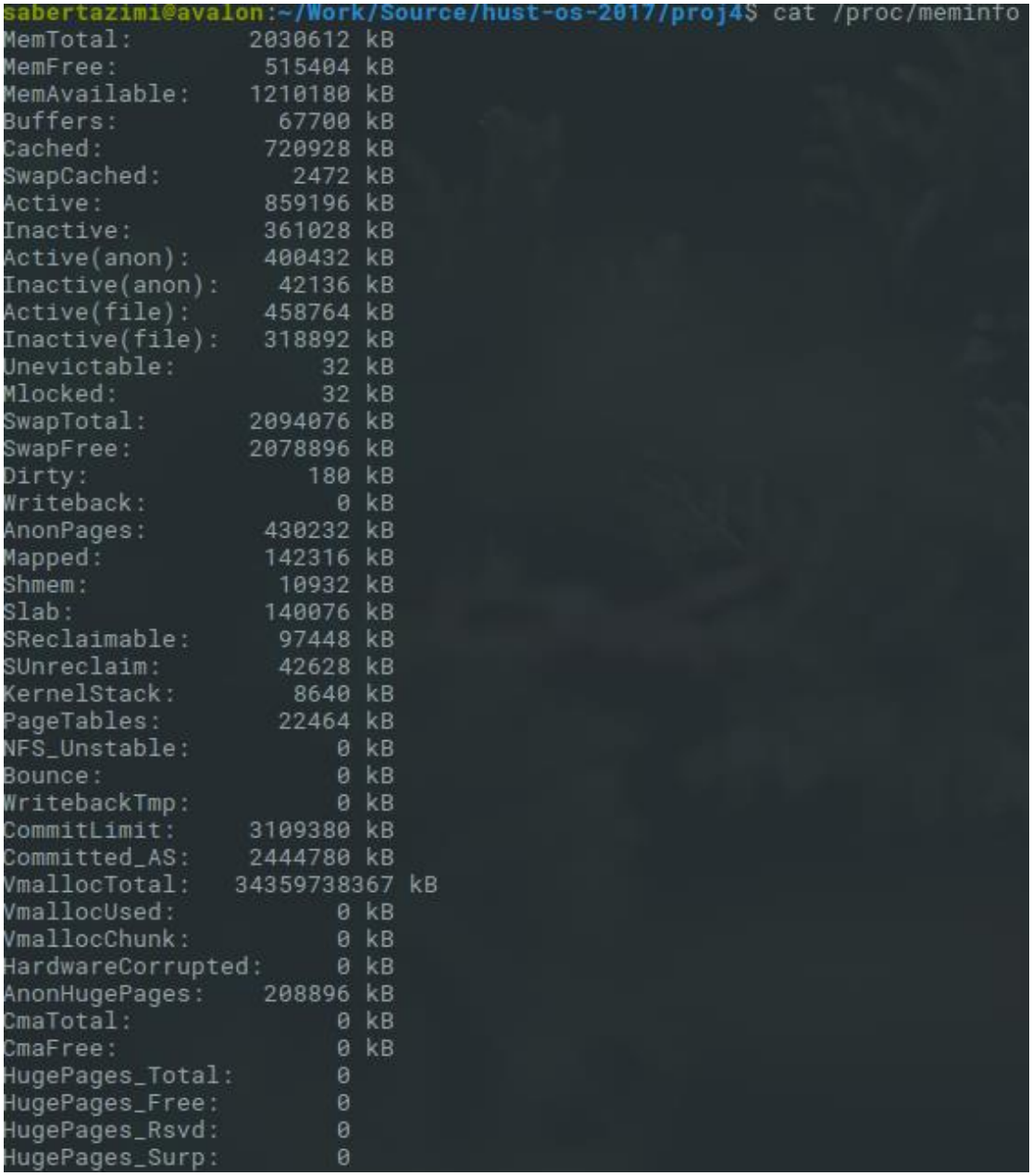


图 38 /proc/meminfo

4. 进程 cpu 利用率的计算方法：计算进程的 cpu 时间所用数据可以在/proc/进程号/stat 文件中读取，如图 39 所示，从该文件中读取程序的用户态运行时间 utime，核态运行时间 stime，将其相加即为该进程中的 cpu 时间，读取两次将结果相减即为当前时间间隔内进程占用的 cpu 时间 ptime。计算公式为：(ptime2-ptime1)/(total2-total1)*100%；

5. 进程内存占有率以及各状态信息的获取：进程相关的内存以及状态信息可从/proc/进程号/status 文件中获取，如图 40 所示。其中 VmRSS 为进程占用的物理内存大小，用其与从/proc/meminfo 文件中获取的总物理内存相比即使进程占用的内存比例。其余信息，直接渲染进 HTML 表格即可；

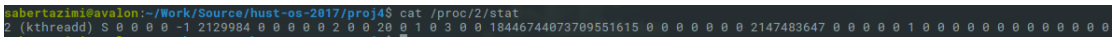


图 39 /proc/pid/stat

```
sabertazimi@avalon:~/Work/Source/hust-os-2017/proj4$ cat /proc/2/status
Name:   kthreadd
State:  S (sleeping)
Tgid:   2
Ngid:   0
Pid:    2
PPid:   0
TracerPid:      0
Uid:    0       0       0       0
Gid:    0       0       0       0
FDSize: 64
Groups:
NStgid: 2
NSpid:  2
NSpgid: 0
NSsid:  0
Threads:        1
SigQ:   0/7780
SigPnd: 0000000000000000
ShdPnd: 0000000000000000
SigBlk: 0000000000000000
SigIgn: ffffffffffffffff
SigCgt: 0000000000000000
CapInh: 0000000000000000
CapPrm: 0000003fffffffff
CapEff: 0000003fffffffff
CapBnd: 0000003fffffffff
CapAmb: 0000000000000000
Seccomp:        0
Cpus_allowed:   ffffffff,ffffffff,ffffffff,ffffffff
Cpus_allowed_list:      0-127
Mems_allowed:   00000000,00000001
Mems_allowed_list:      0
voluntary_ctxt_switches:        393
nonvoluntary_ctxt_switches:     0
```

图 40 /proc/pid/status

6. 将上述所得到的进程信息存储 Map （EcmaScript 内建数据结构），以 pid 为键，将进程信息组织为数组即可，如所示。如此一来，可以十分方便的获取目标进程的信息，且效率较高；

```
const getProcessData = (key, rawData) => {
    const totalCPU = getTotalCPU();
    const processCPU = parseInt(rawData[stat][utime]) + parseInt(rawData[stat][stime]);
    const processName = rawData[stat][comm].replace('(', '').replace(')', '');
    const processState = String.prototype.split.call(rawData[status][state], /\s+/)[2];//.replace('(', '').replace(')', '');
    let processMem = String.prototype.split.call(rawData[status][VmRSS], /\s+/);
    if (processMem[0] === 'VmRSS:') {
        processMem = parseInt(processMem[1]);
        processMem /= 1024;
    } else {
        processMem = 0;
    }

    const processData = [totalCPU, processCPU, 0, key, processName, processMem, processState];

    return processData;
}
```

图 41 获取进程信息的函数

7. 在处理上述数据过程中，有 2 点需要注意的地方。第一，所有进程的 cpu 占用时间的计算应在一次完成，即先把所有进程信息读出，存储下来，再隔一段时间进行取样，得到第二个 cpu 时间用于计算使用率。只有这样做才可使得卡顿感消失；第二，当进程被杀死后，需要将其从 Map 中删除，所以必须在 2 次取样时注意信息的同步即可；

8. 除此之外，利用 child_process 模块，可以调用 kill 命令。利用这个，可以实现杀死进

程功能，在文本框中输入进程 pid，再点击 kill 按钮，即可触发相应函数；
9. UI 绘制。Tab 实现：利用改变不同 div 的可见性，达到 Tab 切换的效果，同时只显示一个 div 即可；表格实现：利用 FlexBox 布局，实现自适应布局的表格，用于展示进程信息；除此之外，利用 CSS 样式添加一些基本的交互效果，如 hover 变色、active 变色等等，加以修饰即可。

# 4 实验调试

## 4.1 实验步骤

1. 了解 proc 伪文件系统的结构，以及不同文件存储的信息：/proc/stat 存储 cpu 信息，/proc/meminfo 存储内存信息,/proc/进程号/stat 存储进程的 cpu 信息，/proc/进程号/status 存储进程的各项状态信息。
2. 完成计算总体 cpu 利用率与内存使用率的函数：从/proc/stat 中读取 cpu 信息，计算出中 cpu 时间与空闲 cpu 时间，计算得到总的 cpu 的利用率；从/proc/meminfo 文件中读取内存使用情况的数据，从而计算出内存的占有率。
3. 完成进程状态信息获取函数：首先将 proc/ 目录下所有数字目录路径存入一个数组；再不断遍历每一个文件夹，从中获取从中获取进程占用的 cpu 时间(一段时间内的差值)，与总 cpu 时间（一段时间内的差值），相比即为进程的 cpu 占有率；打开/proc/pid/status 文件，从中获取进程的名称，状态，以及使用的内存空间量。将数据存入 Map 以供 UI 绘制；
4. 绘制 Tab ，利用 OpenTab 改变不同 Tab 的可见性；利用 FlexBox 布局实现一个表格，每行颜色交替改变，将进程信息渲染至一个巨大的 ul（unorder list）元素中即可。利用 setInterval 函数，加入定时刷新功能。最终效果如图 42 所示。

