操作系统课程设计

内容简介

- 设计目的
- 设计内容
- 实施方法及要求
- 时间安排
- 辅导

设 计 目 的

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

内容简介

- 设计目的
- 设计内容
- 实施方法及要求
- 时间安排
- 辅导

可以用虚拟机来做（切记只能用32位的！！！）。

网上信息良莠不齐，遇到问题，建议去专业网站

https://wiki.ubuntu.com/Kernel/BuildYourOwnKernel

设 计 内 容 (1)

- 要求：熟悉和理解Linux编程环境
- 内容

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

copy <源文件名> <目标文件名>

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

设 计 内 容 (2)

- 要求：掌握添加系统调用的方法

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

实验2　TIPS

- 第一步先不改代码，直接编译内核，编译后的内核能成功启动之后再做第二步，增加系统调用。第一步编译内核的时间比较长，这个是无法避免的，但是只要能成功，就说明你下载的这个内核版本跟你的机器环境是兼容的，可以继续下一步。

- 第二步因为反复修改代码，可能要多次编译内核，但是不需要重复第一步的完整步骤，只需要从make那一步开始，时间不会太长，我的经验是10-20分钟左右。

设 计 内 容 (3)

- 要求：掌握添加设备驱动程序的方法
- 内容
  - 采用模块方法，添加一个新的字符设备驱动程序，实现打开/关闭、读/写等基本操作
  - 编写一个应用程序，测试添加的驱动程序

设 计 内 容 (4)

- 要求：使用GTK/QT实现一个系统监控器（选做）
- 内容
  - 了解/proc文件的特点和使用方法
  - 监控系统状态，显示系统部件的使用情况
  - 用图形界面监控系统状态，包括系统基本信息、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)

设 计 内 容(5)

- 要求：设计并实现一个模拟的文件系统（选做）
- 内容
  - 基于一个大文件(如100M)，模拟磁盘
  - 格式化，建立文件系统管理数据结构
  - 实现文件/目录创建/删除，目录显示等基本功能(可自行扩充文件读/写、用户登录、权限控制、读写保护等其他功能)

内容简介

- 设计目的
- 设计内容
- 实施方法及要求
- 时间安排
- 辅导

实施方法及要求

- **独立**完成课程设计内容

  - 支持借鉴和学习已有的优秀知识！
  - 反对全盘拷贝，不求甚解！
  - 吸收和消化他人经验，做自己的课程设计！

实施方法及要求

- 上机检查：根据要求演示完成的系统，并回答老师的问题或按要求现场修改程序
- 报告提交：
  - 纸质课程设计报告(双面打印)：内容包括实验目的、实验内容、实验设计、实验环境及步骤、调试记录和课程设计心得等
  - 光盘：课程设计报告电子版和程序清单（附注释），每个班一张光盘

要求

- 成绩依据：检查情况+报告撰写情况
- 完成前两题：60-65
- 完成前三题：65-75
- 完成选做题：75以上

第一周周五开始中期检查，检查内容1、2、3题，完成多少检查多少，如果有同学在周五前就完成了前两题的，也可以提前检查。另外，大家做实验的时候不要只追求完成的题数，还要重视每道题的完成度，尤其禁止抄袭，对提交检查的代码必须能解释，否则可能倒扣分。从第3题开始，ppt中给出的评分标准只是参考，实际上考虑问题越完善分数才会越高，3、4、5题都浅尝则止的同学，最后分数可能会比只做了其中两题的同学还低，希望大家了解。

