Linux复习提纲

- Linux概述
  - shell：交互式命令解释程序；用户和内核间交互的桥梁
  - Shell不仅是交互式命令解释程序，还是一种程序设计语言
  - shell是一种命令解释程序，批处理
  - shell是linux的外壳，默认是bash
  - 2.1 Linux基础概念
    - logout和exit、Ctrl -d注销终端
    - Linux是分时多用户系统，系统以账户为基础，构建了多用户工作环境
    - 存放系统用户信息的文件：/etc/passwd
    - 用户口令管理：/etc/shadow
    - 用户组信息：/etc/group
  - 2.2 Linux的启动
    - 7个运行级别
      - 0-停机
      - 1-用户模式
    - （一）内核引导
      - 核心线程init
        - 外设初始化
        - 打开/dev/console，重定向stdin，stdout，stderr到控制台
        - 使用execve()加载执行init程序
    - （二）完成内核引导后，开始执行init的进程号：1
    - （三）系统初始化
    - （四）启动对应级别的守护进程
    - （五）登录系统，启动完成
      - passwd和shadow文件匹配
  - 2.4 Linux的常用命令
    - 命令：可执行的Linux命令、程序、工具或shell脚本
    - 分类：
      - 内部命令：ls，cp，cd，pwd，date
        - 是Shell程序的一部分
        - 随shell装入内存
      - 外部命令：gzip，ftp，telnet
        - 实用程序
        - 运行时才需要加载到内存
    - 多种命令放在一行：实用；隔开
    - 多行命令使用 \ 隔开，\ 表示命令没有结束

- exit / logout / CTRL+d 注销终端，系统重新引导
- reboot 系统重新引导
- halt / shutdown 系统关闭
- poweroff 系统关闭并关闭电源
- **runlevel：显示系统当前和上一级的运行级别，不存在上一级就用N代替**
- **使用systemctl切换运行级别**
- init N：显示系统当前和上一级的运行级别，不存在上一级就用N代替
  - **使用init n也可以临时修改运行级别**
- passwd name：修改系统用户name的密码
- who：列出当前登录上操作系统的用户信心
  - -h
  - who am i：显示本终端注册的用户信息
- su name：切换系统用户，默认切换为root
- id：显示当前登录用户的uid和gid
- useradd：增加用户
  - -c加备注
  - -d指定主目录
  - -u指定用户ID号
- userdel：删除用户号
  - -r：删除此用户的所有文件
- cat：将指定的文件在标准输出上进行显示
  - -A：显示文件的控制符
  - -b：输出每一行的行号
  - -E：在每一行尾部显示结束标志
- more name：一次以一页显示文件
- less：与more类似，但是可以前翻后翻，更灵活
- pwd：显示用户当前的工作目录
- cd
- rm：删除目录
  - -r：包括子目录
- mv
- ls
  - -a：列出所有文件，包括隐藏文件
  - -l：以长列表的方式详细列出文件或目录信息
  - -la
  - 第一项为：drwxrwxrwx的形式
    - d：文件夹
    - -：普通文件

