

**嵌入式操作系统课程设计报告**

|  |  |
| --- | --- |
|  |  |
| 姓 名： | 潘翔 |
| 学 院： | 计算机科学与技术学院 |
| 专 业： | 物联网工程 |
| 班 级： | IOT1601 |
| 学 号： | U20161489 |
| 指导教师： | 石柯 |

|  |  |
| --- | --- |
| 分数 |  |
| 教师签名 |  |

2018 年 7月 12日

**目 录**

[1 设计目的 3](#_Toc1341153448)

[2 文件拷贝 4](#_Toc544645338)

[2.1 设计目的 4](#_Toc1704840036)

[2.2 实验要求和内容 4](#_Toc705564598)

[2.3 环境及步骤 4](#_Toc2025814681)

[2.3.1 开发环境 4](#_Toc828160754)

[2.3.2 开发步骤 4](#_Toc110062500)

[2.4 关键代码 6](#_Toc71436891)

[2.4.1 copyF2F 6](#_Toc18611242)

[2.4.2 copyD2D 7](#_Toc2092477315)

[2.5 调试记录及运行结果 8](#_Toc1152241807)

[2.6 设计感想 8](#_Toc602453600)

[3 图形化进程并发 9](#_Toc47326178)

[3.1 实验要求和内容 9](#_Toc15731743)

[3.2 实验过程与结果 9](#_Toc266536055)

[3.3 实验结果分析 9](#_Toc1446029100)

[3.4 心得与体会 9](#_Toc623979156)

[4 添加系统调用 10](#_Toc1395058600)

[4.1 实验要求和内容 10](#_Toc2083435486)

[4.2 实验步骤 10](#_Toc1918142902)

[4.3 实验过程与结果 11](#_Toc2113874923)

[4.4 实验结果分析 12](#_Toc1457237147)

[4.5 心得与体会 12](#_Toc1589873826)

[5 QT系统监控器 13](#_Toc1657722463)

[5.1 实验要求和内容 13](#_Toc1140529953)

[5.2 实验步骤 13](#_Toc1376817375)

[5.3 实验过程与结果 13](#_Toc420607273)

[6 模拟文件系统设计 14](#_Toc493846564)

[6.1 实验要求和内容 14](#_Toc2050498830)

[6.2 设计架构 14](#_Toc956692194)

[6.3 设计实现 14](#_Toc416685624)

[6.3.1 FreeNode 14](#_Toc1244168630)

[6.3.2 Inode 14](#_Toc1501337532)

[6.3.3 DirEntry 14](#_Toc2121525660)

[6.3.4 myFs(file system operation) 14](#_Toc1949733228)

[6.4 实验总结 15](#_Toc1379668566)

