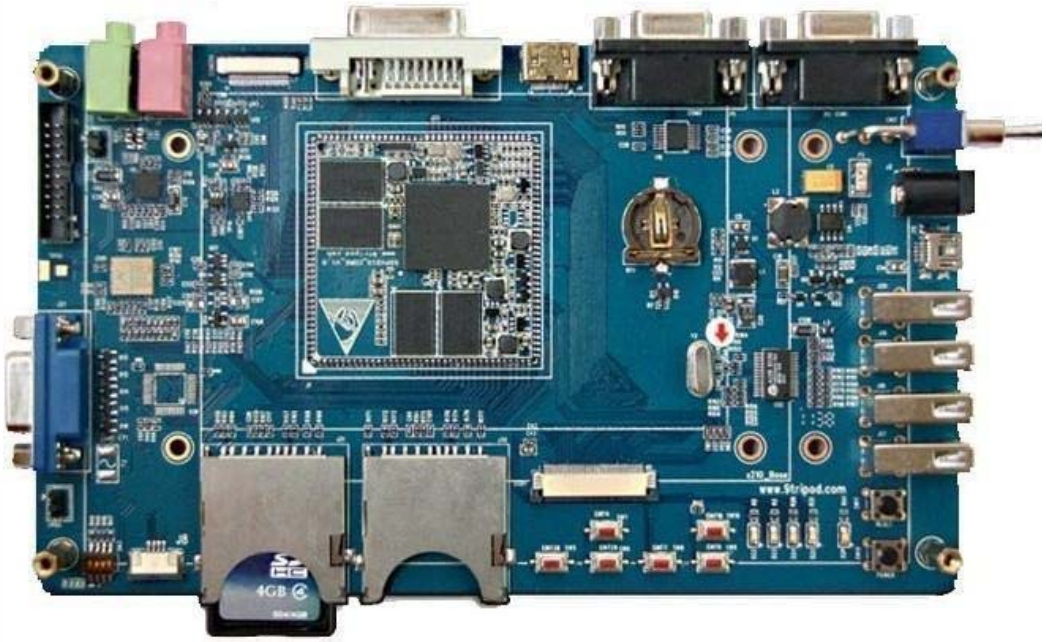
嵌入式系统原理

Principle of Embedded System

计算机与信息学院　　分布智能与物联网研究所

嵌入式的层次划分



- 应用层
- 中间件
- 系统软件层
- 驱动层（中间层）
- 硬件层

嵌入式的层次划分

Linux基本使用、常用命令

✓Linux编程基础

✓Shell编程

✓C语言编程、进程管理、网络编程

嵌入式Linux文件系统

✓ 嵌入式Linux编程基

✓ 交叉编译

✓ 嵌入式程序烧写

✓ 驱动程序设计

本章主要讲述

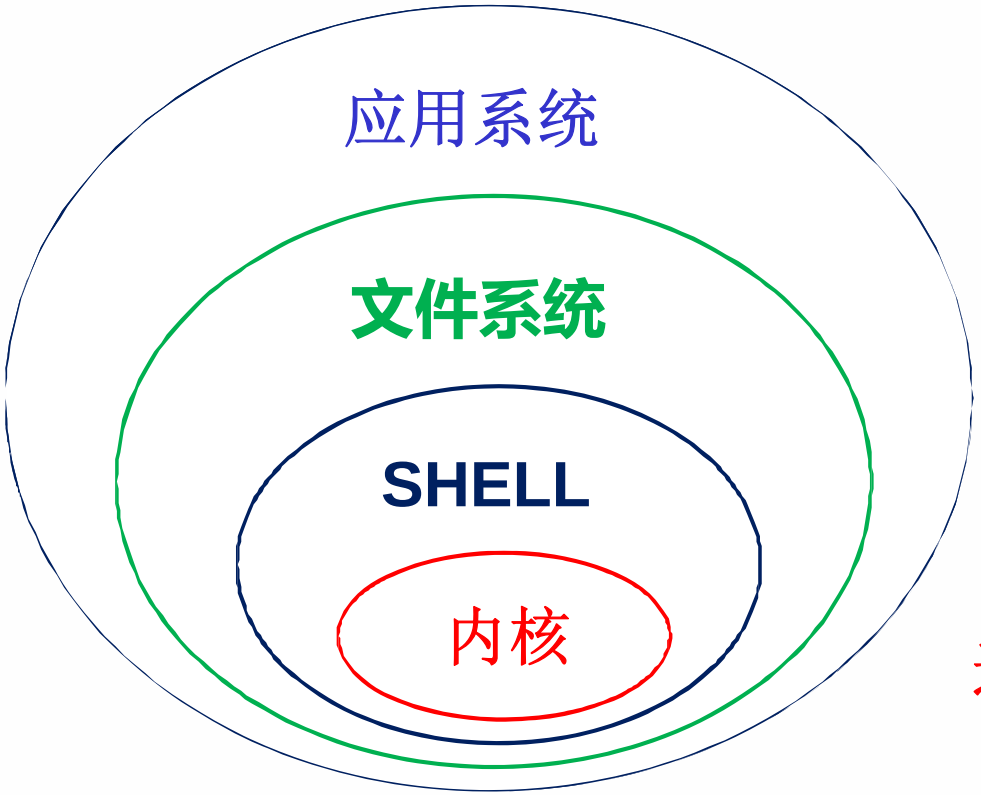
6.1 嵌入式Linux软件设计概述

6.2 Linux简介

6.3 Linux终端命令

6.4 Linux编程基础

嵌入式的层次划分



应用程序的程序集，包括文本编辑器、编程语 言、**X Window**、办公套件、**Internet**工具等

文件存放在磁盘等存储设备上的组织方法。支持ext4、 NFS、ISO9660**等**

用户与内核交互的一种接口，可输入命令等

运行程序和管理硬件设备的核心

1、程序编辑

vi debug.c

```
1 #include <stdio.h>
2 int func(int n)
3 {
4         int sum = 0,i;
5         for(i=0;i<n;i++)
6         {
7                 sum += i;
8         }
9         return sum ;
10 }
11
12 main()
13 {
14         int i;
15         long result=0;
16         for(i=1;i<=100;i++)
17         {
18                 result+=i;
19         }
20         printf("result[1-100]=%d \n",result);
21         printf("result[1-250]=%d \n",func(250));
22 }
```

2、程序编译

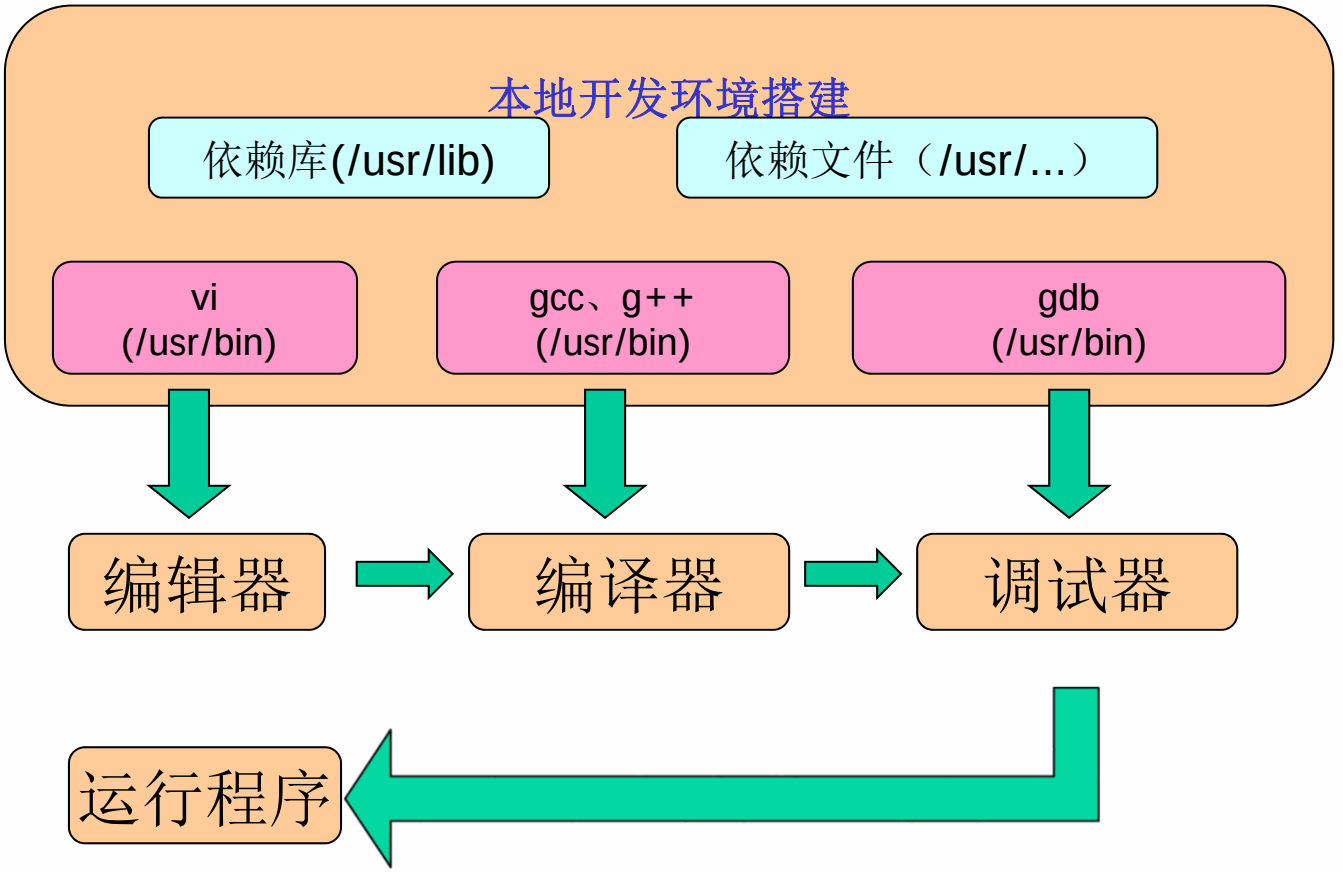
gcc debug.c -o debug -g

3、程序运行

./debug

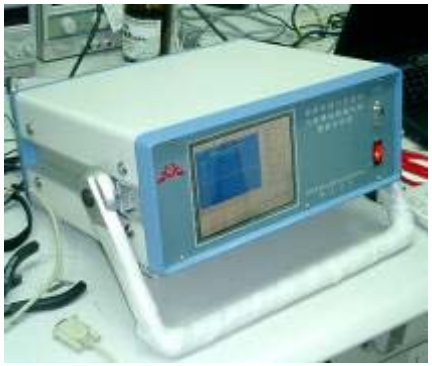
4、程序调试

gdb debug


本地开发环境搭建
依赖库(/usr/lib)
依赖文件（/usr/...）
vi
(/usr/bin)
gcc、g++
(/usr/bin)
gdb
(/usr/bin)
编辑器
编译器
调试器
运行程序

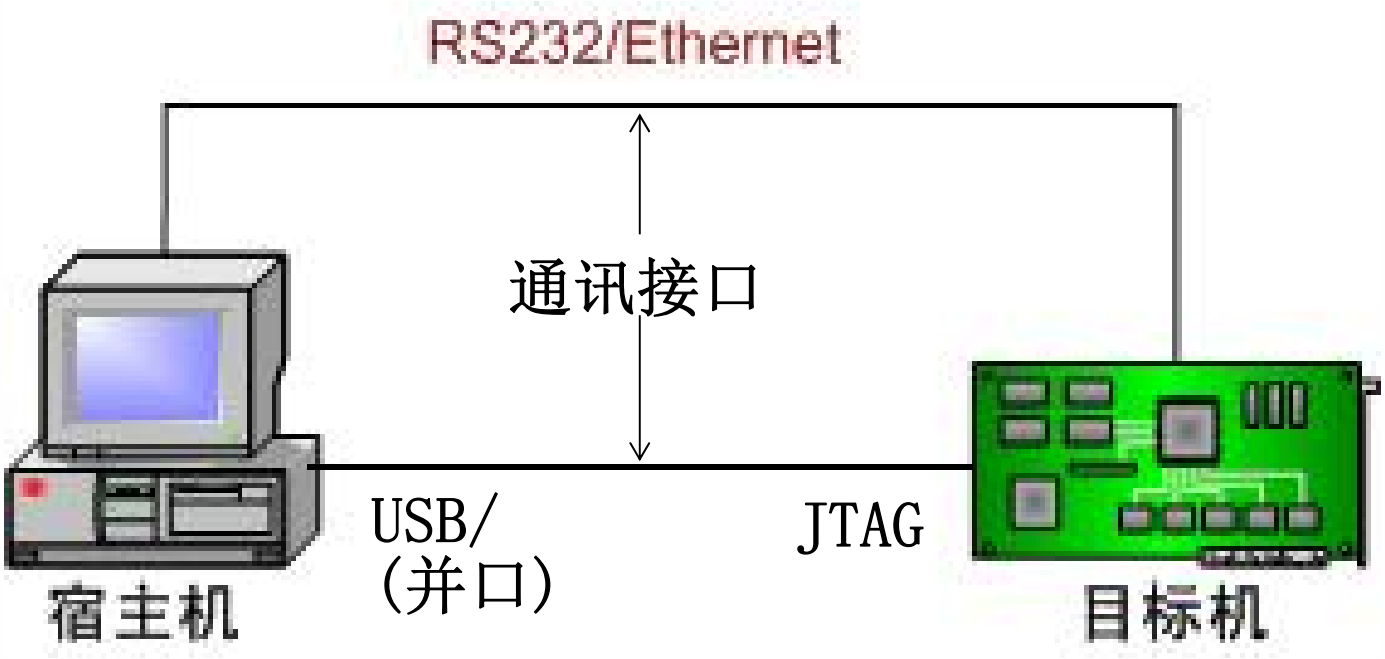
嵌入式系统不具备自举开发能力

- 由于计算、存储、显示等资源受限，嵌入式系统无法完成自举开发。



宿主机－目标机开发模式

- 嵌入式系统采用双机开发模式：宿主机－目标机开发模式，利用资源丰富的PC机来开发嵌入式软件。



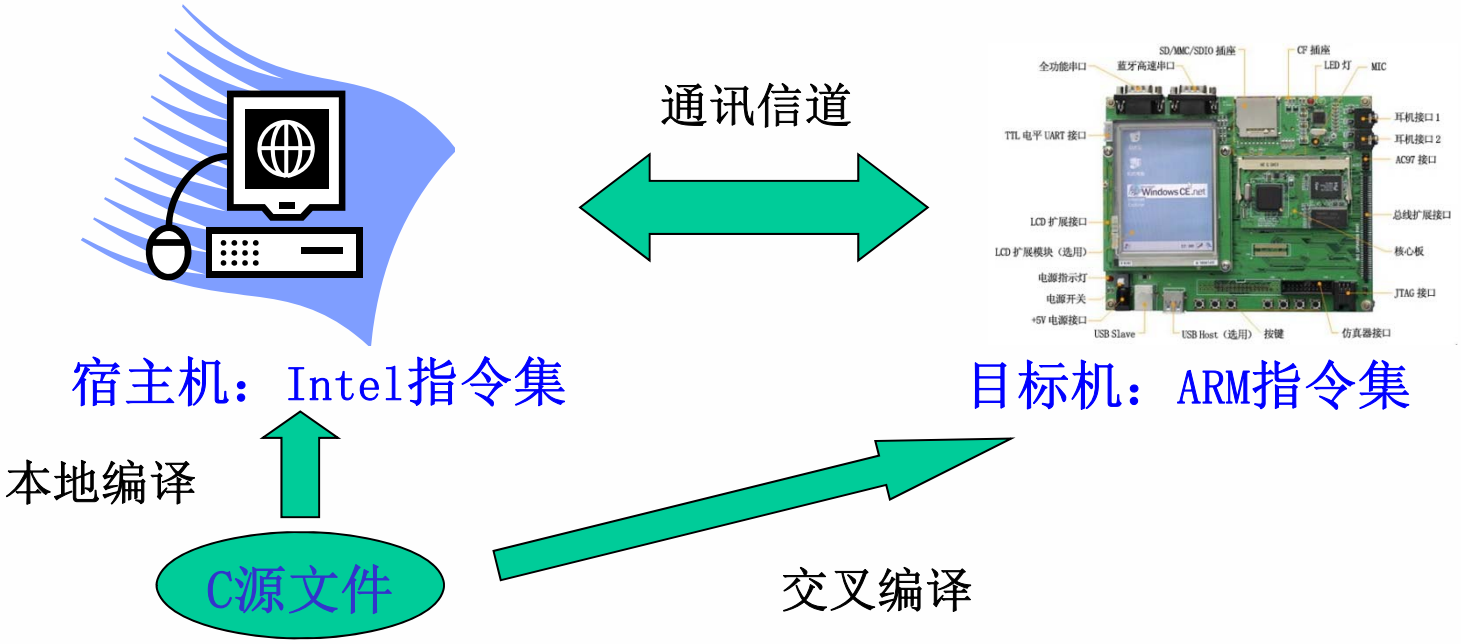
宿主机：资源丰富　　目标机：资源受限

什么是交叉编译

- 在一种平台上编译出能在另一种平台（体系结构不同）上运行的程序；
- 在**PC**平台**(X86)**上编译出能运行在**ARM**平台上的程序，即编译得到的程序在**X86**平台上不能运行，必须放到**ARM**平台上才能运行；
- 用来编译这种程序的编译器就叫交叉编译器；
- 为了不与本地编译器混淆，交叉编译器的名字一般都有前缀，例如：**arm-linux-gcc**。

