



西南交通大学



# 嵌入式系统及应用

西南交通大学

电气工程学院

孙永奎博士

E-mail: [yksun@swjtu.edu.cn](mailto:yksun@swjtu.edu.cn)





# 第6章 Linux系统及文件I/O操作

- ❖ 6.1 Linux 概述
- ❖ 6.2 Linux 基本命令
- ❖ 6.3 Linux 的开发环境
- ❖ 6.4 文件I/O操作
- ❖ 3学时





# Linux入门需要掌握的主要内容

## ❖ Linux 环境的使用

- ◆ 会使用VI编辑器
- ◆ 掌握一些基本的命令

## ❖ Linux编程需要掌握的内容

- ◆ GNU GCC编译工具
- ◆ 进程
- ◆ 文件操作
- ◆ 线程操作
- ◆ 信号处理
- ◆ 消息管理
- ◆ 网络编程





# 嵌入式linux的学习方法

- ❖ 1、为什么要用操作系统？操作系统的主要功能？
- ❖ 2、嵌入式Linux实时性改造？
- ❖ 3、主流的嵌入式操作系统
- ❖ 嵌入式linux的学习方法(初级,入门,没有硬件)
  - 1) 找一本关于操作系统书,最好一本中文的,一本英文的,从头读到尾,知道一些术语,一些概念
  - 2) 找一本linux c语言的书, <Unix环境下的高级编程>
  - 3) 找一本驱动程序的书,<linux 下驱动程序设计>、<linux device drivers>
  - 4) 充分利用网络上的资源





# 嵌入式linux的学习方法

## ❖ 嵌入式linux的学习方法(中级,实践)

- (1) 嵌入式调试流程,
  - 1) 应用程序交叉编译,下载调试
  - 2) 配置Bootloader,linux内核,文件系统以及下载调试
- (2) 嵌入式应用程序开发,图形用户界面开发,简单的驱动程序开发

## ❖ 嵌入式linux的学习方法(高级,实践)

- 1) 复杂的驱动程序开发,(比如USB,编解码器外部硬件的控制等)
- 2) 文件系统的移植(jffs2,yaffs), 内核的移植,Bootloader的移植

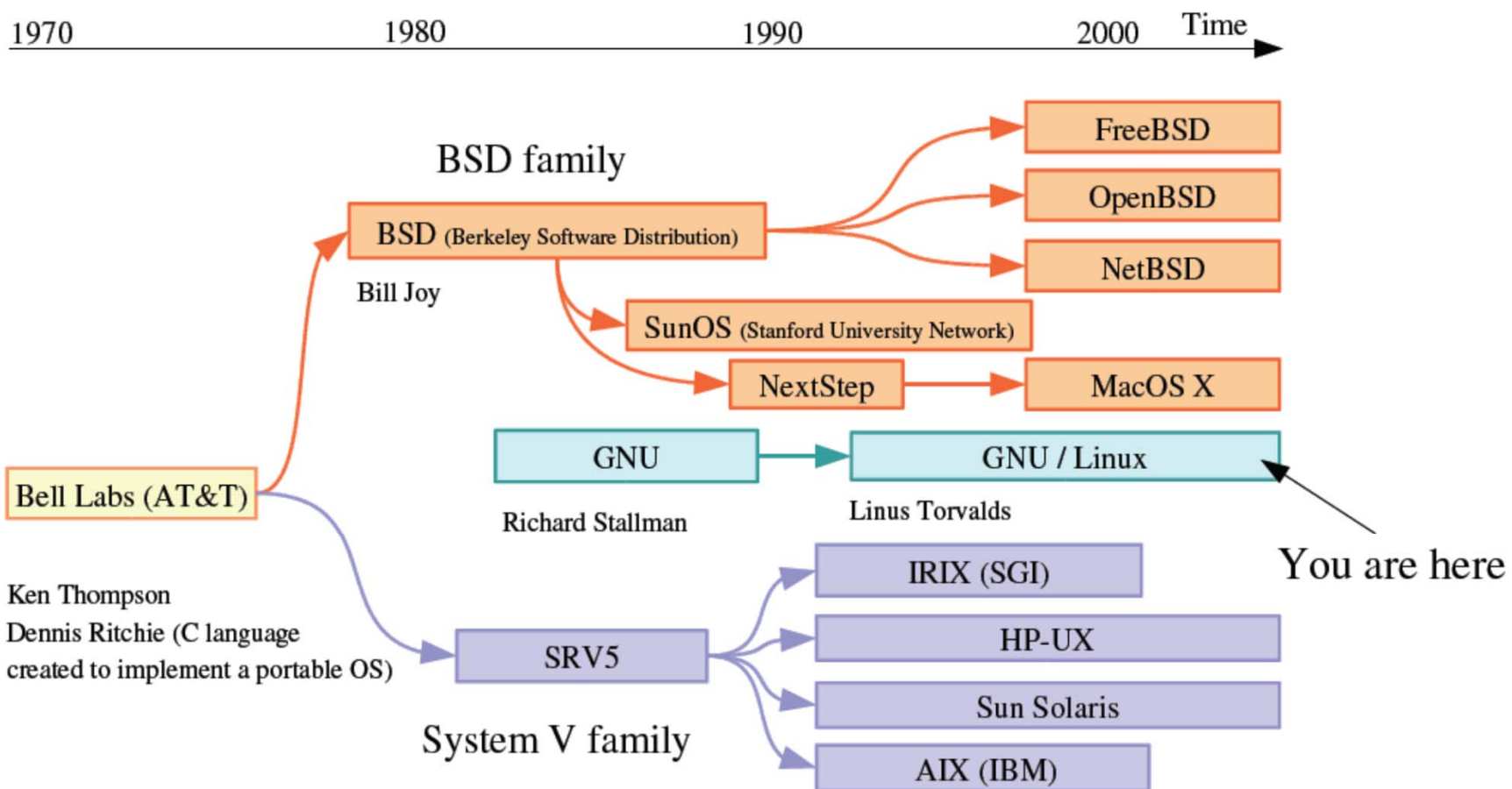




## 6.1 Linux概述

6

### ❖ Unix家族树







## 6.1 Linux概述

7

### ❖ 庞大的嵌入式Linux市场



2021年全球嵌入式软件市场销售额达到了**106亿**美元，预计2028年将达到149亿美元，年复合增长率（CAGR）为4.9%（2022-2028）



## 6.1 Linux概述

8

- ❖ Linux是类UNIX操作系统，该项目由荷兰的Linus Torvalds所启动。
- ❖ Linux是一个Unix兼容的系统，大部分通用的Unix工具和程序都可以在Linux系统下运行
- ❖ 使用GNU工具开发：
  - ◆ gcc, glibc, binutils, make等
- ❖ GNU = GNU is Not Unix 由Richard Stallman在1984创建
  - ◆ 最初的软件：gcc、make、glibc...
- ❖ GPL = General Public License 。







## 6.1 Linux概述

9

### ❖ Linux优点

- ◆ 提供了先进的网络支持
- ◆ 多任务、多用户
- ◆ 符合IEEE POSIX标准
- ◆ 支持数十种文件系统格式
- ◆ 完全运行于保护模式
- ◆ 开放源代码
- ◆ 采用先进的内存管理机制，更加有效地利用物理内存





## 6.1 Linux概述

10

### ❖ Linux作为嵌入式操作系统的优势

- ◆ 一种**开放源码、软实时、多任务**的嵌入式操作系统
- ◆ 可应用于多种硬件平台：X86、PowerPC、ARM、XSCALE、MIPS、SH、68K、Alpha、SPARC等
- ◆ 一种多任务、稳定可靠、**内核可裁剪**的系统
- ◆ 良好的网络支持