        - l：链接文件
        - 按照用户-同组用户-其他用户的顺序显示权限
    - 第二项：硬链接数量
    - 第四项为：组名
- chmod：修改指定文件或目录的权限
    - ugo：主user-同组-其他所有
    - chmod ug+rx file
    - chmod g-x file
    - chmod 777 file
    - chmod 700file
- cp [-ir] s d：文件或目录的拷贝
    - -i：提示是否覆盖已存在文件
    - -r：拷贝指定目录的全部内容
- mkdir [-P]
    - mkdir -P d1/d2/d3/d4 创建完整目录结构
- rmdir [-P]
- chgrp 组名 文件名：改变指定文件的所属用户组
- ln：默认硬链接
    - ln -f file1 file2 硬链接，改变ls第二项的值
    - ln -s file1 file2 符号链接
    - ln -f -s f1 f2：存在则覆盖，否则则创建
- cut：按列截取输入内容
    - -c：按字符截取
    - -f 按单词截取，默认为tab
    - -d 需要自己指定间隔符
    - cut -d : -f 1,3-4 file 首先按照：分割文件，然后选择1,3-4的域输出
- find：查找条件之间可以用-o和-a连接
    - -name：以文件名条件进行查找
    - -type x：查找类型为x的文件，x目前可以去的值有d（目录）、f（文件）
    - -user name：查找属于username的文件
    - -atime n：查找n天前被访问过的文件
    - -mtime n：查找n天前被修改过的文件
    - -print：显示找到文件的路径名称
    - -exec Command{}\; ：执行一个命令，必须用\;结束
    - find / \(-name core -o -name dump\) -atime +3 -exec rm {}\;
- grep：在指定文件中搜索特定内容
    - -c 打印匹配的行数
    - -i 模式不区分大小写
    - -l 只显示包含指定模式的文件名

    - -L 只显示不含特定模式的文件名
    - -n 同时显示行号
    - grep abc * 查找当前目录所有文件中与abc匹配的行
    - grep -n abc file.txt：在file中查找abc并显示行号
- **ps：查看当前运行级别最高的进程信息**
    - `ps aux`：显示所有正在运行的进程，包括系统进程和用户进程。
    - `ps -ef`：显示所有正在运行的进程，使用树形结构展示进程之间的父子关系。
    - `ps -u username`：仅显示指定用户名的进程信息。
    - `ps -p pid`：仅显示指定进程ID（PID）的进程信息。
    - `ps -o`：使用自定义格式输出进程信息，可以指定需要显示的字段和格式。
    - -e：显示每一个进程的信息
    - -f：显示父子进程关联信息，如ppid，执行时间，指令等
    - -l：详细信息
    - -x：没有占用控制台和终端的进程
    - -a：列出所有运行进程
    - 进程状态
        - RSDTZ：运行；睡眠；不可唤醒；停止；僵尸进程
- kill：终止进程
    - Ctrl+C
    - ***可以通过kill -l命令来查看所有信号**
- **nice：设置优先级或改变优先级**
    - -20最高，19最低
- df：磁盘使用情况-已使用空间、空闲空间等
    - Filesystem 1K-blocks Used Available Use% Mounted on
    - -k
    - -m
    - -T
    - -h
- du：查看文件夹或者文件占用内存的情况
- fg %123：作业转为前台运行
- bg %123：作业转为后台运行
- at：在指定时间执行作业程序
    - at 5pm + 2 weeks /bin/ls
    - at 23:59 12/31/2022 echo hello world!
- date：显示和设置系统时间
- cal：显示日历
- ftp：传输文件
- telnet：远程登录

        - ping：测试连通
        - mount：挂载文件系统
            - 访问usb文件系统：mount /dev/sdal /mnt/usb
            - **/etc/fstab是开机自动挂载的配置文件**
            - **etc/mtab是当前分区挂载情况，每次挂载都更新**
        - umount
        - fdisk：查看磁盘分区
        - man
        - touch: 修改文件访问和修改时间；创建文件
        - wc
            - `-l`：仅统计**文件的行数。**
            - `-w`：仅统计文件的**单词数。**
            - `-c`：仅统计文件的字符数。
            - `-m`：仅统计文件的字符数，但是会将多字节字符计算为多个字符。
            - `-L`：显示文件中最长行的长度。
    - 2.5 Linux主要目录解释
        - etc：系统的配置文件存放地址
        - lib：库文件存放地址
        - mnt：挂载文件系统
        - proc：进程信息
        - sbin：系统管理需要的可执行文件
        - bin：基础命令
        - boot：内核以及引导程序
        - dev：设备
    - 2.6 输入输出重定向
        - 文件组织方式按照字节进行
        - 文件之间设备之间的信息传递按字节流完成
        - 三个标准流
            - 标准输入 fd = 0 键盘
            - 标准输出 fd = 1 显示器
            - 标准错误 fd = 2 显示器
            - >>: append
            - >：改变标准输出流
            - <：改变标准输入流
        - 命令的分割
            - |：管道
            - ；多个命令顺序执行
            - & 将命令执行放到后台