内容简介

- 设计目的
- 设计内容
- 实施方法及要求
- 时间安排
- 辅导

时间安排

- 课程设计时间：第一周、第二周
- 课程设计地点：以课表为准
- 中期检查：

  第一周周五上午（8:00-11:00）过时不候
- 最后检查：

  第二周周五上午（8:00-11:00）过时不候
- 实验报告提交：

  第四周

内容简介

- 设计目的
- 设计内容
- 实施方法及要求
- 时间安排
- 辅导

课程设计辅导

- Linux系统的相关知识
- 进程并发
- 添加系统调用
- 添加设备驱动程序
- /proc文件分析

Linux系统的相关知识

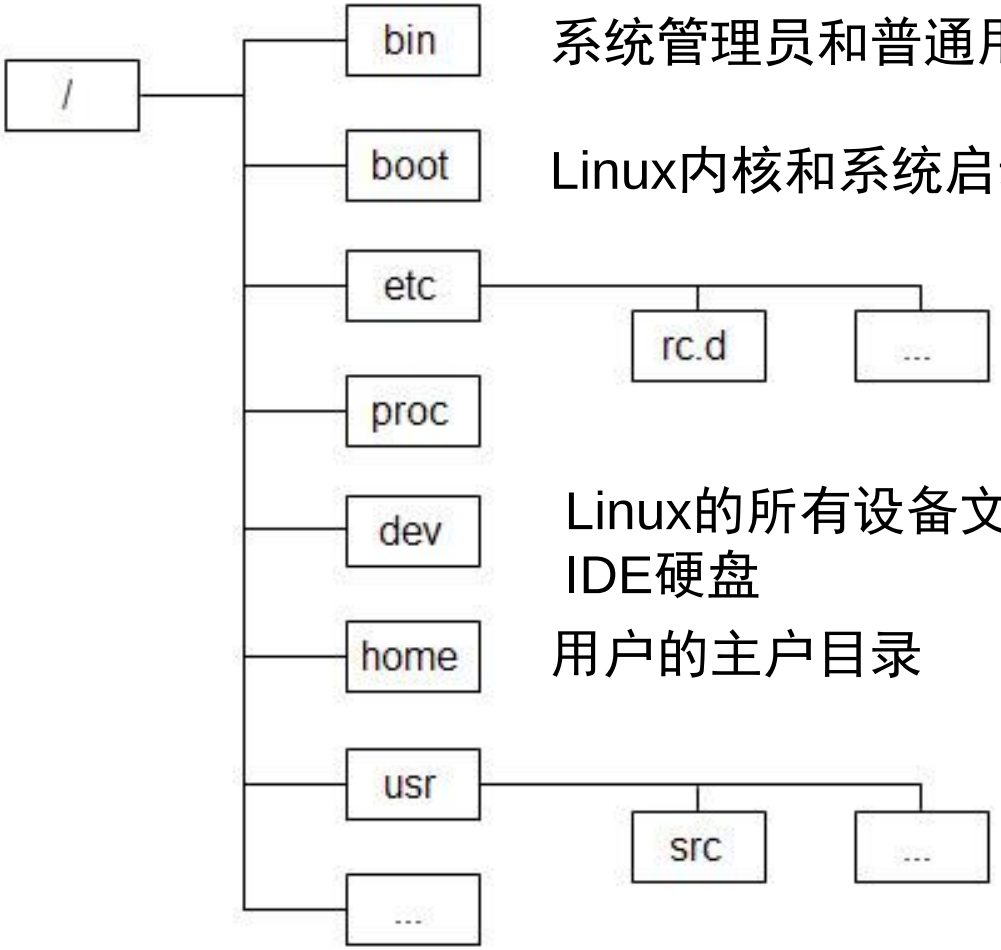
发行版:

内核版本: major.minor.patch-build

- major: 主版本号，有结构性变化时变更
- minor: 次版本号，新增功能时发生变化
  奇数表示开发版，偶数表示稳定版
- patch-build: 修订版本号