# 第6章 Linux系统及文件I/O操作

- ❖ 6.1 Linux 概述
- ❖ 6.2 Linux 基本命令
- ❖ 6.3 Linux 的开发环境
- ❖ 6.4 文件I/O操作
- ❖ 3学时

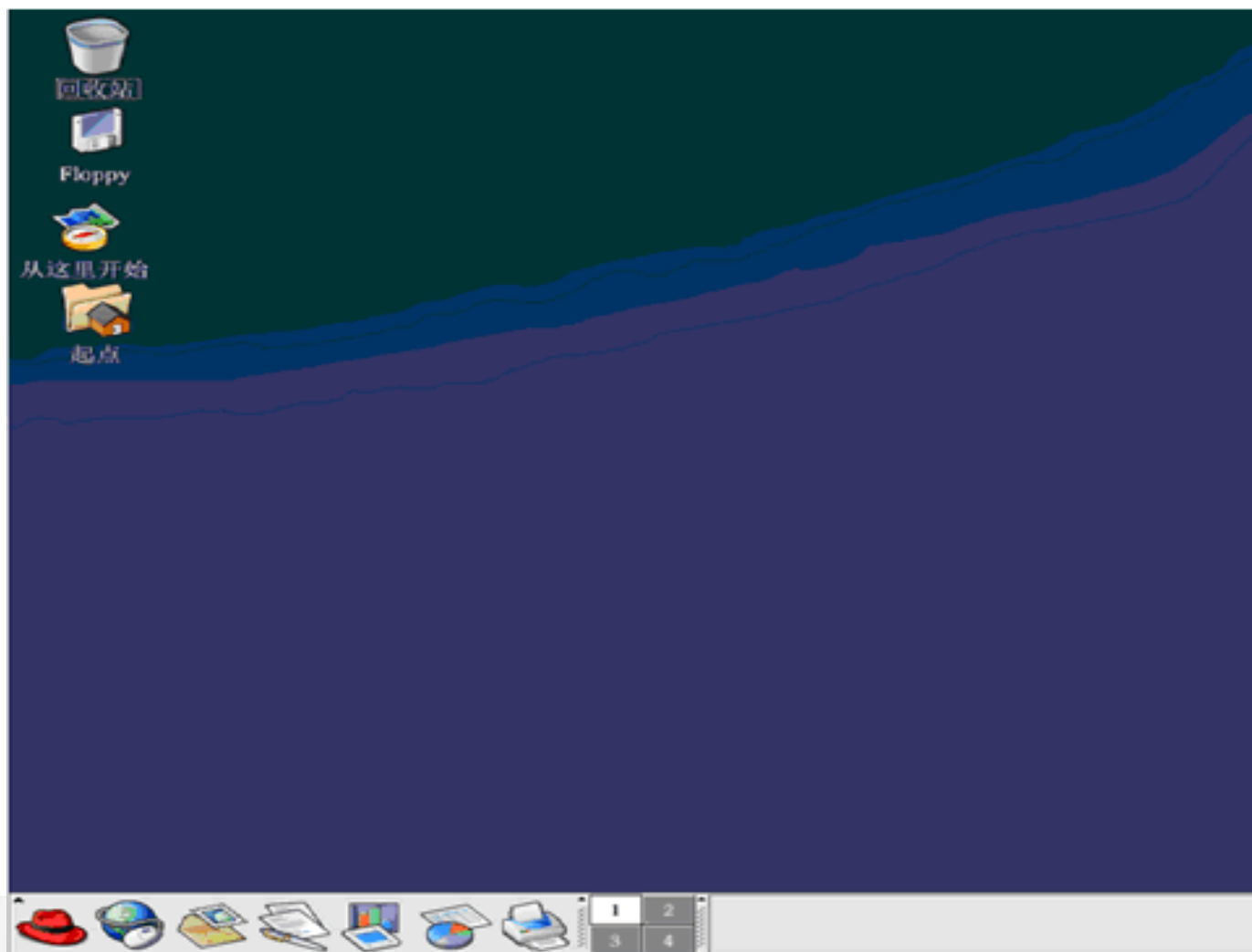




## 6.2 Linux 基本命令

12

### ❖ 1、Linux 操作系统界面





## 6.2 Linux 基本命令

❖ 2、在Linux系统中打开终端的方式有以下两种：

- 1) 是在桌面上依次单击“主程序→系统工具→终端”可打开如图的终端窗口；
- 2) 是在Linux桌面上单击鼠标右键，从弹出的快捷菜单中选择“终端”命令，也可打开终端窗口。





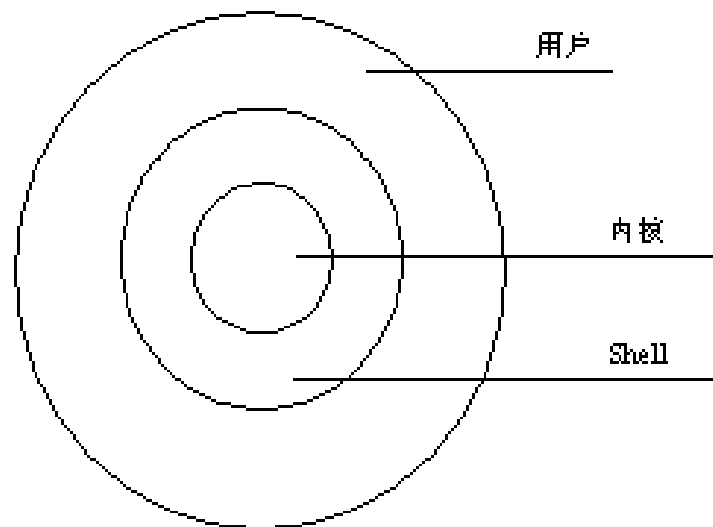
## 6.2 Linux 基本命令

14

### ❖ 内核、Shell和用户的关系：

### ❖ Shell是一种Linux中的命令行解释程序。

- ◆ 为用户提供使用操作系统的接口。用户在提示符下输入的命令都由Shell先解释然后传给Linux内核



### ❖ Linux命令格式说明：

- ◆ 格式中带[]的表明为可选项，其他为必选项。
- ◆ 选项可以多个连带写入。







## 6.2 Linux 基本命令

15

### ❖ 1、用户系统相关命令

#### ❖ 1) 用户切换命令 (su)

- ◆ 变更为其它使用者的身份，主要用于将普通用户身份转变为超级用户

选 项	参 数 含 义
- , -l , --login	为该使用者重新登录，大部分环境变量（如 HOME、SHELL 和 USER 等）和工作目录都是以该使用者（USER）为主。若没有指定 USER，缺省情况是 root
-m , -p	执行 su 时不改变环境变量
-c , --command	变更账号为 USER 的使用者，并执行指令（command）后再变回原来使用者

#### ❖ 使用示例

- ◆ `[linuxlinux@www linux]$ su - root`
- ◆ **Password:**
- ◆ `[root@www root]#`





## 6.2 Linux 基本命令

16

### ❖ 2) 用户管理命令 (useradd和passwd)

- ◆ ① useradd: 添加用户账号。
- ◆ ② passwd: 更改对应用户账号密码。

选 项	参 数 含 义
-g	指定用户所属的群组
-m	自动建立用户的登入目录
-n	取消建立以用户名称为名的群组

### ❖ 使用实例

- ◆ [root@www root]# useradd ycw
- ◆ [root@www root]# passwd ycw





## 6.2 Linux 基本命令

17

### ❖ 3) 系统管理命令 (ps和kill)

- ❖ ① ps: 显示当前系统中由该用户运行的进程列表。

选 项	参 数 含 义
-ef	查看所有进程及其 PID ( 进程号 )、系统时间、命令详细目录、执行者等
-aux	除可显示-ef 所有内容外，还可显示 CPU 及内存占用率、进程状态
-w	显示加宽并且可以显示较多的信息

- ❖ ② kill: 输出特定的信号给指定PID ( 进程号 ) 的进程，并根据该信号而完成指定的行为。其中可能的信号有进程挂起、进程等待、进程终止等

选 项	参 数 含 义
-s	根据指定信号发送给进程
-p	打印出进程号 ( PID )，但并不送出信号
-l	列出所有可用的信号名称



## 6.2 Linux 基本命令

18

### ❖ 使用实例

❖ [root@www root]# ps -ef

UID	PID	PPID	C	STIME	TTY	TIME	CMD
root	1	0	0	2005 ?	00:00:05		<u>init</u>
root	2	1	0	2005 ?	00:00:00		<u>[keventd]</u>
root	3	0	0	2005 ?	00:00:00		<u>[ksoftirqd_CPU0]</u>
root	4	0	0	2005 ?	00:00:00		<u>[ksoftirqd_CPU1]</u>
root	7421	1	0	2005 ?	00:00:00		<u>/usr/local/bin/ntpd -c /etc/ntp.</u>
root	21787	21739	0	17:16 pts/1	00:00:00		<u>grep ntp</u>

### ❖ 使用实例

◆ [root@www root]# kill 7421

❖ 其他用户系统相关命令如磁盘命令(fdisk) 请参考相关资料





## 6.2 Linux 基本命令

19

### ❖ 2、文件目录相关命令

#### ❖ 1) 改变工作目录命令 **cd**

- ◆ 其中的路径为要改变的工作目录，可为相对路径或绝对路径

#### ❖ 使用实例

- ◆ `[root@www uclinux]# cd /home/linux/`

#### ❖ “.”代表当前目录，“../”代表上级目录。





## 6.2 Linux 基本命令

20

### ❖ 2) 列出目录的内容命令 **ls**

选 项↵	参 数 含 义↵
-l , --format=single-column↵	一行输出一个文件（单列输出）↵
-a , -all↵	列出目录中所有文件，包括以“.”开头的文件↵
-d↵	将目录名和其他文件一样列出，而不是列出目录的内容↵
-l, --format=long --format=verbose↵	除每个文件名外，增加显示文件类型、权限、硬链接数、所有者名、组名、大小（Byte）及时间信息（如未指明是其他时间即指修改时间）↵
-f↵	不排序目录内容，按它们在磁盘上存储的顺序列出↵

### ❖ 使用实例

◆ [ycwing@www /]\$ ls -l

```
total 220
drwxr-xr-x  2 root  root    4096 Mar 31  2005 bin
drwxr-xr-x  3 root  root    4096 Apr  3  2005 boot
-rw-r--r--  1 root  root      0 Apr 24  2002 test.run
...
```







## 6.2 Linux 基本命令

21

### ❖ 3) 创建一个目录命令 **mkdir**

选 项	参 数 含 义
-m	对新建目录设置存取权限，也可以用 chmod 命令（在本节后会有详细说明）设置
-p	可以是一个路径名称。此时若此路径中的某些目录尚不存在，在加上此选项后，系统将自动建立好那些尚不存在的目录，即一次可以建立多个目录