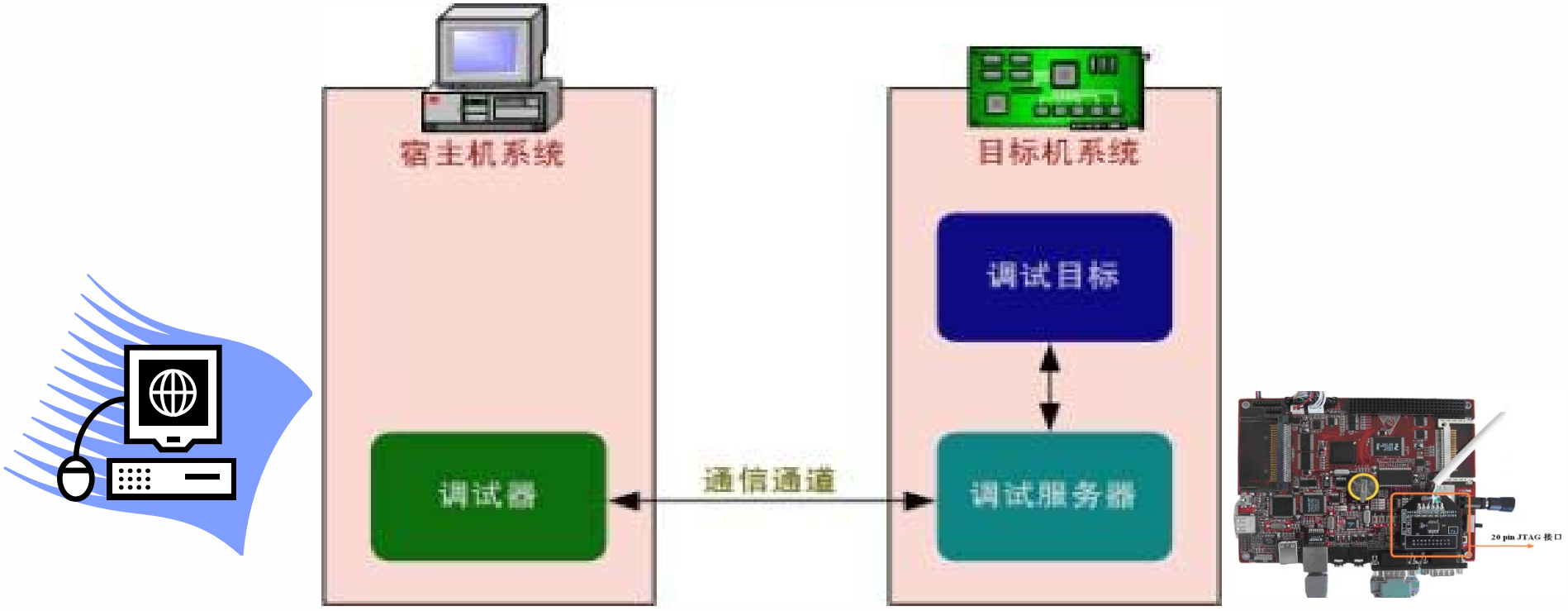
交叉编译与本地编译

- 交叉编译器和交叉链接器是指能够在宿主机上安装，但是能够生成在目标机上直接运行的二进制代码的编译器和链接器



- 基于ARM体系结构的gcc交叉开发环境中，arm-linux-gcc是交叉编译器，arm-linux-ld是交叉链接器

- 一般而言，嵌入式软件需要交叉调试。

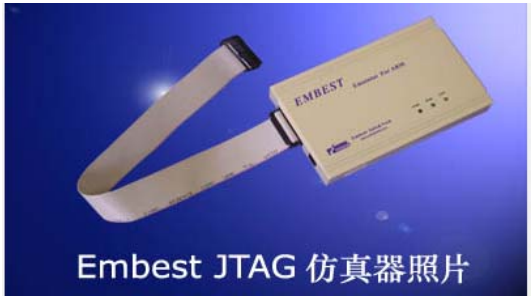


宿主机端的仿真

- 嵌入式应用开发中会出现宿主机操作系统（如**Windows**）与交叉开发环境中要求的宿主机操作系统（如**Linux**）不一致，因此，需要利用虚拟化、仿真化手段建立开发环境，包括：
  - **API**仿真器：**Cygwin**、**MinGW**
  - 虚拟机：**Virtual PC**、**VMWare**、 **Virtualbox**

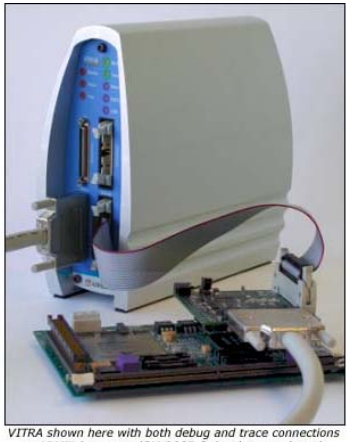
硬件仿真开发——ICE

- **ICE（In-Circuit Emulator）**是一种用于替代目标机上**CPU**的设备，即在线仿真器。
- 它比一般的**CPU**有更多的引出线，能够将内部的信号输出到被控制的目标机。
- **ICE**上的**Memory**也可以被映射到用户的程序空间，即使目标机不存在，也可以进行代码的调试。
- **ICE**可支持软断点和硬件断点的设置、设置各种复杂的断点和触发器、实时跟踪目标程序的运行等。



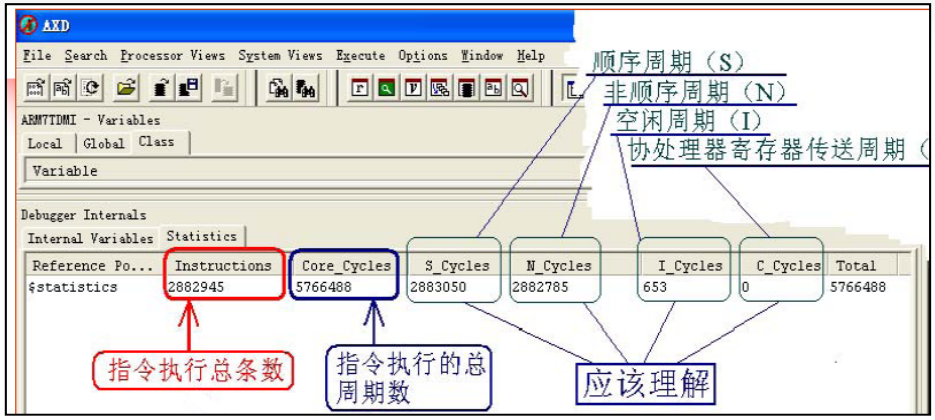
Embest JTAG 仿真器照片

- **OCD（On Chip Debugging）是CPU芯片提供的一种调试功能（片上调试），可以认为是一种廉价的ICE功能。**
- **OCD不占用目标机资源，调试环境和最终目标机运行环境基本一致，支持软硬断点、Trace功能，可提供精确计量程序的执行时间、时序分析等功能。**

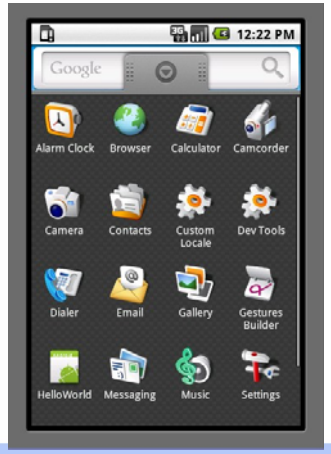


VITRA shown here with both debug and trace connections to ARM™ Integrator/CM-966E-S development board.

软件仿真器举例

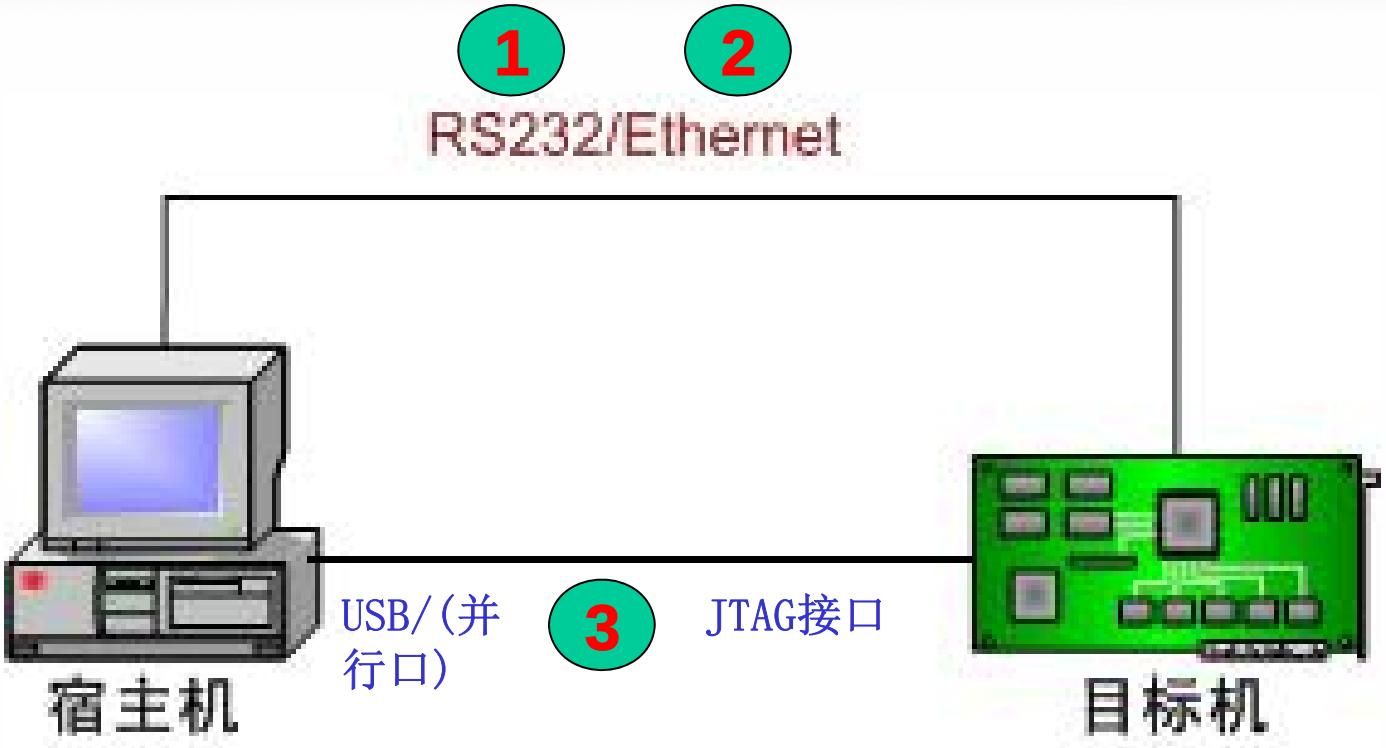


ARM仿真器Armulator



Android仿真器

常见通信模式



通讯模式1－串口通讯

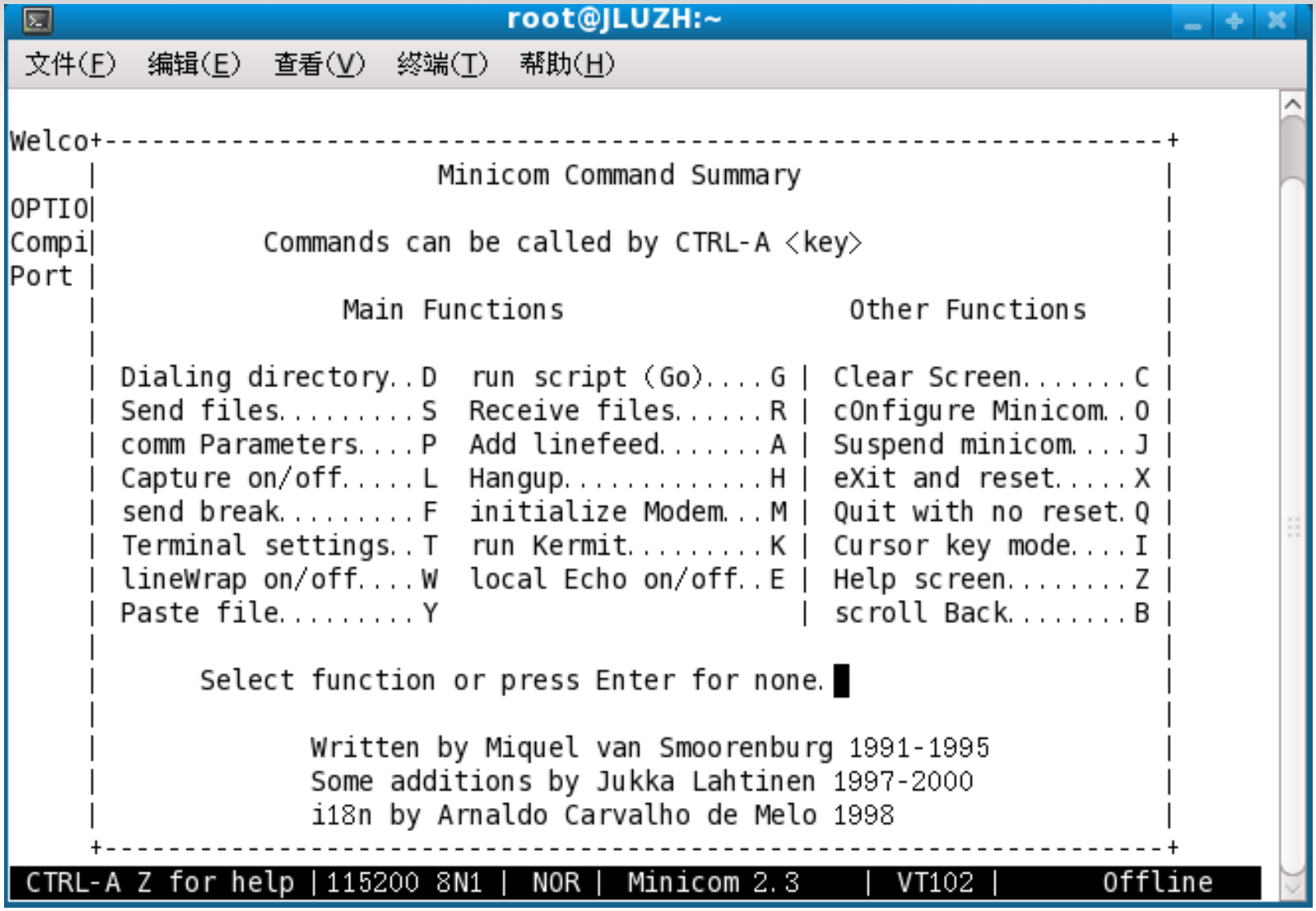
- 特点及应用场合
  - 驱动实现最简单
  - 传输速度慢，距离短，不适合大数据量、长距离数据传输
  - 需要在宿主机、目标机两端均提供驱动
  - 常用于宿主机－目标机的字符流通讯