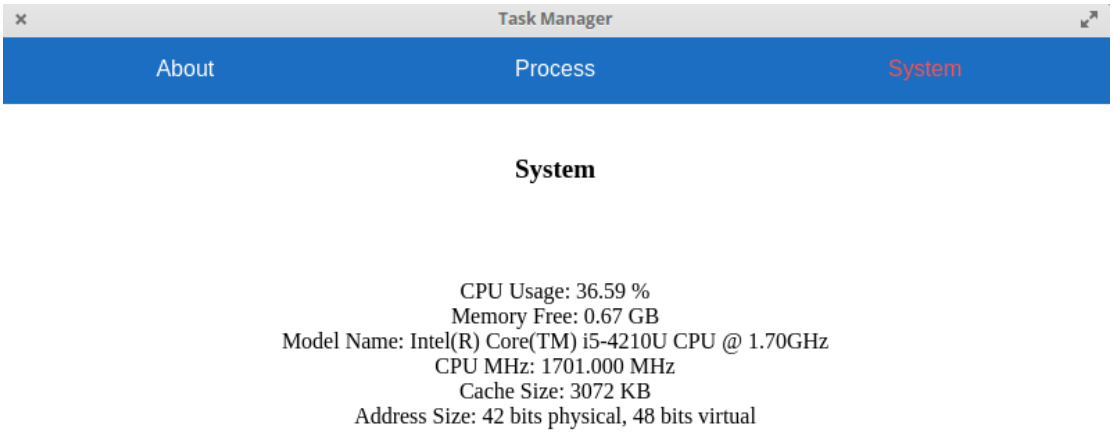


图 42 任务管理器最终效果图

## 4.2 实验调试及结果

**故障：**CPU 使用率计算异常

**故障现象：**CPU 使用率计算结果尝尝为 0/NaN。

**解决方案：**经过比对两次 utime/stime 或者/prco/stat 中数值的取样值发现，停顿 1ms 时 utime/stime 不会发生太大改变，导致计算结果为 0（分子为 0）/NaN（分母为 0）。将两次取样间隔增大至 10ms 即可解决此问题。

## 4.3 实验心得

1. 通过此次实验，基本掌握了利用/proc 伪文件系统获取 linux 系统基本运行信息的方法；
2. 通过此次实验，练习了 html/css UI 编程，并利用 nodejs 与 electron 平台，结合 Web 前沿技术，编写了一个 native app，算是一次有益的尝试。

# 实验五

# 1 实验目的

设计实现一个模拟文件系统，了解文件系统的存盘读盘，以及空间分配与目录管理功能。

# 2 实验内容

1. 研究简易文件系统，建立文件系统管理数据结构；
2. 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/写等其他功能)。

# 3 实验设计

## 3.1 平台环境

硬件平台：Intel i5-4210U(1.70GHz), 8.00GB RAM, 500GB Hard Disk

软件平台：Ubuntu 16.04 LTS（编码平台/测试平台），ElementaryOS （测试平台）

编程环境：Nodejs, Electron, NPM, Atom Editor, git, zsh

## 3.2 方案设计

1. 设计一个基于 JSON 数据格式的内存式实时文件系统。JSON 全称为 JavaScript Object Notation，从名字上就可以看出，其与 JavaScript 具有良好的交互性。利用 JSON，可以十分方便地存储树状数据结构，这恰好适合构建多级目录文件系统，一个基本的文件系统如图 43 所示；

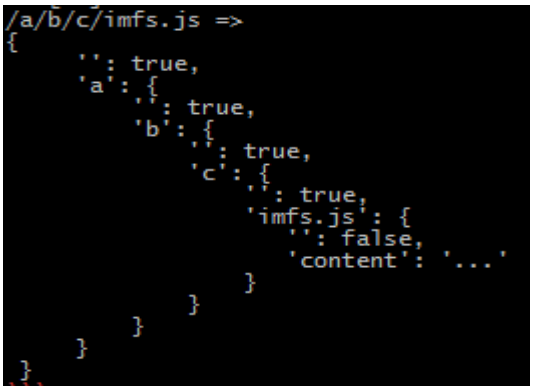


图 43 文件系统基本组织结构

2. 其中，当一个对象的‘’属性（虽然属性名称为空，但确实存在的一个属性）为 true，则表示其为目录；当一个对象的”属性为 false,则表示其为文件。利用这种规定，可以十分方便地将多级目录构建起来，当需要遍历目录时，利用一个 for 循环即可完成；

3. 在基本数据结构建立好的基础上，利用 while 语句与 case 语句编写一个 REPL（read eval print loop，读取-计算-输出循环），作为此文件系统的对外交互界面；

4. 内部的文件系统操作全由 Imfs 对象实现。基本操作包括 chdir/readdir/mkNode/rmNode/readFile/writeFile 等，还包括一些工具函数，如 isFile/isDir/isExist/resolvePath/等；

5. 实现了文件系统操作后，再利用 REPL 进行封装，即可完成对外交互，如图 44 所示。

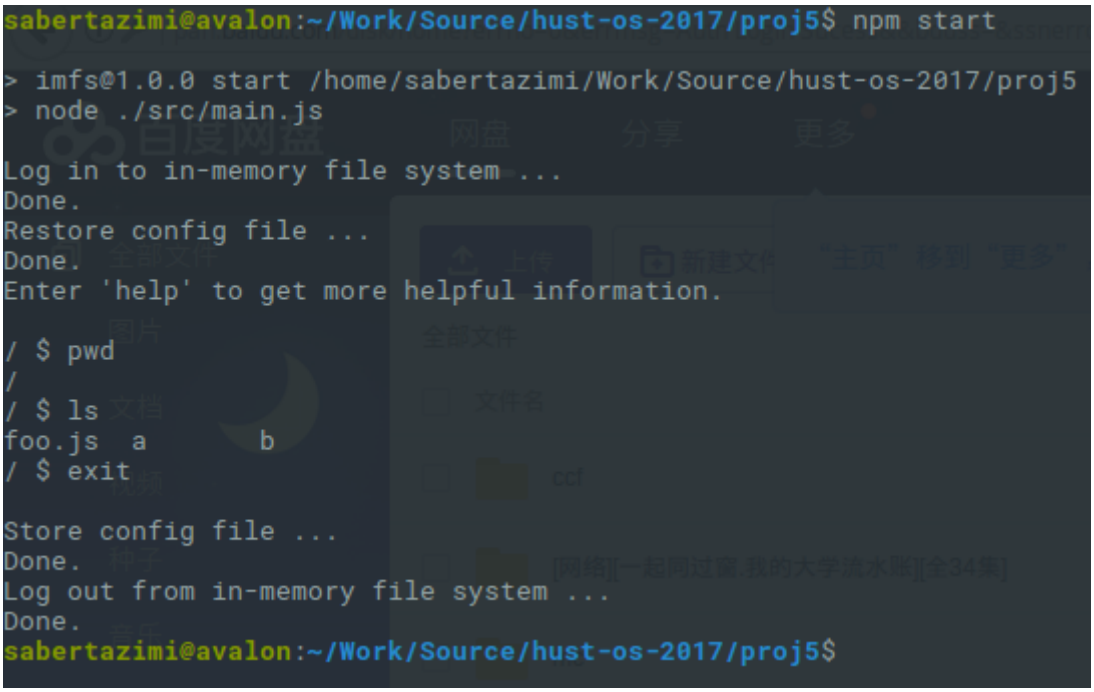


图 44 REPL 运行图

# 4 实验调试

## 4.1 实验步骤

1. 编写 Repl 类，实现交互界面；
2. 编写 Imfs 类，实现基本的工具函数，实现基本的文件系统操作函数；
3. 在实际过程中，以上 2 个类的编写是交叉进行的，逐个功能进行测试，容易找出错误所在，不容易积累错误至一个巨大的坑。先建立起基本的框架，如图 45、图 46 所示，再逐个功能地进行添加，使得编码过程比较顺利。可以看到，初版的 Repl 只有少数几个功能。而初版的 Imfs 中 mkdir 并未将创建文件的情况纳入考虑，已致于后来连接口名字都变为了 mknode。在一定的框架下，无论如何修改，都不会对整体造成太大的影响。未来有机会的话，可以完全不改动 Repl 的代码与 Imfs 的接口，重写 Imfs 的逻辑，实现更为复杂的文件系统（如加入 Inode、Block、Dirent、OpenFileTab 等复杂管理结构）

```
while (!this.exit) {
    let command = readline.question(`${this.cwd} $ `);

    command = command.split(/\s+/);

    switch (command[0]) {
        case 'cd':
            console.log('cd');
            break;
        case 'ls':
            console.log('ls');
            break;
        case 'mkdir':
            console.log('mkdir');
            break;
        case 'rm':
            console.log('rm');
            break;
        case 'touch':
            console.log('touch');
            break;
        case 'cat':
            console.log('cat');
            break;
        case 'write':
            console.log('write');
            break;
        case 'exit':
            console.log('');
            console.log('Log out from in-memory file system ...');
            this.cmd_exit();
            console.log('Done.');
            break;
        default:
            console.log('Unkown command');
            break;
    }
  }
}
```