内核最新版4.9

Linux系统的常用目录



系统管理员和普通用户使用的命令

Linux内核和系统启动文件

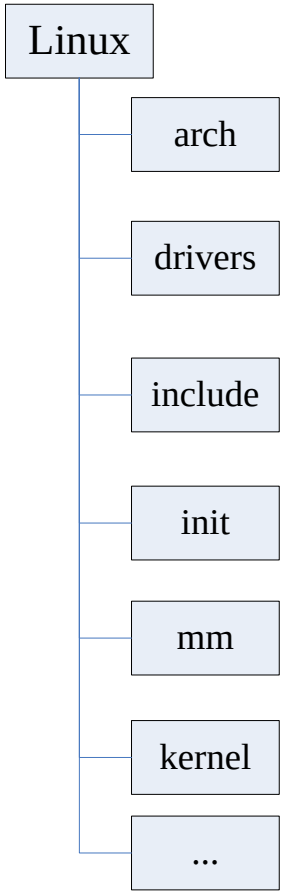
系统管理所需的配置文件和子目录

Linux的所有设备文件，如/dev/hda代表第一个物理IDE硬盘

用户的主户目录

命令、程序库、文档和其他文件

LINUX系统的核心源码



CPU类型相关的核心代码。每一个子目录代表一种**CPU**类型，如**i386**是关于**Intel CPU**及与之相兼容的体系结构的子目录

所有的设备驱动程序;每种驱动程序又各占用一个子目录，如**/block**下为块设备驱动程序

编译核心所需要的大部分头文件。与平台无关的头文件在**include/linux**子目录下，与**Intel CPU**相关的头文件在 **include/asm-i386**子目录下

包含核心的初始化代码，包含两个文件 **main.c**和**version.c**

所有独立于**CPU**体系结构的内存管理代码，与体系结构相关的内存管理代码则位于**arch/*/mm/**下

主要的核心代码，包括大多数**Linux**系统的内核函数， 其中最重要的文件当属进程调度**sched.c**，同样，和体系结构相关的代码在**arch/*/kernel**中

课程设计辅导

- Linux系统的相关知识
- 进程并发
- 添加系统调用
- 添加设备驱动程序
- /proc文件分析

进程并发

- pid=fork()：创建子进程。

  返回值:0　　从子进程返回

  　　>0　　从父进程返回

- exit进程自我终止，进入僵死状态
- wait( ) 等待进程终止(由父进程调用)
- exec( ) 执行一个可执行程序(文件)

FORK()系统调用

```
main()
{
        pid_t p1;
        pid_t t1;

      p1=fork();
        if (p1 == 0)
      {
            puts("sub1 created\n");
        }
      else       //main
      {
          t1=waitpid(p1,&status,0);
        }
}
```

父进程p1>0

```
main()
{       pid_t p1;
        pid_t t1;

      p1=fork();
        if (p1 == 0)
       {    puts("sub1 created\n");
        }
   else
   {  t1=waitpid(p1,&status,0);
   }
}
```

父进程实际执行的程序段

子进程p1==0

```
main()
{       pid_t p1;
        pid_t t1;

      p1=fork();
   if (p1 == 0)
   {  puts("sub1 created\n");
   }
      else       //main
      {   t1=waitpid(p1,&status,0);
       }
}
```

子进程实际执行的程序段

课程设计辅导

- Linux系统的相关知识
- 进程并发
- 添加系统调用
- 添加设备驱动程序
- /proc文件分析

添加系统调用

- Linux系统调用机制
  - Linux内核中设置了一组用于实现各种系统功能的子程序，称为系统调用
  - 用户可以通过系统调用命令在自己的应用程序中调用它们
- 系统调用与普通函数调用的区别
  - 系统调用　　核心态　操作系统核心提供
  - 普通的函数调用　用户态　函数库或用户自己提供

添加系统调用（续）

- Linux系统调用机制
  - int 0x80

    使用寄存器中适当的值跳转到内核中事先定义好的代码中执行：跳转到系统调用的总入口system_call，检查系统调用号，再查找系统调用表sys_call_table，调用内核函数，最后返回

  - 系统调用是靠一些宏，一张系统调用表，一个系统调用入口来完成的

添加系统调用（续）

- 获取linux内核源码，下载地址:
  https://www.kernel.org/pub/linux/kernel/
- 参考资料地址:
  https://wiki.Ubuntu.com/Kernel/BuildYourOwnKernel
- 根据自己安装的linux发行版所用的内核版本，选择下载版本号接近的稳定版本(建议选32位的)
- 解压源码，修改即在此源码版本上进行