        - && 和 || 逻辑
    - 获得磁盘空闲空间块数
        - df | head -2| tail -1 | tr -s "[ ]" | cut -d " " -f 4
        - tr -s "[ ]": 将输入中两个连续空格改为一个空格
        - tr -s "C" ： 将两个连续CC改为C
        - tr -d “C”： 删除所有C
    - 列出当前目录下占用磁盘空间最大的20个文件信息，并输入到tempfile中
        - du -a | sort -nrb | head -20 > tempfile
        - du -a：显示多个文件的磁盘占用
        - du -s：只显示特定文件的占用
        - sort
            - -n：以数值作为大小排序
            - -r：从大到小排序
            - -b：排序时忽略每行前面的空格和制表符
- 第三章：Shell程序设计
    - 3.3 环境变量
        - HOME
        - PATH
        - PWD：当前工作目录的绝对路径名
        - PS1：特权是#，普通是$
        - UID：用户标识符
        - ...
    - 位置变量
        - $0：文件名
        - 第一个参数从$1开始
        - 如果参数超过10，需要。。
    - 预定义变量
        - $#：参数个数
        - $*：参数列表
        - #?：前一个命令返回的状态
        - $$：当前进程的pid
        - $!：最近访问的后台进程的pid
    - 3.4 test 对文件，字符串，整数进行测试
        - 对文件的测试：
            - -e：存在
            - -r：可读
            - -w：可写
            - -x：可执行
            - -s：文件存在且至少有一个字符为真

        - -d：文件存在且为目录
        - -f：文件存在且为普通文件
        - -c：文件存在且为字符型特殊文件
        - -b：文件存在且为块特殊文件
    - 对字符串的测试
        - test s
        - test s1 = s2
    - 对整数的测试
        - test n1 [-eq/ne/lt/le/gt/ge] n2
    - 执行完test后可以通过echo $?获得判断结果
  - 3.5 条件控制语句
  - 向文件中写入时间：date >> file2
  - 文件拷贝：cat file1 >> file2
  - 获取当前时间（H）：hour=`date +%H`
  - 遍历所有.c文件：for file in *.[c]; do echoj ls -l $file
  - 3.8 函数定义
    - 参数$0-$9, $#参数个数，$@所有参数
    - 使用return返回数字；不指定return则返回状态码
    - 使用$?获取函数的执行结果
    - 函数内部声明变量默认为全局变量，使用local声明局部变量，局部变量可以覆盖同名全局变量*
  - 遍历a文件夹和b文件夹下的所有.c和.o文件：find /path/to/user/registration/a /path/to/user/registration/b (-name "*.c" -o -name *".o") -type f
  - 遍历a文件夹和b文件夹下的所有.c和.o文件，for循环实现：for file in /path/to/user/registration/{a,b}/*.{c,o}
  - 统计文件行数：cat file | wc -l
  - 将标准输入中的数据追加到指定文件中，如果该文件不存在则会创建一个新文件：cat >> file
- **进程管理**
  - fork
    - 返回两次，唯一的区别是子进程返回0，父进程返回pid
    - getpid()：获取pid
    - getppid()
  - exit：正常终止进程，把参数status返回给父进程
    - 0：正常退出
    - 非0：异常退出
    - 将一个正常的进程变成一个僵尸进程，并不能将其完全销毁
    - ps –al查看后，S为Z
      - D　不可中断睡眠 (通常是在IO操作)
      - R　正在运行或可运行（在运行队列排队中）
      - S　可中断睡眠 (在等待某事件完成)

