01 - shell操作相关快捷键

shell - - unix -- Bourne shell

bash -- linux -- Bourne again shell

命令解析器作用:

解释用户输入的命令, 执行对应的操作

虚拟终端

输入命令

* 去指定路径搜索用户输入的命令
* 环境变量 -- 路径

shell解析你输入的命令

搜索环境变量：路径

找到了输入的命令，运行这个程序

1. 命令或目录补齐

终端下有对应的输出

a. 快捷键: tab

1. 遍历历史记录相关

* history
* 上 == ctrl+p
* 下 == ctrl+n

1. 光标移动

* 左 -- ctrl+b
* 右-- ctrl+f
* 移动都头部: ctrl+a
* 尾部: ctrl+e

1. 字符删除

* 退格 -- ctrl+h -- 删除光标前边的字符
* 删除后边的字符: ctrl+d = del
  + 光标覆盖的字符
* 删除光标前的字符串: ctrl+u
* 删除光标后的字符串: ctrl+k

快捷键：

遍历历史记录： 向上：ctrl+p

向下：ctrl+n

删除：

删除光标前边字符：backspace， ctrl+h

。。。。后边： ctrl+d

删除。。。前的字符串：ctrl+u

。。。。。后。。。。：ctrl+k

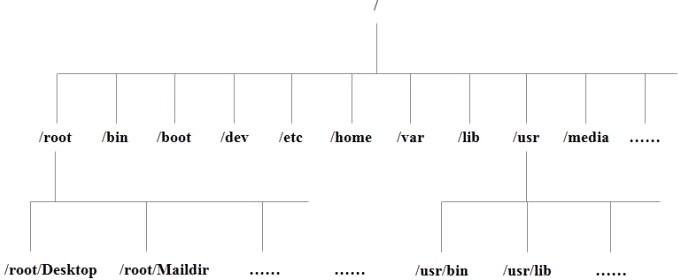
移动：

向前：ctrl+b 向后：ctrl+f

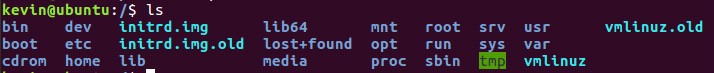
移动到头部：ctrl+a 尾部：ctrl+e

分区 01 - Linux常用命令 的第 3 页

02 - linux系统目录结构



1. 根目录表示方式: /
2. 根目录下常见目录



* + /bin: binary， 二进制文件，可执行程序，shell命令
  + /dev: device，在linux下一切皆文件
    - 硬盘，显卡，显示器。。。
  + /lib: linux运行的时候需要加载的一些动态库
  + /mnt: 手动的挂载目录
  + /media: 外设的自动挂载目录
  + /root: linux 的超级用户的家目录
  + /usr: unix system resource
    - 头文件- stdio.h stdlib.h
    - 游戏
    - 用户安装的应用程序 /usr/local
  + /etc: 存放配置文件
    - /etc/passwd
    - /etc/group
    - man 5 passwd
  + /opt: 安装第三方应用程序
  + /home: linux操作系统所有用户的家目录
    - 用户家目录：( 宿主目录）
      * /home/kevin
  + /tmp: 存放临时文件

03 - 相对路径和绝对路径

用户当前工作目录:

$: 当前用户为普通用户

#: 当前用户为超级用户

1. 相对路径: 从当前的目录开始表示
   * 使用相对路径进入到food目录

./zoo/animal/food

1. 绝对路径: 从根目录/开始表示的路径
   * 使用绝对路径进入到foo的目录

/home/kevin/demo/1Day/zoo/aniaml/food

3. . 和..

* + . 当前目录
  + .. 当前目录的上一级目录cd ../abc

4.

* + kevin: 当前登录的用户
  + @： at， 在
  + ubuntu：安装的时候指定的主机名
  + ~：用户的家目录（宿主目录）
  + ~/demo/1Day: 当前用户的工作目录

04 - 文件目录相关命令

1. tree

○

○

○

○

1. ls

○

○

○

查看目录的内容

tree 查看当前目录

tree dir 查看指定目录

需要安装： sudo apt-get install tree

功能：查看文件或目录语法：

参数:

* + -a 显示所有文件

□

* + -l

□

□

隐藏文件： 文件或目录名前边有一个点

-rwxrw-r-- 1 kevin kevin 3231145 11月 23 23:08 vimplus.tar.gz 第一个字符：文件的类型

* 7种
* 普通文件： -

◊ .txt 压缩包可执行程序

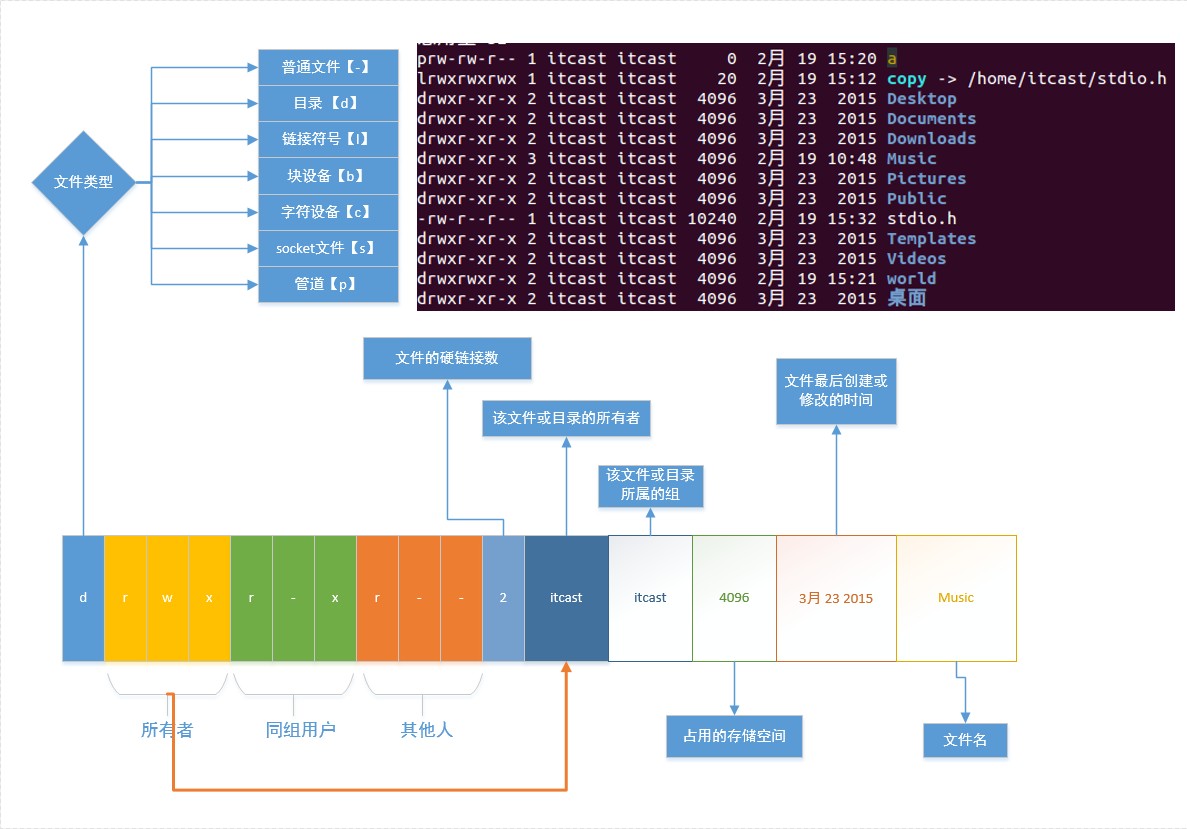
* 目录： d
* 符号链接： l
* 管道： p
* 套接字：s
* 字符设备：c