### ❖ 使用实例

- ◆ [root@www my]# mkdir -m 777 ./why
- ◆ [root@www my]# ls -l

```
total 4
drwxrwxrwx  2 root  root  4096 Jan 14 09:24 why
```

❖ 其他与文件目录相关命令如 **cat**、**cp**、**mv** 和 **rm** 参考其他资料





## 6.2 Linux 基本命令

22

### ❖ 3、压缩打包相关命令

命 令↵	命 令 含 义↵	格 式↵
bzip2↵	bz2 文件的压缩（或解压）程序↵	bzip2[选项] 压缩（解压缩）的文件名↵
bunzip2↵	bz2 文件的解压缩程序↵	bunzip2[选项] bz2 压缩文件↵
bzip2recover↵	用来修复损坏的bz2 文件↵	bzip2recover bz2 压缩文件↵
gzip↵	.gz 文件的压缩程序↵	gzip [选项] 压缩（解压缩）的文件名↵
gunzip↵	解压被 gzip 压缩过的文件↵	gunzip [选项] .gz 文件名↵
unzip↵	解压 winzip 压缩的 zip 文件↵	unzip [选项] zip 压缩文件↵
compress↵	早期的压缩或解压程序（压缩后文件名为.Z）↵	compress [选项] 文件↵
tar↵	对文件目录进行打包或解包↵	tar [选项] [打包后文件名]文件目录列表↵





## 6.2 Linux 基本命令

23

### ❖ 1) 压缩和解压缩命令gzip

选 项	参 数 含 义
-c	将输出信息写到标准输出上，并保留原有文件
-d	将压缩文件解压
-l	对每个压缩文件，显示下列字段：压缩文件的大小、未压缩文件的大小、压缩比、未压缩文件的名字
-r	查找指定目录并压缩或解压缩其中的所有文件
-t	测试，检查压缩文件是否完整
-v	对每一个压缩和解压的文件，显示文件名和压缩比

### ❖ (4) 使用实例

- ◆ [root@www my]# gzip -l hello.c





## 6.2 Linux 基本命令

24

### ❖ Linux常见类型的文件解压命令一览表

文件后缀	解压命令	示例
.a	tar xv	tar xv hello.a
.z	Uncompress	uncompress hello.Z
.gz	Gunzip	gunzip hello.gz
.tar.Z	tar xvZf	tar xvZf hello.tar.Z
.tar.gz/.tgz	tar xvzf	tar xvzf hello.tar.gz
tar.bz2	tar jxvf	tar jxvf hello.tar.bz2
.rpm	安装：rpm -i	安装：rpm -i hello.rpm
	解压：rpm2cpio	解压：rpm2cpio hello.rpm
.deb( Debain 中的文件格式 )	安装：dpkg -i	安装：dpkg -i hello.deb
	解压：dpkg-deb --fsys-tarfile	解压：dpkg-deb --fsys-tarhello hello.deb
.zip	Unzip	unzip hello.zip





## 6.2 Linux 基本命令

25

### ❖ 4、比较合并文件相关命令

#### ❖ 1) diff

- 比较两个不同的文件或不同目录下的两个同名文件功能，并生成补丁文件

选 项	参 数 含 义
-r	对目录进行递归处理
-q	只报告文件是否有不同，不输出结果
-e, -ed	命令格式
-f	RCS (修订控制系统) 命令简单格式
-c, --context	旧版上下文格式
-u, --unified	新版上下文格式
-Z	调用 compress 来压缩归档文件，与-x 联用时调用 compress 完成解压缩





## 6.2 Linux 基本命令

26

### ❖ 5、网络相关命令

选 项↵	参 数 含 义↵	常见选项格式↵
netstat↵	显示网络连接、路由表和网络接口信息↵	netstat [-an]↵
nslookup↵	查询一台机器的 IP 地址和其对应的域名↵	Nslookup [IP 地址/域名]↵
finger↵	查询用户的信息↵	finger [选项] [使用者] [用户@主机]↵
ping↵	用于查看网络上的主机是否在工作↵	ping [选项] 主机名/IP 地址↵
ifconfig↵	查看和配置网络接口的参数↵	ifconfig [选项] [网络接口]↵
ftp↵	利用 ftp 协议上传和下载文件↵	在本节中会详细讲述↵
telnet↵	利用 telnet 协议浏览信息↵	telnet [选项] [IP 地址/域名]↵
ssh↵	利用 ssh 登录对方主机↵	ssh [选项] [IP 地址]↵







## 6.2 Linux 基本命令

27

### ❖ 1. ifconfig

- 用于查看和配置网络接口的地址和参数，包括IP地址、网络掩码、广播地址，它的使用权限是超级用户

选 项↵	参 数 含 义↵
-interface↵	指定的网络接口名，如 eth0 和 eth1↵
up↵	激活指定的网络接口卡↵
down↵	关闭指定的网络接口↵
broadcast address↵	设置接口的广播地址↵
point to point↵	启用点对点方式↵
address↵	设置指定接口设备的 IP 地址↵
netmask address↵	设置接口的子网掩码↵





# 第6章 Linux系统及文件I/O操作

- ❖ 6.1 Linux 概述
- ❖ 6.2 Linux 基本命令
- ❖ 6.3 Linux 的开发环境
- ❖ 6.4 文件I/O操作
- ❖ 3学时






## 6.3.1 Linux编辑器vi (vim)

29

### ❖ 全屏幕编辑器



**vi**  
Cheat Sheet

You're in command mode when you start **vi**.

Pressing **ESC** always takes you back to command mode.

**Cursor Movement Commands**

*vi* only lets you move through text that exists. To extend a line, append spaces. To extend the file, append empty lines.

previous word

← **b** ← **h** **j** **k** → **l** → next word

lowercase ell

beginning of line

← **^** (caret)

Most commands can be preceded with a repeat count.

**\$** → end of line

**5w** move ahead 5 words

**16k** move up 16 lines

**1G** go to line 1

**76G** go to line 76

**G** go to end of file

**ctrl G** where am I?

**Changing Commands**...can include any cursor movement command

**c\$** change to end of line

**d5w** delete 5 words

**dG** delete to end of file

**dd** delete current line

**X** delete this character

**u** undo last change

**U** restore line

**Add-text Commands**...press **ESC** when you're done

**a** append here

**A** append at end of line

**i** insert here

**I** insert at start of line

capital i


**R** overstrike

kiting vi...be in command mode! Press **Enter** to end these commands.

:q quit unless save needed

:q! quit, forget changes

:wq write file then quit



© 2005 K Computing. Copy freely.  
Need a fresh copy?  
<http://www.kcomputing.com/vi.html>  
Contact us at [info@kcomputing.com](mailto:info@kcomputing.com)  
+1 512 858 0380



## 6.3.1 Linux编辑器vi (vim)

### ❖ vi有3种模式：命令行模式、插入模式、底行模式

- ◆ (1) 命令行模式
  - 用户在用vi编辑文件时，最初进入的为一般模式。在该模式中可以通过上下移动光标进行“删除字符”或“整行删除”等操作，也可以进行“复制”、“粘贴”等操作，但无法编辑文字。
- ◆ (2) 插入模式
  - 只有在该模式下，用户才能进行文字编辑输入，用户可按[ESC]键回到命令行模式。
- ◆ (3) 底行模式

在该模式下，光标位于屏幕的底行。用户可以进行文件保存或退出操作，也可以设置编辑环境，如寻找字符串、列出行号等。





## 6.3.1 Linux编辑器vi (vim)

### ❖ vi的基本流程

- ◆ (1) 进入vi，即在命令行下键入vi hello（文件名）。此时进入的是命令行模式，光标位于屏幕的上方。
- ◆ (2) 在命令行模式下键入*i*进入到插入模式。在屏幕底部显示有“插入”表示插入模式，在该模式下可以输入文字信息。
- ◆ (3) 最后，在插入模式中，输入“Esc”，则当前模式转入命令行模式，并在底行行中输入“:wq”（存盘退出）进入底行模式