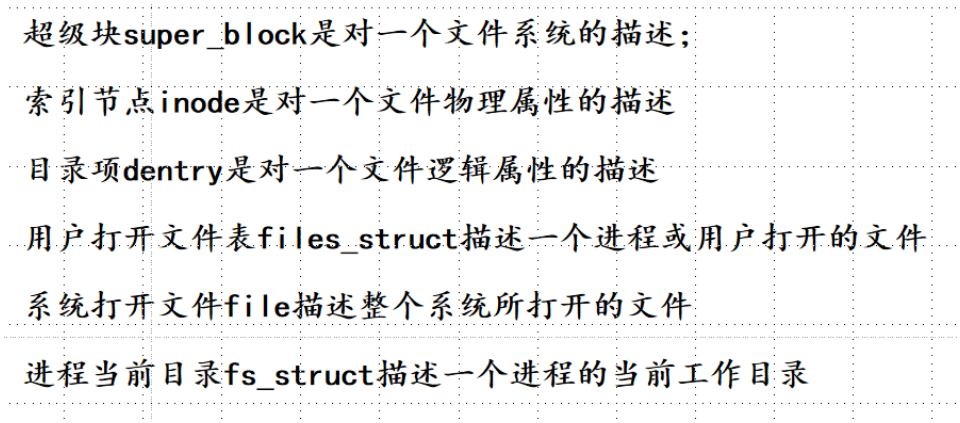
        - T　trace状态
        - W　正在换页(2.6.内核之前有效)
        - X　死进程 (should never be seen)
        - Z　僵尸
    - waitpid调用和wait调用。这两者的作用都是收集僵尸进程留下的信息，同时使这个进程彻底消失。
- _exit：立即退出
    - 传递SIGCHLD信号给父进程，父进程可以由wait函数取得子进程结束状态
- execv、execl...
    - 用新程序的程序映像覆盖进程原来的程序映像。新程序文件名由参数path或file给出，它的程序代码将替代原来的程序代码被执行
- wait：**暂时停止目前进程的执行**，直到有信号来到或子进程结束
    - pid_t wait(int *status)
    - 如果不在意结束状态值，参数 status 可以设成 NULL
    - 子进程的结束状态值会由参数 status
- waitpid
    - pid_t waitpid(pid_t pid, int *status, int options);
    - 参数 status 可以设成 NULL。参数 pid 为欲等待的子进程识别码
    - pid<-1 等待进程组识别码为 pid 绝对值的任何子进程。
    - pid=-1 等待任何子进程，相当于 wait（）。
    - pid=0 等待进程组识别码与目前进程相同的任何子进程。
    - pid>0 等待任何子进程识别码为 pid 的子进程。
    - 参数 option 可以为 0 或下面的 OR 组合:WNOHANG 如果没有任何已经结束的子进程则马上返回, 不予以等待。WUNTRACED 如果子进程进入暂停执行情况则马上返回,但结束状态不予以理会。
    - 子进程的结束状态返回后存于 status,底下有几个宏可判别结束情况:
        - WIFEXITED(status)如果子进程正常结束则为非 0 值。
        - WEXITSTATUS(status)取得子进程 exit()返回的结束代码,一般会先用 WIFEXITED 来判断是否正常结束才能使用此宏。
        - WIFSIGNALED(status)如果子进程是因为信号而结束则此宏值为真
        - WTERMSIG(status) 取得子进程因信号而中止的信号代码,一般会先用
        - WIFSIGNALED 来判断后才使用此宏。
        - WIFSTOPPED(status) 如果子进程处于暂停执行情况则此宏值为真。一般只有使用 WUNTRACED 时才会有此情况。
        - WSTOPSIG(status) 取得引发子进程暂停的信号代码,一般会先用 WIFSTOPPED 来判断后才使用此宏。
- int system(const char * string);
    - system（）会调用 fork（）产生子进程
- pid_t getpgid(pid_t pid);
    - 取得参数 pid 指定进程所属的组识别码。