◊ 键盘，鼠标

* 块设备：b

◊ U盘，硬盘

* + - rwx： 文件所有者权限
    - rw-： 文件所属组用户的权限
    - r--：其他人对文件的操作权限
  + 1： 硬链接计数
  + kevin： 文件所有者
  + kevin： 文件所属组
  + 3231145 ：文件的大小
    - 如果是目录： 4k ，目录本身的大小，不包括里边的内容



* + 11月 23 23:08：日期
  + vimplus.tar.gz：文件名

1. cd -- 切换目录
2. cd 目录
3. 如何进入到家目录:
   * cd 绝对路径（/home/kevin）
   * cd ~
   * cd
4. 在临近的两个目录直接切换:
   * 最后两个相邻的
   * cd -
   * 目录特别长的时候用的比较多
5. pwd - 显示当前的工作目录
6. mkdir

* mkdir 目录名
* -p：创建多级目录

1. touch

* touch 文件名-





1. rmdir

如果不存在， 创建文件，空文件文件存在：更新文件的时间

○

○

1. rm

○

rmdir 空目录的名字矮穷矬

目录

* + rm 目录名-r
    - -r 递归的方式处理
  + 删除的时候提示-i
* 文件
  + rm 文件名-i
* 注意问题：
  + 删除之后，很难恢复

1. cp -- 拷贝

* cp 要拷贝的文件（file1） file（不存在）
  + 创建file，将file1中的内容拷贝到file
* cp file1 file（存在）
  + file1覆盖file
* cp file dir（存在）
  + 拷贝file到dir目录
* cp dir（存在） dir1（存在）
  + 将dir目录拷贝到dir1目录中
  + 包括dir目录
* cp dir（存在） dir1（不存在）
  + 创建dir1
  + 将dir中的内容拷贝到dir1中， 不包括dir目录

1. mv - 改名或者移动文件

mv file1 file2

* 改名
  + mv file（存在） file1（不存在） -- file为文件改名
  + mv dir（存在） dir1（不存在） -- dir为目录
    - 改名
* 移动
  + mv file（文件） dir（存在目录）
    - file-》dir中
  + mv dir（目录存在） dir1（目录存在）
    - dir移动到dir1中
* mv file（存在） file2（存在）
  + file文件覆盖file2文件
  + file文件会被删掉

1. 查看文件内容 -- 了解内容

* cat





cat filename

适用于文件比较小的情况

* more
  + more filename
  + 回车：向下浏览一行
  + 空格：翻页，向下
  + 退出：q
* less
  + less filename
  + 向下滚动一行：
    - 回车，ctrl+n
  + 向上滚动一行：ctrl+p
  + 向下翻页：
    - 空格，pagedown
  + 向上翻页：pageup
  + 退出：q
* head - 从文件的头部显示指定的行数







* tail

head -n 文件名

n不写，默认10行n显示的行数

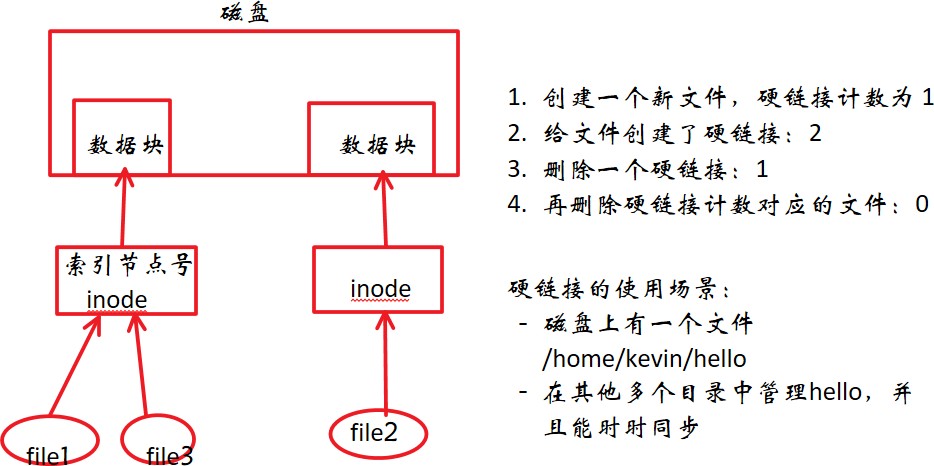
1. 软硬链接 ln

* 软连接 -- 快捷方式
  + ln -s 文件名快捷方式的名字

□ 文件名：使用绝对路径

* + 目录也可以创建软连接
* 硬链接

ln 文件名 硬链接的名字



-- 文件所有者 - 文件所属组 -- 其他人

程序猿

斯 托 曼 比尔盖茨乔 布 斯 苏丙榅

明星

王宝强陈思成林丹

三种类型的用户:

西瓜

文件所有者：user

文件所属组用户：group

其他人：other

路人甲 -- user

路人乙- 文件所属组用户- group 路人丙- group

吃瓜群众

05 - 用户权限, 用户和用户组

1. 修改文件或目录权限 -- chmod
   * 文字设定法
     + chmod who[+|-|=]mode 文件名
       - who:
         * u - user，文件所有者
         * g - group，文件所属组
         * o - other，其他人
         * a - all ， 所有的（默认）

□ +|-|=

* + - * mode
        + r：读
        + w：写
        + x：执行
        + -：没有任何权限
    - 题目: rwxrwxrwx -- file
      * 文件所有者和其他人 减去读写权限
        + 所有者：u
        + 其他人：o
        + 读：r
        + 写：w
        + 减去： -
      * 所有者添加读权限，同组用户减去执行权限
        + 所有者：u + r
        + 同组用户：g-x
  + 数字设定法
    - chmod [+|-|=]mode 文件名
      * mode：八进制的数 0777

0756

 r：4

 w：2

 x：1

* 0 : 八进制
* 7: rwx - 文件所有者
* 5: r-x 文件所属组





* + - 题目:

w：2

x：1

-：0

* 5: r-x 文件所属组
* 6: rw- 其他人
  + --xrwx--x 1 itheima itheima 0 10月 18 15:51 file
    - 所有者和同组用户的权限设置为-wx

◊ wx：3

* + - 其他人只有执行权限 1
  + 文件权限: 777 给所有者和所属组减去r
    - chmod -440 file

1. 修改文件所有者或所属组- chown
   * chown 新的所有者文件名
   * chown 新的所有者：新的组文件名
2. 修改文件所属组-- chgrp
   * chgrp 新的组文件名

06 - 文件查找和检索

1. 根据文件属性查找- find
   * 文件名
     + find 查找的目录-name “查找的文件名”
   * 文件类型
     + find 查找目录 -type 文件类型
       - 普通文件：f
       - 目录：d
       - 符号链接：l
       - 管道：p
       - 套接字：s
       - 字符设备：c
       - 块设备：b
   * 文件大小
     + find 查找目录-size -10M

 +：大于+10k

* + - -：小于-10k
    - 等于10k：10k
    - 单位：
      * k - 小写
      * M - 大写
    - 大于10k小于100k
    - find . -size +10k -size -100k
  + 按日期
    - 创建日期：-ctime -n/+ n
      * -n: n天以内

□ +n：n天以外

* + - * find . -ctime -1
    - 修改日期：-mtime -n/+n
    - 访问日期：-atime -n/+n
  + 深度
    - -maxdepth n(层数）
      * 搜索n层以下的目录
    - -mindepth n（层数）
      * 搜搜n层以上的目录
  + 高级查找

例：查找指定目录，并列出该目录中文件详细信息

* + - find ./ -type d -exec shell 命令（ls -l）{} \;
    - find ./ -type d -ok shell 命令（ls -l）{} \;
      * ok比较安全
    - find ./ -type d | xargs ls -l(shell 命令)

1. 根据文件内容查找- grep w r
   * grep -r（有目录） “查找的内容” 搜索的路径
   * 搜索家目录中带helloworld字符串的文件

grep -r “helloworld” ~ -n（显示行号）

1. 总结:

搜索的内容

搜索的路径

搜索的路径

搜搜的内容

* + find 参数
  + grep 参数

07 - 压缩包管理

1. linux下常见压缩格式:
   * .gz -- gzip
   * .bz2 - bzip2
2. 常用压缩命令:
   * tar - 打包
     + 参数：
       - c - 创建压缩文件
       - x - 释放压缩文件
       - v - 打印提示信息（可不写）
       - f - 指定压缩包的名字
       - z - 使用gzip压缩文件 - xxx.tar.gz
       - j - 使用bzip2的方式压缩文件 -- xxx.tar.bz2
     + 压缩：
       - tar 参数压缩包的名字原材料 -- gz

 tar zcvf test.tar.gz file dir

* + - 解压缩

 tar zxvf test.tar.gz -C 解压目录

* + rar
    - rar需要安装
      * sudo apt-get install rar
    - 压缩：

rar a 压缩包名（不用指定后缀） 压缩内容

* + - * 压缩目录加参数-r
    - 解压缩

rar x 压缩包名解压目录

* + zip/unzip
    - 压缩：

zip 参数压缩包名原材料

* + - * 如果有目录： -r
    - 解压缩：

unzip 压缩包的名字-d 解压目录

1. 总结
   * 压缩：

tar/rar/zip 参数压缩包名 原材料

* + 解压缩

tar/rar/unzip 参数压缩包名参数解压路径

* + - rar 解压缩到指定目录不需要指定参数
    - unzip 不需要解压参数

08 - 软件的安装和卸载

1. 在线安装-- ubuntu apt-get

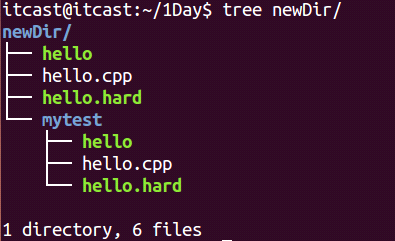
* 安装: sudo apt-(apt) install 软件名字(tree, rar)
* 卸载: sudo apt-get revove 软件名
* 软件列表的更新: sudo apt-get update
* 清空缓存:

1. 软件包安装(Ubuntu下.deb格式)
   * 安装:
   * 卸载:

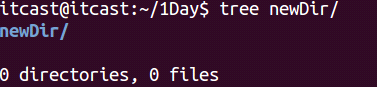
第一天作业

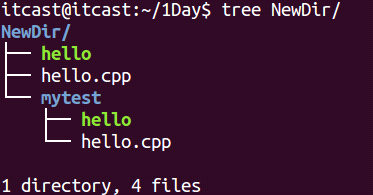
1. 说出Linux下的目录结构
2. 写出下列目录的作用:
   * /dev
   * /etc:
   * /home
   * /usr
   * /bin
   * /lib
   * /mnt
3. itcast@itcast:~/SoftWare/QQ$, 从该字符串中提炼出你所知道的信息。
4. 通过ls -l 命令得到以下文件的详细信息，请写出各个部分代表的意思？
5. 创建test目录，在里面创建aa bb cc三个目录，在aa里创建hello文件，在bb里创建world目录，在cc里创建itcast.c,然后执行tree和ls -R,最后删除test。

要求所有的操作都在同一级目录中完成。

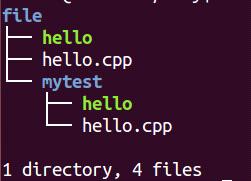
1. 如图：

使用rm命令删除newDir中所有的内容, 使其成为一个空目录.



1. 在指定目录下内容如图所示:

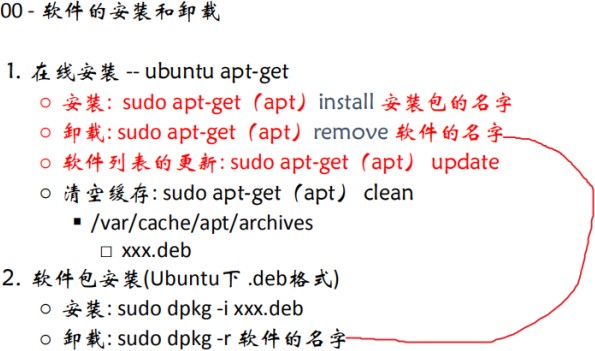
将NewDir目录中的内容拷贝到另一个已经存在的空目录(file)中, 效果如图:



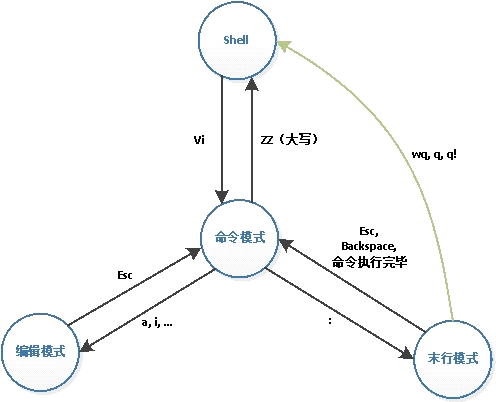
1. 如何给文件创建软连接(举例), 有什么注意事项?
2. 现有一个文件hello.txt, 文件访问权限为 rwxrw-rw-
   * 使用文字设定法将文件所属组和其他人添加执行权限x
   * 使用数字设定法去掉文件所有者的读写权限
3. 写出命令查找宿主目录下文件大于100k小于10M的文件， 并将查到文件的详细信息保存到另一个文件file中。
4. 写出命令查找宿主目录下文件中含有"hello, world"字符串的文件，并显示行号。
5. 写出命令卸载Linux中卸载的tree软件(两种方式)
6. 当前目录下有 a, b, c 三个文件和一个目录Dir
   * 分别使用 tar, rar, zip命令将三个文件和目录压缩为一个文件.
   * 将压缩好的文件解压缩到/home/itheima目录中

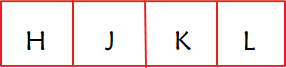
学习目标

1. 掌握vim命令模式下相关命令的使用
2. 掌握从命令模式切换到编辑模式的相关命令
3. 掌握vim末行模式下相关命令的使用
4. 能够说出gcc的工作流程和掌握常见参数的使用
5. 熟练掌握Linux下的静态库的制作和使用
6. 熟练掌握Linux下的共享库的制作和使用



01 - vim相关

1. 需要自己安装vim
   * shell敲命令：vimtutor
2. vim的三种工作模式
3. 命令模式
4. 编辑模式
5. 末行模式
6. vim命令模式下的相关操作
   * 保存退出: ZZ
   * 代码格式化: gg=G
   * 光标的移动
     + 上下左右:



* + - 光标移动到行首: 0（零）
    - 行尾: $
    - 光标移动到文件首部: gg
    - 文件尾部: G
    - 行跳转123行: 123G
      * 行号n回车: 当前行向下移动n行
  + 删除命令
    - 删除字符:
      * 光标前的字符：X
      * 。。后。。。：x
    - 单词: dw
      * 删除整个单词光标应该在单词的最前边
    - 删除光标前字符串: d0
    - 删除光标后 d$(D)
    - 删除行: dd, 删除光标所在行
    - 删除多行: ndd
      * n行数
  + 撤销和反撤销
    - 撤销: u
    - 反撤销: ctrl+r
  + 复制和粘贴
    - 复制:
      * 1行: yy
      * 多行: nyy
    - 粘贴:
      * p: 粘贴到光标的下边一行
      * P：。。。。上边一行
  + 可视模式：v
    - 移动光标:
      * hjkl
    - 复制: y
    - 删除: d
    - 粘贴:
      * p：粘贴到光标的后边
      * P：粘贴到光标的前边
  + 替换操作
    - r：替换一个，光标盖住的字符
    - R：替换多个，从光标盖住的这个往后替换
  + 查找命令
    - /xxx
    - ?xxx

□

* + - #:

□

关键字切换：n/N

光标移动到待搜索关键字上边，键盘输入#

* + 查看man文档
    - 章节号+ K（shift+k）

1. 命令模式切换到文本编辑模式
   * a：从光标后开始插入字符
   * A：行尾
   * i：光标前插入
   * I：行首
   * o：光标下创建新行
   * O：。。。上
   * s：删除光标盖住的字符
   * S：删除光标所在行
2. vim末行模式下相关操作

a. 从命令模式 -> 末行模式

1. 键盘录入冒号（：）
   * 保存退出
     + 保存不退出：w
     + 退出：q
     + 退出不保存：q！
     + 保存退出：wq == x
   * 替换
     + 替换光标所在行的字符串:
       - :s/old/new/gc
         * g: 替换当前行所有的old
         * c：替换的时候添加提示信息
     + 替换一个范围
       - :x,ys/old/new/gc
         * xy是一个范围（行数）
     + 替换当前文档所有的
       - :%s/old/new/gc
   * 分屏操作
     + 当前文件分屏
       - 水平：sp
       - 垂直：vsp
     + 两个屏幕显示不同的文件
       - 水平：sp 文件名
       - 垂直：vsp 文件名
     + 屏幕的关闭:
       - 关闭所有：qall
       - 保存关闭所有：wqall
       - 保存所有：wall
     + 屏幕的切换:
       - ctrl+w+w
     + 打开的时候分屏
       - 水平：vim -on 文件名 文件名
       - 垂直：vim -On 文件名文件名
         * n可以省略的，表示分屏的个数
   * 执行shell命令
   * 行跳转
     + ：行号（回车）
   * 末行模式 -> 命令模式:
     + 两次esc
     + 末行模式下执行一个命令
2. vim配置文件
   * 用户级别：
     + ~/.vimrc
3. - gcc相关
4. gcc工作流程
   * 预处理- -E
     + 宏替换
     + 头文件展开
     + 注释去掉
     + xxx.c -> xxx.i

□ c文件

* + 编译- -S
    - xxx.i -> xxx.s
    - 汇编文件
  + 汇编- c
    - xxx.s -> xxx.o
    - 二进制文件
  + 链接
    - xxx.o -> xxx( 可执行）

1. gcc常用参数
   * -v/--version
   * -I: 编译的时候指定头文件路径
   * -c:
     + 将汇编文件生成二进制文件，得到了一个.o文件
     + 源代码
   * -o: 指定生成的文件的名字
   * -g:

gdb调试的时候需要加

* + -D

在编译的时候指定一个宏

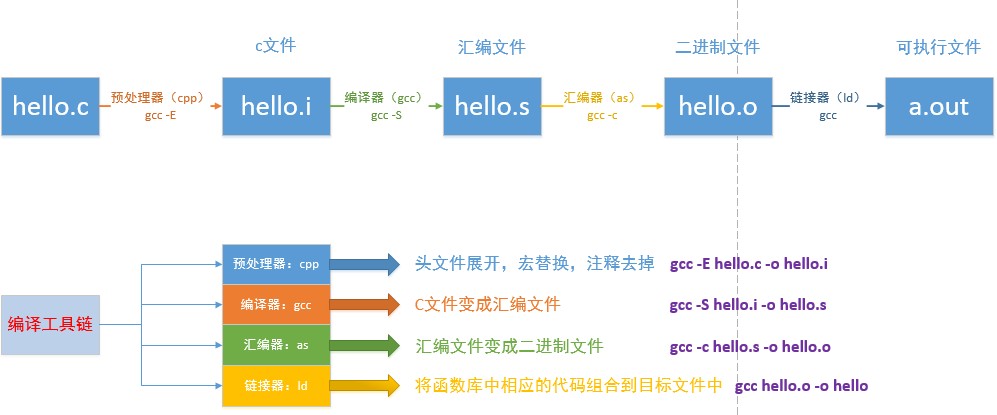
* + - 使用场景：测试程序的时候用
  + -Wall:
    - 添加警告信息
  + -On:
    - 优化代码，n是优化级别：1,2,3

int a = 10； int b = a； int c = b；

printf（“%d”，c）；

int c = 10；

printf（“%d”，10）；



1. - 静态库和动态库的制作和使用
2. 库是什么?
   * 二进制文件
   * 将源代码- 》二进制格式的源代码
     + .c .cpp
   * 加密
3. 库制作出来之后, 如何给用户使用?
   * 头文件
   * 制作出的库
4. 静态库的制作和使用
   * 命名规则: libtest.a
     + lib
     + xxx - 库的名字
     + .a
   * 制作步骤:
     + 原材料：源代码.c .cpp
     + 将.c文件生成.o
       - gcc a.c b.c -c
     + 将.o 打包
       - ar rcs 静态库的名字 原材料
       - ar rcs libtest.a a.o b.o
         * ar - archive
   * 库的使用:
     + gcc test.c -I ./ -L./lib -lmycalc -o app
       - -L: 指定库的路径
       - -l：指定库的名字取得lib和.a
5. 动态库的制作和使用
   * 命名规则
     + libxxx.so
   * 制作步骤
     + 将源文件生成.o

gcc a.c b.c -c -fpic(fPIC)

* + - 打包

gcc -shared a.o b.o -o libxxx.so

* + 库的使用
    - 头文件a.h
    - 动态库libtest.so
    - 参考函数声明编程测试程序main.c

gcc main.c -I ./ -L ./ -l test -o app

* + 动态库无法加载:
    - 使用环境变量
      * 临时设置： 在终端：

export LD\_LIBRARY\_PATH=动态库的路径:$LD\_LIBRARY\_PATH

* + - * 永久设置：
        + 用户级别：

◊ ~/.bashrc

 配置完成：

重启终端

source ~/.bashrc

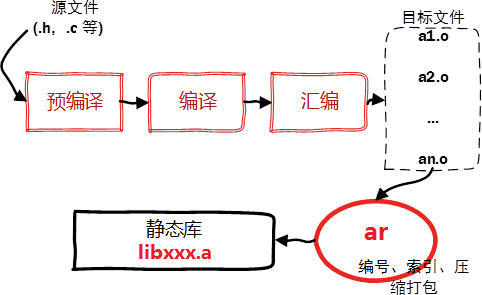
* + - * + 系统级别：

◊ /etc/profile

 source /etc/profile

* + - /etc/ld.so.cache文件列表
      * 找到一个配置文件
        + /etc/ld.so.conf
        + 把动态库的绝对路径添加到文件中
      * 执行一个命令：
        + sudo ldconfig -v
  + 知识点拓展：
    - dlopen, dlclose, dlsym

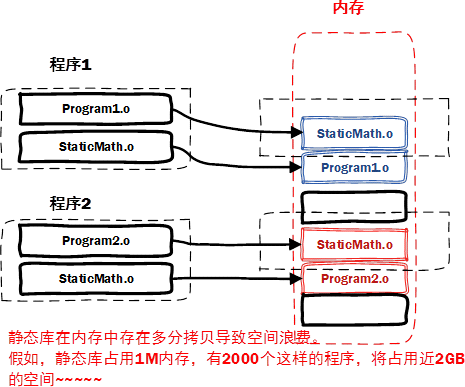
-- 静态库创建过程



-- 静态库，动态库优缺点

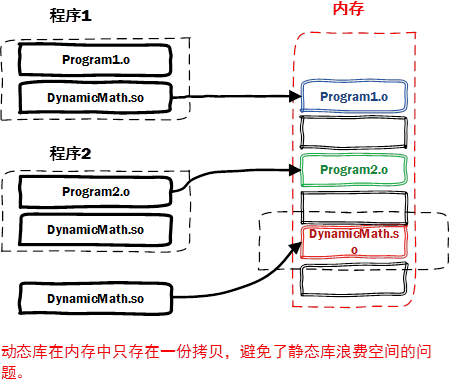
静态库：

* 优点：
  + 静态库被打包到应用程序中加载速度快
  + 发布程序无需提供静态库，移植方便
* 缺点：
  + 销毁系统资源，浪费内存
  + 更新、部署、发布麻烦。