Minicom对串口数据传输的配置

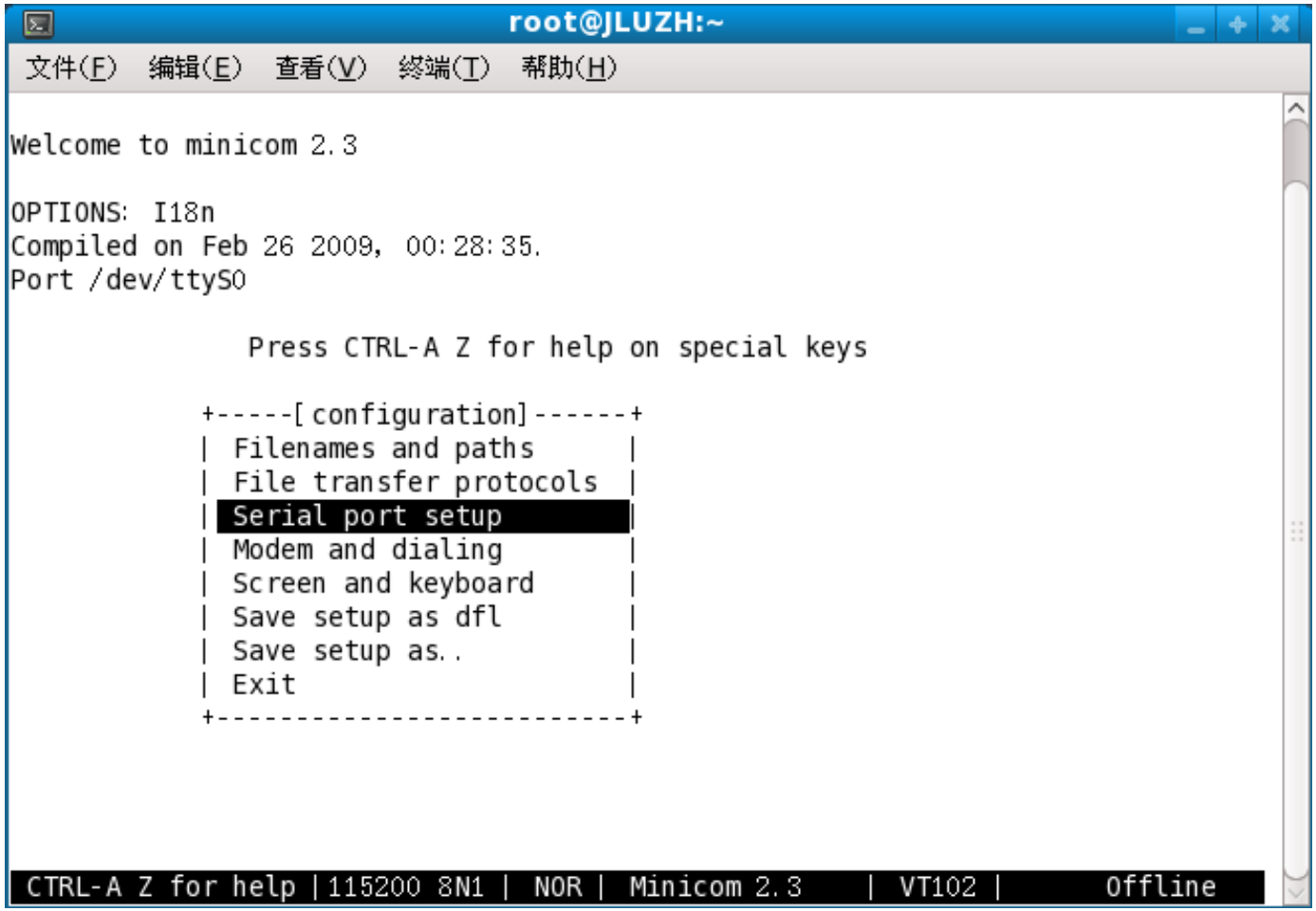
```
[root@XSBase home]# minicom -s
```

- 若目标机接在COM1上，则输入/dev/ttyS0;若接在COM2上则输入/dev/ttyS1。
- Speed为115200
- Parity bit为No
- Data bit为8
- Stop bits为1

Linux宿主机串口通讯简介－minicom



设置正确后，目标板启动显示信息如下：

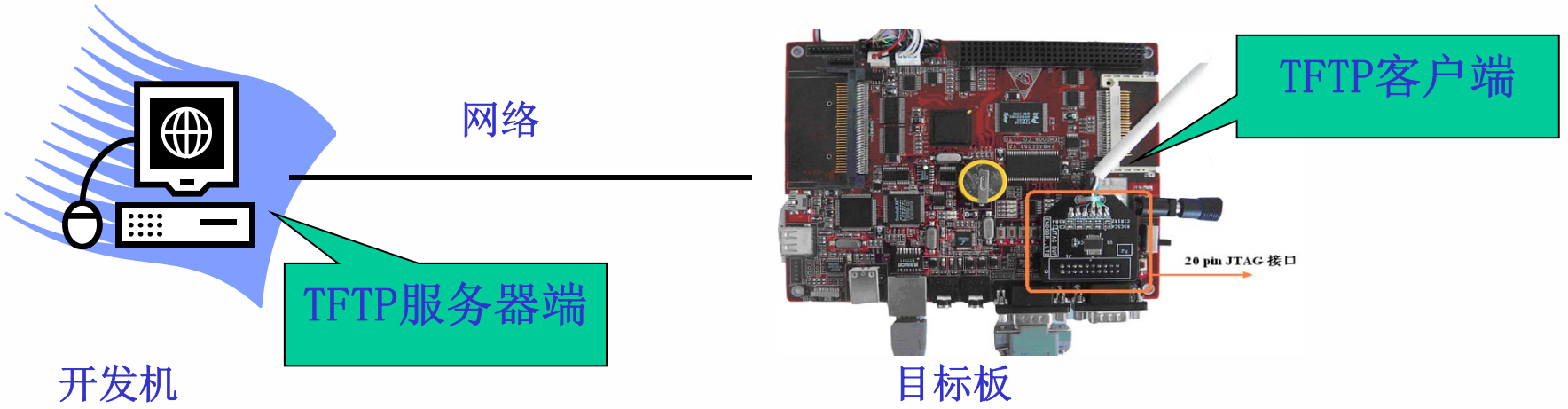


通讯模式2－网络通讯

- 特点及应用场合
  - 驱动实现相对复杂，一般采用精简的网络通讯协议，如TFTP进行通讯
  - 常用于宿主机－目标机的大数据量数据传输，可以作为串口通讯的补充
  - 需要在宿主机、目标机两端均提供驱动
  - 宿主机端实现服务器，目标机端提供客户端

宿主机端网络通讯简介－TFTP协议

- TFTP服务的全称是简单文件传输协议（Trivial File Transfer Protocol）
- TFTP可以看成一个简化了的FTP
- TFTP服务器端安装在宿主机，TFTP客户端由目标板实现，目标板需要获取IP地址



本章主要讲述

6.1 嵌入式Linux软件设计概述

6.2 Linux简介

6.3 Linux终端命令

6.4 Linux编程基础

操作系统

Linux

VxWorks®

WIND RIVER

µC/OS-II The Real-Time Kernel

完全开源，软件资源丰富，高性能，稳定

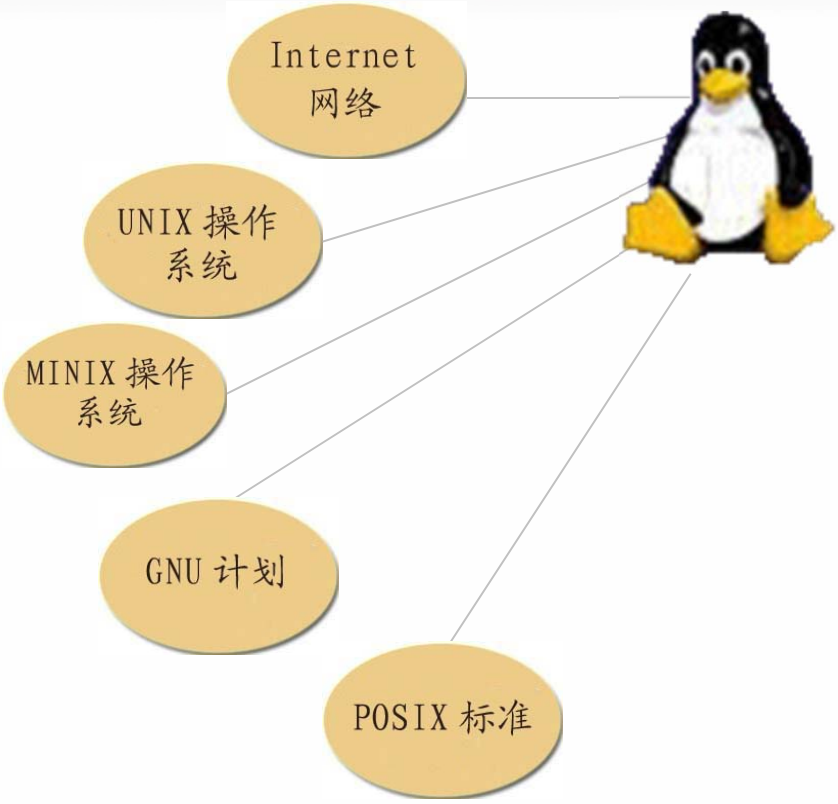
实时性强，价格高，开发维护成本高

实时，对教育开源，仅包含基本功能， 无网络功能，但扩展性好，适合中小型应用

ANDROID

Microsoft Windows CE

Linux诞生

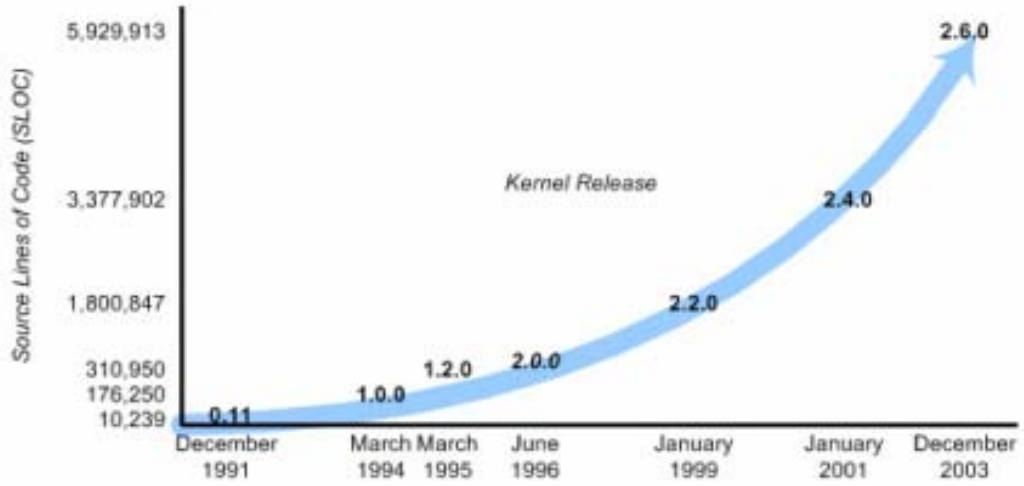


- UNIX 操作系统是美国贝尔实验室于1969年夏在DEC PDP-7 小型计算机上开发的一个分时操作系统
- MINIX 系统是由 Andrew S. Tanenbaum（AST）1987 年开发的，主要用于学生学习操作系统原理
- 如果没有 Internet 网，没有遍布全世界的无数计算机黑客通过网络的无私奉献，那么 Linux 绝对不可能发展到现在的水平。

Linux版本历史

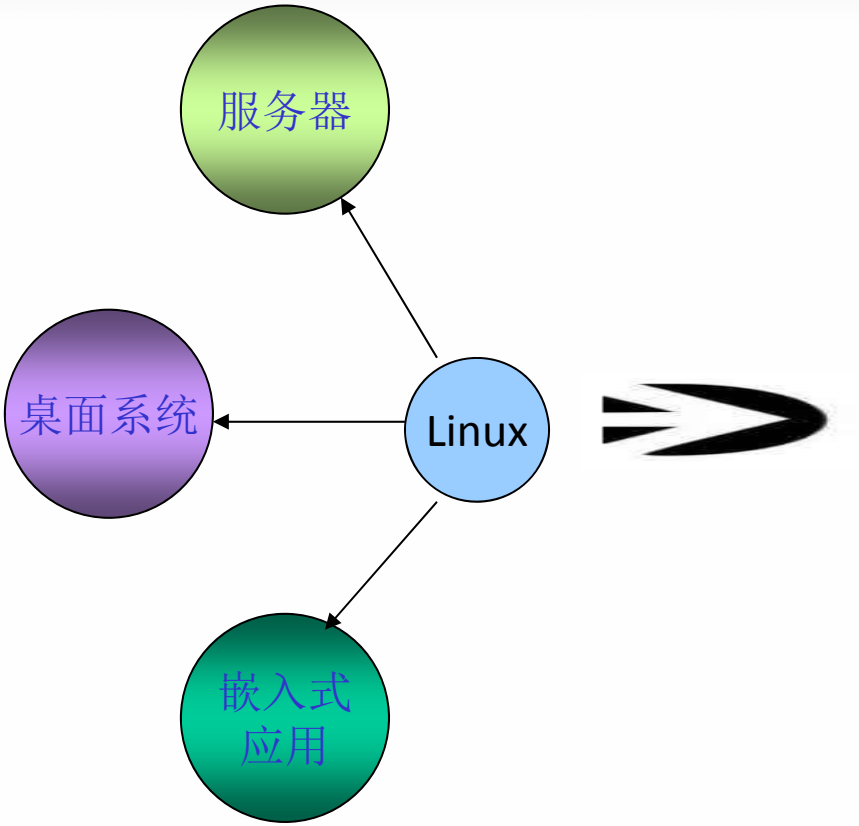
- 从 Linux 诞生开始，Linux 内核就从来没有停止过升级，从0.02 版本到 1999年具有里程碑意义的2.2 版本，一直到我们现在看到的x.x.xx版本。

从未停止过升级

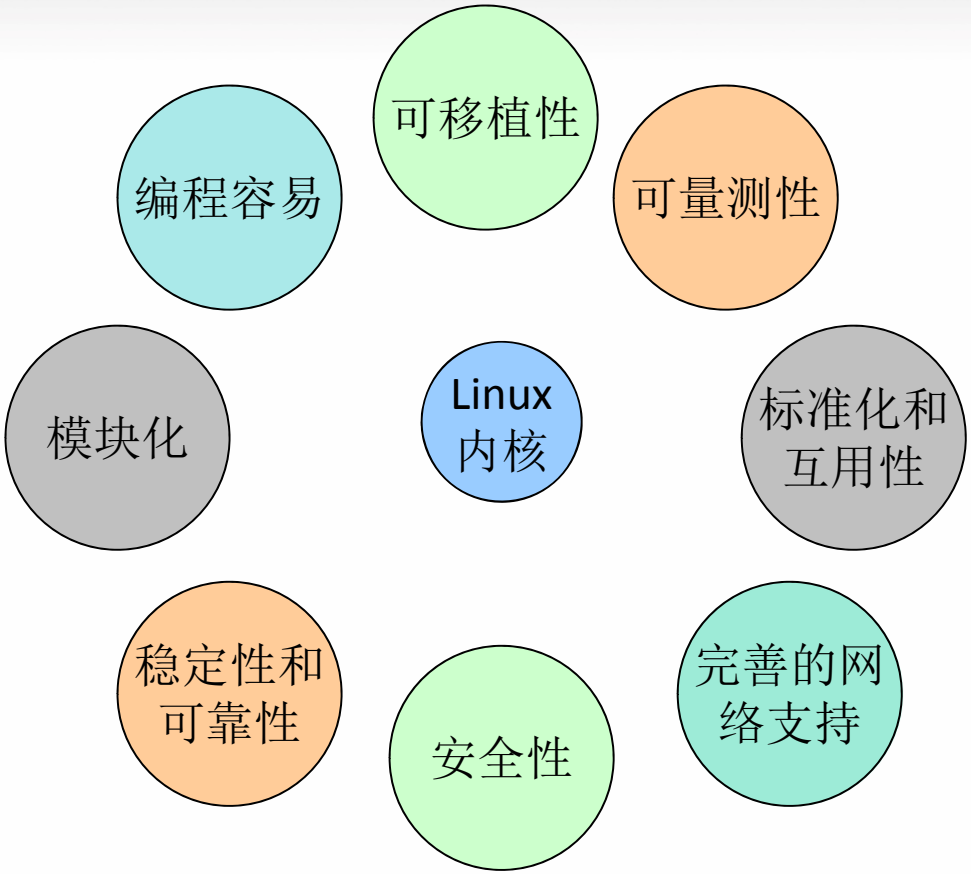


- Linux 内核版本有两种：
  - - 稳定版和开发版
- Linux内核的命名机制:num.num.num。
  - - 第一个数字是主版本号
  - - 第二个数字是次版本号
  - - 第三个数字是修订版本号