        - 如果参数 pid 为 0，则会取得目前进程的组识别码。
    - pid_t getpgrp(void);
        - 取得目前进程所属的组识别码
    - int setpgid(pid_t pid,pid_t pgid);
    - int setpgrp(void);
- **进程间通信**
    - 信号量
        - 信号是进程中**异步发生事件**时发出的提示信息或传送给进程的一种事件通知，是UNIX操作系统用来通知进程发生了某种事件的一种手段。
        - 信号也可以用于进程之间进行通信和实现进程同步处理
        - 某些系统调用将产生信号：kill，raise
            - SIGTERM
            - **可以通过kill -l命令来查看所有信号**
        - 当用户在终端按下某些键时，产生终端生成的信号
        - 硬件例外会产生信号
        - 发生了某种必须让进程知道的情况时也会生成信号
        - 系统对信号可以采用3种处理方式
            - 忽略
            - 调用默认动作
            - 捕获信号
        - 信号的产生，可以使用三个系统调用
            - int raise(int sig);
            - unisigned int alarm(unsigned int seconds);
            - **int kill(pid_t pid，int sig);**
                - pid>0 将信号传给进程识别码为pid 的进程
                - pid=0 将信号传给和目前进程相同进程组的所有进程
                - pid=-1 将信号广播传送给系统内所有的进程
                - pid<0 将信号传给进程组识别码为pid绝对值的所有进程
        - 子进程的结束状态返回后存于status，底下有几个宏可判别结束情况
            - **WIFEXITED(status):如果子进程正常结束则为非0值。**
            - **WEXITSTATUS(status):取得子进程exit()返回的结束代码。一般会先用 WIFEXITED 来判断是否正常结束才能使用此宏。**
            - **WIFSIGNALED(status):如果子进程是因为信号而结束则此宏值为真。**
            - WTERMSIG(status):取得子进程因信号而中止的信号代码，一般会先用 WIFSIGNALED 来判断后才使用此宏。
            - WIFSTOPPED(status):如果子进程处于暂停执行情况则此宏值为真。一般只有使用 WUNTRACED 时才会有此情况。
            - **WSTOPSIG(status):取得引发子进程暂停的信号代码。**
        - void (*signal(int sig,void (*func)(int)))(int)

        - 在有些情况下，进程不允许随意被打断。因此对执行中的进程进行信号捕获和设定特殊的处理是非常必要的
    - 进程正确的执行,而不想进程受到信号的影响：**信号屏蔽**
        - 信号集sigset_t
  - 信号量
    - 用以控制进程之间的同步控制，用来对资源做锁定的工作；使进程间共用一个计数值
    - 信号量与信号的区别
  - 管道：提供进程之间单向通信
    - 先进先出（FIFO）的特殊文件
    - 管道的两个文件描述符和普通文件一样，只是内核的inode中记文件类型为特殊文件-管道文件
    - 创建两个文件描述符fd[0]和fd[1]，其中fd[0]固定用于读管道，而fd[1]固定用于写管道，这样就构成了一个半双工的通道
    - 无名管道只能用于具有亲缘关系的进程之间的通信
    - 管道内的数据只能被读出1次。管道不允许进行文件的定位操作，读和写操作都是顺序的。
    - 无名管道可以通过调用pipe()来实现，关闭用close().
    - 有名管道有文件名，可以用ls –l查看，其**文件属性为p**
    - 有名管道和无名管道的对比
  - 消息队列
    - 消息队列不是存放在磁盘，是**存放在内存中**的数据对象
    - 消息队列一旦创建，内核中的数据对象就存在，即使创建消息队列的进程终止，消息队列仍旧存在
    - 能将格式化的数据送往任意的进程
    - 消息队列主要由三部分组成：消息队列表、消息头、消息文本
    - 多个进程之间的通信数据必须经过内核复制，当数据量极大时不如共享内存
  - 共享存储
    - 为了在多个进程间交换信息，内核专门留出了一块内存区。这段内存区可以由需要访问的进程将其映射到自己的私有地址空间。进程就可以直接读写这一内存区而不需要进行数据的拷贝，从而大大提高了效率
    - **共享存储提供了进程间共享数据的最快途径**
    - 由于多个进程共享一段内存，因此也需要依靠某种同步机制，如互斥锁和信号量等。
- **文件管理**
  - 程序在开始读写一个文件的内容之前，必须首先在程序与文件之间**建立连接**或**通信通道**
    - 文件描述符
      - 文件描述符表示为int类型的对象
      - 对**特定设备**进行控制操作，必须使用文件描述符方式，没有函数能对流进行这类操作
      - 需要按照特殊的方式进行I/O操作（例如非阻塞的方式），必须使用文件描述符方式
      - 进行文件读写时，系统会自动打开三个标准文件描述符是0、1、2，代表标准输入流、标准输出流，标准错误流