动态库：

* 优点：
  + 可实现进程间资源共享
  + 程序升级简单
  + 程序猿可以控制何时加载动态库
* 缺点：
  + 加载速度比静态库慢



【总线】：

(计算机，各个部件之间传送信息的公共通路。计算机以总线来连接各个功能部件的。(它是cpu、内存、输入、输出设备传递信息的公用通道。

(如果将主板（Mother Board）比作一座城市，那么总线就像是城市里的公共汽车（bus）， (能按照固定行车路线，传输来回不停运作的比特（bit）。

(总线可以划分为数据总线、地址总线和控制总线，分别用来传输数据、数据地址和控制信号。(

=================================

【FreeBSD】：

(是一种类UNIX操作系统，是由BSD(1977至1995年间由加州大学伯克利分校开发,发布)发展而来的Uni x的一个重要分支。

(苹果公司的MacOS X就是借鉴了该系统研发而来。

==================================

【System V】：

(是一种类UNIX操作系统，曾经也被称为 AT&T System V，是Unix操作系统众多版本中另外一个重要分支。 (它最初由 AT&T 开发，在1983年第一次发布。其四个重要版本依次是：SVR1、SVR2、SVR3、SVR4 =====

============================= (

【ISO】：

(国际标准化组织（International Organization for Standardization，简称ISO） (是一个全球性的非政府组织，是国际标准化领域中一个十分重要的组织。

(ISO负责目前绝大部分领域（包括军工、石油、船舶等垄断行业）的标准化活动。

(中国是ISO的正式成员，代表机构是中国国家技术监督局(CSBTS）

(我们很常见的“ISO9001质量体系标准”

==================================

【IEC】：

(国际电工委员会（International Electro technical Commission，简称IEC）

(成立于1906年，是世界上成立最早的非政府性国际电工标准化机构。IEC负责有关电工、

(电子领域的国际标准化工作，其他领域则由ISO负责。IEC的宗旨是促进电工标准的国际统一。

==================================

【IEEE】：(电气和电子工程师协会(Institute of Electrical and Electronics Engineers，简称IEEE)(IEEE的两个前身AIEE（美国电气工程师协会）和 IRE（无线电工程师协会）。(是一个国际性的电子技术与信息科学工程师的协会，是目前全球最大的非营利性专业技术学会。 (IEEE的标准制定内容包括电气与电子设备、试验方法、原器件、符号、定义以及测试方法等多个领域。

【ANSI】：

(美国国家标准学会(American National Standards Institute, 简称ANSI)，协商与标准有关的活动， (审议美国国家标准，并努力提高美国在国际标准化组织中的地位。

(是国际标准化委员会（ISO）和国际电工委员会（IEC）5个常任理事成员之一。

==================================

【POSIX 标准】：

(POSIX 标准最初由IEEE开发 ，是为了提高 UNIX 环境下应用程序的可移植性。(后来许多其它操作系统也陆续支持该标准。

==================================

【C99】：

(C语言的官方标准第二版。1999年12月1日，国际标准化组织（ISO）和国际电工委员会（IEC） (旗下的C语言标准委员会（ISO/IEC JTC1/SC22/WG14）正式发布了这个标准文件。

(新的C语言标准是 ISO 和 IEC 在2011年12月8日正式发布的C11标准。

==================================

【libc库】:

(libc是Linux下的ANSI C的函数库。

==================================

【glibc】:

(glibc是GNU发布的libc库。glibc 包括 libc 标准 C 库、libm 数学库、(libcrypt 加密与编码库、libpthread 线程库、libnss 网路服务库 等等。

(

==================================

【GNU】：new

(由Richard Stallman在1983年9月27日公开发起的。它的目标是创建一套完全自由的操作系统。

(GNU软件可以自由地“使用、复制、修改和发布”。GNU通用公共许可证（GNU General Public License， GPL）。

(即“反版权”（或称Copyleft）概念。

(1985年Richard Stallman又创立了自由软件基金会（Free Software Foundation）来为GNU计划提供技术、法律以及财政支持。

(1991年Linus Torvalds编写出了与UNIX兼容的Linux操作系统内核并在GPL条款下发布。1992年Linux与其他GNU软件结合，

(完全自由的操作系统正式诞生。被称为“GNU/Linux”或简称Linux。

==================================

【gcc】：

(GNU编译器套件（GNU Compiler Collection）。是以GPL许可证所发行的自由软件，也是 GNU计划的关键部分。

(现已被大多数类Unix操作系统（如Linux、BSD、Mac OS X等）采纳为标准的编译器，GCC同样适用于微软

的Windows

学习目标

1. 熟练使用规则编写简单的makefile文件
2. 熟练使用makefile中的变量
3. 熟练使用makefile中的函数
4. 熟练掌握gdb相关调试命令的使用
5. 了解概念: pcb和文件描述符,虚拟地址空间
6. 熟练掌握Linux系统IO函数的使用

了解阻塞和非阻塞的概念

01 - makefile

(1. make

(2. makefile文件的命名

(3. makefile中的规则

(4. makefile的编写

1. gdb调试

(

1. 启动gdb

(2. 查看代码 -- list

(3. 断点操作 - break / b

((○ 设置断点:

((○ 查看断点((○ 删除断点((○ 设置断点无效((○ 断点生效((○ 设置条件断点:

(4. 调试相关命令

1. GDB简介

GDB（GNU Debugger）是GCC的调试工具。其功能强大，现描述如下： GDB主要帮忙你完成下面四个方面的功能：

1. 启动你的程序，可以按照你的自定义的要求随心所欲的运行程序。
2. 可让被调试的程序在你所指定的调置的断点处停住。（断点可以是条件表达式）
3. 当程序被停住时，可以检查此时你的程序中所发生的事。
4. 动态的改变你程序的执行环境。
5. 生成调试信息

一般来说GDB主要调试的是C/C++的程序。要调试C/C++的程序，首先在编译时， 我们必须要把调试信息加到可执行文件中。

使用编译器（cc/gcc/g++）的 -g 参数可以做到这一点。

如： gcc -g hello.c -o hello g++ -g hello.cpp -o hello

如果没有-g，你将看不见程序的函数名、变量名，所代替的全是运行时的内存地址。当你用-g把调试信息加入之后，并成功编译目标代码以后，让我们来看看如何用gdb 来调试他。

1. 启动GDB 的方法gdb program

program 也就是你的执行文件，一般在当前目录下。

1. 程序运行上下文
   1. 程序运行参数

set args 可指定运行时参数。（如：set args 10 20 30 40 50 ） show args 命令可以查看设置好的运行参数。 run 启动程序

* 1. 工作目录

cd 相当于shell的cd命令。 pwd 显示当前的所在目录。

1. 设置断点
   1. 简单断点

break 设置断点，可以简写为b

b 10 设置断点，在源程序第10行

b func 设置断点，在func函数入口处

* 1. 多文件设置断点

在进入指定函数时停住:

C++中可以使用class::function或function(type,type)格式来指定函数名。如果有名称空间，可以使用namespace::class::function或者function (type,type)格式来指定函数名。

break filename:linenum -- 在源文件filename的linenum行处停住break filename:function -- 在源文件filename的function函数的入口处

停住

break class::function或function(type,type) -- 在类class的function函数的入口处停住

break namespace::class::function -- 在名称空间为namespace的类class的function函数的入口停住

* 1. 查询所有断点info b

6条件断点

一般来说，为断点设置一个条件，我们使用if关键词，后面跟其断点条件。设置一个条件断点

b test.c:8 if intValue == 5

1. 维护停止点delete [range...]