添加系统调用（续）

- 与系统调用相关的内核代码文件:
  - 系统调用服务例程定义

    如 arch/kernel/sys.c
  - 系统调用函数声明

    如 include/linux/syscalls.h
  - 系统调用表（为每个系统调用分配唯一号码）

    如 arch/x86/syscalls/syscall_32.tbl

本课件以ubuntu 14.4.04、内核源码4.0.1版为例，
不同版本Linux的文件名和存放位置会有所不同！

添加系统调用（续）

- 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
 }
```

添加系统调用（续）

- 步骤_2　连接新的系统调用

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。

添加系统调用（续）

- 步骤_3　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
```

添加系统调用（续）

- 步骤_4 修改/etc/default/grub文件，注释掉 GRUB_HIDDEN_TIMEOUT=0，然后运行update-grub命令
- 步骤_5 重启，选择新修改的内核
- 步骤_6 编写应用程序，测试新增系统调用

添加系统调用（续）

- 编写内核代码时的注意事项
  - 打印调试信息应该用内核提供的函数printk，不能用printf。printk打印的信息可以用dmesg命令查看。
  - 文件操作应使用对应的内核函数sys_open、sys_read等，在调用这些函数时，为了避免内存保护检查错误，要暂时将访问限制值设置为内核的内存访问范围，然后再修改为原来的值。使用以下语句：

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

课程设计辅导

- Linux系统的相关知识
- 进程并发
- 添加系统调用
- 添加设备驱动程序
- /proc文件分析

添加设备驱动程序

- 设备驱动程序
  - 一组常驻内存的具有特权的共享库，是低级硬件处理例程
  - 每个设备文件有两个设备号
    - 主设备号标识驱动程序
    - 从设备号表示使用同一个设备驱动程序的不同硬件设备
  - 设备驱动程序的功能
    - 对设备初始化和释放
    - 把数据从内核传送到硬件和从硬件读取数据
    - 读取应用程序传给设备文件的数据和回送应用程序请求的数据
    - 检测和处理设备出现的错误

添加设备驱动程序（续）

- Linux支持的设备类型
  - 字符设备—— c

    存取时没有缓存；对字符设备发出读写请求时，实际的I/O就发生了。如：鼠标、键盘等。
  - 块设备—— b

    利用一块系统内存区域作缓冲区，当用户进程对设备请求能满足用户要求时，返回请求数据，否则，调用请求函数进行实际的I/O操作。如：硬盘、软盘、CD-ROM等。
  - 网络设备

添加设备驱动程序（续）

- 注册设备：向系统登记设备及驱动程序的入口点
  - int register_chrdev (unsigned int major, const char *name, struct file_operations *fops);

    //向系统的字符设备表登记一个字符设备

    //major：希望获得的设备号，为0时系统选择一个没有被占用的设备号返回。

    //name：设备名

    //fops：登记驱动程序实际执行操作的函数的指针

    //登记成功，返回设备的主设备号，否则，返回一个负值

  - int register_blkdev (unsigned int major, const char *name, struct file_operations *fops);

    //向系统的块设备表登记一个块设备

添加设备驱动程序（续）

- 设备卸载
  - int unregister_chrdev (unsigned int major, const char *name);
    //卸载字符设备
    //major：要卸载设备的主设备号
    //name：设备名
  - int unregister_blkdev (unsigned int major, const char *name);
    //卸载块设备

添加设备驱动程序（续）

- Linux系统采用一组固定的入口点来实现驱动设备的功能。
  - open入口点：打开设备。对将要进行的I/O操作做好必要的准备工作，如清除缓冲区等
  - close入口点：关闭一个设备
  - read入口点：从设备上读数据
  - write入口点：往设备上写数据
  - ioctl入口点：执行读、写之外的操作
  - select入口点：检查设备，看数据是否可读或设备是否可用于写数据

添加设备驱动程序（续）

- 内核模块(LKM, Loadable Kernel Modules)
  - Linux核心是一种monolithic类型的内核，即单一的大核心
  - linux内核是一个整体结构，因此向内核添加或者删除某些功能，都十分困难。为了解决这个问题，引入了模块机制，从而可以动态的在内核中添加或删除模块