Linux应用领域



Linux内核特点



- **掌握Linux基础知识**
  - ✓ 常用命令
  - ✓ 文件系统
  - ✓ Linux编程
- **通过实践巩固基础知识**
  - ✓ **以需求为导向**
  - ✓ **应用程序为主**
  - ✓ **熟悉内核和引导系统**
  - ✓ **强化进程、网络编程**
  - ✓ **熟悉Shell编程**

一个典型的Linux发行版包括：

- ✓ **Linux**内核
- ✓ 一些**GNU**程序库和工具
- ✓ 命令行**shell**
- ✓ 图形界面和桌面环境，如**KDE/GNOME**
- ✓ 数千种从办公套件、编译器、文本编辑器到科学工具的应用软件。

- Debian
- 红帽(Redhat)
- Ubuantu
- Suse
- Fedora

- **Linux的发展离不开GNU（GNU is Not Unix）,GNU 计划又称革奴计划，是由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计划开发出许多高质量的免费软件，如GCC、GDB、Bash Shell等，这些软件为Linux的开发创造了基本的环境，是Linux发展的重要基础。因此，严格来说， Linux应该称为GNU/ Linux。**

Linux的安装有三种方式：

- **纯Linux**
- **双操作系统**
- **基于虚拟机的安装**

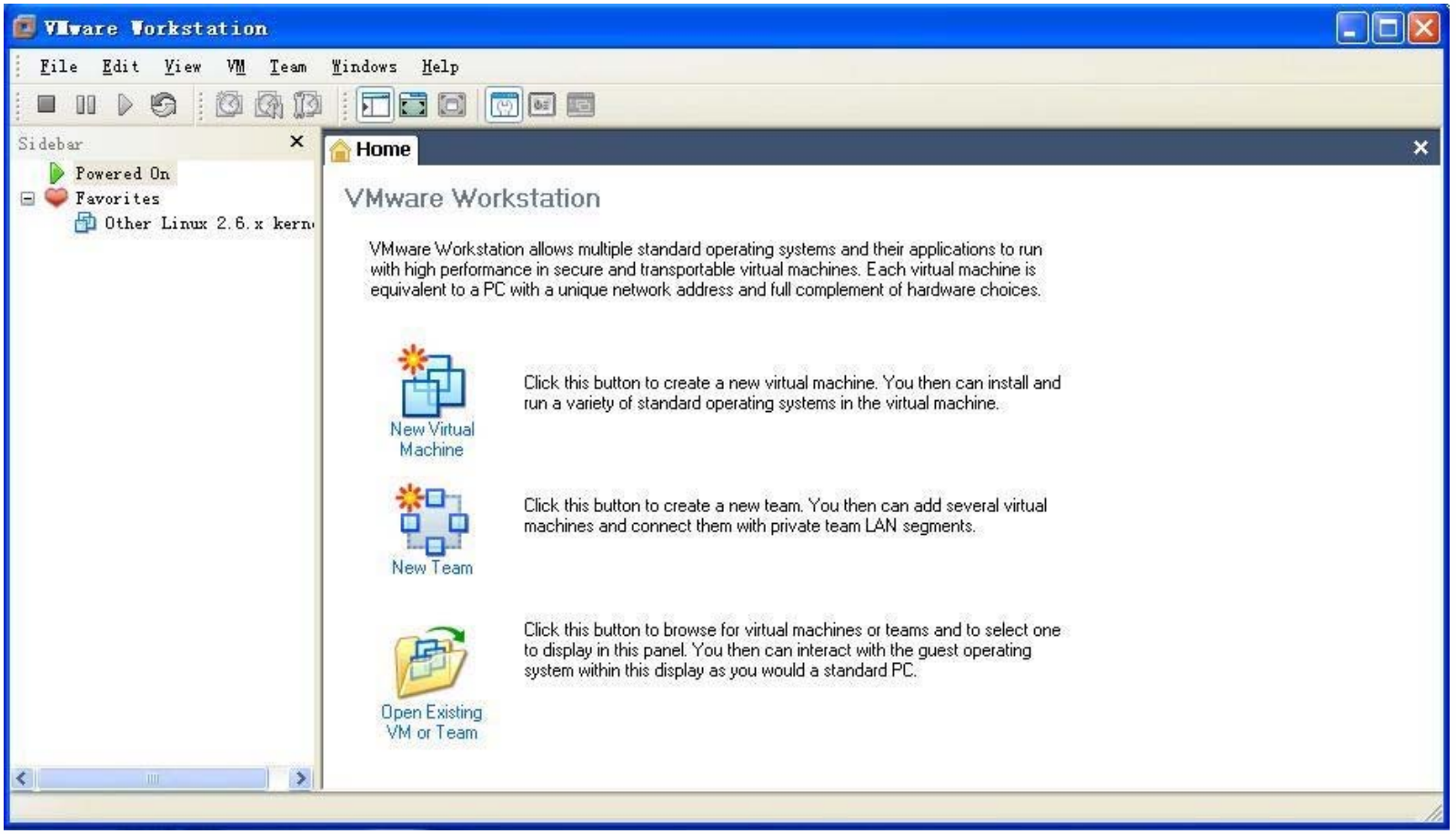
✓ 模拟出硬件设备，然后在该硬件设备基础上安装系统

✓ 安装Vmware/Virtual Box等虚拟机软件

✓ 建立虚拟机

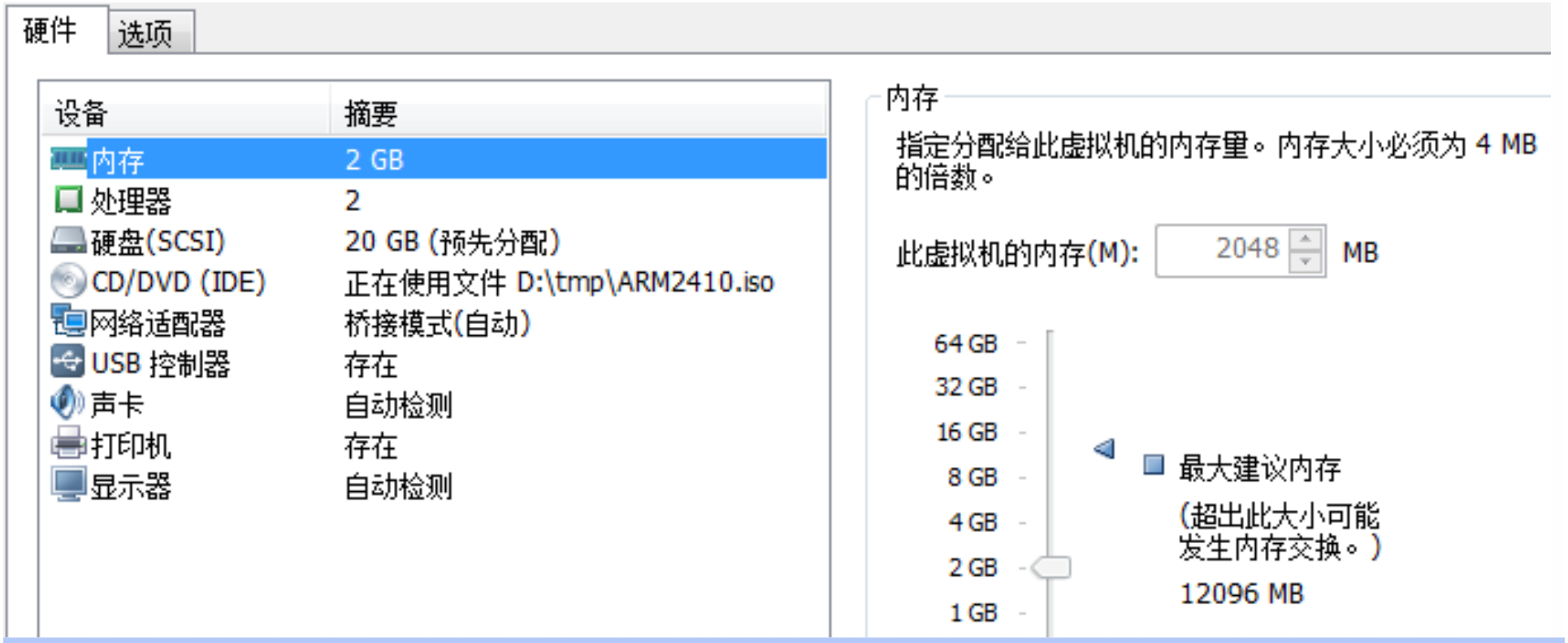
✓ 在虚拟机下安装Linux

建立虚拟机


VMware Workstation
File Edit View VM Team Windows Help
Sidebar
Powered On
Favorites
Other Linux 2.6.x kern
Home
VMware Workstation
VMware Workstation allows multiple standard operating systems and their applications to run with high performance in secure and transportable virtual machines. Each virtual machine is equivalent to a PC with a unique network address and full complement of hardware choices.
New Virtual Machine
Click this button to create a new virtual machine. You then can install and run a variety of standard operating systems in the virtual machine.
New Team
Click this button to create a new team. You then can add several virtual machines and connect them with private team LAN segments.
Open Existing VM or Team
Click this button to browse for virtual machines or teams and to select one to display in this panel. You then can interact with the guest operating system within this display as you would a standard PC.

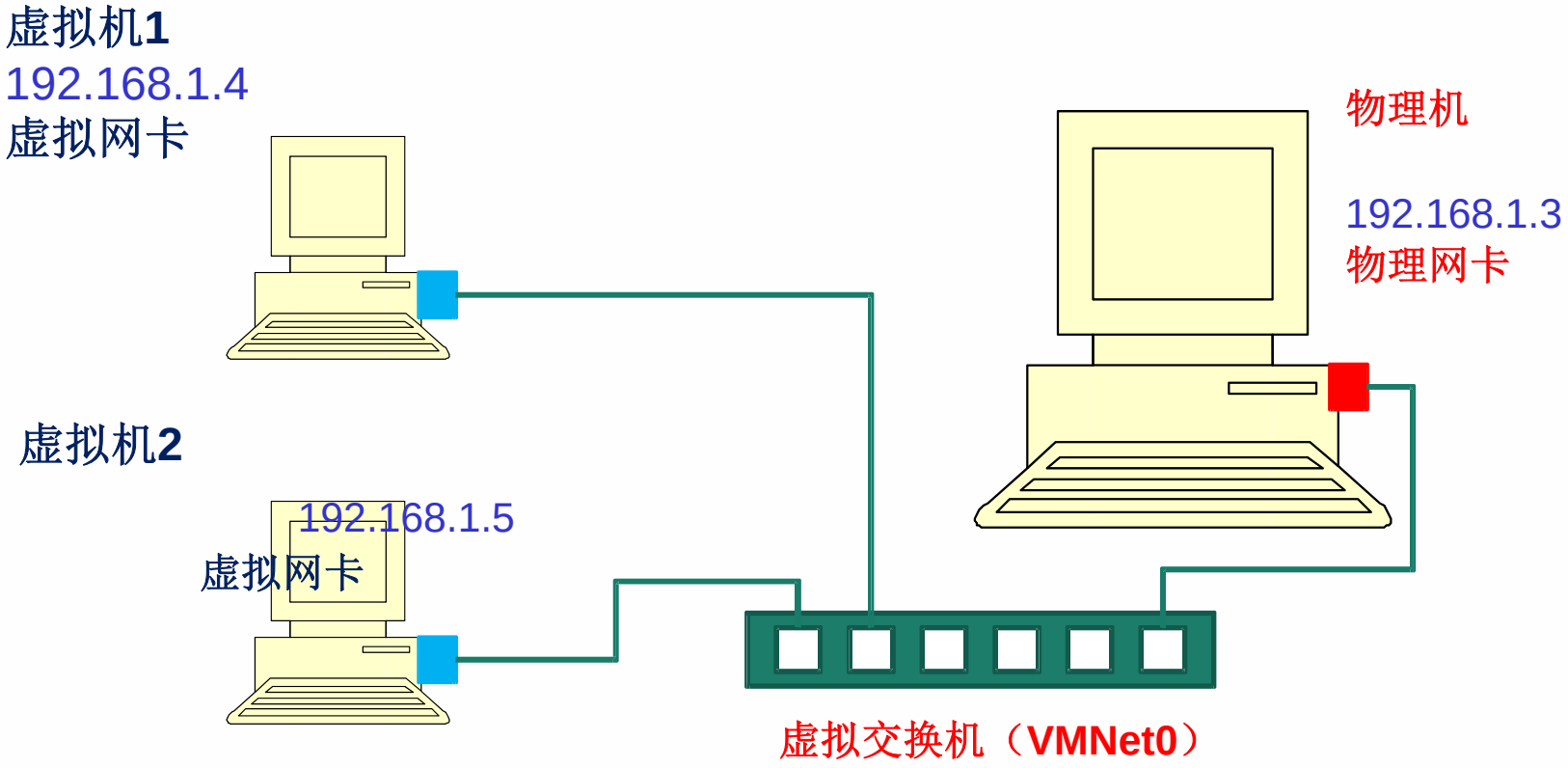
硬件设置

- 模拟出的硬件包括内存、硬盘、光驱、 网卡、声卡、串口、USB口等
- VMWare模拟出来的各种硬件与主机没 有关系
- VMWare可以直接用ISO文件作为虚拟光驱使用