删除指定的断点，如果不指定断点号，则表示删除所有的断点。range 表示断点号的范围（如：3-7）。其简写命令为d。

比删除更好的一种方法是disable停止点，disable了的停止点，GDB不会删除

，当你还需要时，enable即可，就好像回收站一样。disable [range...]

disable所指定的停止点，如果什么都不指定，表示disable所有的停止点。简写命令是dis.

enable [range...]

enable所指定的停止点，如果什么都不指定，表示enable所有的停止点。简写命令是ena.

分区 02 - vim-gcc-静态库-动态库 的第 6 页

1. 调试代码

run 运行程序，可简写为r

next 单步跟踪，函数调用当作一条简单语句执行，可简写为n step 单步跟踪，函数调进入被调用函数体内，可简写为s finish 退出进入的函数

until 在一个循环体内单步跟踪时，这个命令可以运行程序直到退出循环体,可简写为u。continue 继续运行程序，可简写为c

1. 查看运行时数据

print 打印变量、字符串、表达式等的值，可简写为p p count 打印count的值

1. 自动显示

你可以设置一些自动显示的变量，当程序停住时，或是在你单步跟踪时，这些变量会自动显示。相关的GDB命令是display。

display 变量名

info display -- 查看display设置的自动显示的信息。undisplay num（info display时显示的编号）

delete display dnums⋯ -- 删除自动显示，dnums意为所设置好了的自动显式的编号。如果要同时删除几个，编号可以用空格分隔，如果要删除一个范围内的编号，可以用减号表示（如：2-5）

disable display dnums⋯ enable display dnums⋯

disable和enalbe不删除自动显示的设置，而只是让其失效和恢复。

1. 查看修改变量的值

(gdb) ptype width -- 查看变量width的类型type = double

(gdb) p width -- 打印变量width 的值

$4 = 13

你可以使用set var命令来告诉GDB，width不是你GDB的参数，而是程序的变量名，如： (gdb) set var width=47

在你改变程序变量取值时，最好都使用set var格式的GDB命令。

12 显示源代码

GDB 可以打印出所调试程序的源代码，当然，在程序编译时一定要加上 –g 的参数，把源程序信息编译到执行文件中。不然就看不到源程序了。当程序停下来以后，GDB会报告程序停在了那个文件的第几行上。你可以用list命令来打印程序的源代码。默认打印10行，还是来看一看查看源代码的GDB命令吧。

list linenum

Print lines centered around line number linenum in the current source file. list function 显示函数名为function的函数的源程序。

list 显示当前行后面的源程序。

list - 显示当前行前面的源程序。 一般是打印当前行的上5行和下5行，如果显示函数是是上2行下8行，默认是10行，当然，你也可以定制显示的范围，使用下面命令可以设置一次显示源程序的行数。

set listsize count 设置一次显示源代码的行数。show listsize 查看当前listsize的设置。

1. 文件IO

(1. open/close

((函数原型:

(((§ int open(const char \*pathname, int flags);

(((§ int open(const char \*pathname, int flags, mode\_t mode);

((参数:

(((flags

((((必选项 O\_RDONLY, O\_WRONLY, O\_RDWR

((((可选项

(((((创建文件: O\_CREAT

(((((创建文件时检测文件是否存在: O\_EXCL

((((( 如果文件存在, 返回-1 ((((( 必须与O\_CREAT一起使用(((((追加文件: O\_APPEND

(((((文件截断: O\_TRUNC

(((((设置非阻塞: O\_NONBLOCK

(( mode (2. read

((函数原型: ssize\_t read(int fd, void \*buf, size\_t count);

(((§ 参数:

(((§ 返回值:

(3. write

((函数原型: ssize\_t write(int fd, const void \*buf, size\_t count);( (( 参 数 :

(( 返 回 值 :

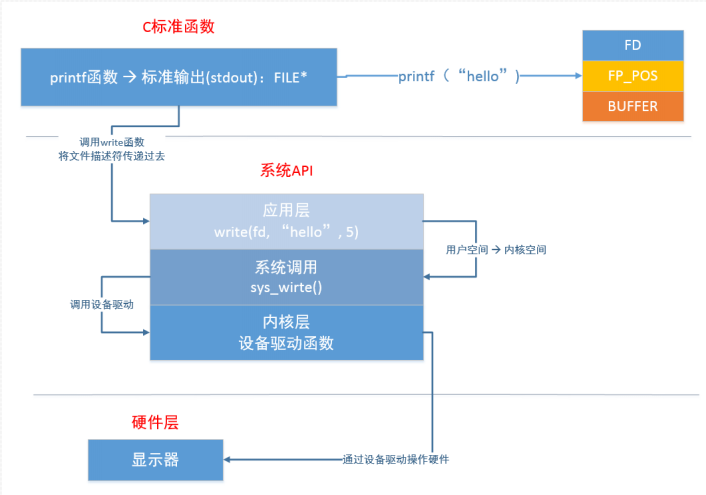
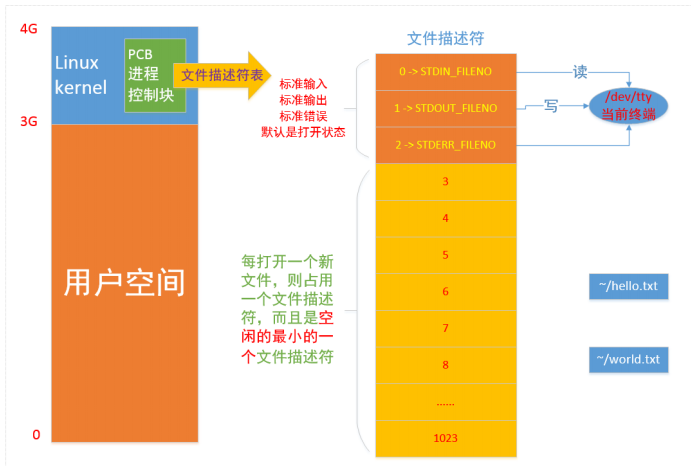
(4. lseek

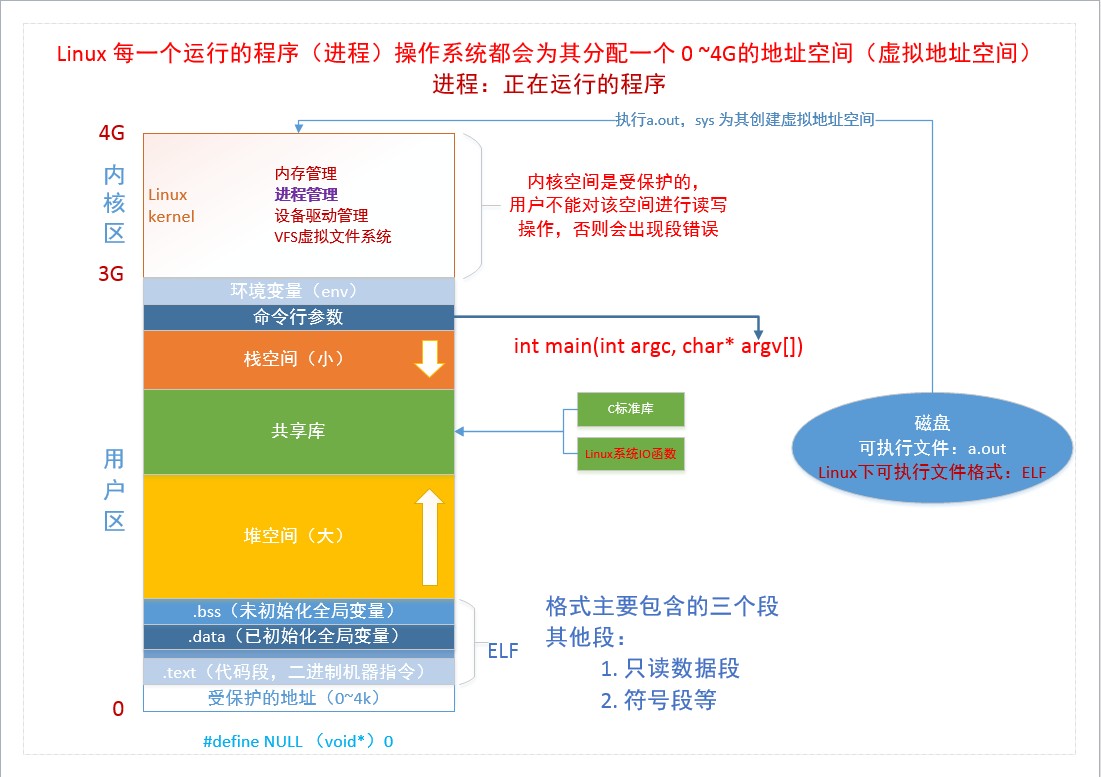
(( 函数原型: off\_t lseek(int fd, off\_t offset, int whence);