        - read、write、open等I/O系统调用在默认情形下均是阻塞的
            - 有两种方法指定非阻塞I/O：
                - 调用open时指定O_NONBLOCK文件状态标志
                - 对已经打开的描述字，可以调用fcntl函数打开O_NONBLOCK文件状态标志
        - lseek函数：移动文件指针位置
        - tell函数： 报告当前文件的指针位置
        - lockf：文件锁和解锁
        - dup函数：复制文件描述字old至一个新**描述字**
            - 重复一文件描述字的主要用途是实现输入或输出重定向，即改变一特定文件描述字对应的文件或管道
        - dup2函数：复制描述字old至编号为**new的描述字**
            - dup2(fd,STDOUT_FILENO)，fd是标准输出的描述字，printf可以输入到文件中
        - fcntl函数：对打开的文件描述字执行各种控制操作
            - 重复一个文件描述字，查询或设置文件描述字标签，查询或设置文件状态标签，操纵文件锁等。
    - 流
        - 流则表示为指向结构FILE的指针FILE* ，因此流也称为“文件指针”
        - **流函数是通过文件描述符函数来实现的**
        - FILE对象用来描述打开的流的**所有内部状态信息**；
        - **所有的流输入输出函数都使用FILE对象来对流进行操作**
        - FILE对象的内容：
            - 进行实际I/O的文件描述符
            - 文件位置指针
            - 缓冲区大小及指针
        - c=fgetc(stdin)；字符输入
        - **fgets(buf,sizeof(buf),stream); 字符串输入一行**
        - fputs(“Hello world”,stream);
        - **fputs(question,stdout); 字符串输出一行**
        - fputs(“Please answer y or n:”,stdout);
        - fputc(‘ ’,stdout); 字符输出
        - 每一个流对象内部都保持着两个指示器：
            - 一个为错误指示器，当读写文件出错时该指示器被设置；
            - 另一个为文件结束指示器，当遇到文件尾时该指示器被设置。
            - ferror和feof两个函数分别对这两个指示器进行检查。
        - 格式化I/O：sprintf、sscanf
        - tmpnam返回一个与系统中已经存在的文件名不相同的临时文件名
- **文件系统**
    - 文件结构是文件存放在磁盘等存储设备上的组织方法，体现在对**文件和目录**的组织上