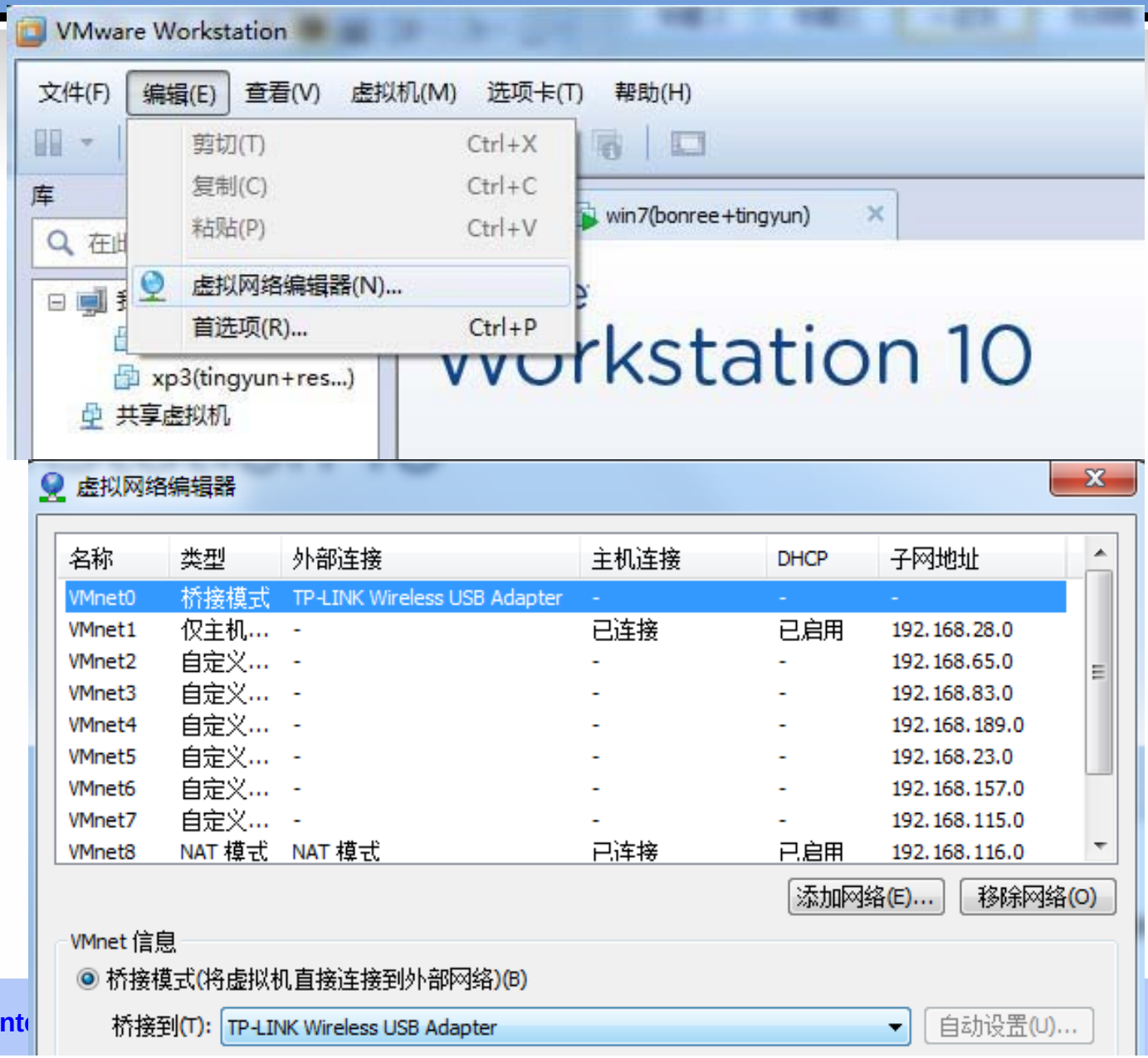


网络设置

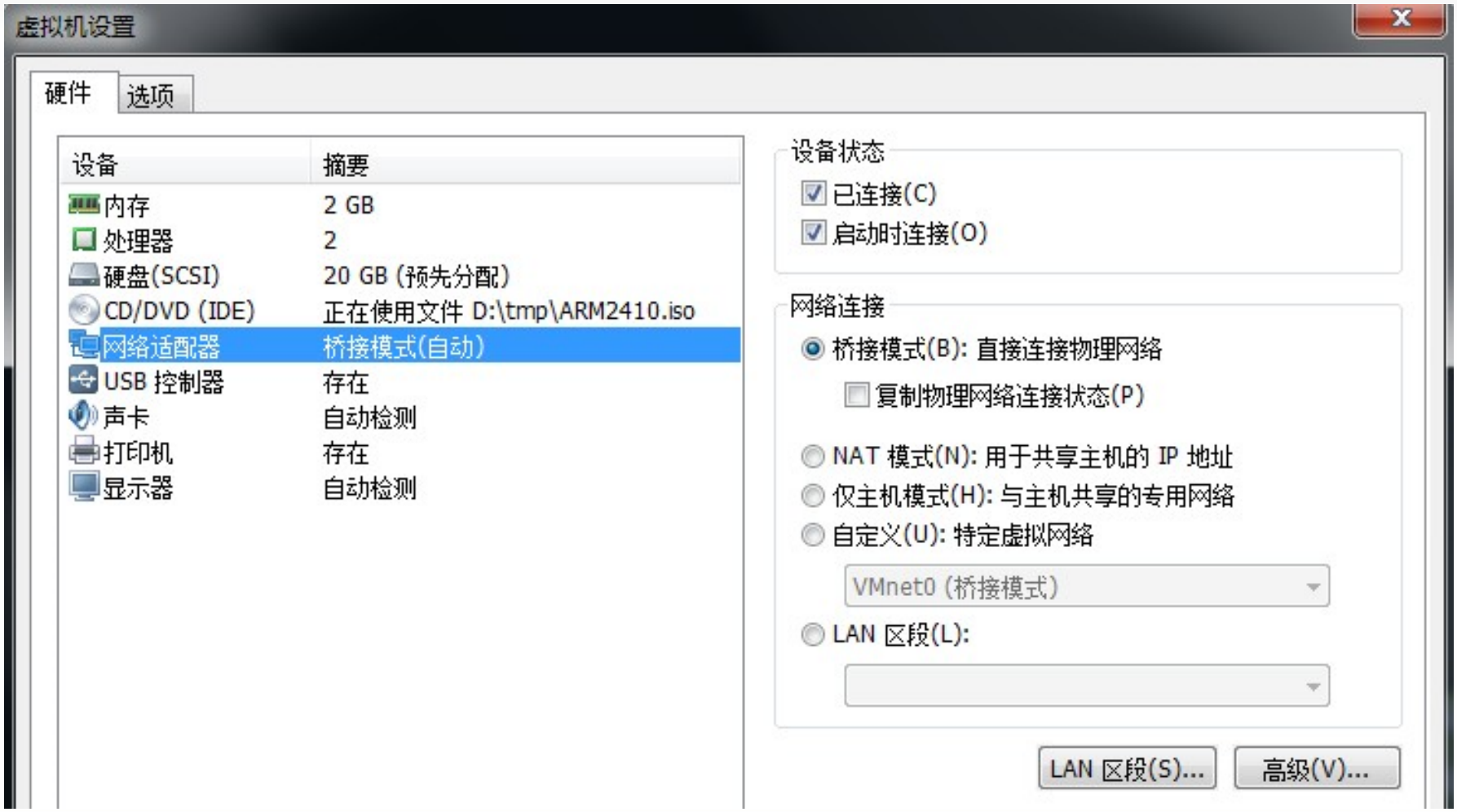
- Bridged方式
- NAT方式
- Host Only方式



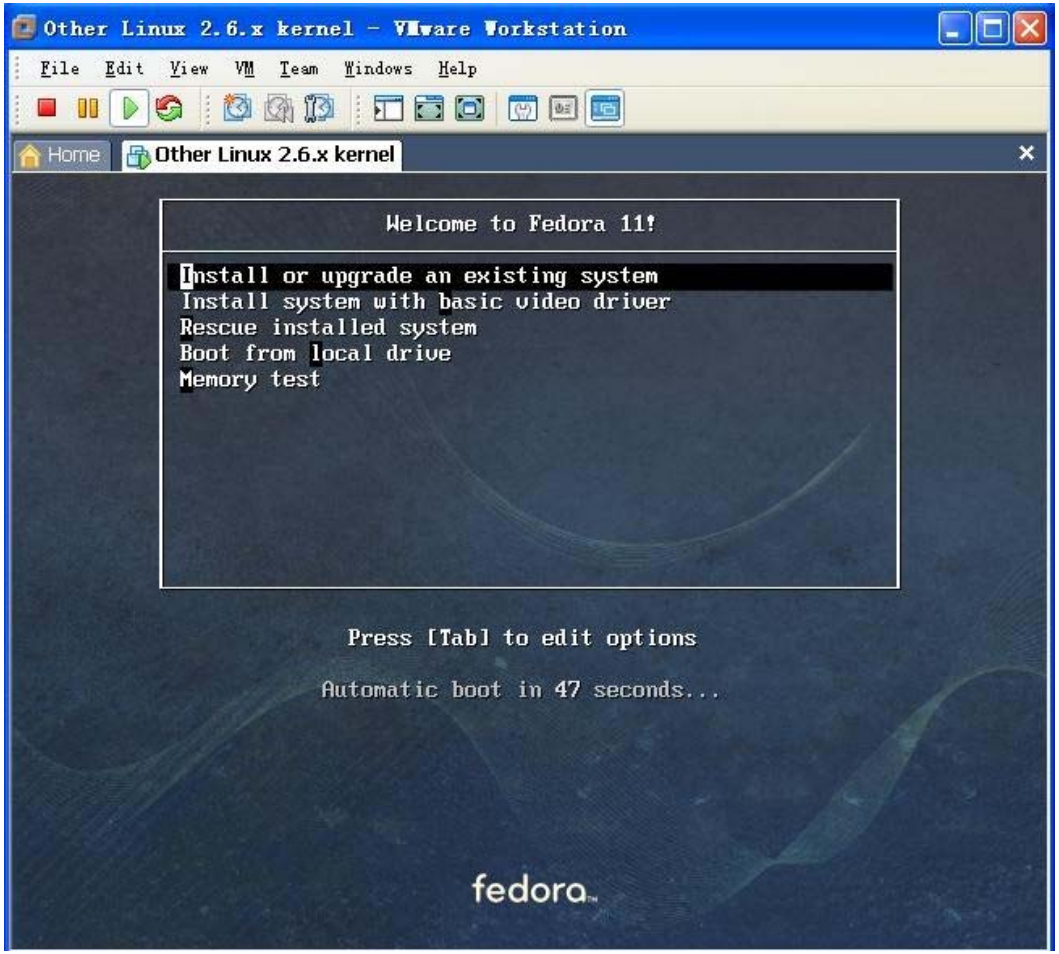
网络设置


VMware Workstation
文件(F)
编辑(E)
查看(V)
虚拟机(M)
选项卡(T)
帮助(H)
剪切(T) Ctrl+X
复制(C) Ctrl+C
粘贴(P) Ctrl+V
虚拟网络编辑器(N)...
首选项(R)... Ctrl+P
库
win7(bonree+tingyun)
xp3(tingyun+res...)
共享虚拟机
Workstation 10
虚拟网络编辑器
名称 类型 外部连接 主机连接 DHCP 子网地址
VMnet0 桥接模式 TP-LINK Wireless USB Adapter - - -
VMnet1 仅主机... - 已连接 已启用 192.168.28.0
VMnet2 自定义... - - - 192.168.65.0
VMnet3 自定义... - - - 192.168.83.0
VMnet4 自定义... - - - 192.168.189.0
VMnet5 自定义... - - - 192.168.23.0
VMnet6 自定义... - - - 192.168.157.0
VMnet7 自定义... - - - 192.168.115.0
VMnet8 NAT 模式 NAT 模式 已连接 已启用 192.168.116.0
添加网络(E)...
移除网络(O)
VMnet 信息
桥接模式(将虚拟机直接连接到外部网络)(B)
桥接到(T): TP-LINK Wireless USB Adapter
自动设置(U)...

网络设置


虚拟机设置
硬件
选项
设备
摘要
内存
2 GB
处理器
2
硬盘(SCSI)
20 GB (预先分配)
CD/DVD (IDE)
正在使用文件 D:\tmp\ARM2410.iso
网络适配器
桥接模式(自动)
USB 控制器
存在
声卡
自动检测
打印机
存在
显示器
自动检测
设备状态
已连接(C)
启动时连接(O)
网络连接
桥接模式(B): 直接连接物理网络
复制物理网络连接状态(P)
NAT 模式(N): 用于共享主机的 IP 地址
仅主机模式(H): 与主机共享的专用网络
自定义(U): 特定虚拟网络
VMnet0 (桥接模式)
LAN 区段(L):
LAN 区段(S)...
高级(V)...

启动虚拟机并安装Linux



本章主要讲述

6.1 嵌入式Linux软件设计概述

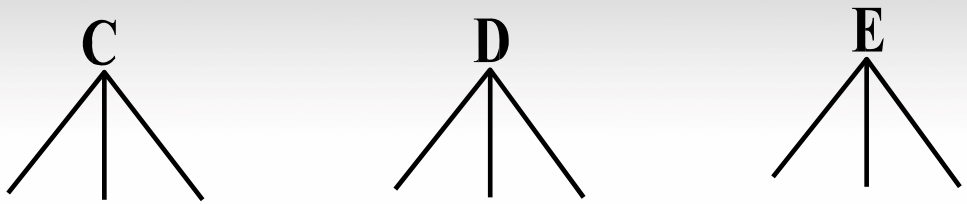
6.2 Linux简介

6.3 Linux终端命令

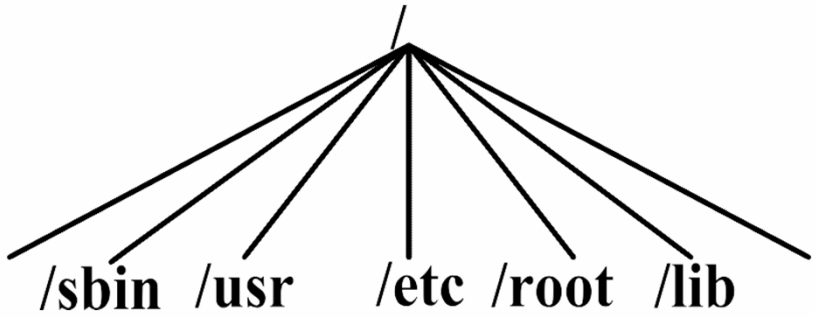
6.4 Linux编程基础

- 操作系统中负责管理和存储文件信息的软件机构称为文件管理系统，简称文件系统
- 文件系统由三部分组成：与文件管理有关的软件、 被管理的文件以及实施文件管理所需的数据结构
- **Linux**支持许多不同的文件系统

✓ FAT16、FAT32； NTFS；
✓ ext2、ext3、ext4
✓ swap；
✓ NFS；
✓ ISO9660



目录树结构，以每个分区为树根， 有几个分区就有几个树型结构



单一目录树结构，最上层是根目录，所有目录都是从根目录出发而生成的

本质不同：Windows文件系统只负责文件存 储，Linux文件系统管理所有软硬件资源

- **Linux系统中有三种基本的文件类型**
  - 普通文件：又分为文本文件和二进制文件；
  - 目录文件：目录文件存储了一组相关文件的位置、大小等与文件有关的信息；
  - 设备文件：**Linux系统把每一个I/O设备都看成一个文件，与普通文件一样处理，这样可以使文件与设备的操作尽可能统一** 。

- **Linux系统以目录的方式来组织和管理系统中的所有文件**

✓ **Linux系统通过目录将系统中所有的文件分级、分层组织在一起，形成了Linux文件系统的树型层次结构。以根目录“/”为起点，所有其他的目录都由根目录派生而来。**

✓ **特殊目录:“.”代表该目录自己，“..”代表该目录的父目录，对于根目录，“.”和“..”都代表其自己。**

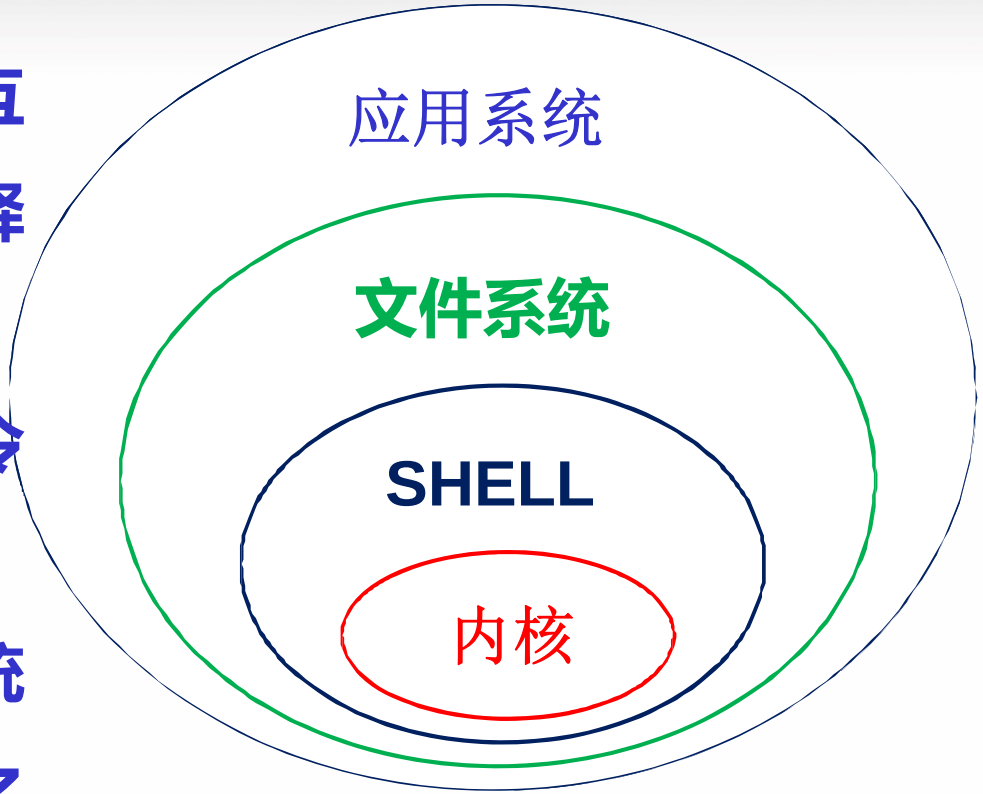
- 工作目录：用户登录到Linux系统后，每时每刻都处在某个目录之中，此目录被称为“工作目录” 或“当前目录”