添加设备驱动程序(续)

- 模块的实现机制
  - 模块初始化(注册)　　int init_module( ){ };
  - 模块卸载(注销)　　int cleanup_module( ){ };
  - 操作
    - unsigned long sys_create_module (char *name, unsigned long size);
      //重新分配内存
    - int sys_delete_module (char *name);　　//卸载
    - int sys_query_module (const char *name, int which, void *buf,
      size_t bufsize, size_t *ret);　　//查询

添加设备驱动程序（续）

- 模块编程实例
  - hello.c源码

```
#include "linux/kernerl.h"
#include "linux/module.h"
/*处理版本问题CONFIG_MODVERSIONS */
#if CONFIG_MODVERSIONS==1
#define MODVERSIONS
#include "linux/version.h"
#end if
int init_module() /* 模块初始化*/
{
  printk("hello world !\n");
  printk("I have runing in a kernel mod! \n");
  return 0;
}
void cleanup_module() /* 模块卸载 */
{
  printk(" I will shut down myself in kernel mode!\n)";
}
```

添加设备驱动程序（续）

- 模块的实现机制
  - 模块加入： insmod modulename.ko
  - 查看模块： lsmod
  - 删除模块： rmmod modulename

添加设备驱动程序（续）

- 添加设备驱动程序的方法

1. 编写设备驱动程序mydev.c
2. 设备驱动模块的编译　　Makefile文件的使用
3. 加载设备驱动模块: insmod mydev.ko

若加载成功，在文件/proc/devices中能看到新增加的设备，包括设备名mydev和主设备号。

4. 生成设备文件: mknod /dev/test c 254 0

其中，test为设备文件名，254为主设备号，0为从设备号，c表示字符设备

```
ifneq ($(KERNELRELEASE),)
#kbuild syntax.
mymodule-objs :=test.o        //模块的文件组成
obj-m :=mymodule.o        //生成的模块文件名
else
PWD :=$(shell pwd)
KVER :=$(shell uname -r)
KDIR :=/lib/modules/$(KVER)/build
all:
        $(MAKE) -C $(KDIR) M=$(PWD)
clean:
        rm -f *.cmd *.o *.mod *.ko
endif
```

通用的Makefile

添加设备驱动程序（续）

- 编写应用程序，测试驱动程序

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int main()
{
    int testdev;
    int i;
    char buf[10];
    testdev =
      open("/dev/test",O_RDWR);
    if ( testdev == -1 )
    {
        printf("Cann't open file \n");
        exit(0);
     }
     read(testdev,buf,10);
     for (i = 0; i < 10;i++)
     printf("%d\n",buf[i]);
     close(testdev);
}
```

课程设计辅导

- Linux系统的相关知识
- 进程并发
- 添加系统调用
- 添加设备驱动程序
- /proc文件分析

/PROC文件分析

- proc文件系统
  - 进程文件系统和内核文件系统组成的复合体
  - 将内核数据对象化为文件形式进行存取的一种内存文件系统
  - 监控内核的一种用户接口，拥有一些特殊的纯文本文件，从中可以获取系统状态信息
    - 系统信息：与进程无关，随系统配置的不同而不同
    - 进程信息：系统中正在运行的每一个用户级进程的信息

/PROC文件分析

- 系统信息
  - /proc/cmd/line: 内核启动的命令行
  - /proc/cpuinfo: CPU信息
  - /proc/stat: CPU的使用情况、磁盘、页面、交换、所有的中断、最后一次的启动时间等
  - /proc/meminfo: 内存状态的有关信息
- 进程信息
  - /proc/$pid/stat
  - /proc/$pid/status
  - /proc/$pid/statm
  - ……

/PROC文件分析

- 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

模拟文件系统设计（选做）

- 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。

- 要求：写清楚设计思路、设计框架、设计方案等

课程设计辅导——参考资料

- 计算机操作系统实验指导（Linux版），郑然，庞丽萍编著，人民邮电出版社

- 其他各种网络、书籍资源