### ❖ 一个简单的vi操作流程

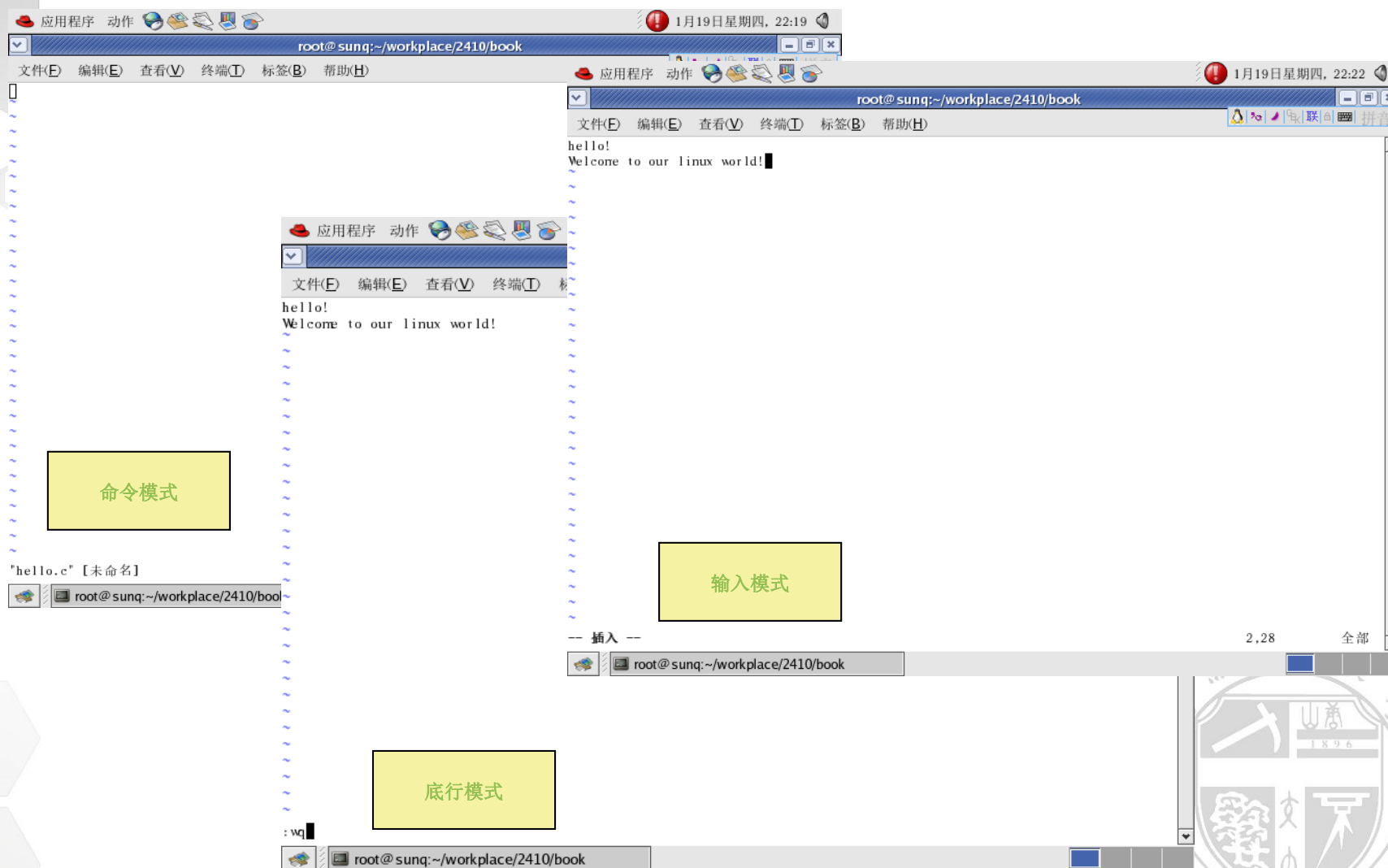
- ◆ 命令行模式→插入模式→底行模式。





## 6.3.1 Linux编辑器有vi (vim)

32







## 6.3.2 gcc 编译器

- ❖ GNU CC（简称为Gcc）是GNU项目中符合ANSI C标准的编译系统，能够编译如C、C++、Object C、Java、Fortran、Pascal、Modula-3和Ada等多种语言，而且gcc又是一个交叉平台编译器，它能够在当前CPU平台上为多种不同体系结构的硬件平台开发软件。
- ❖ 下表是gcc支持编译源文件的后缀及其解释。

后 缀 名↗	所对应的语言↗	后 缀 名↗	所对应的语言↗
.c↗	C 原始程序↗	.s/.S↗	汇编语言原始程序↗
.C/.cc/.cxx↗	C++原始程序↗	.h↗	预处理文件（头文件）↗
.m↗	Objective-C 原始程序↗	.o↗	目标文件↗
.i↗	已经过预处理的 C 原始程序↗	.a/.so↗	编译后的库文件↗
.ii↗	已经过预处理的 C++原始程序↗	↗	↗





## 6.3.2 gcc 编译器

34

### ❖ gcc编译流程分为4个步骤

#### ❖ 预处理 (Pre-Processing)

- ◆ 编译器将上述代码中的stdio.h编译进来

#### ❖ 编译 (Compiling)

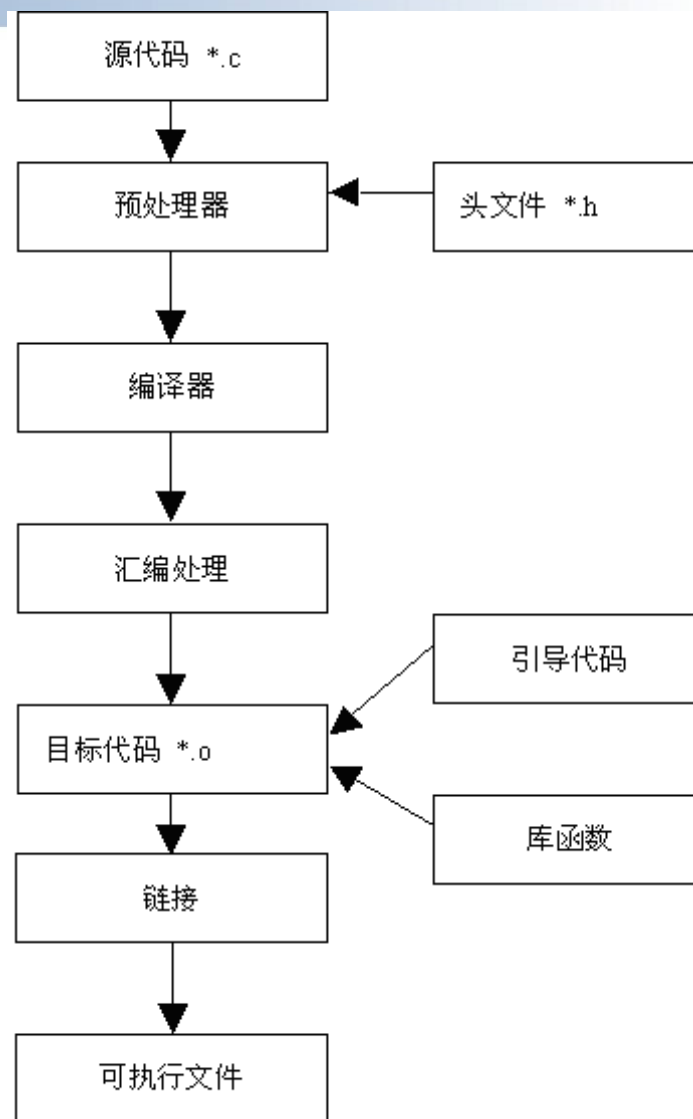
- ◆ gcc首先要检查代码的规范性、是否有语法错误等，gcc把代码翻译成汇编语言

#### ❖ 汇编 (Assembling)

- ◆ 汇编阶段是把编译阶段生成的“.s”文件转成目标文件

#### ❖ 链接 (Linking)

- ◆ 把目标文件和标准库文件进行链接生产可执行的文件





## 6.3.2 gcc 编译器

35

### ❖ gcc --- 常用选项

选项	含义
-c	只编译不链接，生成目标文件“.o”
-S	只编译不汇编，生成汇编代码
-E	只进行预编译，不做其他处理
-g	在可执行程序中包含标准调试信息
-o file	指定将 file 文件作为输出文件
-v	打印出编译器内部编译各过程的命令行信息和编译器的版本
-I dir	在头文件的搜索路径列表中添加 dir 目录





## 6.3.2 gcc 编译器

36

### ❖ gcc --- 体系结构相关选项

选 项	含 义
-mcpu=type	针对不同的 CPU 使用相应的 CPU 指令。可选择的 type 有 i386、i486、pentium 及 i686 等。
-mieee-fp	使用 IEEE 标准进行浮点数的比较。
-mno-ieee-fp	不使用 IEEE 标准进行浮点数的比较。
-msoft-float	输出包含浮点库调用的目标代码。
-mshort	把 int 类型作为 16 位处理，相当于 short int。
-mrtld	强行将函数参数个数固定的函数用 ret NUM 返回，节省调用函数的一条指令。





## 6.3.2 gcc 编译器

37

### ❖ gcc ---库选项

选 项	含 义
-static	进行静态编译，即链接静态库，禁止使用动态库。
-shared	1. 可以生成动态库文件。 2. 进行动态编译，尽可能地链接动态库，只有没有动态库时才会链接同名的静态库（默认选项，即可省略）。
-L dir	在库文件的搜索路径列表中添加 dir 目录。
-lname	链接称为 libname.a（静态库）或者 libname.so（动态库）的库文件。若两个库都存在，则根据编译方式（-static 还是-shared）而进行链接。
-fPIC（或-fpic）	生成使用相对地址的位置无关的目标代码（Position Independent Code）。然后通常使用 gcc 的-static 选项从该 PIC 目标文件生成动态库文件。

### ❖ 函数库分为静态库和动态库两种，

- ◆ 静态库是一系列的目标文件（.o文件）的归档文件（文件名格式为libname.a），如果在编译某个程序时链接静态库，则链接器将会搜索静态库，从中提取出它所需要的目标文件并直接拷贝到该程序的可执行二进制文件(ELF格式文件)之中；
- ◆ 动态库（文件名格式为libname.so[.主版本号.次版本号.发行号]）在程序编译时并不会被链接到目标代码中，而是在程序运行时才被载入。