- 用户主目录（Home Directory）：是系统管理员在增加用户时为该用户建立起来的目录，每个用户都有自己的主目录。 使用符号~表示。

- 路径是指从树型目录结构中的某个目录到某个文件的一条道路。此路径的主要构成是目录名称，中间用“/”分开。
  - 绝对路径是指从“根”开始的路径，也称为完全路径；
  - 相对路径是指从用户工作目录开始的路径。
- 通配符
  - 通配符 *
  - 通配符 ？
  - 字符组模式：通配符“[”、“]”、“-”用于构成字符组模式。（[abc],[a-e]。）
  - 转义字符 \。（*。）

- 设备是指计算机中的外围硬件装置，即除了CPU和内存以外的所有设备。
- 在Linux环境下，文件和设备都遵从按名访问的原则，因此用户可以用使用文件的方法来使用设备。
- 设备名以文件系统中的设备文件的形式存在。
- 所有的设备文件存放在/dev目录下。
- 常用设备名
  - /dev/hd*
  - /dev/sd*
  - /dev/lp*

- Shell是系统的用户界面，提供了用户与内核进行交互操作的一种接口(命令解释器)。
- Shell接收用户输入的命令并把它送入内核去执行。
- Shell起着协调用户与系统的一致性和在用户与系统之间进行交互的作用。

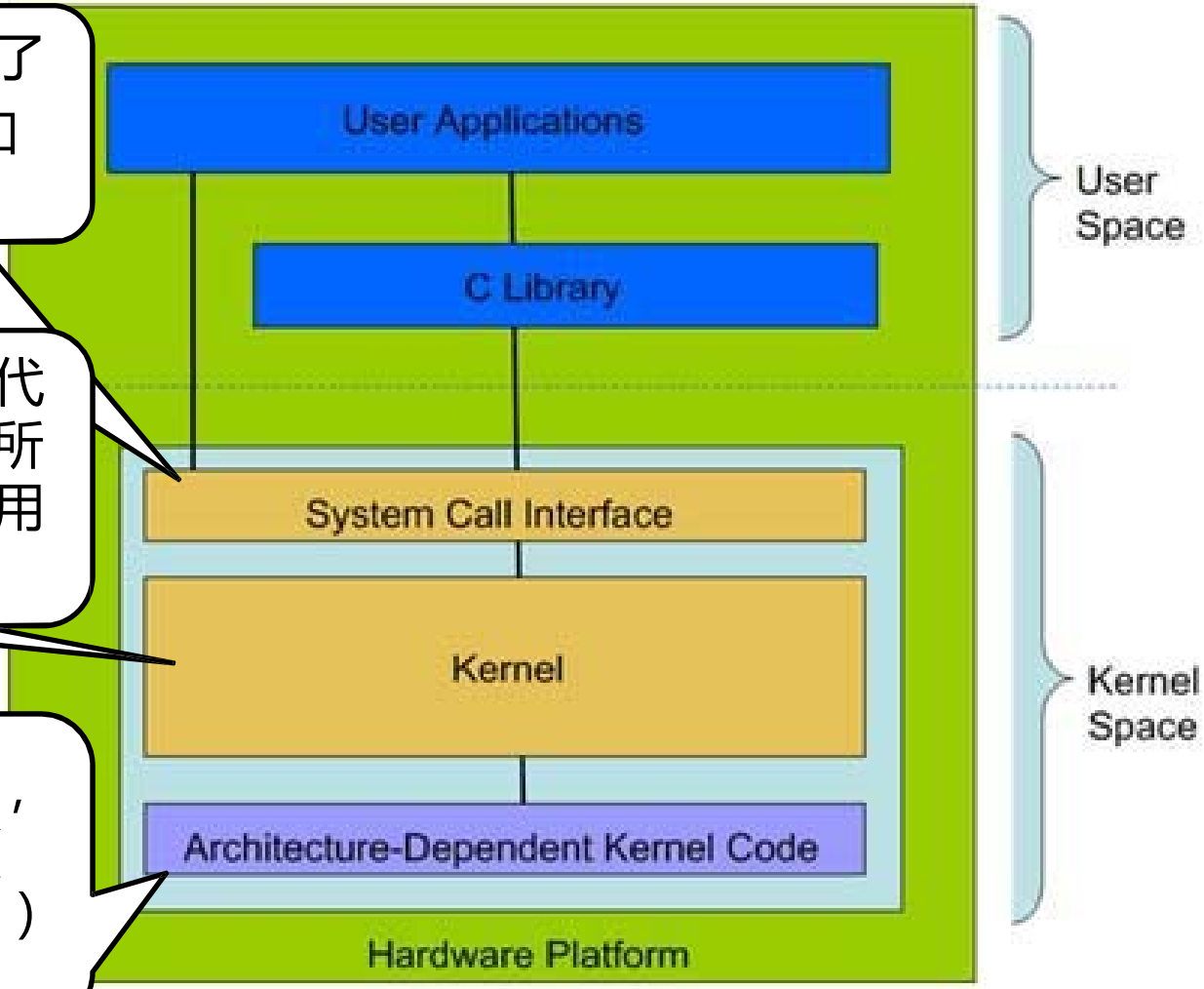


Linux内核结构

系统调用接口，它实现 了一些基本的功能，例 如 read 和 write

独立于体系结构的内 核代码。是 Linux 所 支持的所有处理器体 系结构所通用的

依赖于体系结构的代 码，构成了通常称为 BSP（Board Support Package）的部分



- 五个主要子系统

(1)进程调度

(2)进程间通信

(3)内存管理

(4)虚拟文件系统

(5)网络接口

- 其他部分

(6) 各子系统需要对应的设备驱动程序

(7) 依赖体系结构的代码

进程调度

■进程调度负责控制进程访问CPU

✓保证进程公平地使用CPU

✓ 保证内核能够准时执行一些必要的硬件操作

■所有其他子系统都依赖于进程调度

■Linux采用基于优先级的进程调度方法

✓SCI（系统调用接口） 层提供了某些机制执行 从用户空间到内核的函数调用

✓通过优先级确保重要进程优先执行

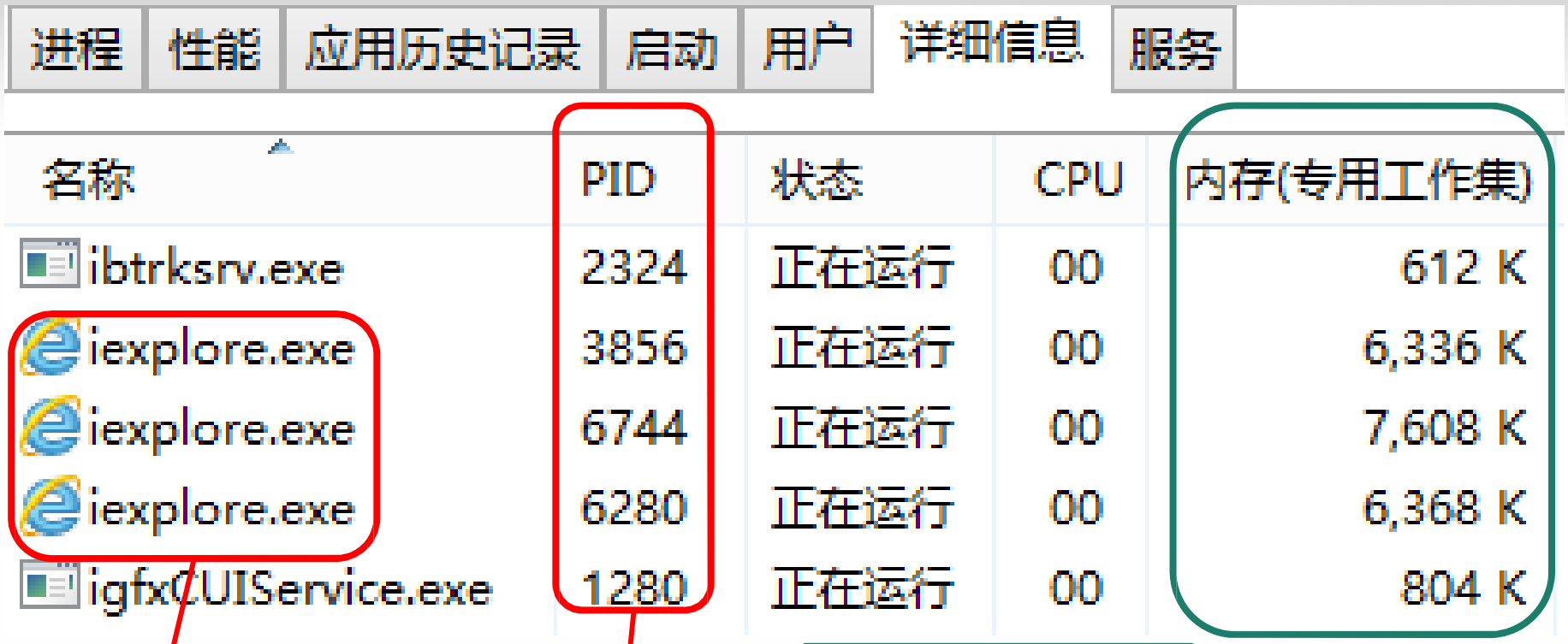
一组指令的有序集合

程序是静态的

具有独立功能的程序的一次运行过程

进程是动态的，有生命周期

一个程序可以有多个进程



程序名不能唯一标识进程

ID可唯一标识进程

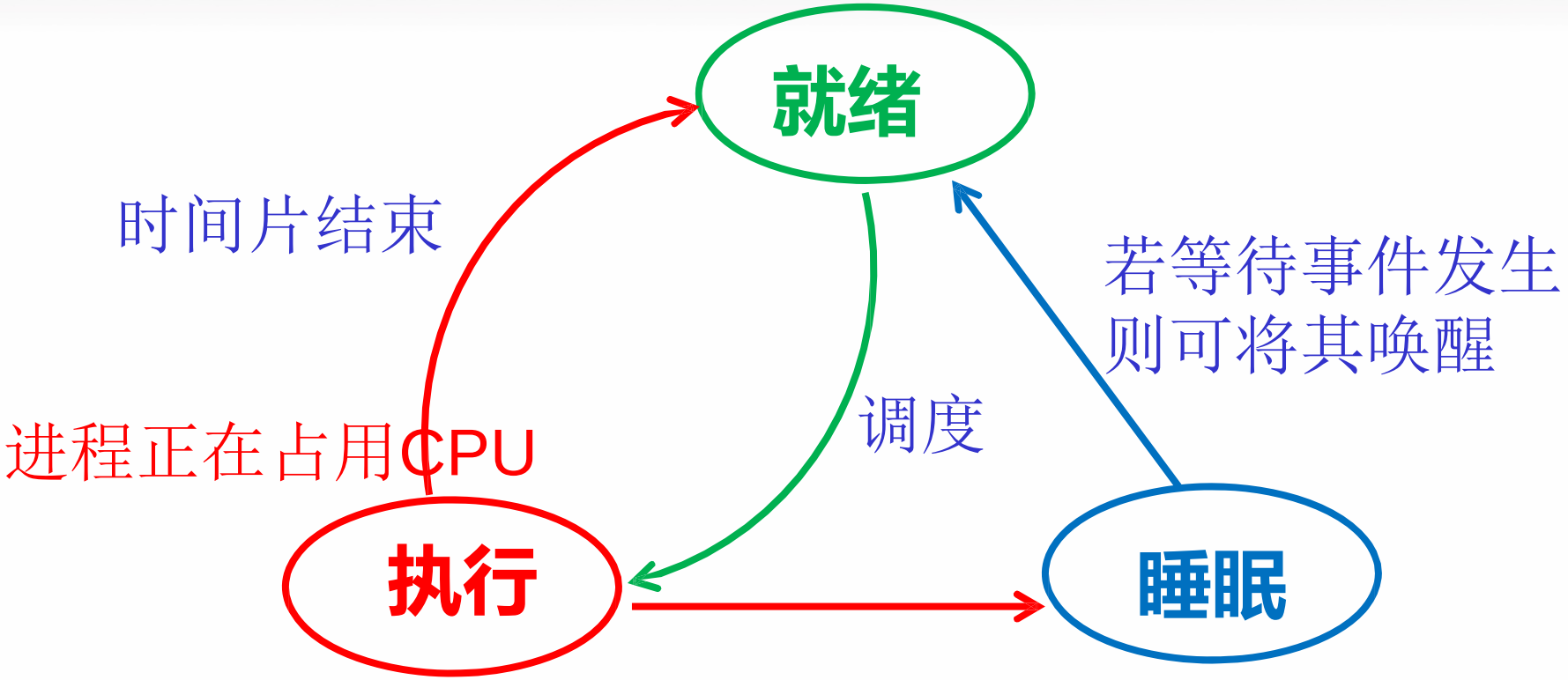
独立性

动态性

并发性

```
[root@localhost ~]# ps a
  PID TTY      STAT   TIME COMMAND
 4320 tty1     Ss+    0:00 /sbin/mingetty tty1
 4321 tty2     Ss+    0:00 /sbin/mingetty tty2
 4322 tty3     Ss+    0:00 /sbin/mingetty tty3
 4324 tty4     Ss+    0:00 /sbin/mingetty tty4
 4326 tty5     Ss+    0:00 /sbin/mingetty tty5
 4327 tty6     Ss+    0:00 /sbin/mingetty tty6
 4429 tty7     Ss+    0:05 /usr/bin/Xorg :0 -br
 4703 pts/1    Ss     0:00 bash
 5021 pts/1    R+     0:00 ps a
```

进程已经具备执行的条件，等待CPU的处理时间片



进程调度概念

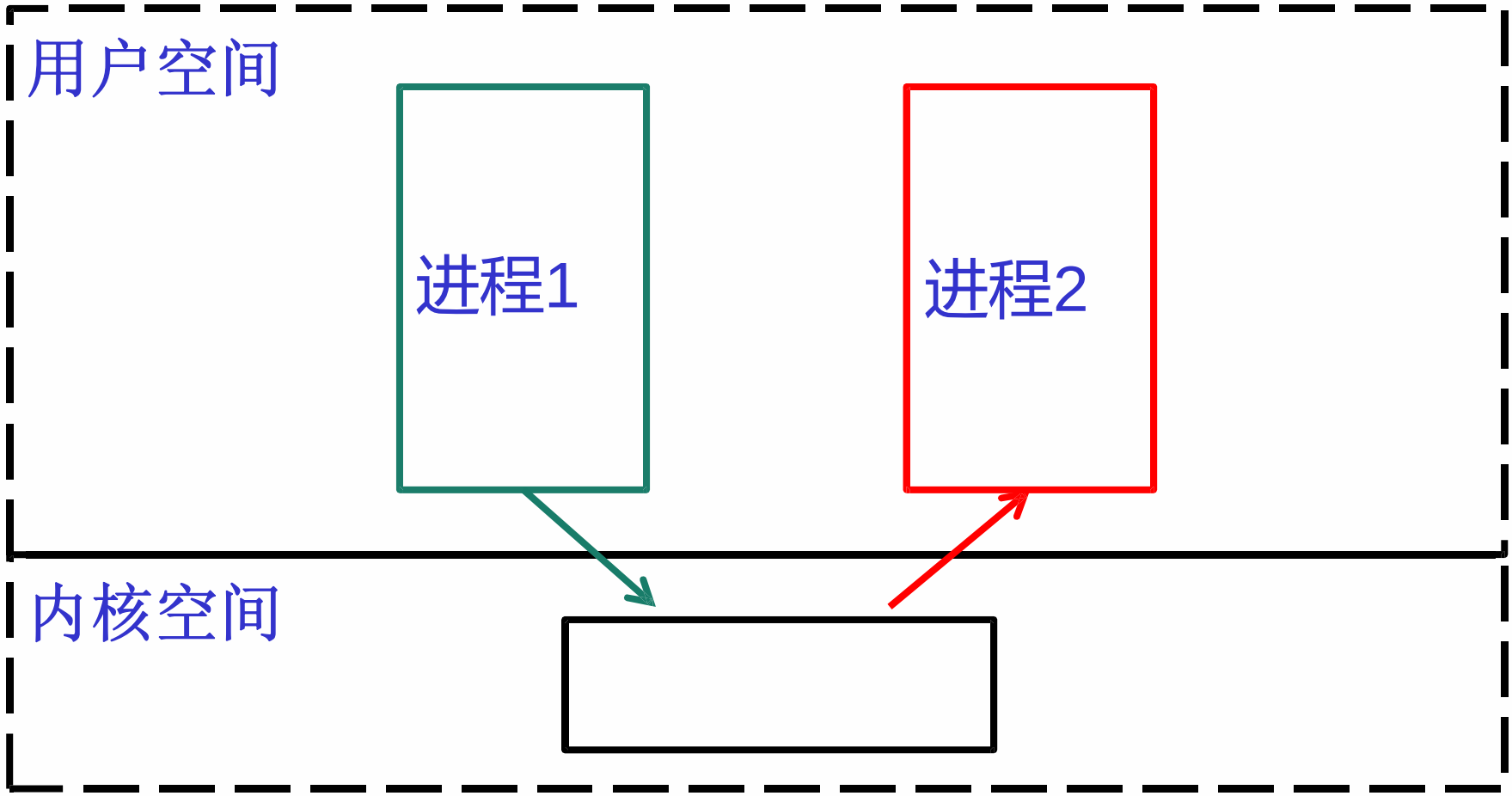
- Linux提供了抢占式的多任务模式
  - ✓ 由调度程序来决定什么时候停止一个进程的运行，以便其他进程能够得到执行机会
  - ✓ 这个强制的挂起动作叫做抢占
  - ✓ 进程被抢占之前能够运行的时间是预先设 置好的，叫进程的时间片
  - ✓ 时间片实际上就是分配给每个可运行进程的处理器时间段