图 45 Repl 基本框架

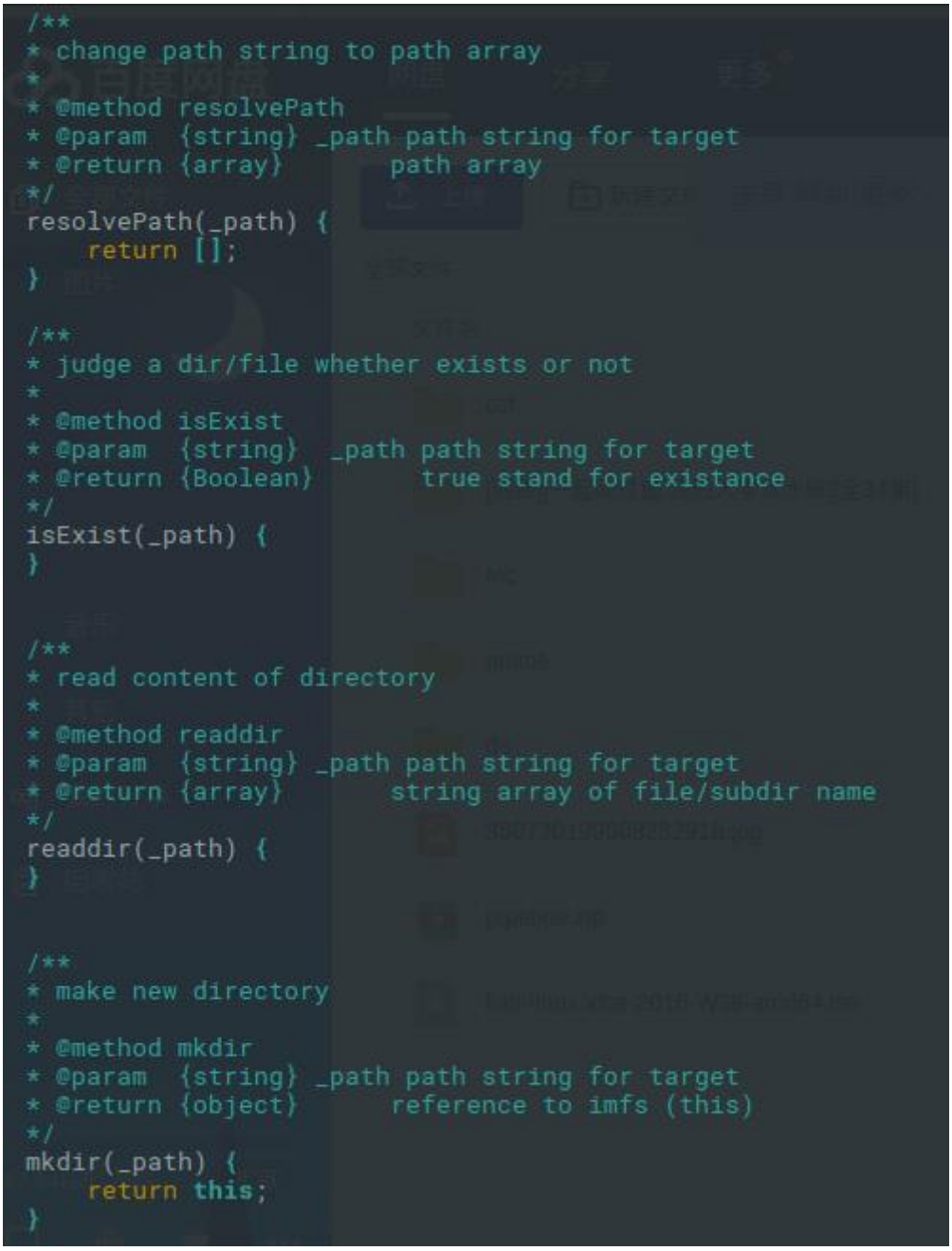


图 46 lmfs 基本框架

## 4.2 实验调试及结果

**故障 1：**cd 功能异常

**故障现象：**当进行 cd ../ 想回到根目录时，出现回不去的现象，如图 47 所示。

**解决方案：**编写 isExist 函数时，当 path 字符串从终端读取下来，利用 nodejs 内建模块处理路径后，如果为'/'目录，会被处理为空字符串。当未在 isExist 里对返回字符串进行长度判断，直接将其误判为不存在的目录，导致此现象。只要添加一个长度判断即可，当长度为 0 时，表示为根目录。后续所有类似函数，都应进行此类判断。解决了此问题，有效地避免了

后续所有功能函数中出现同样的逻辑错误。

```
● sabertazimi@avalon: proj5 03/04/17-17:31:30
→ npm start

> imfs@1.0.0 start /home/sabertazimi/Work/Source/hust-os-2017/proj5
> node ./src/main.js

Log in to in-memory file system ...
Done.

/ $ mkdir a/b/c
/ $ cd a/b/c
c $ cd ../
b $ cd ../
a $ cd ../
Error: path not exists.
a $
```

图 47 cd 异常

**故障 2：**ls 功能异常

**故障现象：**当切换路径后，调用 ls 仍然显示/目录下的文件清单，如图 48 所示。

**解决方案：**为 ls 添加新的功能，即接受多个参数，可以打印多个目录时，忘记对输入参数进行检查。当参数为 0 时，误把 argv[1]传给 ls 处理函数，导致 resolvePath 将其解析为'/'根目录，导致一直输出根目录的清单。当传入参数为 0 时，将传给 ls 处理函数的值改为 imfs.cwd 即可解决此问题，如图 49 所示。

```
● sabertazimi@avalon: proj5 03/04/17-19:15:10
→ npm start

> imfs@1.0.0 start /home/sabertazimi/Work/Source/hust-os-2017/proj5
> node ./src/main.js

Log in to in-memory file system ...
Done.
Enter 'help' to get more helpful information.

/ $ touch a/a.txt
Success: create file '/a/a.txt'.
/ $ cd a
a $ pwd
/a
a $ ls
a
a $
```

图 48 ls 异常

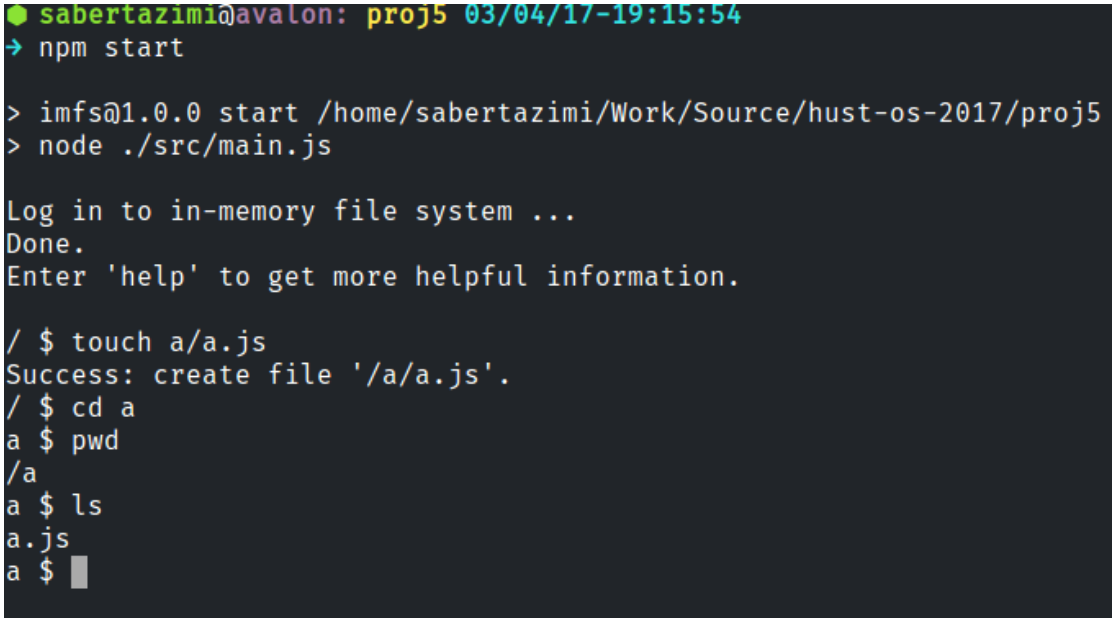


图 49 ls 正常

## 4.3 实验心得

1. 通过此次实验，基本掌握了 JSON 这一数据格式，并利用其特性较为快速地建立了树型数据结构；

2. 通过此次实验，再一次在 nodejs 平台上编写代码，利用其内建的 path、readFile、process 等内建模块，较为轻松的完成了一个简易的内存式文件系统（In Memory File System）。

# 附录　实验代码

**实验一**

**1.semaphore**

```
#ifndef SEMAPHORE_H
#define SEMAPHORE_H

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

typedef struct _semaphore_ *semaphore_t;

union semun {
        int                     val;       ///< value for SETVAL
        struct semid_ds *buf;      ///< buffer for IPC_STAT, IPC_SET
        unsigned short    *array;    ///< array for GETALL, SETALL
        struct seminfo    *__buf;    ///< buffer for IPC_INFO (Linux-specific)
};

/// \brief an OO struct for encapsulating semget/semop/semctl functions
///
/// implement semaphore with OO pattern
struct _semaphore_ {
        int semid;                              ///< id of semaphore
        union semun semun;                      ///< struct for semctl function
        struct sembuf sembuf;                   ///< struct from stand sem.h for semop function
        void (*P)(semaphore_t self);       ///< function pointer pointing to P function
        void (*V)(semaphore_t self);       ///< function pointer pointing to V function
        void (*del)(semaphore_t self);    ///< function pointer pointing to destructor
};

/// \brief constructor for semaphore struct
/// \param semval initial value of semaphore
/// \return pointer pointing to an allocated semaphore
semaphore_t semnew(key_t key, int semval);

#endif /* !SEMAPHORE_H */

#include <stdio.h>
#include <stdlib.h>
#include "semaphore/semaphore.h"

/// \brief P function
/// \param self semaphore pointer
/// \return void
static void semP(semaphore_t self);

/// \brief V function
/// \param self semaphore pointer
/// \return void
static void semV(semaphore_t self);
```

```
/// \brief destructor for semaphore
/// \param self semaphore pointer
/// \return void
static void semdel(semaphore_t self);

semaphore_t semnew(key_t key,int semval) {
	// initialize a new semphore
	semaphore_t sem = (semaphore_t)malloc(sizeof(*sem));

	// create/get a semaphore IPC
	if ((sem->semid = semget(key, 1, IPC_CREAT | IPC_EXCL | 0666)) == -1) {
		// get a exist semaphore IPC
		if ((sem->semid = semget(key, 1, 0)) == -1) {
			perror("semget error\n");
			return NULL;
		}
	} else {
		sem->semun.val = semval;			// initial value of semaphore for semctl function
		if (semctl(sem->semid, 0, SETVAL, sem->semun) < 0) {
			perror("semctl error\n");
			return NULL;
		}
	}

	sem->sembuf.sem_num = 0;			// set operation index to sem[0](all semaphores are with single demension)
	sem->sembuf.sem_flg = SEM_UNDO;		// automatically undone when the process terminates

	sem->P = semP;					// set up P function pointer
	sem->V = semV;					// set up V function pointer
	sem->del = semdel;				// set up destructor pointer

	return sem;
}

static void semP(semaphore_t self) {
	self->sembuf.sem_op = -1;
	if (semop(self->semid, &(self->sembuf), 1) < 0) {
		perror("P error\n");
	}
}

static void semV(semaphore_t self) {
	self->sembuf.sem_op = +1;
	if (semop(self->semid, &(self->sembuf), 1) < 0) {
		perror("V error\n");
	}
}

static void semdel(semaphore_t self) {
	// check to prevent multiple-destruction
	if (self->P != NULL || self->V != NULL || self->del != NULL) {
		// delete semaphore IPC
		if (semctl(self->semid, 0, IPC_RMID) < 0) {
```

```
                perror("semctl error\n");
                return ;
            }

            // set all pointer member to NULL
            self->P = NULL;
            self->V = NULL;
            self->del = NULL;

            // free heap
            free(self);
        }
}
```