# 

# 1 设计目的

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

# 2 文件拷贝

## 2.1 设计目的

## 2.2 实验要求和内容

要求：熟悉和理解Linux编程环境

内容：

编写一个C程序，用read、write等系统调用实现文件拷贝功能。命令形式：

copy <源文件名> <目标文件名>

## 2.3 环境及步骤

### 2.3.1 开发环境

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

### 2.3.2 开发步骤

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

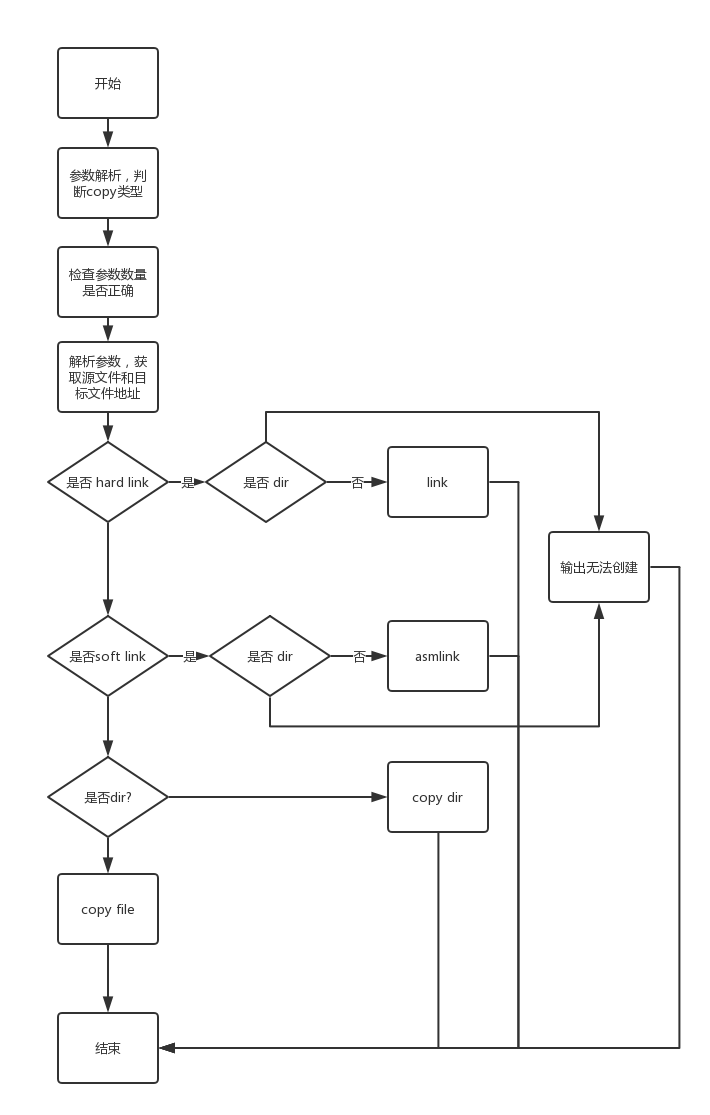


图2-1 copy函数流程图

## 2.4 关键代码

### 2.4.1 copyF2F

1. 函数原型

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

1. 函数流程

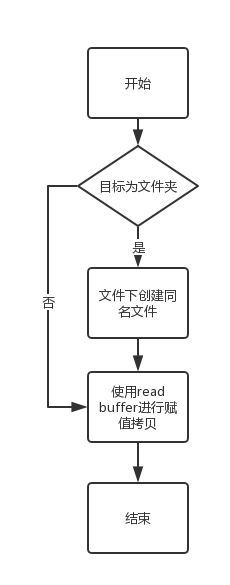


图2-2 copyF2F函数流程图

1. 关键代码

while ((n\_chars = read(in\_fd, buf, BUFFERSIZE)) > 0)

{

if (write(out\_fd, buf, n\_chars) != n\_chars)

{

printf("%s write file fail！", dest\_file);

return 1;

}

if (n\_chars == -1)

{

printf("%s read file fail！", src\_file);

return 1;

}

}

### 2.4.2 copyD2D

1. 函数原型

int copyD2D(char \*src\_dir, char \*dest\_dir)函数流程

1. 函数流程

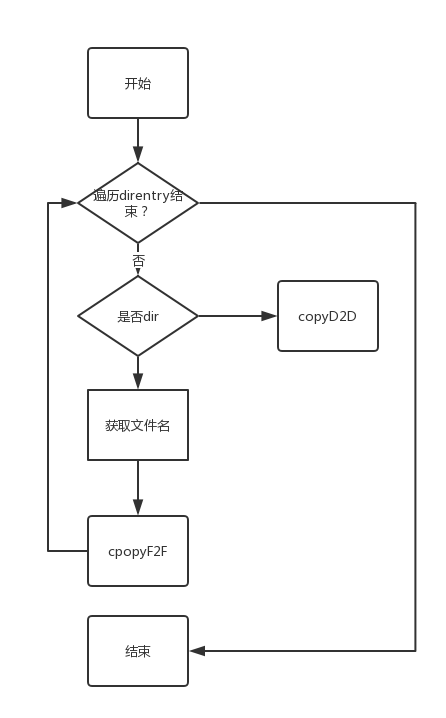


图2-3 copyD2D函数流程图

1. 关键代码

//open dir

if ((dp = opendir(src\_dir)) == NULL)

return 1;

else

{

//get dirent

while ((dirp = readdir(dp)))