(5. 阻塞和非阻塞

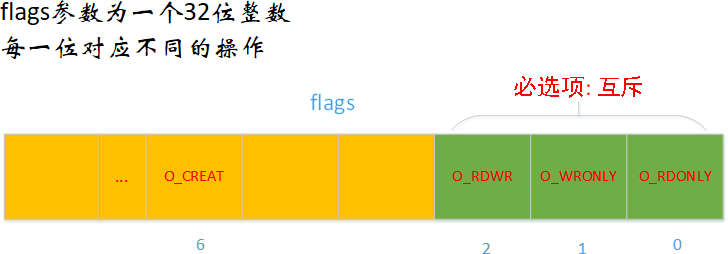
(( 阻塞和非阻塞是文件的属性还是read函数的属性?

# 3-1 C库IO函数工作流程

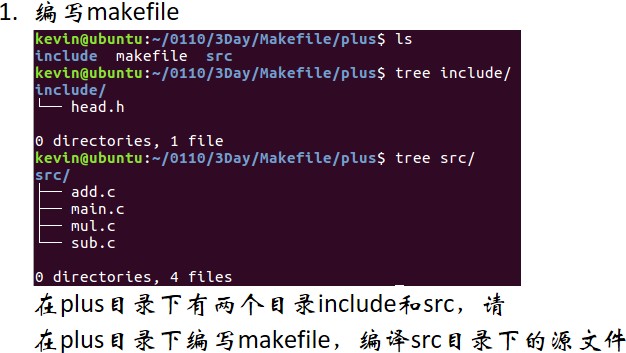
* 1. C库函数与系统函数的关系
  2. 虚拟地址空间



* 1. pcb和文件描述符表
  2. open函数的flags参数



练习



1. 文件IO
   1. open/close
      * 函数原型:
        + int open(const char \*pathname, int flags);
        + int open(const char \*pathname, int flags, mode\_t mode);
      * 参数:
        + flags
          - 必选项 O\_RDONLY, O\_WRONLY, O\_RDWR
          - 可选项

创建文件: O\_CREAT

◊ 创建文件时检测文件是否存在: O\_EXCL

◊ 如果文件存在, 返回-1

◊ 必须与O\_CREAT一起使用

追加文件: O\_APPEND

文件截断: O\_TRUNC

设置非阻塞: O\_NONBLOCK

* + - * mode -- 指定0777
        + 八进制数
        + 最终权限: mode & ~umaks
        + umask 0002

000000010

~

111111101

111111111

&

111111101

775

* 1. read
     + 函数原型: ssize\_t read(int fd, void \*buf, size\_t count);
       - 参数:
         * fd -- open的返回值
         * buf - 缓冲区, 存放读取的数据
         * count -- 缓冲区的最大容量 sizeof(buf)
       - 返回值:
  2. write

□ -1: 失败

* >0: 读出的字节数

□ =0: 文件读完了

* + 函数原型: ssize\_t write(int fd, const void \*buf, size\_t count);
    - 参数:
      * fd: 文件描述符, open 返回值
      * buf: 要往文件中写的数据
      * count: 有效数据的长度
    - 返回值:
  1. lseek

□ -1: 失败

* >0: 写入的字节数
  + 函数原型: off\_t lseek(int fd, off\_t offset, int whence);
    - SEEK\_SET
    - SEEK\_CUR
    - SEEK\_END
  + 使用:

1. 文件指针移动到头部: lseek(fd, 0, SEEK\_SET);
2. 获取文件指针当前的位置:

int len = lseek(fd, 0, SEEK\_CUR);

1. 获取文件长度:

int len = lseek(fd, 0, SEEK\_END);

* 文件拓展
  + 文件原大小100k， 拓展为1100k lseek(fd, 1000, SEE\_END);
  + 最后做一次写操作

write(fd, "a", 1);

* 1. 阻塞和非阻塞
     + 阻塞和非阻塞是文件的属性还是read函数的属性?
       - 文件的属性
       - 普通文件：hello.c
         * 默认不阻塞
       - 终端设备：/dev/tty
         * 默认阻塞
         * 管道
         * 套接字



1. stat/lstat函数
2. 获取文件属性

- struct stat {

dev\_t st\_dev; //文件的设备编号

ino\_t st\_ino; //节点

mode\_t st\_mode; //文件的类型和存取的权限

nlink\_t st\_nlink; //连到该文件的硬连接数目，刚建立的文件值为1

uid\_t st\_uid; //用户ID

gid\_t st\_gid; //组ID

dev\_t st\_rdev; //(设备类型)若此文件为设备文件，则为其设备编号

off\_t st\_size; //文件字节数(文件大小)

blksize\_t st\_blksize; //块大小(文件系统的I/O 缓冲区大小)

blkcnt\_t st\_blocks; //块数

time\_t st\_atime; //最后一次访问时间

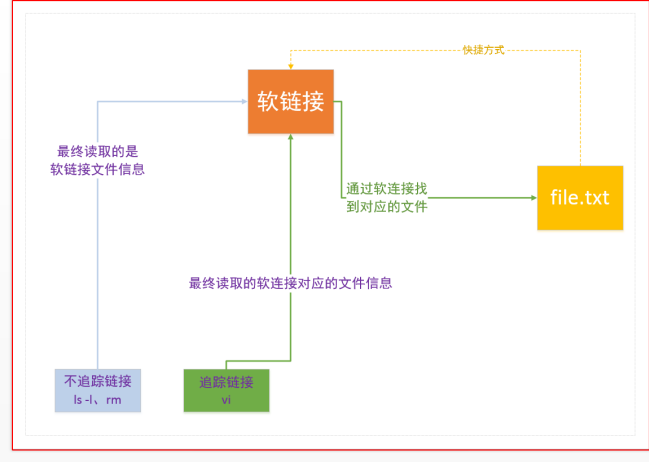
time\_t st\_mtime; //最后一次修改时间

time\_t st\_ctime; //最后一次改变时间(指属性)

};

* st\_mode -- 16位整数
  + 0-2 bit -- 其他人权限
    - S\_IROTH 00004 读权限
    - S\_IWOTH 00002 写权限
    - S\_IXOTH 00001 执行权限
    - S\_IRWXO 00007 掩码, 过滤 st\_mode中除其他人权限以外的信息
  + 3-5 bit -- 所属组权限
    - S\_IRGRP 00040 读权限
    - S\_IWGRP 00020 写权限
    - S\_IXGRP 00010 执行权限
    - S\_IRWXG 00070 掩码, 过滤 st\_mode中除所属组权限以外的信息
  + 6-8 bit -- 文件所有者权限
    - S\_IRUSR 00400 读权限
    - S\_IWUSR 00200 写权限
    - S\_IXUSR 00100 执行权限
    - S\_IRWXU 00700 掩码, 过滤 st\_mode中除文件所有者权限以外的信息
  + 12-15 bit -- 文件类型
    - S\_IFSOCK 0140000 套接字
    - S\_IFLNK 0120000 符号链接（软链接）
    - S\_IFREG 0100000 普通文件
    - S\_IFBLK 0060000 块设备
    - S\_IFDIR 0040000 目录
    - S\_IFCHR 0020000 字符设备
    - S\_IFIFO 0010000 管道
    - S\_IFMT 0170000 掩码,过滤 st\_mode中除文件类型以外的信息(st\_mode & S\_IFMT) == S\_IFREG
  + int stat(const char \*path, struct stat \*buf);
  + int lstat(const char \*path, struct stat \*buf);
    - lstat读取的链接文件本身的属性
    - stat读取的是连接文件指向的文件的属性