边听课，边记笔记

进程间通信

- 进程间通信就是在不同进程之间传播或交换信息
- 每个进程各自有不同的用户地址空间，进程间并不能直接通信，所以进程之间要交换数据必须通 过内核
- 内核中开辟一块缓冲区，进程1把数据从用户空间拷到内核缓冲区，进程2再从内核缓冲区把数据读走，内核提供的这种机制称为进程间通信（IPC，InterProcess Communication）

进程间通信



进程间通信

- 管道及有名管道
  - ✓ 管道可用于具有亲缘关系进程间的通信；
  - ✓ 有名管道（name_pipe），除了管道的功能外，还可以在许多并不相关的进程之间进行通讯
- 共享内存

  使多个进程可以访问同一块内存空间，是最快的可用IPC形式
- 套接字（Socket）

  更为一般的进程间通信机制，可用于不同机器之间的进程间通信
- 信号量、信号、报文（Message）队列（消息队列）

内存管理

- 内存管理可以使多个进程安全地共享内存

  IPC中的共享内存方式依赖于内存管理

- 管理虚拟内存

  Linux 包括了管理可用内存的方式，以及物理 和虚拟映射所使用的硬件机制。

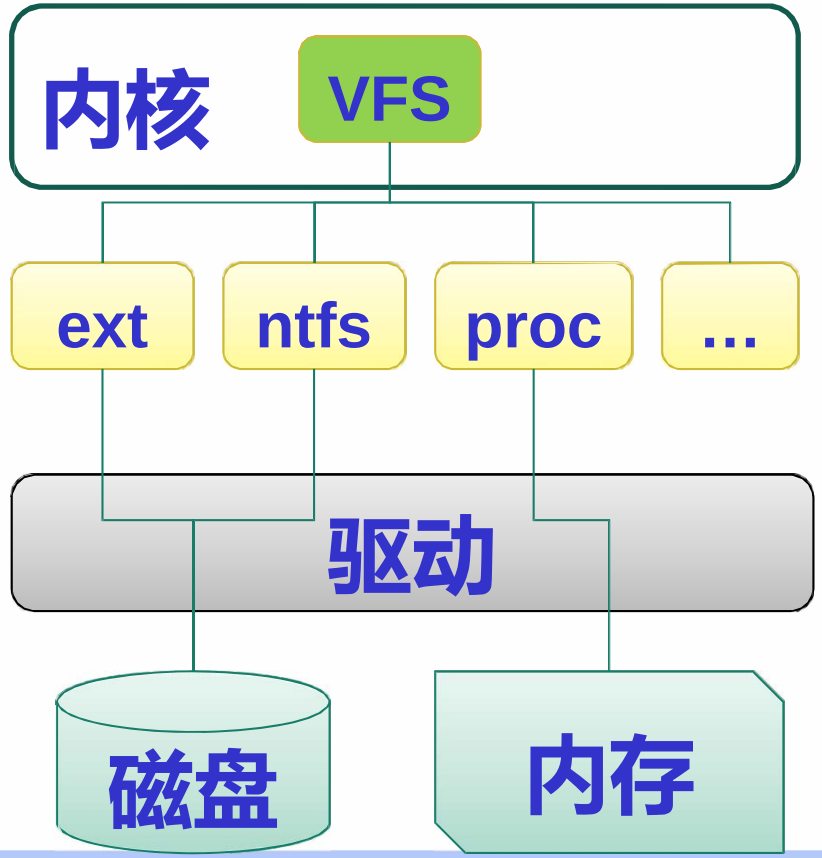
虚拟文件系统

- 虚拟文件系统（VFS）为文件系统提供了一 个通用的接口抽象
- VFS

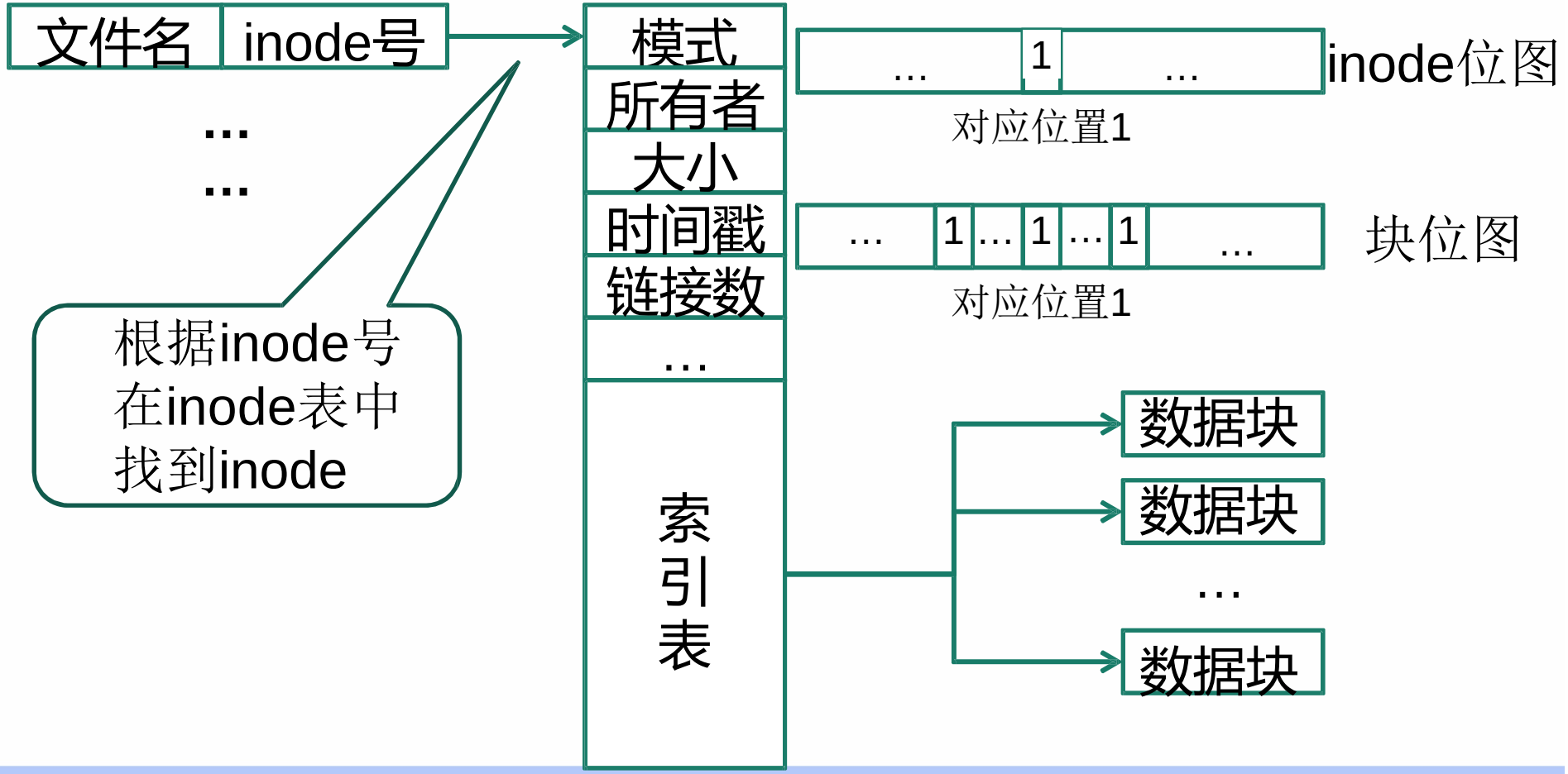
  逻辑文件系统

  EXT4、NFS等

  设备驱动程序



虚拟文件系统


文件名
inode号
...
...
根据inode号
在inode表中
找到inode
模式
所有者
大小
时间戳
链接数
...
索
引
表
...
1
...
inode位图
对应位置1
...
1
...
1
...
1
...
块位图
对应位置1
数据块
数据块
...
数据块

网络接口

- 网络接口在设计上遵循模拟协议本身的分层体系结构
  - ✓ 网络协议
  - ✓ 网络设备驱动程序
- Linux网络支持TCP/IP模型
  - ✓ IP协议
  - ✓ TCP协议
  - ✓ UDP协议

内核移植

(1) 下载内核及其关于 ARM 平台的补丁（如： linux- 3.4.6.tar.gz 和 patch-3.4.6.gz ）。

(2)给内核打补丁：zcat ../ patch-3.4.6.gz | patch

(3)准备交叉编译环境

(4)修改相关的配置文件，如修改makefile文件中关于交叉编译工具相关的内容，此后就可以使用 这个makefile进行编译了

(5)修改Linux内核源码，主要是修改和CPU相关的部分

(6)内核的裁剪，根据项目的需要裁剪内核模块

(7)内核的编译，将裁剪好的内核进行编译，生成二进制映像文件

(8) 将生成的二进制映像文件，烧写到目标平台

内核裁剪

1.make config

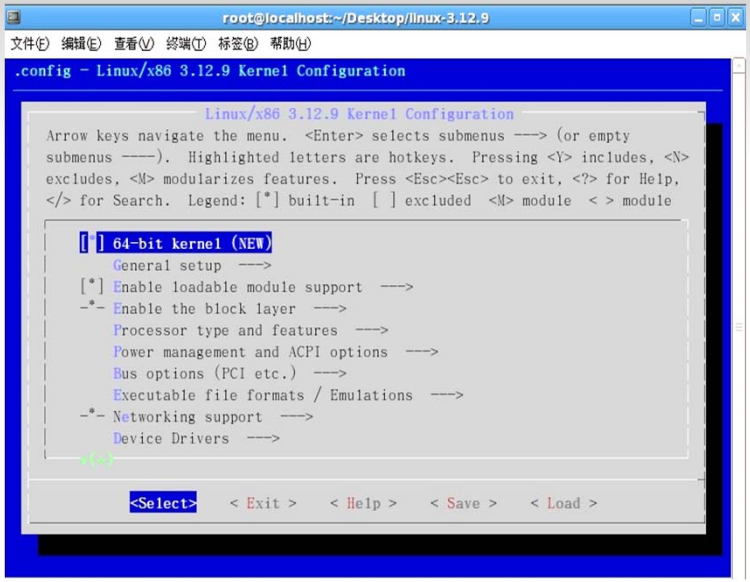
字符界面

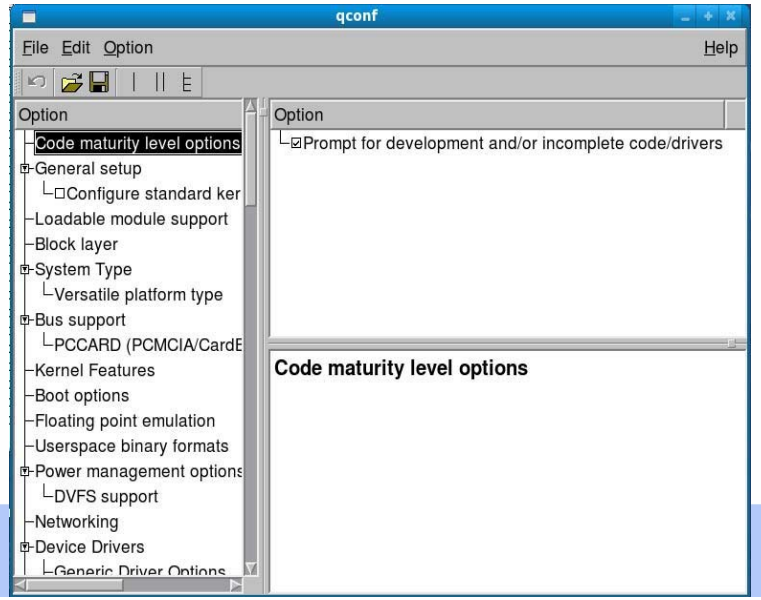
2.make menuconfig

文本图形

3.make xconfig

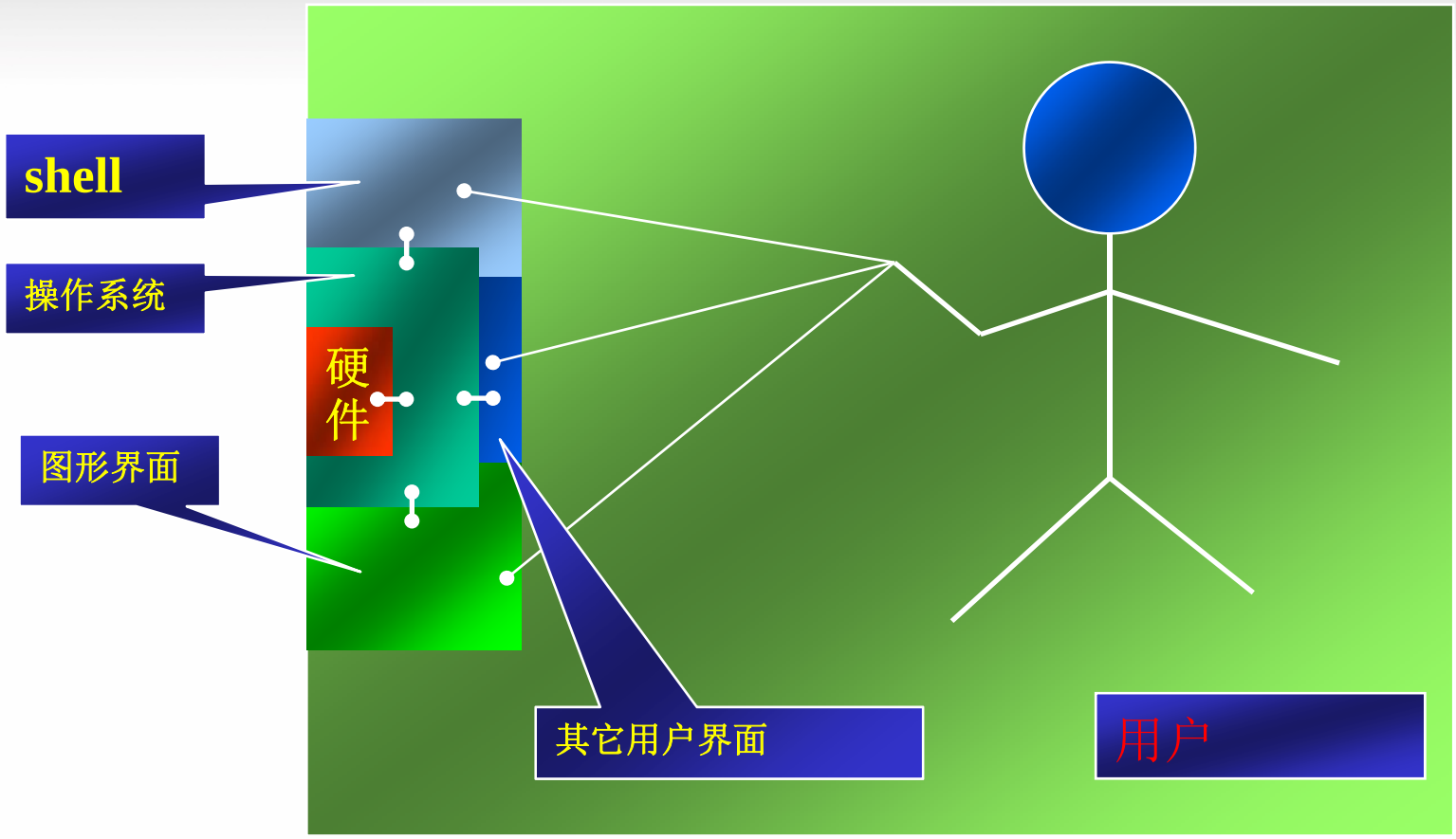
图形界面





内核烧写

- 通过串口
  - ✓ minicom
  - ✓ 与bootloader的烧写类似
- 通过网络
  - ✓ Uboot下使用tftp烧写
  - ✓ 通过JTAG接口