**2.writebuf**

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/shm.h>
#include "semaphore/semaphore.h"

#define BUF_SIZE 10                 ///< size of shared buffer

const key_t buf_key = 233;    ///< key of shared memory to buffer
semaphore_t buf_notfull;          ///< initial value: 1, key: 1
semaphore_t buf_notempty;             ///< initial value: 0, key: 2
semaphore_t mutex;                ///< mutex for buf_map[2](number of data)

int main(int argc, char **argv) {
    FILE *fp;                       ///< src file pointer
    int buf_sid;                   ///< shm id of shared memory as buffer
    char *buf_map;                  ///< map address of shm to as buffer

    // get semaphores
    buf_notfull = semnew(1, 1);
    buf_notempty    = semnew(2, 0);
    mutex = semnew(3, 1);

    // get shm
    buf_sid = shmget(buf_key, BUF_SIZE+4, IPC_CREAT | 0666);

    if (buf_sid == -1) {
        perror("shmget error\n");
        return -1;
    }

    // attach shm
    buf_map = (char *)shmat(buf_sid, NULL, 0);

    // open src file
    if (argc <= 1) {
        if ((fp = fopen("./default.src", "a+")) == NULL) {
            perror("fopen error\n");
            return -1;
        }
    } else {
        if ((fp = fopen(argv[1], "a+")) == NULL) {
            perror("fopen error\n");
```

```
            return -1;
        }
    }

    // get data from src file to buffer
    while (1) {
        // test whether buf is full or not
        while (buf_map[2] == BUF_SIZE) {
            buf_notfull->P(buf_notfull);
        }

        // get write buf pointer
        int iwrite = buf_map[0];

        // write data to buffer, move write pointer
        char ch;
        if (fread(&ch, sizeof(char), 1, fp) <= 0) {
            // set end flag
            buf_map[3] = 1;
        }
        buf_map[4+iwrite++] = ch;
        iwrite %= BUF_SIZE;
        buf_map[0] = iwrite;

        // add number of data
        mutex->P(mutex);
        buf_map[2]++;
        mutex->V(mutex);

        fprintf(stdout, "write %c from src file to buffer... \n", ch);

        // break condition: read the end of file
        if (buf_map[3] == 1) {
            buf_notempty->V(buf_notempty);
            break;
        } else {
            buf_notempty->V(buf_notempty);
        }
    }

    // close src file
    if (fp != NULL) {
        fclose(fp);
    }

    usleep(500);

    return 0;
}
```

**3.readbuf**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/shm.h>
#include "semaphore/semaphore.h"
```

```
#define BUF_SIZE 10                ///< size of shared buffer

const key_t buf_key = 233;    ///< key of shared memory to buffer
semaphore_t buf_notfull;          ///< initial value: 1, key: 1
semaphore_t buf_notempty;             ///< initial value: 0, key: 2
semaphore_t mutex;                ///< mutex for buf_map[2](number of data)

int main(int argc, char **argv) {
	FILE *fp;                         ///< dist file pointer
	int buf_sid;                     ///< shm id of shared memory as buffer
	char *buf_map;                   ///< map address of shm to as buffer

	// get semaphores
	buf_notfull = semnew(1, 1);
	buf_notempty    = semnew(2, 0);
	mutex = semnew(3, 1);

	// get shm
	buf_sid = shmget(buf_key, BUF_SIZE+4, IPC_CREAT | 0666);

	if (buf_sid == -1) {
		perror("shmget error\n");
		return -1;
	}

	// attach shm
	buf_map = (char *)shmat(buf_sid, NULL, 0);

	// open dist file
	if (argc < 2) {
		if ((fp = fopen("./default.dist", "w+")) == NULL) {
			perror("fopen error\n");
			return -1;
		}
	} else if (argc == 2) {
		if ((fp = fopen(strcat(argv[1], ".dist"), "w+")) == NULL) {
			perror("fopen error\n");
			return -1;
		}
	} else {
		if ((fp = fopen(argv[2], "w+")) == NULL) {
			perror("fopen error\n");
			return -1;
		}
	}

	// put data from buffer to dist file
	while (1) {
		// test whether there is empty or not
		while (buf_map[2] == 0) {
			buf_notempty->P(buf_notempty);
		}

		// get buffer read pointer
		int iread = buf_map[1];

		// write to dist file from buffer, move read pointer
```

```
            char ch = buf_map[4+iread++];
            iread %= BUF_SIZE;
            buf_map[1] = iread;

            // sub number of data
            mutex->P(mutex);
            buf_map[2]--;
            mutex->V(mutex);

            // break condition: buffer empty && writebuf finish
            if (buf_map[2] <= 0 && buf_map[3] == 1 ) {
                buf_notfull->V(buf_notfull);
                break;
            } else {
                fputc(ch, fp);                      // write character into dist file
                fprintf(stdout, "read %c from buffer to dist file... \n", ch);
                buf_notfull->V(buf_notfull);
            }
        }

        // close dist file
        if (fp != NULL) {
            fclose(fp);
        }

        // detach shm
        shmdt(buf_map);

        usleep(500);

        return 0;
}
```

**4.proj1_1 main**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/shm.h>
#include <sys/wait.h>
#include "semaphore/semaphore.h"

#define BUF_SIZE 10                ///< size of shared buffer

const key_t buf_key = 233;    ///< key of shared memory to buffer
semaphore_t buf_notfull;          ///< initial value: 1, key: 1
semaphore_t buf_notempty;             ///< initial value: 0, key: 2
semaphore_t mutex;                ///< mutex for buf_map[2](number of data)

int main(int argc, char **argv) {
        int status;                    ///< wait status

        pid_t writebuf_pid;         ///< return pid of fork function
        pid_t readbuf_pid;          ///< return pid of fork function

        int buf_sid;                   ///< shm id of shared memory as buffer
        char *buf_map;                 ///< map address of shm to as buffer
```

```
        // create semaphore
        buf_notfull = semnew(1, 1);
        buf_notempty    = semnew(2, 0);
        mutex = semnew(3, 1);

        // create shm
        // buf[0] stores write pointer, buf[1] stores read pointer, buf[2] stores number of data, buf[3]
stores end flag, buf[3:BUF_SIZE+2] stores truly data
        buf_sid = shmget(buf_key, BUF_SIZE+4, IPC_CREAT | 0666);

        // get shm failed
        if (buf_sid == -1) {
                perror("shmget error\n");
                return -1;
        }

        // attach shm
        buf_map = (char *)shmat(buf_sid, NULL, 0);

        // wrie empty characters into buf_shm
        memset(buf_map, '\0', BUF_SIZE+4);

        // detach shm
        shmdt(buf_map);

        fprintf(stdout, "start copy data... \n");

        while ((writebuf_pid = fork()) == -1) ;
        if (writebuf_pid == 0) {                        // sub
                execlp("./writebuf", "writebuf", argv[1], argv[2], argv[argc], NULL);
        } else {                                                // main
                while ((readbuf_pid = fork()) == -1) ;
                if (readbuf_pid == 0) {                 // sub
                        execlp("./readbuf", "readbuf", argv[1], argv[2], argv[argc], NULL);
                } else {                                        // main
                        // wait for finish of children process
                        for (int i = 0; i < 2; i++) {
                                waitpid(-1, &status, 0);
                        }

                        fprintf(stdout, "done.\n");

                        // remove shm
                        shmctl(buf_sid, IPC_RMID, 0);

                        // remove semaphore
                        buf_notfull->del(buf_notfull);
                        buf_notempty->del(buf_notempty);
                        mutex->del(mutex);

                        return 0;
                }
        }
}
```

**5.TimeWindow**

```
#ifndef TIMEWINDOW_H
#define TIMEWINDOW_H
```

```
#include <QMainWindow>
#include <QLabel>

class TimeWindow : public QMainWindow
{
	Q_OBJECT

public:
	explicit TimeWindow(QWidget *parent = 0);
	~TimeWindow(void);

private:
	QLabel *label;
	QMainWindow &setText(const char *txt);
	const char* getTime(void);

private slots:
	void updateTime(void);
};

#endif // TIMEWINDOW_H
```

```
#include <QString>
#include <QTimer>
#include <ctime>
#include "timewindow.h"

using namespace std;

TimeWindow::TimeWindow(QWidget *parent) : QMainWindow(parent)
{
	move(QPoint(400, 300));
	label = new QLabel(this);
	label->setFixedSize(220,50);
	label->setText("Time = now");

	// using slots to implement update
	QTimer *timer = new QTimer(this);
	connect(timer, SIGNAL(timeout()), this, SLOT(updateTime()));
	timer->start(1000);
}

TimeWindow::~TimeWindow(void)
{
	delete label;
}

const char* TimeWindow::getTime(void) {
	time_t ct;
	ct = time(NULL);
	return ctime(&ct);
}

void TimeWindow::updateTime(void) {
	label->setText(QString(getTime()));
}
```

**6.SumWindow**

```
#ifndef SUMWINDOW_H
#define SUMWINDOW_H

#include <QMainWindow>
#include <QLabel>

class SumWindow : public QMainWindow
{
	Q_OBJECT

public:
	explicit SumWindow(QWidget *parent = 0);
	~SumWindow(void);

private:
	QLabel *label;
	int sum;
	int cnt;
	char *sumTxt;
	QMainWindow &setText(const char *txt);
	const char *getSum(void);

private slots:
	void updateSum(void);
};

#endif // SUMWINDOW_H

#include <QString>
#include <QTimer>
#include <cstdio>
#include "sumwindow.h"

using namespace std;

SumWindow::SumWindow(QWidget *parent) : QMainWindow(parent)
{
	move(QPoint(800, 300));
	label = new QLabel(this);
	label->setText("Sum = 0");
	label->setFixedSize(220,50);
	sum = 0;
	cnt = 0;
	sumTxt = new char[10];