- 追踪，穿透



1. 文件属性函数
   1. 测试当前用户指定文件是否具有某种属性
      * 当前用户, 使用哪个用户调用这个函数, 这个用户就是当前用户
      * int access(const char \*pathname, int mode);
        + 参数:

pathname: 文件名mode: 4种权限

R\_OK -- 读

W\_OK -- 写

X\_OK -- 执行

F\_OK -- 文件是否存在

* + - * 返回值:

0 - 有某种权限, 或者文件存在

-1 - 没有, 或文件不存在

* 1. 修改文件权限
     + int chmod(const char \*filename, int mode);
       - 参数:
         * filename: 文件名
         * mode: 文件权限, 八进制数
  2. 修改文件所有者和所属组
     + int chown(const char \*path, uid\_t owner, gid\_t group);
     + 函数参数:
       - path -- 文件路径
       - owner -- 整形值, 用户ID
         * /etc/passwd
       - group -- ....., 组ID
         * /etc/group
  3. 修改文件大小
     + int truncate(const char \*path, off\_t length);
     + 参数:
       - path -- 文件名
       - length -- 文件的最终大小

1. 比原来小, 删掉后边的部分
2. 比原来大, 向后拓展
3. 目录操作相关函数
4. 文件重命名
   * int rename(const char \*oldpath, const char

\*newpath);

1. 修改当前进程(应用程序)的路径 cd
   * int chdir(const char \*path);
     + 参数: 切换的路径
2. 获取当前进程的工作目录 pwd
   * char \*getcwd(char \*buf, size\_t size);
     + 返回值:
       - 成功: 当前的工作目录
       - 失败: NULL
     + 参数:
       - buf: 缓冲区, 存储当前的工作目录
       - size: 缓冲区大小
3. 创建目录 mkdir
   * int mkdir(const char \*pathname, mode\_t mode);
     + 参数:
       - pathname: 创建的目录名
       - mode: 目录权限, 八进制的数, 实际权限:

mode & ~umask

1. 删除一个空目录
   * int rmdir(const char \*pathname);
     + 参数: 空目录的名字
2. 目录遍历相关函数
   1. 打开一个目录
      * DIR \*opendir(const char \*name);
        + 参数: 目录名
        + 返回值: 指向目录的指针
      * FILE\* fp = fopen（）
      * fread（buf， len， len，fp）；
   2. 读目录

struct dirent

{

ino\_t d\_ino; // 此目录进入点的inode

ff\_t d\_off; // 目录文件开头至此目录进入点的位移signed short int d\_reclen; // d\_name 的长度, 不包含NULL 字符unsigned char d\_type; // d\_name 所指的文件类型

har d\_name[256]; // 文件名

};

d\_type

* DT\_BLK - 块设备
* DT\_CHR - 字符设备
* DT\_DIR - 目录
* DT\_LNK - 软连接
* DT\_FIFO - 管道
* DT\_REG - 普通文件
* DT\_SOCK - 套接字
* DT\_UNKNOWN - 未知
  + - struct dirent \*readdir(DIR \*dirp);
      * 参数: opendir的返回值
      * 返回值: 目录项结构体
  1. 关闭目录
     + int closedir(DIR \*dirp);
  2. 独立完成递归读目录中指定类型文件个数的操作.

1. dup-dup2-fcntl
   1. 复制文件描述符
      * int dup(int oldfd);
        + oldfd - 要复制的文件描述符
        + 返回值: 新的文件描述符
        + dup调用成功:
          - 有两个文件描述符指向同一个文件
        + 返回值: 取最小的且没被占用的文件描述符
      * int dup2(int oldfd, int newfd);
        + oldfd -》hello
        + newfd -》world
          - 假设newfd已经指向了一个文件，首先断开close与那个文件的链接，newfd指向oldfd指向的文件

文件描述符重定向

oldfd和newfd指向同一个文件

* + - * + newfd没有被占用，newfd指向oldfd指向的文件
  1. 改变已经打开的文件的属性: fcntl
     + 变参函数
     + 复制一个已有的文件描述符

int ret = fcntl(fd, F\_DUPFD);

* + - 获取/设置文件状态标志
      * open 的flags参数

1. 获取文件状态标识

int flag = fcntl(fd, F\_GETFL)

1. 设置文件状态标识
   * flag = flag | O\_APPEND;
   * fcntl(fd, F\_SETFL, flag)
     + 可以更改的几个标识: O\_APPEND、O\_NONBLOCK （常用）

分区 04 - 文件和目录操作函数-dup-fcntl 的第 11 页

作业

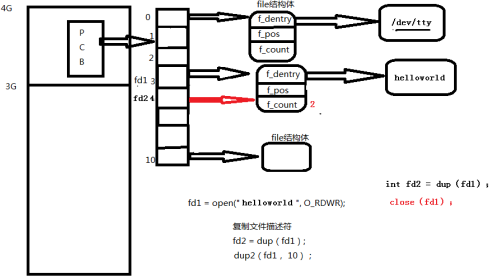
2016年11月28日 17:30

1. ls -l

a. stat(lstat)

1. 验证dup和dup2

a. 验证fcntl



复习

2017年3月18日 9:09

1. makefile
   * 一个规则

目标：依赖

（tab缩进） 命令

* + - 可以有多个规则
      * make - 终极目标
        + 第一个规则对应的目标
      * make 子目标的名字
    - 工作原理：
      * 依赖不存在：
        + 向下查找子规则，找到了用来生成依赖的规则，执行规则中的命令
      * 依赖存在->更新检查：
        + 更新条件：目标时间 < 依赖的时间
  + 两个函数：
    - wildcard
      * src = $(wildcard ./src/\*.c)
      * src=./src/a.c ./src/b.c ./src/c.c
    - patsubst
      * obj=$(patsubst %.c, %.o, $(src))
      * obj=./src/a.o ./src/b.o ./src/c.o
  + 三个自动变量
    - $@: 规则中的目标
    - $<: 规则中的依赖（第一个）
    - $^: 规则中的所有的依赖
      * 只能在命令中使用

app：a.o b.o c.o

gcc a.o b.o c.o -o app

* + - 模式规则：

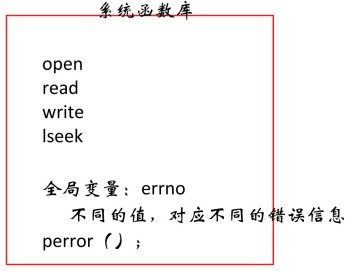
%.o:%.c

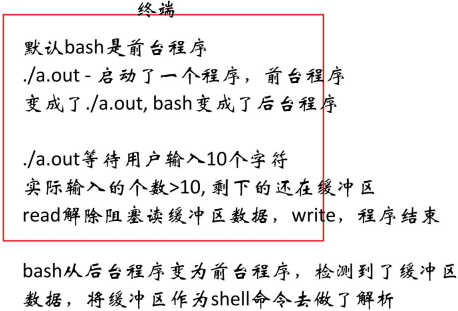
gcc -c $< -o $@

* + 清空项目：
    - 声明伪目标：

.PHONY:clean clean：

rm \*.o





2017年3月18日 15:54

目录

进到目录之后：opendir，读内容需要循环的读目录中的内容while（readdir）

{

}

最后关闭目录

closedir（dir）；