- Linux使用标准的**目录结构-树形结构**，无论操作系统管理几个磁盘分区，这样的**目录树只有一个**
- 文件系统：文件存在的物理空间。Linux中每个分区都是一个文件系统，都有自己的目录层次结构
- Linux文件系统使用**索引节点来记录文件信息**
- **文件系统通过索引节点号识别一个文件**，指向位于任意一个文件系统的任意文件，甚至可以指向一个不存在的文件。
- 硬链接和符号链接
- 将一个文件系统的顶层目录挂到另一个文件系统的子目录上，使它们成为一个整体，称为**安装或挂载（mount）**。把该子目录称为"安装点或挂载点（mount point）
- **安装一个文件系统可以用mount命令**
- **EXT2**是Linux的标准文件系统，系统把它的磁盘分区作为系统的**根文件系统**
- linux的主要文件类型
  - 常规文件：文本文件和二进制文件
  - 目录文件：将文件的名称和它的索引节点号结合在一起的一张表
  - 设备文件：每种I/O设备对应一个设备文件
  - 管道文件：主要用于在进程间传递数据，又称为**先进先出（FIFO）文件**。
  - 链接文件：又称符号链接（软链接）文件，提供了共享文件的一种方法。
- Linux对文件的访问设定了**三类权限**：
  - 文件所有者、与文件所有者同组的用户，其他用户。
- 对文件的访问主要是**三种操作**：读取、写入和执行
- 为了支持其他各种不同的文件系统，Linux提供了一种统一的框架，就是**虚拟文件系统**
  - 三层：应用层、虚拟层、实现层
  - VFS数据对象：
    - 超级快：存放系统中已安装文件系统的有关信息
    - 索引节点：存放关于具体文件的一般信息
      - Inode对文件是唯一的。在同一个文件系统中，文件名可以更改，索引节点号是唯一的
      - inode结构代表的是物理意义上的文件，记录的是物理上的属性，对于一个具体的文件系统，其inode结构在磁盘上就有对应的映像
    - 目录项：存放目录项与对应文件对应链接的信息
      - dentry结构代表的是逻辑意义上的文件，描述的是文件逻辑上属性，目录项对象在磁盘上并没有对应的映像
    - 文件对象：存放打开文件与进程之间进行交互的有关信息
      - 进程是通过文件描述符来访问文件的
      - Linux中专门用了一个file文件对象来保存打开的文件与进程的交互信息，这些信息仅当进程访问文件期间才存于主存中。
      - 每个进程用一个**files_struct结构**来记录文件描述符的使用情况，files_struct称为用户打开文件表，它是进程的私有数据。
      - 每个进程都有一个当前工作目录和当前工作目录所在文件系统的根目录

        - 每个进程用**fs_struct结构**来记录进程的**当前工作目录**
    - 系统打开文件表是由file对象组成的链表，它是内核态，由内核控制
    - 用户打开文件表是文件描述符表，在进程用户空间，是进程的私有数据
- 

    - 文件系统的注册和注销：通过**insmod/rmmod命令**在装入该文件系统模块时向VFS注册/注销
    - 文件系统的缓存机制：
        - VFS inode 缓存
            - VFS提供缓存，把当前使用的inode保存起来，同时采用散列技术，以快速找到所需的inode。
        - VFS目录高速缓存
        - 页面高速缓存
- EXT2文件系统
    - 一个分区的容量最大可达4TB
    - 文件大小限制为2GB
    - 使用变长的目录项，支持255个字符的长文件名，可扩展到1012个字符
    - 使用位图来管理数据块和节点的使用情况
    - Ext2将磁盘分区看成是由块组组成，每个块组包含一个或多个块；每个块组大小相同，且顺序存放
- proc文件系统
    - /proc文件系统不是普通意义上的文件系统，它是一个伪文件系统
    - 通过proc文件系统可以了解进程的地址空间信息、系统硬件信息、系统的中断信息以及系统的I/O信息等
    - 大多数文件是只读的，用户只能从这些文件中读取内核中的信息。**有一些proc文件可写**，用户通过写这些proc文件将参数传递给内核

```
proc 文件系统可以被用于收集有用的关于系统和运行中的内核的信息。下面是一些重要的文件：
/proc/cpuinfo : CPU 的信息（型号，家族，缓存大小等）
/proc/meminfo : 物理内存、交换空间等的信息
/proc/mounts : 已加载的文件系统的列表
/proc/devices : 可用设备的列表
/proc/filesystems : 被支持的文件系统
/proc/modules : 已加载的模块
/proc/version : 内核版本
/proc/cmdline : 系统启动时输入的内核命令行参数
```