	// using slots to implement update
	QTimer *timer = new QTimer(this);
	connect(timer, SIGNAL(timeout()), this, SLOT(updateSum()));
	timer->start(3000);
}

SumWindow::~SumWindow(void)
{
	delete label;
	delete sumTxt;
}
```

```
const char *SumWindow::getSum(void) {
	if (cnt <= 1000) {
		sum += (++cnt);
		sprintf(sumTxt,"%d", sum);
	}

	return sumTxt;
}

void SumWindow::updateSum(void) {
	label->setText(QString(getSum()));
}
```

**7.CPUWindow**

```
#ifndef CPUWINDOW_H
#define CPUWINDOW_H

#include <QMainWindow>
#include <QLabel>

class CPUWindow : public QMainWindow
{
	Q_OBJECT

public:
	explicit CPUWindow(QWidget *parent = 0);
	~CPUWindow(void);

private:
	QLabel *label;
	char *cpuTxt;
	QMainWindow &setText(const char *txt);
	const char* getCPU(void);

private slots:
	void updateCPU(void);
};

#endif // CPUWINDOW_H

#include <QString>
#include <QTimer>
#include <cstdio>
#include <unistd.h>
#include "cpuwindow.h"

using namespace std;

CPUWindow::CPUWindow(QWidget *parent) : QMainWindow(parent)
{
	move(QPoint(600, 300));
	label = new QLabel(this);
	label->setText("CPU Usage = 0%");
	label->setFixedSize(220,50);
	cpuTxt = new char[10];

	// using slots to implement update
```

```
	QTimer *timer = new QTimer(this);
	connect(timer, SIGNAL(timeout()), this, SLOT(updateCPU()));
	timer->start(2000);
}

CPUWindow::~CPUWindow(void)
{
	delete label;
	delete cpuTxt;
}

const char* CPUWindow::getCPU(void) {
	FILE *proc_stat = NULL;
	char buf[50];
	int total = 0, idle = 0;

	// open /proc/stat
	proc_stat = fopen("/proc/stat","r");
	if (proc_stat == NULL) {
		exit(0);
	}

	fscanf(proc_stat, "%s", buf);

	for(int i = 0; i < 10; i++) {
		fscanf(proc_stat, "%s", buf);
		total += atoi(buf);

		// column 4: idle spare time
		if (i == 3) {
			idle = atoi(buf);
		}
	}

	fseek(proc_stat, 0, SEEK_SET);	// reset seek pointer to 0 (read file again)
	sleep(1);	// update cpu stat
	fscanf(proc_stat, "%s", buf);

	for (int i = 0; i < 10; i++) {
		fscanf(proc_stat, "%s", buf);
		total -= atoi(buf);

		// column 4: idle spare time
		if (i == 3) {
			idle-=atoi(buf);
		}
	}

	fclose(proc_stat);

	sprintf(cpuTxt, "CPU Usage = %.2f%%\n", 100.0*((float)(idle-total)) / ((float)(-total)));

	return cpuTxt;
}

void CPUWindow::updateCPU(void) {
	label->setText(QString(getCPU()));
```

```
}
```

**8.Qt main**

```
#include <QApplication>
#include <unistd.h>
#include <sys/wait.h>
#include "timewindow.h"
#include "cpuwindow.h"
#include "sumwindow.h"

int main(int argc, char *argv[])
{
	pid_t tw_t, cw_t, sw_t;

	while ((tw_t = fork()) == -1);

	if (tw_t == 0) {
		QApplication ta(argc, argv);
		TimeWindow tw;
		tw.show();
		ta.exec();
	} else {
		while ((cw_t = fork()) == -1);

		if (cw_t == 0) {
			QApplication tc(argc, argv);
			CPUWindow cw;
			cw.show();
			tc.exec();
		} else {
			while ((sw_t = fork()) == -1);

			if (sw_t == 0) {
				QApplication ts(argc, argv);
				SumWindow sw;
				sw.show();
				ts.exec();
			} else {
				for (int i = 0; i < 3; i++) {
					waitpid(-1, NULL, 0);
				}
			}
		}
	}

}
```

**实验二**

**1.kernel.sh**

```
# exec under /usr/src

# cd /usr/src
# su root
# wget https://www.kernel.org/pub/linux/kernel/v4.x/linux-4.4.31.tar.gz
# tar -zxvf linux-4.4.31.tar.gz
# cd linux-4.4.31/

apt-get install libncurses5-dev libssl-dev
```

```
make mrproper
make clean

make menuconfig              # 编译新内核的配置文件
make bzImage -j 4 && make modules -j 4 && make modules_install -j 4 && make install -j 4

# comments GRUB_HIDDEN_OUT = 0 out in /etc/default/grub
# update-grub                # 更新引导文件
# reboot                     # 重新启动
```

**2.dragoncopy.c 系统调用实现**

```
#include <linux/kernel.h>
#include <linux/syscalls.h>
#include <linux/fs.h>
#include <asm/segment.h>
#include <asm/uaccess.h>

asmlinkage long sys_dragoncopy (const char *src, const char *dst) {
	struct file *srcp;
	struct file *dstp;
	loff_t read_pos = 0, write_pos = 0;
	int num = 0;
	char buf[50];

	mm_segment_t old_fs = get_fs();
	set_fs(KERNEL_DS);

	srcp = filp_open(src, O_CREAT | O_RDWR, S_IRWXU | S_IRWXG | S_IRWXO);
	if (IS_ERR(srcp)) {
		printk("Dragon copy open file failed.\n");
		return -1;
	}

	dstp = filp_open(dst, O_CREAT | O_RDWR, S_IRWXU | S_IRWXG | S_IRWXO);

	num = vfs_read(srcp, buf, 40, &read_pos);
	while (num != 0) {
		vfs_write(dstp, buf, num, &write_pos);
		num = vfs_read(srcp, buf, 40, &read_pos);
	}

	filp_close(srcp, 0);
	filp_close(dstp, 0);

	set_fs(old_fs);

	return 0;
}
```

**3.dragoncopy_test.c 系统调用测试**

```
#define _GNU_SOURCE
#include <unistd.h>
#include <sys/syscall.h>

int main(void) {
	syscall(548, "dragoncopy.c", "dragoncopy.copy");
	return 0;
}
```

**4.测试用 Makefile**

```
test:
	rm -fr dragoncopy.copy
	rm -fr a.out
	gcc dragoncopy_test.c
	./a.out
	cat dragoncopy.copy

clean:
	rm -fr dragoncopy.copy
	rm -fr a.out
```

**实验三**

**1.dragondev.c 驱动程序**

```
#include <linux/fs.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <asm/uaccess.h>

// 处理版本问题 CONFIG_MODVERSIONS
#if CONFIG_MODVERSIONS == 1
#define MODVERSIONS
#include "linux/version.h"
#endif

#define BUF_SIZE 1000      ///< buffer size
char buf[BUF_SIZE];          ///< device charaters buffer
int dev_num = 0;               ///< device number
int buf_size = 0;             ///< buffer current size containing characters
int seek_pos = 0;             ///< current

static int dragondev_open(struct inode *inode, struct file *filp) {
	seek_pos = 0;
	printk("Dragon device open success.\n");
	return 0;
}

static int dragondev_release (struct inode *inode, struct file *filp) {
	seek_pos = 0;
	printk("Dragon device release success.\n");
	return 0;
}

static ssize_t dragondev_read (struct file *filp, char __user *target , size_t tsize, loff_t *offset) {
	size_t read_size = buf_size - seek_pos;
	read_size = read_size < tsize ? read_size : tsize;

	if (!copy_to_user((char *)target, buf+seek_pos, read_size)) {
		seek_pos += read_size;
		printk("Dragon device read success.\n");
		return read_size;
	} else {
		printk("Dragon device read failed.\n");
		return -1;
	}
}

static ssize_t dragondev_write (struct file *filp,const char __user *target , size_t tsize, loff_t
*offset) {
```

```
        size_t write_size = BUF_SIZE - seek_pos;
        write_size = write_size < tsize ? write_size : tsize;

        if(!copy_from_user((char *)buf+seek_pos, target, write_size)) {
                seek_pos += write_size;
                buf_size += write_size;
                printk("Dragon device write success.\n");
                return write_size;
        } else {
                printk("Dragon device write failed.\n");
                return -1;
        }
}

static const struct file_operations dragondev_fops = {
        .owner = THIS_MODULE,
        .open = dragondev_open,
        .read = dragondev_read,
        .write = dragondev_write,
        .release = dragondev_release
};


int init_module(void) {
        if ((dev_num = register_chrdev(0, "dragondev", &dragondev_fops)) < 0) {
                printk("Dragon device register failed.\n");
        } else {
                printk("Dragon device register success.\n");
        }

        return 0;
}

void cleanup_module(void) {
        unregister_chrdev(dev_num, "dragondev");
        printk("Dragon device unregister success.\n");
}

MODULE_AUTHOR("sabertazimi");
MODULE_DESCRIPTION("dragon device");
MODULE_LICENSE("MIT");
MODULE_VERSION("V1.0");
```

**2.dragondev_test.c 驱动测试程序**

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int main(void) {
        int dragondev;
        char buf[11];

        dragondev = open("/dev/dragondev", O_RDWR);
        if (dragondev == -1) {
                perror("Test failed.\n");
```

```
        exit(0);
    }

    for (int i = 1; i <= 127; i++) {
        buf[0] = i;
        printf("write_size = %d\t", (int)write(dragondev, buf, 1));
    }

    close(dragondev);
    printf("\n\n");

    dragondev = open("/dev/dragondev", O_RDWR);
    if (dragondev == -1) {
        perror("Test failed.\n");
        exit(0);
    }

    for (int i = 1; i <= 12; i++) {
        printf("read_size = %d, ", (int)read(dragondev, buf, 10));
        printf("%s\n", buf);
    }

    close(dragondev);

    return 0;
}
```