{

struct stat file\_stat;

if (!isdir(dirp->d\_name))

{

//link name

strcat(tempDest, dirp->d\_name);

strcat(tempSrc, dirp->d\_name);

//copy file

copyF2F(tempSrc, tempDest);

//recover name

strcpy(tempDest, dest\_dir);

strcpy(tempSrc, src\_dir);

}

}

//close dir

closedir(dp);

return 0;

}

## 2.5 调试记录及运行结果



图2-4 copyF2F Test



图2-5 copyD2D Test

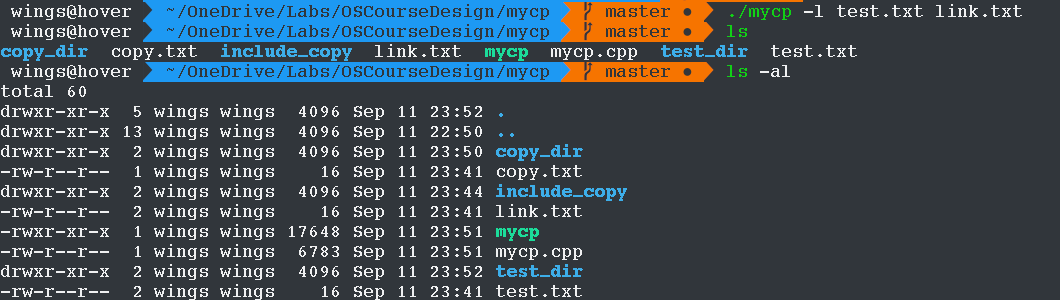


图2-6 link Test

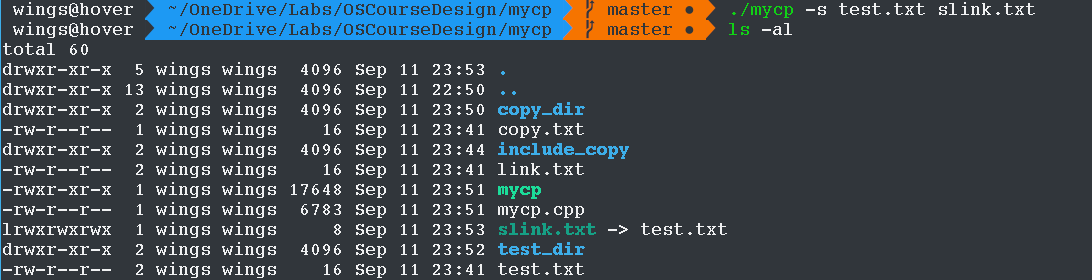


图2-7 syblink Test

## 2.6 设计感想

参照linux cp进行设计，其中D2D进行递归操作，类似于ls，需要注意栈空间的占用

# 3 图形化进程并发

## 3.1 实验要求和内容

要求：熟悉和理解Linux编程环境

内容：

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

## 3.2 实验过程与结果

## 3.3 实验结果分析

## 3.4 心得与体会

# 4 添加系统调用

## 4.1 实验要求和内容

要求：熟悉和理解Linux编程环境

内容：

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

## 4.2 实验步骤

1. 添加源代码

编写添加到内核中的源程序，函数名以sys\_开头。

如：mycall(int num)，在arch/kernel/sys.c文件中添加如下代码：

asmlinkage long sys\_mysyscall(int number)

{

return number; //该系统调用仅返回一个整型值

}

1. 添加新的系统调用

使内核的其余部分知道该系统调用的存在。为此，需编辑两个文件：

include/linux/syscalls.h ——系统调用定义

增加新系统调用的函数定义：

asmlinkage long sys\_mysyscall(int number);

arch/x86/syscalls/syscall\_32.tbl ——系统调用表

在系统调用表中为新增的系统调用分配一个系统调用号和系统调用名。

1. 重建Linux内核

#make menuconfig //生成内核配置文件

如编译中报错缺少软件包，则先安装：

#sudo apt-get install package

#apt-get install libncurses5-dev

#make bzImage //编译内核映像

#make modules //编译内核模块

#make modules\_install //生成并安装模块

#make install //安装新的系统

1. 修改/etc/default/grub文件，注释掉 GRUB\_HIDDEN\_TIMEOUT=0，然后运行update-grub命令
2. 重启，选择新修改的内核
3. 编写应用程序，测试新增系统调用