## 6.3.2 gcc 编译器

38

### ❖ gcc --- 警告选项

选 项	含 义
-ansi	支持符合 ANSI 标准的 C 程序
-pedantic	允许发出 ANSI C 标准所列的全部警告信息
-pedantic-error	允许发出 ANSI C 标准所列的全部错误信息
-w	关闭所有告警
-Wall	允许发出 gcc 提供的所有有用的报警信息
-Werror	把所有的告警信息转化为错误信息，并在告警发生时终止编译过程





## 6.3.3 gdb调试器

39

- ❖ gdb调试器是一款GNU开发组织并发布的UNIX/Linux下的程序调试工具。虽然，它没有图形化的友好界面，但是它强大的功能也足以与微软的VC工具等媲美。

```
$ gcc -g test.c -o test
$ gdb test
GNU gdb Red Hat Linux (6.3.0.0-1.21rh)
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB. Type "show warranty" for details.
This GDB was configured as "i386-redhat-linux-gnu"...Using host libthread_db library "/lib/libthread_db.so.1".
(gdb)
```





## 6.3.3 gdb调试器

40

### ❖ gdb使用流程

- ◆ 首先，用编辑器vi，编辑代码**vi test.c**
- ◆ 使用gcc对test.c进行编译 **gcc -g test.c -o test**
- ◆ 接下来就启动gdb进行调试。 **gdb test**

### ❖ 在 “（gdb）” 开头的命令行界面

- ◆ （1）查看文件，在gdb中键入 “l”（list）就可以查看所载入的文件
- ◆ （2）设置断点，在gdb中设置断点非常简单，只需在 “b”后加入对应的行号即可
- ◆ （3）查看断点情况，在设置完断点之后，用户可以键入 “info b”来查看设置断点情况，在gdb中可以设置多个断点
- ◆ （4）运行代码，gdb默认从首行开始运行代码，可键入 “r”（run）即可
- ◆ （5）查看变量值，在gdb中只需键入 “p”+变量值即可
- ◆ （6）单步运行，单步运行可以使用命令 “n”（next）或 “s”（step）
- ◆ （7）恢复程序运行，可以使用命令 “c”（continue）恢复程序的正常运行了







## 6.3.3 gdb调试器

41

```
(gdb) l
1  #include <stdio.h>
2  int sum(int m);
3  int main()
4  {
5      int i,n = 0;
6      sum(50);
7      for(i = 1; i <= 50; i++)
8      {
9          n += i;
10     }
(gdb) l
11     printf("The sum of 1~50 is %d \n", n );
12
13 }
14 int sum(int m)
15 {
16     int i, n = 0;
17     for(i = 1; i <= m; i++)
18     {
19         n += i;
20     }
21     printf("The sum of 1~m is = %d\n", n);
22 }
```

(gdb) b 6

Breakpoint 1 at 0x804846d: file test.c, line 6.

(gdb) info b

Num	Type	Disp	Enb	Address	What
1	breakpoint	keep	y	0x0804846d	in main at test.c:6

(gdb) b 19

(gdb) c

Breakpoint 2, sum(m=50) at test.c:19

19 printf("The sum of 1-m is %d\n", n);

(gdb) bt

#0 sum(m=50) at test.c:19

#1 0x080483e8 in main() at test.c:6

(gdb) r

Starting program: /root/workplace/gdb/test

Reading symbols from shared object read from target memory...done.

Loaded system supplied DSO at 0x5fb000

Breakpoint 1, main () at test.c:6

6 sum(50);





## 6.3.3 gdb调试器

42

```
(gdb) p n
$1 = 0
(gdb) p i
$2 = 134518440
```

```
(gdb) n
The sum of 1-m is 1275
7      for (i = 1; i <= 50; i++)
(gdb) s
sum (m=50) at test.c:16
16      int i, n = 0;
```

```
(gdb) c
Continuing.
The sum of 1-50 is :1275
Program exited with code 031.
```

### (gdb) help

List of classes of commands:

aliases -- Aliases of other commands

breakpoints -- Making program stop at certain points

data -- Examining data

files -- Specifying and examining files

internals -- Maintenance commands

...

**Type "help" followed by a class name for a list of commands in that class.**

Type "help" followed by command name for full documentation.  
Command name abbreviations are allowed if unambiguous.

### (gdb) help call

Call a function in the program.

The argument is the function name and arguments, in the notation of the

current working language. The result is printed and saved in the value

history, if it is not void.



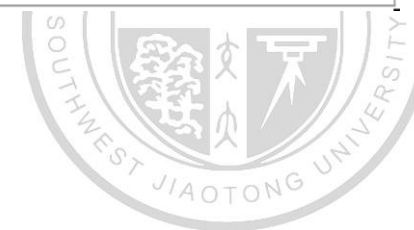


## 6.3.3 gdb调试器

43

### ❖ gdb工作环境相关命令

命令格式	含 义
<code>set args</code> 运行时的参数	指定运行时参数，如 <code>set args 2</code>
<code>show args</code>	查看设置好的运行参数
<code>path dir</code>	设定程序的运行路径
<code>show paths</code>	查看程序的运行路径
<code>set environment var [=value]</code>	设置环境变量
<code>show environment [var]</code>	查看环境变量
<code>cd dir</code>	进入到 <code>dir</code> 目录，相当于 <code>shell</code> 中的 <code>cd</code> 命令
<code>pwd</code>	显示当前工作目录
<code>shell command</code>	运行 <code>shell</code> 的 <code>command</code> 命令





## 6.3.3 gdb调试器

44

### ❖ gdb设置断点与恢复命令

命令格式	含 义
<code>info b</code>	查看所设断点
<code>break [文件名:]行号或函数名 &lt;条件表达式&gt;</code>	设置断点
<code>tbreak [文件名:]行号或函数名 &lt;条件表达式&gt;</code>	设置临时断点，到达后被自动删除
<code>delete [断点号]</code>	删除指定断点，其断点号为“ <code>info b</code> ”中的第一栏。若缺省断点号则删除所有断点
<code>disable [断点号]</code>	停止指定断点，使用“ <code>info b</code> ”仍能查看此断点。同 <code>delete</code> 一样，省断点号则停止所有断点
<code>enable [断点号]</code>	激活指定断点，即激活被 <code>disable</code> 停止的断点
<code>condition [断点号] &lt;条件表达式&gt;</code>	修改对应断点的条件
<code>ignore [断点号] &lt;num&gt;</code>	在程序执行中，忽略对应断点 <code>num</code> 次
<code>Step</code>	单步恢复程序运行，且进入函数调用
<code>Next</code>	单步恢复程序运行，但不进入函数调用
<code>Finish</code>	运行程序，直到当前函数完成返回
<code>c</code>	继续执行函数，直到函数结束或遇到新的断点



## 6.3.3 gdb调试器

45

### ❖ gdb中源码查看相关命令

命令格式	含义
<code>list &lt;行号&gt; &lt;函数名&gt;</code>	查看指定位置代码
<code>file [文件名]</code>	加载指定文件
<code>forward-search 正则表达式</code>	源代码前向搜索
<code>reverse-search 正则表达式</code>	源代码后向搜索
<code>dir dir</code>	停止路径名
<code>show directories</code>	显示定义了的源文件搜索路径
<code>info line</code>	显示加载到 <code>gdb</code> 内存中的代码





## 6.3.3 gdb调试器

46

### ❖ gdb中查看运行数据的相关命令

命令格式	含 义
<code>print 表达式 变量</code>	查看程序运行时对应表达式和变量的值
<code>x &lt;n/f/u&gt;</code>	查看内存变量内容。其中 <b>n</b> 为整数表示显示内存的长度， <b>f</b> 表示显示的格式， <b>u</b> 表示从当前地址往后请求显示的字节数
<code>display 表达式</code>	设定在单步运行或其他情况中，自动显示的对应表达式的内容
<code>backtrace</code>	查看当前栈的情况，即可以查到调用哪些函数尚未返回。





## 6.3.4 make工程管理器

47

- ❖ 工程管理器，顾名思义，是指管理较多的文件
- ❖ Make工程管理器也就是个“自动编译管理器”，
  - ◆ “自动”是指它能根据文件时间戳自动发现更新过的文件而减少编译的工作量，同时，它通过读入Makefile文件文件的内容来执行大量的编译工作
- ❖ Make大大提高了实际项目的工作效率，而且几乎所有Linux下的项目编程均会涉及它





## 6.3.4 make工程管理器

48

### ❖ 1、Makefile基本结构

❖ Makefile是Make读入的惟一配置文件，在一个Makefile中通常包含如下内容：

- ◆ 需要由make工具创建的目标体（target），通常是目标文件或可执行文件；
- ◆ 要创建的目标体所依赖的文件（dependency\_file）；
- ◆ 创建每个目标体时需要运行的命令（command），**这一行必须以制表符（tab键）开头。**