    - /proc文件系统中，代表各个文件节点的是**proc_dir_entry**结构
    - 初始化：建立 proc文件系统树
    - 创建了由proc_dir_entry结构形成的proc文件系统树
    - 内核从proc文件系统树中读取该文件inode信息
    - proc文件系统的超级块也和普通文件系统的超级块不同，**它并不需要从硬件中获得超级块的数据，而是在内核启动时直接初始化超级块数据**，从而完成系统对proc文件系统操作函数的初始化过程。
- **内核模块**
    - 模块（module）是一段可以被动态链接的目标代码（.ko），它可由**insmod**命令动态的装载并链接到正在运行的内核。链接后，它就成了内核的一部分，直到用**rmmod**命令解除链接并卸载
    - **可以被单独编译，但不能独立运行**
    - 需要提供**入口函数和出口函数**
    - 使用Linux内核里定义的函数，不能使用glibc的库
    - 这些函数可以用来完成**硬件访问**等操作
    - 内核模块工作在**内核空间**，而应用程序工作在用户空间
    - 由多个回调函数组成的“**被动**”代码集合体，采用了“**事件驱动模型**”；而应用程序总是从头至尾的执行单个任务。
    - 内核模块可使用的栈很小(一般只有4096字节)
    - 入口：module_init()
        - 通过insmod或modprobe命令加载内核模块时，模块的加载函数会自动被内核执行
    - 出口：module_exit()
        - 当通过rmmod命令卸载某模块时，模块的卸载函数会自动被内核执行
    - 编译：gcc –c –D_KERNEL_ -DMODULE
    - 链接：insmod
    - MODULE_LICENSE()---模块许可证声明

头文件 #include <linux/init.h> #include <linux/module.h>	必选
许可声明 MODULE_LICENSE("Dual BSD/GPL");	必选
加载函数 module_init( )	必选
卸载函数 module_exit( )	必选
模块参数 module_param(num, int, S_IRUGO)	可选
模块导出符号 EXPORT_SYMBOL(add_integer)	可选
模块作者等信息声明 MODULE_AUTHOR(“author_name”)	可选

- 设备驱动程序
  - 所有设备控制操作都由与被控制设备相关的代码来完成，这段代码就叫做设备驱动程序。
  - 设备驱动程序就是外部设备的软件抽象，或者说是软件表现，是系统看到的设备，是虚拟的设备
  - 字符设备
  - 块设备
    - 块设备的接口必须支持挂载(mount)文件系统
  - 网络接口设备
  - **file结构**是在<linux/fs.h>中定义的一个数据结构，用**来代表一个打开的文件**（包括设备文件和普通文件）
  - **file_operations结构**是一个定义在<linux/fs.h>中的函数指针数组。每个文件都通过file结构中的f_op字段与它自己的函数集相关联。这些函数负责系统调用的实现，而这个结构负责系统调用的映射。
  - **设备类型和主设备号唯一标识驱动程序**，次设备号是由主设备号确定的驱动程序来使用的
  - 设备文件节点：应用程序看到的设备和文件是一样的，他们都是文件系统上的一个节点，有着相同的接口。
    - **mknod 创建设备文件节点**：mknod /dev/mydev1 c 254 0
    - mydev1就是设备的名字，它是/dev目录下的一个文件节点，c表示字符设备，254是主设备号，0是次设备号

驱动程序步骤step123

step1:设计驱动程序

① 实现各个设备文件操作函数

② 声明函数指针数组

③ 完成初始化函数，并向系统注册。完成清除函数。

step2:建立设备文件节点

```
mknod /dev/testdev   c    xxx    0
```

step3:设计测试应用程序

step4:加载驱动程序

```
insmod testdev.o
```

step5:运行应用程序进行测试

step6:卸载驱动程序

```
rmmod testdev
```