## 4.3 实验过程与结果

1. 查看内核版本

cat /proc/version



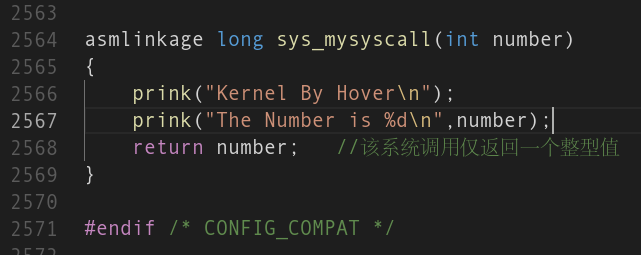
图4-1 查看内核版本

1. 获取内核源码

访问<https://www.kernel.org/> 下载4.16.9版本内核

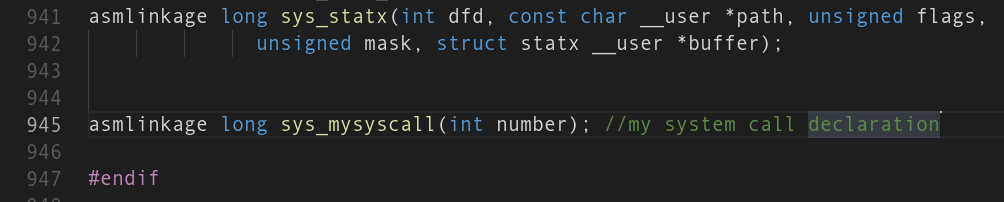
1. 添加调用服务例程定义

在arch/kernel/sys.c中添加调用服务例程定义



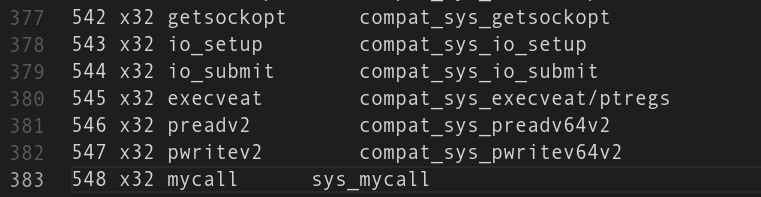
1. 添加调用函数声明

在include/linux/syscalls.h中添加调用函数声明



1. 系统调用表

在/arch/x86/entry/syscalls/syscall\_64.tbl 系统调用表中为新增的系统调用分配一个系统调用号和系统调用名。



1. 编译内核

make mrproper

make menuconfig

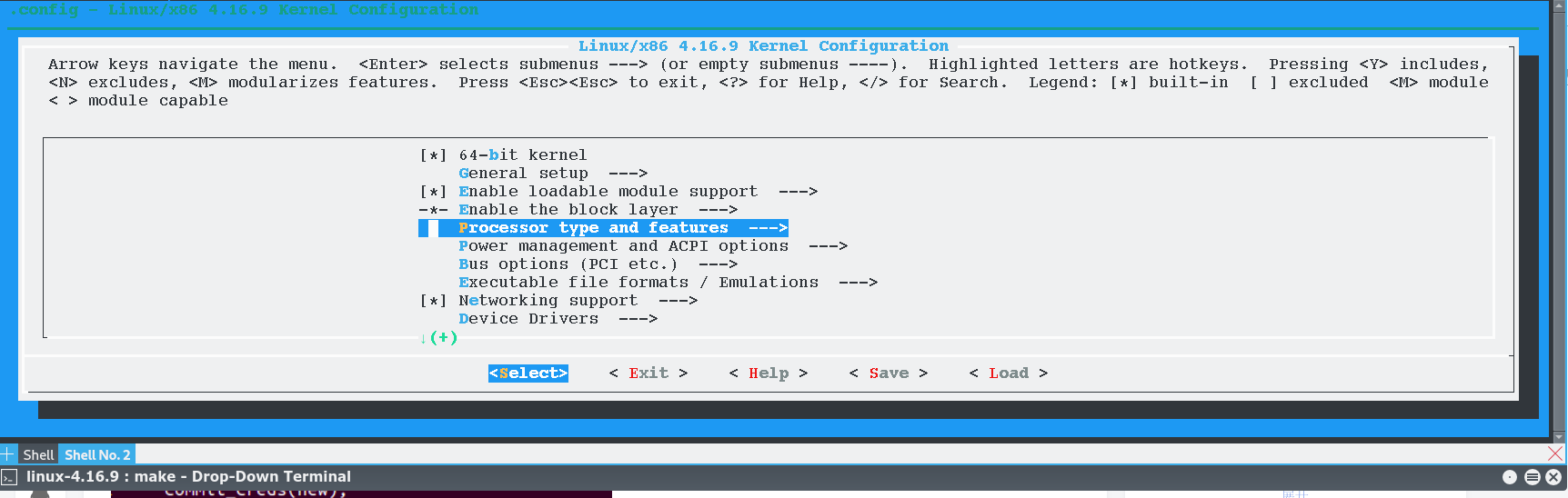
make bzImage –j4

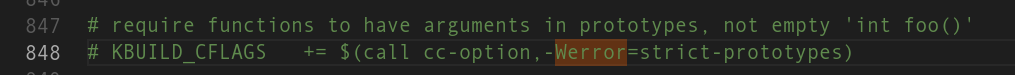
make modules –j4

make modules\_install –j4

make install –j4

update-grub





## 4.4 实验结果分析

## 4.5 心得与体会

# 5 QT系统监控器

## 5.1 设计目的

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

## 5.2 设计内容

监控系统功能

通过读取proc文件系统，获取系统各种信息，并以比较容易理解的方式显示出来

C语言开发，图形界面直观展示

具体包括：

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

----参照WINDOWS的任务管理器，实现其中的部分功能

## 5.3 环境及步骤

### 5.3.1 开发环境

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