**3.Makefile**

```
DEVICE_NAME=dragondev

ifneq ($(KERNELRELEASE),)
    obj-m := $(DEVICE_NAME).o
else
    KERNELDIR ?= /lib/modules/$(shell uname -r)/build
    PWD := $(shell pwd)
default:
    sudo rm -fr /usr/src/linux-4.10/drivers/misc/$(DEVICE_NAME).c
    sudo cp -fr $(DEVICE_NAME).c /usr/src/linux-4.10/drivers/misc
    $(MAKE) -C $(KERNELDIR) M=$(PWD) modules
endif

obj-m += $(DEVICE_NAME).o

install:
    sudo insmod $(DEVICE_NAME).ko

showdev:
    cat /proc/devices | grep $(DEVICE_NAME)
    ls -alh /dev/$(DEVICE_NAME)

showmsg:
    dmesg -c | grep Dragon

# mknod:
# mknod /dev/$(DEVICE_NAME) c $(DEVICE_NUM) 0

uninstall:
    sudo rm -fr /dev/$(DEVICE_NAME)
```

```
	sudo rmmod $(DEVICE_NAME)

clean:
	sudo rm -fr /dev/$(DEVICE_NAME)
	rm -fr *.o *.ko .*.cmd *.mod.c .tmp_versions modules.order Module.symvers

test:
	gcc -Wall -Wextra -o $(DEVICE_NAME)_test $(DEVICE_NAME)_test.c
	sudo ./$(DEVICE_NAME)_test
```

**实验四**

**1.package.json 依赖管理文件**

```
{
  "name": "electron-taskmanager",
  "version": "1.0.0",
  "description": "A task manager based on Electron",
  "main": "./js/main.js",
  "scripts": {
    "start": "./node_modules/.bin/electron ."
  },
  "repository": "https://github.com/sabertazimi/hust-os-2017",
  "keywords": [
    "Electron",
    "Task Manager"
  ],
  "author": "sabertazimi",
  "license": "MIT",
  "devDependencies": {
    "electron": "^1.6.1"
  },
  "dependencies": {}
}
```

**2.index.html 基本页面结构**

```
<!DOCTYPE html>
<html>

	<head>
		<meta charset="UTF-8">
		<title>Task Manager</title>
		<link rel="stylesheet" type="text/css" href="./css/Tab.css">
		<link rel="stylesheet" type="text/css" href="./css/Flex.css">
		<link rel="stylesheet" type="text/css" href="./css/Process.css">
	</head>

	<body>
		<ul class="tab">
			<li><a href="javascript:void(0)" onclick="openTab(event,
'About')">About</a></li>
			<li><a href="javascript:void(0)" onclick="openTab(event,
'Process')">Process</a></li>
			<li><a href="javascript:void(0)" onclick="openTab(event,
'System')">System</a></li>
		</ul>

		<div id="About" class="tabcontent active">
			<section class="flex-container">
				<h3>About</h3>
				<p>author: sabertazimi</p>
```

```
                <p>email: sabertazimi@gmail.com</p>
            </section>
        </div>

        <div id="Process" class="tabcontent">
            <section>
                <input type="text" name="pid" class="kill-pid" id="kill-pid"
placehold="input pid to kill">
                <input type="button" name="kill" class="kill-button" value="kill"
onclick="killProcess()">
            </section>
            <div class="process-table">
                <ul class="process-list">
                </ul>
            </div>
        </div>

        <div id="System" class="tabcontent">
            <section class="flex-container">
                <h3>System</h3>
                <p id="cpu-usage">CPU Usage</p>
                <p id="mem-usage">Memory Usage</p>
                <p id="model-name">Model Name</p>
                <p id="cpu-mhz">CPU MHz</p>
                <p id="cache-size">Cache Size</p>
                <p id="addr-size">Addr Size</p>
            </section>
        </div>

        <script src="./js/view.js"></script>
    </body>
</html>
```

**3.Flex.css FlexBox 布局格式**

```
.flex-container {
    display: flex;
    flex-direction: column;
    align-items: center;
    text-align: left;
}

.flex-container h3 {
    padding-top: 5vh;
    flex: 0 0 20vh;
}

.flex-container p {
    flex: 1 0 auto;
}
```

**4.Tab.CSS 标签页样式**

```
* {
    box-sizing: border-box;
    margin: 0;
    padding: 0;
}

ul.tab {
    list-style-type: none;
```

```
        background-color: #1b6ec2;
        display: flex;
        justify-content: space-around;
}

ul.tab li {
        display: flex;
        flex-direction: column;
        justify-content: center;
        align-items: center;

        min-height: 8vh;
        min-width: 8vw;

        border-bottom: 2px solid transparent;
        transition: border-bottom 0.3s linear;
}

ul.tab li a {
        color: white;
        font-family: sans-serif;
        text-decoration: none;
        transition: color 0.3s linear;
}

ul.tab li:hover {
        border-bottom: 2px solid #fa5252;
}

ul.tab a:hover {
        color: #fa5252;
}

ul.tab a:focus {
        color: #fa5252;
}

.tabcontent {
        display:none;
        padding: 6px 12px;
}

.active {
        display: block;
}
```

**5.Process.css 进程信息表格样式表**

```
div.process-table {
        display: flex;
        flex-direction: column;
}

ul.process-list {
        list-style-type: none;
        display: flex;
        justify-content: space-between;
}
```

```
ul.process-list li {
	min-height: 5vh;
	/*min-width: 30%;*/
	/*max-width: 35%;*/
	flex: 1;
}

ul.process-list li:last-child {
	text-align: right;
}

ul.process-list:nth-child(2n) {
	background-color: #f2f2f2;
}

.kill-pid {
	position: fixed;
	bottom: 2vh;
	right: 10vw;
	height: 25px;
}

.kill-button {
	position: fixed;
	bottom: 2vh;
	right: 10vw;

	backgroud-color: inherit;
	border: 1px solid #f44336;
	color: #f44336;
	padding: 5px 12px;
	text-align: center;
	display: inline-block;
	cursor: pointer;
}
```

**6.CPU.js CPU 信息获取**

```
const fs = require('fs');
const path = require('path');

const readCPUUsage = (coreNumber, sleepTime = 100) => {
	let total = 0;
	let idle = 0;
	let usage = 0.0;
	let data = '';
	let stat = '';
	let cpustat = '';

	data = fs.readFileSync('/proc/stat', 'utf8');
	stat = String.prototype.split.call(data, '\n');
	cpustat = String.prototype.split.call(stat[coreNumber], /\s+/);

	for (let i = 1; i < 10; i++) {
		total += parseInt(cpustat[i], 10);

		// column 4: idle spare time
		if (i == 4) {
			idle = parseInt(cpustat[i], 10);
```

```
            }
        }

        // sleep
        const date = new Date();
        let curDate = null;
        do {
            curDate = new Date();
        } while (curDate - date < sleepTime);

        data = fs.readFileSync('/proc/stat', 'utf8');
        stat = String.prototype.split.call(data, '\n');
        cpustat = String.prototype.split.call(stat[coreNumber], /\s+/);

        for (let i = 1; i < 10; i++) {
            total -= parseInt(cpustat[i], 10);

            // column 4: idle spare time
            if (i == 4) {
                idle -= parseInt(cpustat[i], 10);
            }
        }

        usage = 100.0 * (idle*1.0 - total*1.0) / (-total*1.0);
        // console.log(`${coreNumber} usage ${usage}`);

        return usage;
};

module.exports = readCPUUsage;
```

**7.Memory.js 内存信息获取**

```
const fs = require('fs');
const path = require('path');

const readMemoryUsage = (memKind, memUnit) => {
        const data = fs.readFileSync('/proc/meminfo', 'utf8');
        const stat = String.prototype.split.call(data, '\n');
        const memstat = String.prototype.split.call(stat[memKind], /\s+/);
        let memData = memstat[1];

        if (memUnit == memMB || memUnit == memGB) {
            memData /= 1024;
        }

        if (memUnit == memGB) {
            memData /= 1024;
        }

        return memData;
};

module.exports = readMemoryUsage;
```

**8.System.js 系统信息获取**

```
const fs = require('fs');
const path = require('path');

const getSystemInfo = () => {
```

```
    const data = fs.readFileSync('/proc/cpuinfo', 'utf8');
    const stat = String.prototype.split.call(data, '\n');
    const modelName = String.prototype.split.call(stat[4], /\s+:\s+/)[1];
    const cpuMHz = String.prototype.split.call(stat[7], /\s+:\s+/)[1] + " MHz";
    const cacheSize = String.prototype.split.call(stat[8], /\s+:\s+/)[1];
    const addrSize = String.prototype.split.call(stat[24], /\s+:\s+/)[1];

    return [modelName, cpuMHz, cacheSize, addrSize];
};

module.exports = getSystemInfo;
```