## 6.3.4 make工程管理器

49

### ▶ makefile格式

```
target: dependency_files  
< TAB >command /* 该行必须以tab键开头*/
```

### ▶ 例子

```
hello.o: hello.c hello.h  
gcc -c hello.c -o hello.o
```

### ▶ 使用makefile

```
$ make hello.o  
gcc -c hello.c -o hello.o  
$ ls  
hello.c hello.h hello.o makefile
```





## 6.3.4 make工程管理器

50

### ❖ 2、创建和使用makefile变量

- ❖ 用来代替一个文本字符串
- ❖ 变量定义的两种方式
  - ◆ 递归展开方式VAR=var
  - ◆ 简单方式 VAR: =var
- ❖ 变量使用\$(VAR)

```
OBJS = kang.o yul.o
```

```
CC = gcc
```

```
CFLAGS = -Wall -O -g
```

```
david : $(OBJS)
```

```
    $(CC) $(OBJS) -o david
```

```
kang.o : kang.c kang.h
```

```
    $(CC) $(CFLAGS) -c kang.c -o kang.o
```

```
yul.o : yul.c yul.h
```

```
    $(CC) $(CFLAGS) -c yul.c -o yul.o
```



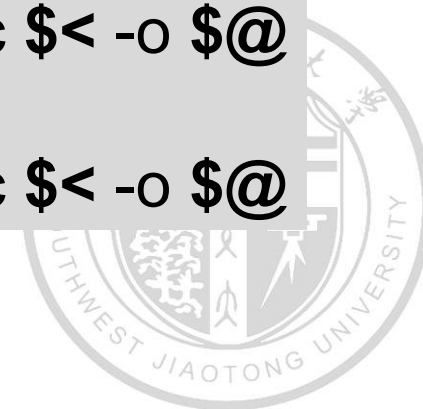
## 6.3.4 make工程管理器

51

### ❖ 变量种类

- ◆ 用户自定义变量
- ◆ 预定义变量
- ◆ 自动变量
- ◆ 环境变量

```
OBJS = kang.o yul.o
CC = gcc
CFLAGS = -Wall -O -g
david : $(OBJS)
    $(CC) $^ -o $@
kang.o : kang.c kang.h
    $(CC) $(CFLAGS) -c $< -o $@
yul.o : yul.c yul.h
    $(CC) $(CFLAGS) -c $< -o $@
```





## 6.3.4 make工程管理器

52

### ❖ makefile中常用的预定义变量

命令格式	含 义
AR	库文件维护程序的名称，默认值为 <code>ar</code>
AS	汇编程序的名称，默认值为 <code>as</code>
CC	C 编译器的名称，默认值为 <code>cc</code>
CPP	C 预编译器的名称，默认值为 <code>\$(CC) -E</code>
CXX	C++编译器的名称，默认值为 <code>g++</code>
FC	FORTTRAN 编译器的名称，默认值为 <code>f77</code>
RM	文件删除程序的名称，默认值为 <code>rm -f</code>
ARFLAGS	库文件维护程序的选项，无默认值
ASFLAGS	汇编程序的选项，无默认值
CFLAGS	C 编译器的选项，无默认值
CPPFLAGS	C 预编译的选项，无默认值
CXXFLAGS	C++编译器的选项，无默认值
FFLAGS	FORTTRAN 编译器的选项，无默认值



## 6.3.4 make工程管理器

53

### ❖ makefile中常见的自动变量 和环境变量

自动变量	含 义
\$* ↕	不包含扩展名的目标文件名称↕
\$+ ↕	所有的依赖文件，以空格分开，并以出现的先后为序，可能包含重复的依赖文件↕
\$< ↕	第一个依赖文件的名称↕
\$? ↕	所有时间戳比目标文件晚的依赖文件，并以空格分开 ↕
\$@ ↕	目标文件的完整名称↕
^ ↕	所有不重复的依赖文件，以空格分开↕
% ↕	如果目标是归档成员，则该变量表示目标的归档成员名称↕

- ◆ make在启动时会自动读取系统当前已经定义了的环境变量，并且会创建与之具有相同名称和数值的变量
- ◆ 如果用户在makefile中定义了相同名称的变量，那么用户自定义变量将会覆盖同名的环境变量





## 6.3.4 make工程管理器

54

### ❖ 3、makefile规则

- ❖ makefile的规则是make进行处理的依据，它包括了目标体、依赖文件及其之间的命令语句。
- ❖ Makefile中的一条语句就是一个规则。
  - ◆ 如 “\$(CC) \$(CFLAGS) -c \$< -o \$@”，
- ❖ 隐式规则、模式规则





## 6.3.4 make工程管理器

55

### ❖ 隐式规则

- ◆ 告诉make怎样使用传统的技术完成任务，使用它们时就不必详细指定编译的具体细节，而只需把目标文件列出即可。
- ◆ Make会自动搜索隐式规则目录来确定如何生成目标文件。

```
OBJS = kang.o yul.o
CC = gcc
CFLAGS = -Wall -O -g
david : $(OBJS)
        $(CC) $^ -o $@
```

对应语言后缀名	隐式规则
C 编译: .c 变为.o	$$(CC) -c $(CPPFLAGS) $(CFLAGS)$
C++编译: .cc 或.C 变为.o	$$(CXX) -c $(CPPFLAGS) $(CXXFLAGS)$
Pascal 编译: .p 变为.o	$$(PC) -c $(PFLAGS)$
Fortran 编译: .f 变为.o	$$(FC) -c $(FFLAGS)$





## 6.3.4 make工程管理器

56

### ❖ 模式规则

- ◆ 用来定义相同处理规则的多个文件。
- ◆ 模式规则还能引入用户自定义变量，为多个文件建立相同的规则，从而简化了makefile的编写
- ◆ 相关文件前必须用 “%”标明

```
OBJS = kang.o yul.o
CC = gcc
CFLAGS = -Wall -O -g
david : $(OBJS)
    $(CC) $^ -o $@
%.o : %.c
    $(CC) $(CFLAGS) -c $< -o $@
```







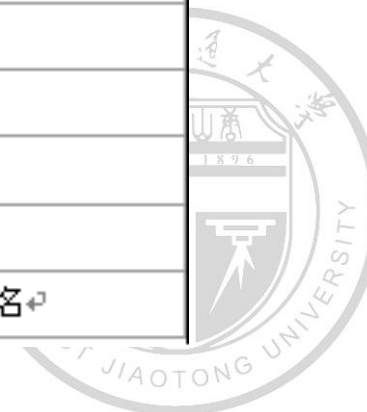
## 6.3.4 make工程管理器

57

### ❖ 4、Make管理器的使用

- ❖ 在make命令的后面键入目标名即可建立指定的目标，
  - 直接运行make，则建立Makefile中的第一个目标。
- ❖ make的命令行选项

命令格式↵	含 义↵
-C dir↵	读入指定目录下的 Makefile↵
-f file↵	读入当前目录下的 file 文件作为 Makefile↵
-i↵	忽略所有的命令执行错误↵
-I dir↵	指定被包含的 Makefile 所在目录↵
-n↵	只打印要执行的命令，但不执行这些命令↵
-p↵	显示 make 变量数据库和隐含规则↵
-s↵	在执行命令时不显示命令↵
-w↵	如果 make 在执行过程中改变目录，则打印当前目录名↵





# 第6章 Linux系统及文件I/O操作

- ❖ 6.1 Linux 概述
- ❖ 6.2 Linux 基本命令
- ❖ 6.3 Linux 的开发环境
- ❖ 6.4 文件I/O操作
- ❖ 3学时





## 6.4.1 Linux系统调用及用户编程接口（API）

59

- ❖ 系统调用是指操作系统提供给用户程序调用的一组“特殊”接口，用户程序可以通过这组“特殊”接口来获得操作系统内核提供的服务。
- ❖ Linux中将程序的运行空间分为**内核空间**和**用户空间**（也就是常称的内核态和用户态），它们分别运行在不同的级别上，在逻辑上是相互隔离的。
- ❖ 用户空间的进程需要获得一定的系统服务（调用内核空间程序），这时操作系统就必须利用系统提供给用户的“特殊接口”——系统调用规定用户进程进入内核空间的具体位置。
- ❖ 系统调用时，程序运行空间需要从用户空间进入内核空间，处理完后再返回到用户空间。



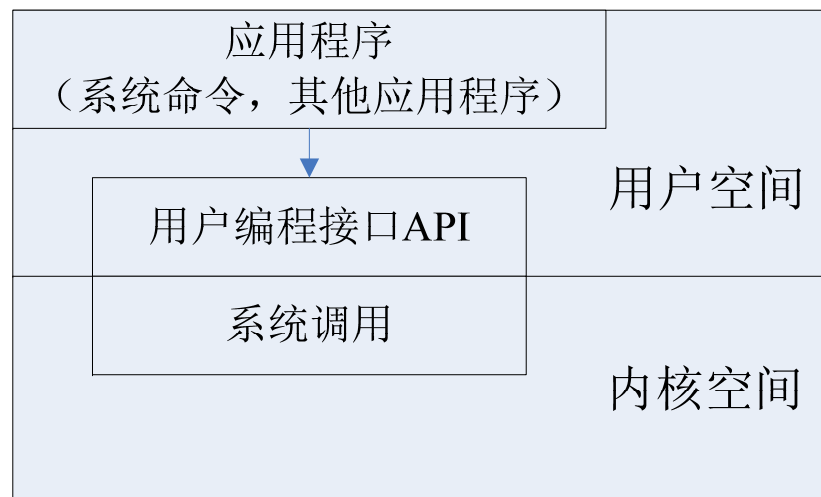


## 6.4.1 Linux系统调用及用户编程接口（API）

60

### ❖ 用户编程接口（API）

- ❖ 系统调用并不是直接与程序员进行交互的，它仅仅是一个通过软中断机制向内核提交请求，以获取内核服务的接口（API）
- ❖ 系统命令相对API更高了一层，它实际上一个可执行程序，它的内部引用了用户编程接口（API）来实现相应的功能。