### 5.3.2 运行环境

基于Qt跨平台特性，可运行于基于Linux的平台

### 5.3.3 开发步骤

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

## 5.4 关键代码

### 5.4.1 CPU

1. 更新过程
2. 利用率计算
3. 图表绘制

### 5.4.2 Process

1. 数据结构
2. 更新过程
3. 列表维护
4. 列表绘制

### 5.4.3 Memory

1. 更新过程
2. 利用率计算
3. 图表绘制

### 5.4.4 Net

1. 更新过程
2. 利用率计算
3. 图表绘制

## 5.5 调试记录及运行结果

## 5.6 设计感想

设计过程中，参考GNOME-MONITOR，使用Qt的定时器机制进行操作

# 6 模拟文件系统设计

## 设计目的

熟悉Linux文件系统

## 设计内容

用磁盘中的一个文件（大小事先指定）来模拟一个磁盘

确定文件目录项的结构

空闲块的管理（每个块＝连续的Ｎ个文件字节）

扩充系统调用命令实现文件的操作：open、 close、 read、 write、 cp、 rm等

选择支持：多用户、树形目录。

## 环境及步骤

### 开发环境

### 系统总体设计

### SuperBlock

### Inode

### Sector

### Buffer

系统分模块设计

## 设计实现

### FreeNode

1. 类描述
2. 构造函数

### Inode

1. 类描述
2. 构造函数

### DirEntry

### myFs(file system operation)

## 实验总结

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

在编写过程中，考虑参数传入，使用C++ std模板类进行传参，同时对于参数的的个数进行动态的解析，

参考文献：

https://www.boost.org/doc/libs/1\_57\_0/libs/filesystem/doc/tutorial.html

Linux文件系统

FUSE文件系统