**9.Process.js 进程信息获取**

```
const fs = require('fs');
const path = require('path');

const readProcessDir = (dir) => {
    let results = [];
    let list = '';
    let len;

    list = fs.readdirSync(dir, 'utf8');

    if (list.length === 0) {
        return null;
    }

    list.forEach((file) => {
        let stat;
        file = path.resolve(dir, file);

        stat = fs.statSync(file);

        // find a process info directory, push it into results
        if (stat && stat.isDirectory && /^[0-9]*$/.test(file.split('/')[2])) {
            results = results.concat(file);
        }
    });

    return results;
};

const getTotalCPU = () => {
    let cpuData = fs.readFileSync('/proc/stat', 'utf8');
    let cpuStat = String.prototype.split.call(cpuData, '\n');
    let cpuUsage = String.prototype.split.call(cpuStat[0], /\s+/);

    let total = 0;

    for (let i = 1; i < 10; i++) {
        total += parseInt(cpuUsage[i], 10);
    }

    return total;
};

const readProcessRawData = () => {
    let processRawData = new Map();
```

```
    const processDirs = readProcessDir('/proc');
    processDirs.forEach((dir) => {
        // important file: /proc/*/stat => status info of process
        const file = path.resolve(dir, 'stat');
        const data = fs.readFileSync(file, 'utf8');
        const stat = String.prototype.split.call(data, /\s+/);

        // important file: /proc/*/status => memory info of process
        const files = path.resolve(dir, 'status');
        const datas = fs.readFileSync(files, 'utf8');
        const status = String.prototype.split.call(datas, '\n');

        const rawData = [stat, status];

        processRawData.set(stat[pid], rawData);
    });

    return processRawData;
};

// enum for stat[XXX]
const pid = 0;
const comm = 1;
const tty = 6;
const utime = 13;
const stime = 14;
const priority = 18;
const vsize = 22;
const rss = 23;
const sleepTime = 10;

// enum for status
const VmRSS = 21;
const state = 2;

const stat = 0;
const status = 1;

const P_TOTAL = 0;
const P_CPU = 1;

const getProcessData = (key, rawData) => {
    const totalCPU = getTotalCPU();
    const processCPU = parseInt(rawData[stat][utime]) + parseInt(rawData[stat][stime]);
    const processName = rawData[stat][comm].replace('(', '').replace(')', '');
    const processState = String.prototype.split.call(rawData[status][state], /\s+/)[2];//.replace('(',
'').replace(')', '');
    let processMem = String.prototype.split.call(rawData[status][VmRSS], /\s+/);
    if (processMem[0] === 'VmRSS:') {
        processMem = parseInt(processMem[1]);
        processMem /= 1024;
    } else {
        processMem = 0;
    }

    const processData = [totalCPU, processCPU, 0, key, processName, processMem,
processState];
```

```
    return processData;
}

const getProcessItems = () => {
    let processItems = new Map();
    let processRawData = readProcessRawData();

    for (let key of processRawData.keys()) {
        const rawData = processRawData.get(key);
        const [totalCPU, processCPU, cpuUsage, proceesID, processName, processMem, processState] = getProcessData(key, rawData);
        const processItem = [totalCPU, processCPU, cpuUsage, processName, processName, processMem, processState];
        processItems.set(key, processItem);
    }

    const date = new Date();
    let curDate = null;
    do {
        curDate = new Date();
    } while (curDate - date < sleepTime);

    processRawData = readProcessRawData();

    for (let key of processRawData.keys()) {
        if (processItems.has(key)) {
            // update information
            const rawData = processRawData.get(key);
            let [totalCPU2, processCPU2, cpuUsage, processID, processName, processMem, processState] = getProcessData(key, rawData);

            // calculate cpu usage
            // update cpu usage
            const processItem = processItems.get(key);
            const totalCPU1 = processItem[P_TOTAL];
            const processCPU1 = processItem[P_CPU];
            cpuUsage = 100 * (processCPU2 * 1.0 - processCPU1 * 1.0) / (totalCPU2 * 1.0 - totalCPU1 * 1.0);

            const newProcessItem = [totalCPU2, processCPU2, cpuUsage, processID, processName, processMem, processState];
            processItems.set(key, newProcessItem);
        } else {
            // add new process
            const rawData = processRawData.get(key);
            const [totalCPU, processCPU, cpuUsage, processID, processName, processMem, processState] = getProcessData(key, rawData);
            const processItem = [totalCPU, processCPU, cpuUsage, processID, processName, processMem, processState];
            processItems.set(key, processItem);
        }
    }

    // remove missing process
    for (let key of processItems.keys()) {
        if (!processRawData.has(key)) {
```

```
                processItems.delete(key);
            }
        }

        return processItems;
};

module.exports = getProcessItems;
```
**10.view.js DOM 操作（UI 绘制）**
```
const execSync = require('child_process').execSync;
const readCPUUsage = require('./js/CPU.js');
const readMemUsage = require('./js/Memory.js');
const getProcessItems = require('./js/Process.js');
const getSystemInfo = require('./js/System.js');

const coreTotal = 0;
const core0 = 1;
const core1 = 2;
const core2 = 3;
const core3 = 4;

const memTotal = 0;
// const memFree = 1;
const memAvail = 2;
const memCached = 4;
const memKB = 0;
const memMB = 1;
const memGB = 2;

const S_modelName = 0;
const S_cpuMHz = 1;
const S_cacheSize = 2;
const S_addrSize = 3;

const P_TOTAL = 0;
const P_CPU = 1;
const P_CPUUsage = 2;
const P_PID = 3;
const P_NAME = 4;
const P_MemUsage = 5;
const P_State = 6;

const systemUpdateInterval = 2000;
const processUpdateInterval = 3000;

const updateSystemInfo = () => {
        const SystemDiv = document.querySelector('div#System');

        if (SystemDiv.className === 'tabcontent') {
                // non-active
                return;
        }

        const CPUUsagePara = document.querySelector('div#System p#cpu-usage');
        const MemUsagePara = document.querySelector('div#System p#mem-usage');
        const modelNamePara = document.querySelector('div#System p#model-name');
        const CPUMHzPara = document.querySelector('div#System p#cpu-mhz');
```

```
    const cacheSizePara = document.querySelector('div#System p#cache-size');
    const addrSizePara = document.querySelector('div#System p#addr-size');
    const [modelName, CPUMHz, cacheSize, addrSize] = getSystemInfo();

    CPUUsagePara.innerHTML = `CPU Usage: ${readCPUUsage(coreTotal).toFixed(2)} %`;
    MemUsagePara.innerHTML = `Memory Free: ${readMemUsage(memAvail,
memGB).toFixed(2)} GB`;
    modelNamePara.innerHTML = `Model Name: ${modelName}`;
    CPUMHzPara.innerHTML = `CPU MHz: ${CPUMHz}`;
    cacheSizePara.innerHTML = `Cache Size: ${cacheSize}`;
    addrSizePara.innerHTML = `Address Size: ${addrSize}`;

    // console.log(CPUUsagePara);
    // console.log(MemUsagePara);
};

const updateProcessInfo = () => {
    const ProcessDiv = document.querySelector('div#Process');

    if (ProcessDiv.className === 'tabcontent') {
        // non-active
        return;
    }

    const processTable = document.querySelector('div#Process div.process-table');
    const processTableHeader = '<ul class="process-list">      \
    <li>Name</li>                  \
    <li>PID</li>                   \
    <li>State</li>                 \
    <li>CPU</li>                   \
    <li>Memory</li>                \
    </ul>';

    // <li>Disk</li>                  \
    // <li>Network</li>               \
    const processItems = getProcessItems();

    processTable.innerHTML = processTableHeader;
    for (let key of processItems.keys()) {
        const processData = processItems.get(key);
        // console.log(processData);
        processTable.innerHTML += `<ul class="process-list">                          \
        <li>${processData[P_NAME]}</li>               \
        <li>${processData[P_PID]}</li>                \
        <li>${processData[P_State]}</li>              \
        <li>${processData[P_CPUUsage].toFixed(2)}</li>        \
        <li>${processData[P_MemUsage].toFixed(2)} MB</li>        \
        </ul>`;
    }
};

const openTab = (evt, id) => {
    const tabcontent = document.getElementsByClassName('tabcontent');

    for (let i = 0; i < tabcontent.length; i++) {
        tabcontent[i].style.display = 'none';
        tabcontent.className = 'tabcontent';
```

```
        }

        document.getElementById(id).style.display = 'block';
        document.getElementById(id).className += ' active';
};

const killProcess = () => {
        const pidText = document.getElementById('kill-pid');

        if (pidText.value && /^[0-9]*$/.test(pidText.value)) {
                execSync(`kill ${pidText.value}`);
        }
};

const updateUI = () => {
        setInterval(updateSystemInfo, systemUpdateInterval);
        setInterval(updateProcessInfo, processUpdateInterval);
};

updateUI();
```

**11.main.js Electron 应用入口**

```
const electron = require('electron');
const app = electron.app;
const BrowserWindow = electron.BrowserWindow;

const path = require('path');
const url = require('url');

let mainWindow;

const createWindow = () => {
   mainWindow = new BrowserWindow({
      width: 800,
      height: 600
   });

   // mainWindow.webContents.openDevTools();

   mainWindow.loadURL(url.format({
      pathname: path.join(__dirname, '../index.html'),
      protocol: 'file:',
      slashes: true
   }));

   mainWindow.on('closed', () => {
      mainWindow = null
   });
}

app.on('ready', createWindow);

app.on('window-all-closed', () => {
   if (process.platform !== 'darwin') {
      app.quit()
   }
});
```

```
app.on('activate', function () {
  if (mainWindow === null) {
    createWindow()
  }
});

app.on('browser-window-created', (e, window) => {
  window.setMenu(null);
});
```