## 6.4.1 Linux系统调用及用户编程接口（API）

### ❖ 概念：

#### ❖ (1) 用户空间和内核空间（用户态和内核态）

#### ❖ (2) shell命令解释器

#### ❖ (3) 系统调用

- ◆ 用户程序通常不能直接访问系统内核提供的服务，在Linux中，为了保护内核空间，将程序的运行空间分为用户空间和内核空间
- ◆ 也就是常称的用户态和内核态。
- ◆ 用户进程通常情况下不允许访问内核数据，也无法使用内核函数，它们只能在用户空间操作用户数据，调用用户空间的函数。





## 6.4.2 Linux中文件及文件描述符概述

62

- ❖ 在Linux系统中一切皆可以看成是文件，Linux把包括硬件设备在内的能够进行流式字符操作的内容都定义为文件。
- ❖ Linux系统中文件的类型包括：普通文件，目录文件，连接文件，管道（FIFO）文件、设备文件（块设备、字符设备）和套接字。





## ❖ 文件描述符

- ❑ 对于Linux而言，所有对设备和文件的操作都是使用文件描述符来进行的。
- ❑ 文件描述符是一个非负的整数，它是一个索引值，并指向在内核中每个进程打开文件的记录表。
- ❑ 当打开一个现存文件或创建一个新文件时，内核就向进程返回一个文件描述符；当需要读写文件时，也需要把文件描述符作为参数传递给相应的函数。





## 6.4.3 底层文件I/O操作

64

❖ 主要有5个函数：

- ◆ `open()`;
- ◆ `read()`;
- ◆ `write()`;
- ◆ `lseek()`;
- ◆ `close()`。
- ◆ 还有 `fctl()`, `select()` 等







## 6.4.3 底层文件I/O操作

65

### ❖ 函数说明

- ❑ `open()`函数是用于打开或创建文件，在打开或创建文件时可以指定文件的属性及用户的权限等各种参数。
- ❑ `close()`函数是用于关闭一个被打开的文件。当一个进程终止时，所有被它打开的文件都由内核自动关闭，很多程序都使用这一功能而不显示地关闭一个文件。
- ❑ `read()`函数是用于将从指定的文件描述符中读出的数据放到缓存区中，并返回实际读入的字节数。若返回0，则表示没有数据可读，即已达到文件尾。读操作从文件的当前指针位置开始。当从终端设备文件中读出数据时，通常一次最多读一行。





## 6.4.3 底层文件I/O操作

66

- ❖ `write()`函数是用于向打开的文件写数据，写操作从文件的当前指针位置开始。对磁盘文件进行写操作，若磁盘已满或超出该文件的长度，则`write()`函数返回失败。
- ❖ `lseek()`函数是用于在指定的文件描述符中将文件指针定位到相应的位置。它只能用在可定位（可随机访问）文件操作中。管道、套接字和大部分字符设备文件是不可定位的，所以在这些文件的操作中无法使用`lseek()`调用。





## 6.4.3 底层文件I/O操作

67

### ❖ open()函数格式

所需头文件	#include <sys/types.h> /* 提供类型 pid_t 的定义 */ #include <sys/stat.h> #include <fcntl.h>	
函数原型	int open(const char *pathname, int flags, int perms)	
函数传入值	pathname	被打开的文件名（可包括路径名）
	flag: 文件打开的方式	O_RDONLY: 以只读方式打开文件
		O_WRONLY: 以只写方式打开文件
		O_RDWR: 以读写方式打开文件
		O_CREAT: 如果该文件不存在, 就创建一个新的文件, 并用第三个参数为其设置权限
		O_EXCL: 如果使用 O_CREAT 时文件存在, 则可返回错误消息。这一参数可测试文件是否存在。此时 open 是原子操作, 防止多个进程同时创建同一个文件。
		O_NOCTTY: 使用本参数时, 若文件为终端, 那么该终端不会成为调用 open() 的那个进程的控制终端
		O_TRUNC: 若文件已经存在, 那么会删除文件中的全部原有数据, 并且设置文件大小为 0。
函数传入值	perms	O_APPEND: 以添加方式打开文件, 在打开文件的同时, 文件指针指向文件的末尾, 即将写入的数据添加到文件的末尾。
		被打开文件的存取权限。 可以用一组宏定义: S_I(R/W/X)(USR/GRP/OTH) 其中 R/W/X 分别表示读/写/执行权限 USR/GRP/OTH 分别表示文件所有者/文件所属组/其他用户。 例如 S_IRUSR   S_IWUSR 表示设置文件所有者的可读可写属性。八进制表示法中 600 也表示同样的权限。
函数返回值	成功: 返回文件描述符 失败: -1	



## 6.4.3 底层文件I/O操作

68

### ❖ close()和read()函数

所需头文件↵	#include <unistd.h>↵
函数原型↵	int close(int fd)↵
函数输入值↵	fd: 文件描述符↵
函数返回值↵	0: 成功↓ -1: 出错↵

所需头文件↵	#include <unistd.h>↵
函数原型↵	ssize_t read(int fd, void *buf, size_t count)↵
函数传入值↵	fd: 文件描述符↵
	buf: 指定存储器读出数据的缓冲区↵
	count: 指定读出的字节数↵
函数返回值↵	成功: 读到的字节数↓ 0: 已到达文件尾↓ -1: 出错↵





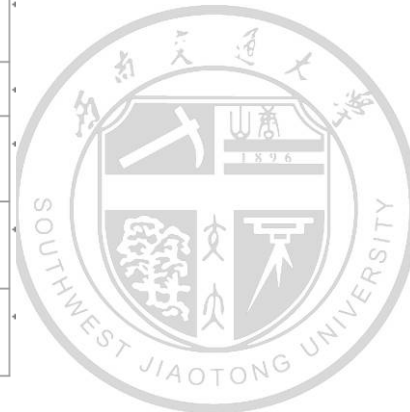
## 6.4.3 底层文件I/O操作

69

### ❖ write() 和lseek()函数

所需头文件	#include <unistd.h>
函数原型	ssize_t write(int fd, void *buf, size_t count)
函数传入值	fd: 文件描述符
	buf: 指定存储器写入数据的缓冲区
	count: 指定读出的字节数
函数返回值	成功: 已写的字节数 -1: 出错

所需头文件	#include <unistd.h> #include <sys/types.h>				
函数原型	off_t lseek(int fd, off_t offset, int whence)				
函数传入值	fd: 文件描述符				
	offset: 偏移量, 每一读写操作所需要移动的距离, 单位是字节, 可正可负 (向前移, 向后移)				
	<table><tr><td rowspan="3">whence: 当前位置的基点</td><td>SEEK_SET: 当前位置为文件的开头, 新位置为偏移量的大小</td></tr><tr><td>SEEK_CUR: 当前位置为文件指针的位置, 新位置为当前位置加上偏移量</td></tr><tr><td>SEEK_END: 当前位置为文件的结尾, 新位置为文件的大小加上偏移量的大小</td></tr></table>	whence: 当前位置的基点	SEEK_SET: 当前位置为文件的开头, 新位置为偏移量的大小	SEEK_CUR: 当前位置为文件指针的位置, 新位置为当前位置加上偏移量	SEEK_END: 当前位置为文件的结尾, 新位置为文件的大小加上偏移量的大小
whence: 当前位置的基点	SEEK_SET: 当前位置为文件的开头, 新位置为偏移量的大小				
	SEEK_CUR: 当前位置为文件指针的位置, 新位置为当前位置加上偏移量				
	SEEK_END: 当前位置为文件的结尾, 新位置为文件的大小加上偏移量的大小				
函数返回值	成功: 文件的当前位移 -1: 出错				





## 6.4.3 底层文件I/O操作

70

### ❖ 文件锁

- ❑ 文件锁包括**建议性锁**和**强制性锁**。建议性锁要求每个上锁文件的进程都要检查是否有锁存在，并且尊重已有的锁。
- ❑ 强制性锁是由内核执行的锁，当一个文件被上锁进行写入操作的时候，内核将阻止其他任何文件对其进行读写操作。
- ❑ 在Linux中，实现文件上锁的函数有lockf()和fcntl()，其中lockf()用于对文件施加建议性锁，而**fcntl()**不仅可以施加建议性锁，还可以施加强制锁。同时，fcntl()还能对文件的某一记录上锁，也就是记录锁。
- ❑ 记录锁又可分为读取锁和写入锁，其中读取锁又称为共享锁，它能够使多个进程都能在文件的同一部分建立读取锁。而写入锁又称为排斥锁，在任何时刻只能有一个进程在文件的某个部分上建立写入锁。





## 6.4.3 底层文件I/O操作

71

### ❖ fcntl()函数

所需头文件	#include <sys/types.h> #include <unistd.h> #include <fcntl.h>	
函数原型	int fcntl(int fd, int cmd, struct flock *lock)	
函数传入值	fd:	文件描述符
	cmd	F_DUPFD: 复制文件描述符
		F_GETFD: 获得 fd 的 close-on-exec 标志, 若标志未设置, 则文件经过 exec()函数之后仍保持打开状态
		F_SETFD: 设置 close-on-exec 标志, 该标志由参数 arg 的 FD_CLOEXEC 位决定
		F_GETFL: 得到 open 设置的标志
		F_SETFL: 改变 open 设置的标志
		F_GETLK: 根据 lock 描述, 决定是否上文件锁
		F_SETLK: 设置 lock 描述的文件锁
		F_SETLKW: 这是 F_SETLK 的阻塞版本 (命令名中的 w 表示等待 (wait))。在无法获取锁时, 会进入睡眠状态; 如果可以获取锁或者捕捉到信号则会返回。
	lock:	结构为 flock, 设置记录锁的具体状态, 后面会详细说明
函数返回值	成功: 0 -1: 出错	







## 6.4.3 底层文件I/O操作

### ❖ Linux下文件操作示例

◆ `int creat(const char *pathname, mode_t mode);`

以mode方式创建一个以pathname为文件名的文件，返回新的文件句柄fd，错误返回-1及错误代码errno。

◆ `size_t read(int fd, void *buf, size_t count);`

把fd指向的文件传送count字节到buf指针所指向的内存中，正确返回实际写入的字节数，错误返回-1及错误代码errno。

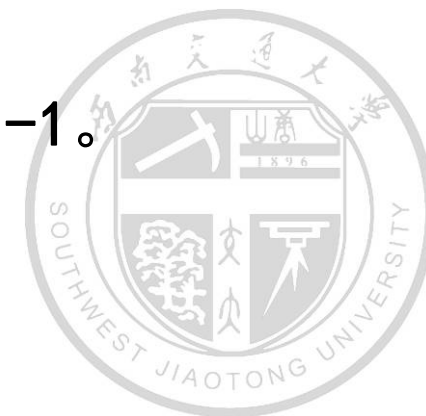






## 6.4.3 底层文件I/O操作

- ◆ `size_t write(int fd, void *buf, size_t count);`  
把buf指针指向的内存count字节传送到fd指向的文件中，正确返回读到的字节数或0，错误返回-1及代码errno。
- ◆ `off_t lseek(int fd, off_t offset, int where);`  
将fd所指文件的读写指针在where位置移动offset个位移量
- ◆ `int close(int fd);`  
关闭fd所指文件，顺利关闭返回0，错误返回-1。





## 6.4.3 底层文件I/O操作

### Linux下文件操作编程实例（教材P135）

```
❖ #include <unistd.h>
❖ #include <sys/stat.h>
❖ #include <fcntl.h>
❖ #include <stdlib.h>
❖ int main()
❖ {
    ◆ char c;
    ◆ int in, out;
    ◆ in = open("file.in", O_RDONLY);
    ◆ out = open("file.out", O_WRONLY|O_CREAT);
    ◆ while(read(in,&c,1) == 1)
    ◆ write(out,&c,1);
    ◆ exit(0);
❖ }
```

❖ 思考题：如何给一个文件上锁？





## 6.4.3 底层文件I/O操作

75

### ❖ I/O处理的模型

#### ❖ 阻塞I/O模型：

- ◆ 若所调用的I/O函数没有完成相关的功能，则会使进程挂起，直到相关数据到达才会返回。如常见对管道设备、终端设备和网络设备进行读写时经常会出现这种情况。

#### ❖ 非阻塞模型：

- ◆ 当请求的I/O操作不能完成时，则不让进程睡眠，而且立即返回。非阻塞I/O使用户可以调用不会阻塞的I/O操作，如open()、write()和read()。
- ◆ 如果该操作不能完成，则会立即返回出错（例如：打不开文件）或者返回0（例如：在缓冲区中没有数据可以读取或者没空间可以写入数据）。





## 6.4.3 底层文件I/O操作

76

### ❖ I/O多路转接模型：

- ◆ 如果请求的I/O操作阻塞，且它不是真正阻塞I/O，而是让其中的一个函数等待，在这期间，I/O还能进行其他操作。

### ❖ 信号驱动I/O模型

- ◆ 通过安装一个信号处理程序，系统可以自动捕获特定信号的到来，从而启动I/O。这是由内核通知用户何时可以启动一个I/O操作决定的。

### ❖ 异步I/O模型：

- ◆ 当一个描述符已准备好，可以启动I/O时，进程会通知内核。现在，并不是所有的系统都支持这种模型。





## 6.4.4 标准I/O编程

77

### ❖ 标准I/O编程概述

- ❑ 系统调用是操作系统直接提供的函数接口。
- ❑ 因为运行系统调用时，Linux必须从用户态切换到内核态，执行相应的请求，然后再返回到用户态，所以应该尽量减少系统调用的次数，从而提高程序的效率。
- ❑ 标准I/O提供流缓冲的目的是尽可能减少使用read()和write()等系统调用的数量。
- ❑ 标准I/O提供了3种类型的缓冲存储。





## 6.4.4 标准I/O编程

78

### ❖ 全缓冲:

- ◆ 当填满标准I/O缓存后才进行实际I/O操作。对于存放在磁盘上的文件通常是由标准I/O库实施全缓冲的。

### ❖ 行缓冲:

- ◆ 当在输入和输出中遇到行结束符时，标准I/O库执行I/O操作。这允许我们一次输出一个字符（如fputc()函数），但只有写了一行之后才进行实际I/O操作。标准输入和标准输出就是使用行缓冲的典型例子。

### ❖ 不带缓冲:

- ◆ 标准I/O库不对字符进行缓冲。如果用标准I/O函数写若干字符到不带缓冲的流中，则相当于用系统调用write()函数将这些字符全写到被打开的文件上





## 6.4.4 标准I/O编程

79

### ❖ 1、打开文件

#### ❖ 有三个标准函数

- ◆ `fopen()`: 可以指定打开文件的路径和模式
- ◆ `fdopen()`: 可以指定打开的文件描述符和模式
- ◆ `freopen()`: 可指定打开的文件、模式外, 还可指定特定的I/O流。

❖ 它们以不同的模式打开, 但都返回一个指向FILE的指针, 该指针指向对应的I/O流。

❖ 对文件的读写都是通过这个FILE指针来进行





## 6.4.4 标准I/O编程

80

所需头文件	<code>#include &lt;stdio.h&gt;</code>
函数原型	<code>FILE * fopen(const char * path, const char * mode)</code>
函数传入值	<b>Path:</b> 包含要打开的文件路径及文件名
	<b>mode:</b> 文件打开状态（后面会具体说明）
函数返回值	成功：指向 <b>FILE</b> 的指针 失败：NULL

### ❖ mode定义打开文件的访问权限

<b>r</b> 或 <b>rb</b>	打开只读文件，该文件必须存在
<b>r+</b> 或 <b>r+b</b>	打开可读写的文件，该文件必须存在
<b>w</b> 或 <b>wb</b>	打开只写文件，若文件存在则文件长度清为 0，即会擦写文件以前的内容。若文件不存在则建立该文件
<b>w+</b> 或 <b>w+b</b>	打开可读写文件，若文件存在则文件长度清为 0，即会擦写文件以前的内容。若文件不存在则建立该文件
<b>a</b> 或 <b>ab</b>	以附加的方式打开只写文件。若文件不存在，则会建立该文件；如果文件存在，写入的数据会被加到文件尾，即文件原先的内容会被保留
<b>a+</b> 或 <b>a+b</b>	以附加方式打开可读写的文件。若文件不存在，则会建立该文件；如果文件存在，写入的数据会被加到文件尾后，即文件原先的内容会被保留







## 6.4.4 标准I/O编程

81

### ❖ 2、关闭文件

- ◆ 关闭标准流文件的函数为**fclose()**，该函数将缓冲区内的数据全部写入到文件中，并释放系统所提供的文件资源

所需头文件	<code>#include &lt;stdio.h&gt;</code>
函数原型	<code>int fclose(FILE * stream)</code>
函数传入值	<b>stream</b> : 已打开的文件指针
函数返回值	成功: 0 失败: EOF





## 6.4.4 标准I/O编程

82

### ❖ 3、读操作的函数为 **fread()** :

所需头文件	<code>#include &lt;stdio.h&gt;</code>
函数原型	<code>size_t fread(void * ptr, size_t size, size_t nmemb, FILE * stream)</code>
函数传入值	<b>ptr</b> : 存放读入记录的缓冲区
	<b>size</b> : 读取的记录大小
	<b>nmemb</b> : 读取的记录数
	<b>stream</b> : 要读取的文件流
函数返回值	成功: 返回实际读取到的 <b>nmemb</b> 数目 失败: <b>EOF</b>





## 6.4.4 标准I/O编程

83

### ❖ 4、fwrite()函数

所需头文件	<code>#include &lt;stdio.h&gt;</code>
函数原型	<code>size_t fwrite(const void * ptr, size_t size, size_t nmemb, FILE * stream)</code>
函数传入值	<b>ptr</b> : 存放写入记录的缓冲区
	<b>size</b> : 写入的记录大小
	<b>nmemb</b> : 写入的记录数
	<b>stream</b> : 要写入的文件流
函数返回值	成功: 返回实际写入到的 <b>nmemb</b> 数目 失败: <b>EOF</b>





# 作业

84

❖ P164 思考题1

❖ 假设有源文件old.txt,请使用IO完成该文件的拷贝new.txt。