**实验五**

**1.Repl.js 交互界面**

```
'use strict';

const fs = require('fs');
const path = require('path');
const process = require('process');
const readline = require('readline-sync');

class Repl {
    constructor(imfs) {
        this.imfs = imfs;
        this.exit = false;
    }

    start() {
        console.log('Log in to in-memory file system ...');
        console.log('Done.');
        console.log('Restore config file ...');

        const jsonFile = fs.openSync('imfs.json', 'a+');
        const jsonData = fs.readFileSync(jsonFile, 'utf8');

        if (jsonData) {
            this.imfs.data = JSON.parse(jsonData);
        }

        fs.closeSync(jsonFile);

        console.log('Done.');
        console.log(`Enter 'help' to get more helpful information.`);
        console.log('');

        this.repl();
    }

    repl() {
        while (!this.exit) {
            const prompt = ((this.imfs.cwd === '/') ? '/' : path.basename(this.imfs.cwd));
            const command = readline.question(`${prompt} $ `).split(/\s+/);
            const cmd = command[0];
            const pathstr = command[1];
            const paths = command.slice(1);
            const content = command.slice(2).join(' ');

            switch (cmd) {
                case 'cd':
                    this.cmd_cd(pathstr);
```

```
                        break;
                case 'ls':
                        if (paths.length === 0) {
                                this.cmd_ls(this.imfs.cwd);
                        } else {
                                paths.forEach((pathstr) => {
                                        this.cmd_ls(pathstr);
                                });
                        }
                        break;
                case 'pwd':
                        this.cmd_pwd();
                        break;
                case 'mkdir':
                        paths.forEach((pathstr) => {
                                this.cmd_mkdir(pathstr);
                        });
                        break;
                case 'rm':
                        paths.forEach((pathstr) => {
                                this.cmd_rm(pathstr);
                        });
                        break;
                case 'touch':
                        paths.forEach((pathstr) => {
                                this.cmd_touch(pathstr);
                        });
                        break;
                case 'cat':
                        paths.forEach((pathstr) => {
                                this.cmd_cat(pathstr);
                        });
                        break;
                case 'write':
                        this.cmd_write(pathstr, content);
                        break;
                case 'clear':
                        this.cmd_clear();
                        break;
                case 'exit':
                        this.cmd_exit();
                        break;
                case 'help':
                        this.cmd_help();
                        break;
                default:
                        console.log(`Error: unkown command '${cmd}'.`);
                        break;
        }
    }
}

cmd_cd(_path) {
    try {
        this.imfs.chdir(_path);
    } catch (err) {
        console.log(err.message);
```

```
        }
}

cmd_ls(_path) {
        try {
                _path = _path || '/';
                const nodes = this.imfs.readdir(_path);
                nodes.forEach((node) => {
                        process.stdout.write(`${node}\t`);
                });
                console.log('');
        } catch (err) {
                console.log(err.message);
        }
}

cmd_pwd() {
        console.log(this.imfs.cwd);
}

cmd_mkdir(_path) {
        try {
                this.imfs.mkNode(_path, 0);
        } catch (err) {
                console.log(err.message);
        }
}

cmd_rm(_path) {
        try {
                this.imfs.rmNode(_path);
        } catch (err) {
                console.log(err.message);
        }
}

cmd_touch(_path) {
        try {
                this.imfs.mkNode(_path, 1);
        } catch (err) {
                console.log(err.message);
        }
}

cmd_cat(_path) {
        try {
                console.log(this.imfs.readFile(_path));
        } catch (err) {
                console.log(err.message);
        }
}

cmd_write(_path, content) {
        try {
                this.imfs.writeFile(_path, content);
        } catch (err) {
                console.log(err.message);
```

```
        }
    }

    cmd_clear() {
        console.log('\x1Bc'); // \033c
    }

    cmd_exit() {
        console.log('');
        console.log('Store config file ...');

        const data = JSON.stringify(this.imfs.data);

        if (data) {
            fs.writeFileSync('imfs.json', data, 'utf8');
        }

        this.exit = true;

        console.log('Done.');
        console.log('Log out from in-memory file system ...');
        console.log('Done.');
    }

    cmd_help() {
        console.log('Welcome to in-memory file system');
        console.log('\tcd       <path>                  : change current working directory.');
        console.log('\tls       <directory ...>    : list content of directory.');
        console.log('\tpwd                                  : print current working directory.');
        console.log('\tmkdir <directory ...>    : create new directory.');
        console.log('\trm       <target ...>         : remove file or directory.');
        console.log('\ttouch <file ...>             : create new file.');
        console.log('\tcat      <file ...>             : print content of file.');
        console.log('\twrite <file> <content> : write content to file.');
        console.log('\tclear                                : clear screen.');
        console.log('\texit                                 : log out from in-memory file system.');
    }
}

module.exports = Repl;
```

**2.Imfs.js 文件系统实现**

```
'use strict';

const path = require('path');      // utils for resolve path

class Imfs {
    constructor(data) {
        this.data = data || {};
        this.cwd = '/';
    }


    /**
    * judge whether current node is directory or not
    *
    * @method isDir
    * @param    {object} node current node
```

```
* @return {Boolean}            true stand for is directory
*/
isDir(node) {
        if (typeof node !== 'object') {
                return false;
        } else {
                return node[''] === true;
        }
}

/**
* judge whether current node is file or not
*
* @method isFile
* @param    {object} node current node
* @return {Boolean}            true stand for is file
*/
isFile(node) {
        if (typeof node !== 'object') {
                return false;
        } else {
                return node[''] === false;
        }
}

/**
* change path string to path array
*
* @method resolvePath
* @param    {string} _path path string for target
* @return {string}              normalized absolute path
*/
resolvePath(_path) {
        let formatPath = _path;

        // combine path to absolute path
        if (!path.isAbsolute(formatPath)) {
                formatPath = path.join(this.cwd, formatPath);
        }

        formatPath = path.normalize(formatPath);

        return formatPath;
}

/**
* change normalized absolute path to array
*
* @param    {string} formatPath normalized absolute path
* @return {array}                       path array
*/
path2arr(formatPath) {
        let patharr = formatPath.substr(1).split('/');

        // remove tail '/' when from relative path
        if (!patharr[patharr.length - 1]) {
                patharr.pop();
```

```
        }

        return patharr;
}

/**
* change current working directory
*
* @method chdir
* @param    {string}    _path path string for target
* @return {object}          reference to imfs (this)
*/
chdir(_path) {
        const formatPath = this.resolvePath(_path);

        if (this.isExist(formatPath)) {
                this.cwd = formatPath;
        } else {
                throw new Error(`Error: directory '${formatPath}' not exists.`);
        }
}

/**
* judge a dir/file whether exists or not
*
* @method isExist
* @param    {string}    _path path string for target
* @return {Boolean}             true stand for existance
*/
isExist(_path) {
        const formatPath = this.resolvePath(_path);
        const patharr = this.path2arr(formatPath);

        // root directory
        if (patharr.length === 0) {
                return true;
        }

        let cache = this.data;
        let i = 0;

        for (; i < patharr.length - 1; i++) {
                if (!this.isDir(cache[patharr[i]])) {
                        return false;
                }

                cache = cache[patharr[i]];
        }

        return !!cache[patharr[i]];
}

/**
* read content of directory
*
* @method readdir
* @param    {string} _path path string for target
```

```
* @return {array}            string array of file/subdir name
*/
readdir(_path) {
    const formatPath = this.resolvePath(_path);
    const patharr = this.path2arr(formatPath);

    // root directory
    if (patharr.length === 0) {
        return Object.keys(this.data).filter(Boolean);
    }

    let cache = this.data;
    let i = 0;

    for (; i < patharr.length - 1; i++) {
        if (!this.isDir(cache[patharr[i]])) {
            throw new Error(`Error: directory '${formatPath}' not exists.`);
        }

        cache = cache[patharr[i]];
    }

    if (!this.isDir(cache[patharr[i]])) {
        throw new Error(`Error: directory '${formatPath}' not exists.`);
    }

    return Object.keys(cache[patharr[i]]).filter(Boolean);
}

/**
* make new directory/file
*
* @method mkNode
* @param    {string} _path path string for target
* @param    {Boolean} type 0 for directory, 1 for file
* @return {object}          reference to imfs (this)
*/
mkNode(_path, type) {
    const formatPath = this.resolvePath(_path);
    const patharr = this.path2arr(formatPath);

    // root directory
    if (patharr.length === 0) {
        return this;
    }

    let cache = this.data;
    let i = 0;

    for (; i < patharr.length - 1; i++) {
        if (this.isFile(cache[patharr[i]])) {
            throw new Error(`Error: homonymous file '${patharr[i]}' exists.`);
        } else if (!this.isDir(cache[patharr[i]])) {
            // create new directory when non-exist
            cache[patharr[i]] = {": true};
        }
```

```
            cache = cache[patharr[i]];
        }

        if (this.isDir(cache[patharr[i]])) {
            throw new Error(`Error: directory '${patharr[i]}' exists.`);
        }
        if (type) {
            cache[patharr[i]] = {'': false, 'content': ''};
            console.log(`Success: create file '${formatPath}'.`);
        } else {
            cache[patharr[i]] = {'': true};
            console.log(`Success: create directory '${formatPath}'.`);
        }

        return this;
    }

    /**
    * delete directory/file
    *
    * @method rmNode
    * @param    {string} _path path string for target
    * @return {object}          reference to imfs (this)
    */
    rmNode(_path) {
        const formatPath = this.resolvePath(_path);
        const patharr = this.path2arr(formatPath);

        if (patharr.length === 0) {
            throw new Error(`Error: can't remove '/' directory`);
        }

        let cache = this.data;
        let i = 0;

        for (; i < patharr.length - 1; i++) {
            if (!this.isDir(cache[patharr[i]])) {
                throw new Error(`Error: directory '${patharr[i]}' not exists.`);
            }

            cache = cache[patharr[i]];
        }

        delete cache[patharr[i]];

        return this;
    }

    /**
    * read content of file
    *
    * @method readFile
    * @param    {string} _path path string for target
    * @return {string}            content of file
    */
    readFile(_path) {
        const formatPath = this.resolvePath(_path);
```

```
        const patharr = this.path2arr(formatPath);


        let cache = this.data;
        let i = 0

        for (; i < patharr.length - 1; i++) {
                if (!this.isDir(cache[patharr[i]])) {
                        throw new Error(`Error: directory '${patharr[i]}' not exists.`);
                }

                cache = cache[patharr[i]];
        }

        if (!this.isFile(cache[patharr[i]])) {
                throw new Error(`Error: file '${patharr[i]}' not exists.`);
        }

        return cache[patharr[i]]['content'].toString();
}

/**
* write content to file
*
* @method writeFile
* @param    {string} _path path string for target
* @param    {string} content content to write
* @return {object}          reference to imfs (this)
*/
writeFile(_path, content) {
        if (!content) {
                throw new Error(`Error: no content.`);
        }

        const formatPath = this.resolvePath(_path);
        const patharr = this.path2arr(formatPath);

        if (patharr.length === 0) {
                throw new Error(`Error: file '${formatPath}' not exists.`);
        }

        let cache = this.data;
        let i = 0

        for (; i < patharr.length - 1; i++) {
                if (!this.isDir(cache[patharr[i]])) {
                        throw new Error(`Error: directory '${patharr[i]}' not exists.`);
                }

                cache = cache[patharr[i]];
        }

        if (this.isDir(cache[patharr[i]])) {
                throw new Error(`Error: file '${formatPath}' not exists.`);
        }

        cache[patharr[i]]['content'] = content;
```

```
            return this;
        }
}

module.exports = Imfs;
```

**3.main.js Nodejs 应用入口**

```
'use strict';

const Imfs = require('./Imfs.js');
const Repl = require('./Repl.js');

const imfs = new Imfs();
const repl = new Repl(imfs);

repl.start();
```