shell
操作系统
硬件
图形界面
其它用户界面
用户

■ **Shell的种类**

– **ash：是贝尔实验室开发的shell，bsh是对ash的符号链接。**

– **bash：是GNU的Bourne Again shell，是GNU默认的shell。sh以及bash2都是对它的符号链接。**

– **tcsh：是Berkeley UNIX C shell。csh是对它的符号链接 。**

■ **Shell的主要功能**

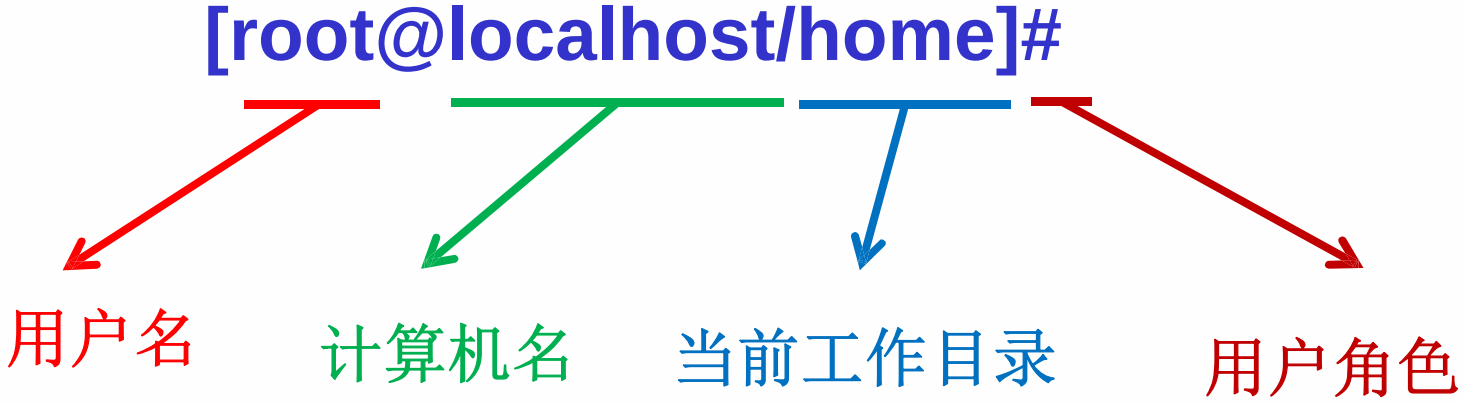
命令解释器、命令通配符、命令补全、别名机制

、命令历史

Linux中执行Shell命令的方式

虚拟终端，**Ctrl+Alt+(F1~F6)**

图形界面的终端模拟器(terminal)



- 命令行技巧

  Tab键可补全命令

  上下键调用命令历史记录

- 命令的终止

  大部分命令，ctrl+c可终止执行

  部分命令，例如man，用“q”退出

登陆

- 进入Linux系统，必须要进入用户的帐号，在系统安装过程中可以创建以下帐号：
  - 1 root-超级用户帐号（系统管理员），使用这个帐号可以在系统中作任何事情。
  - 2 普通用户-这个帐号供普通用户使用，可以进行有限的操作。
- 一般的Linux使用者均为普通用户，而系统管理员一般使用超级用户帐号完成一些系统管理的工作。

- 用户登陆分两步
  - 第一步，输入用户名
  - 第二步，输入用户的口令
  - 当用户正确地输入用户名和口令后，就能合法地进入系统。屏幕显示：
  - **[root@loclhost/root]#**
  - 这时就可以对系统做各种操作了。超级用户的提示符是“#”，其他用户的提示符是“$”。

控制台切换

- **Linux是一个多用户操作系统，它可以同时接受多个用户登录。 Linux还允许一个用户进行多次登陆，因为Linux提供了虚拟控制台的访问方式。**
- 一般从图形界面--〉文本界面
  - **Ctrl+Alt+Fn n=1-6**
- 文本界面--〉图形界面
  - **Alt+F7**

Linux命令格式

- **cmd [-参数] [操作对象]**
  - **cmd**是命令名
  - 单字符参数前使用一个减号（-），单词参数前使用两个减号（--）。
  - 多个单字符参数前可以只使用一个减号。
  - 最简单的**Shell**命令只有命令名，复杂的**Shell**命令可以有多个参数。
  - 操作对象可以是文件也可以是目录，有些命令必须使用多个操作对象， 如**cp**命令必须指定源操作对象和目标操作对象。
  - 命令名、参数和操作对象都作为**Shell**命令执行时的输入，它们之间用空格分隔开。

增加用户

■useradd

- 格式 **useradd [选项] 用户名**
- 范例
- **useradd sjg**
- 添加名字为**sjg**的用户

切换用户

- **su**
- 格式 **su [选项] [用户名]**
- 范例
- **su -root**
- 切换到**root**用户，并将**root**的环境变量同时代入

关机

- **shutdown**
- 格式 **shutdown [-t seconds] [-rkhncfF] time [message]**
- 范例
- **shutdown now**
- 立即关机

拷贝

■ **cp**

- 格式 **:cp [选项] 源文件或目录 目标文件或目录**
- 范例
- **1. cp /home/test /tmp**
- **将/home目录下test文件copy到/tmp目录下**
- **2. cp –r /home/lky /tmp/**
- **将/home目录下的lky目录copy到/tmp目录下**

移动或更名

- **mv**
- 格式 :**mv** [选项] 源文件或目录 目标文件或目录
- 范例
- **1. mv /home/test /home/test1**
- 将**/home**目录下**test**文件更名为**/tmp**目录下
- **2. mv /home/lky /tmp/**
- 将**/home**目录下的**lky**目录移动（剪切）到**/tmp**目录下

删除

■ rm

- 格式 :rm [选项] 源文件或目录
- 范例
- 1. rm /home/test
- 将/home目录下test文件删除
- 2. rm –r /home/lky
- 将/home目录下的lky目录删除

创建目录

■ **mkdir**

- 格式 :**mkdir [选项]** 目录名
- 范例
- **1. mkdir /home/test**
- 在**/home**目录下创建**tes**目录
- **2. mkdir –p /home/lky/tmp/**
- 创建**/home/lky/tmp**目录，如果**lky**不存在，先创建**lky**

改变工作目录

■cd

- 格式 :cd 目录名
- 范例
- **cd /home**
- 进入/home目录

查看当前路径

■**pwd**

- 格式 **:pwd**
- 范例
- **pwd**
- 显示当前工作目录的绝对路径

查看目录

■ **ls**

- 格式 **:ls [选项] [目录或文件]**
- 范例
- **1. ls /home**
- 显示**/home**目录下文件与目录(不包含隐藏文件)
- **2. ls –a /home**
- 显示**/home**目录下所有文件与目录(包含隐藏文件)

类似：**ll** 指令

打包与压缩

■ tar

- 格式 :tar [选项] 目录或文件
- 范例
- **1. tar cvf tmp.tar /home/tmp**
- 将**/home/tmp**目录下所有文件与目录打包成一个**tmp.tar**文件
- **2. tar xvf tmp.tar**
- 将打包文件**tmp.tar**在当前目录下解开

打包与压缩

- **3. tar cvzf tmp.tar.gz /home/tmp**
- 将**/home/tmp**目录下所有文件与目录打包并压缩成一个**tmp.tar.gz**文件
- **4. tar xvzf cvf tmp.tar.gz**
- 将打包压缩文件**tmp.tar.gz**在当前目录下解开

压缩与解压

■ zip与unzip

- 格式：
- **zip [参数] [压缩包名] [压缩的目录或者文件的路径]**
- **unzip [参数] [压缩文件]**
- 范例
- **1. zip tmp.zip tmp**
- **2. unzip tmp.zip**

压缩、解压文件

访问权限

- 系统中的每个文件和目录都有访问许可权限，用它来确定谁可以通过何种方式对文件和目录进行访问。文件或目录的访问权限分为读、写和可执行三种。
- 有三种不同类型的用户对文件或目录进行访问：文件所有者、与所有者同组的用户、其他用户。所有者一般是文件的创建者。

访问权限

- 每一文件或目录的访问权限都有3种，每组用三位表示，分别为文件所有者的读、写和执行权限；与所有者同组的用户的读、写和执行权限；系统中其他用户的读、写和执行权限。当用ls –l命令显示文件的详细信息时，最左边的一列为文件的访问权限。

改变访问权限

- **chmod**
- 格式 **:chmod [who] [+|-|=][mode]**文件名
- 参数
- **Who**
- **u** 表示文件的所有者
- **g** 表示与文件的所有者同组的用户
- **o** 表示其他用户
- **a** 表示所有用户，它是系统默认值
- **Mode:**
- **+** 添加权限， **-** 取消权限， **=**赋予权限
- 如：**chmod g + w hello.c**

改变访问权限

- **Mode所表示的权限可使用8进制数表示**
- **r=4, w=2, x=1，分别以数字之和表示权限**
- **范例:**

  chmod 761 hello.c

  chmod 777 hello

查看目录大小

- **du**
- 格式 :**du** [选项] 目录
- 范例：
- **Du –b ipc**
- 以字节为单位显示**ipc**这个目录的大小

网络配置

- **ifconfig**
- 格式 **:ifconfig [网络接口]**
- 范例：
- **1. ifconfig etho 192.168.0.1**
- 配置**eth0**这一网卡的**ip**地址
- **2.ifconfig eth0 down**
- 暂停**eth0**这一网卡的工作
- **3. ifconfig eth0 up**
- 恢复**eth0**这一网卡的工作

查看网络状态

- **netstat**
- 格式 **:netstat [选项]**
- 范例**:**
- **netstat –a**
- 查看系统中所有网络监听端口

软件安装

- **rpm**
- 格式 :**rpm [选项][安装文件]**
- 范例：
- **1. rpm –ivh tftp.rpm**
- 安装**tftp**
- **2.rpm –qa**
- 列出所有已安装的**rpm**包
- **3. rmp –e name**
- 卸载名为**name**的**rpm**包

挂载

- **mount**
- 格式 **:mount [选项] 设备源 目标目录**
- 范例：
- **mount /dev/cdrom /mnt**
- 将光盘挂载到**/mnt**目录下
- **umount**
- 格式 **:umount 设备源 /目标目录**
- 范例：
- **umount /mnt**
- 取消 光盘在**/mnt**目录下的挂载

查找字符串

- **grep**
- 格式 **:grep** [选项] 字符串
- 范例：
- **1. grep “file” ./ -rn**
- 在当前目录及其子目录中，查找包含**file**字符串的文件
- **2. netstat –a | grep tftp**
- 查看所有端口中用于**tftp**的端口

动态查看CPU使用

- **top**
- 格式 **:top**
- 范例：
- **top**
- 查看系统中进程对**cpu**、内存等的占用情况。

查看进程

- **ps**
- 格式 :**ps[选项]**
- 范例：
- **ps aux**
- 查看系统中所有进程。

杀死进程

■ **kill**

- 格式 **:kill[选项]进程号**
- **-l** 信号，若果不加信号的编号参数，则使用“**-l**”参数会列出全部的信号名称
- **-a** 当处理当前进程时，不限制命令名和进程号的对应关系
- **-p** 指定**kill** 命令只打印相关进程的进程号，而不发送任何信号
- **-s** 指定发送信号
- **-u** 指定用户

范例：

kill –s SIGKILL 4096

杀死**4096**号进程。

帮助

- **man**
- 格式 :**man** 命令名
- 范例:
- **man grep**

目录和文件的基本操作

- 文件查看和连接命令**cat**

  cat [选项] <file1> …

- 分屏显示命令**more**

  more [选项] <file>…

- 按页显示命令**less**

  less [选项] <filename>

  etc/lftp.conf

常用命令

- 显示文字命令**echo**

  echo [-n] <字符串>

- 显示日历命令**cal**

  cal [选项] [[月] 年]

- 日期时间命令**date**

  显示日期和时间的命令格式为：

  date [选项] [+FormatString]

  – 设置日期和时间的命令格式为：

  date <SetString>

- 清除屏幕命令**clear**

常用命令示例

- 显示日历命令cal

  cal [选项] [[月] 年]

cal -y 2013

cal 2013

```
[root@localhost ~]# cal -y 2013
                                2013

      一月                  二月                  三月
日 一 二 三 四 五 六   日 一 二 三 四 五 六   日 一 二 三 四 五 六
       1  2  3  4  5                     1  2                     1  2
 6  7  8  9 10 11 12   3  4  5  6  7  8  9   3  4  5  6  7  8  9
13 14 15 16 17 18 19  10 11 12 13 14 15 16  10 11 12 13 14 15 16
20 21 22 23 24 25 26  17 18 19 20 21 22 23  17 18 19 20 21 22 23
27 28 29 30 31        24 25 26 27 28        24 25 26 27 28 29 30
                                            31

      四月                  五月                  六月
日 一 二 三 四 五 六   日 一 二 三 四 五 六   日 一 二 三 四 五 六
    1  2  3  4  5  6                  1  2  3  4                     1
 7  8  9 10 11 12 13   5  6  7  8  9 10 11   2  3  4  5  6  7  8
14 15 16 17 18 19 20  12 13 14 15 16 17 18   9 10 11 12 13 14 15
21 22 23 24 25 26 27  19 20 21 22 23 24 25  16 17 18 19 20 21 22
28 29 30              26 27 28 29 30 31     23 24 25 26 27 28 29
                                            30

      七月                  八月                  九月
日 一 二 三 四 五 六   日 一 二 三 四 五 六   日 一 二 三 四 五 六
    1  2  3  4  5  6                  1  2  3   1  2  3  4  5  6  7
 7  8  9 10 11 12 13   4  5  6  7  8  9 10   8  9 10 11 12 13 14
14 15 16 17 18 19 20  11 12 13 14 15 16 17  15 16 17 18 19 20 21
21 22 23 24 25 26 27  18 19 20 21 22 23 24  22 23 24 25 26 27 28
28 29 30 31           25 26 27 28 29 30 31  29 30

      十月                  十一月                十二月
日 一 二 三 四 五 六   日 一 二 三 四 五 六   日 一 二 三 四 五 六
       1  2  3  4  5                     1  2   1  2  3  4  5  6  7
 6  7  8  9 10 11 12   3  4  5  6  7  8  9   8  9 10 11 12 13 14
13 14 15 16 17 18 19  10 11 12 13 14 15 16  15 16 17 18 19 20 21
20 21 22 23 24 25 26  17 18 19 20 21 22 23  22 23 24 25 26 27 28
27 28 29 30 31        24 25 26 27 28 29 30  29 30 31
```

本章主要讲述

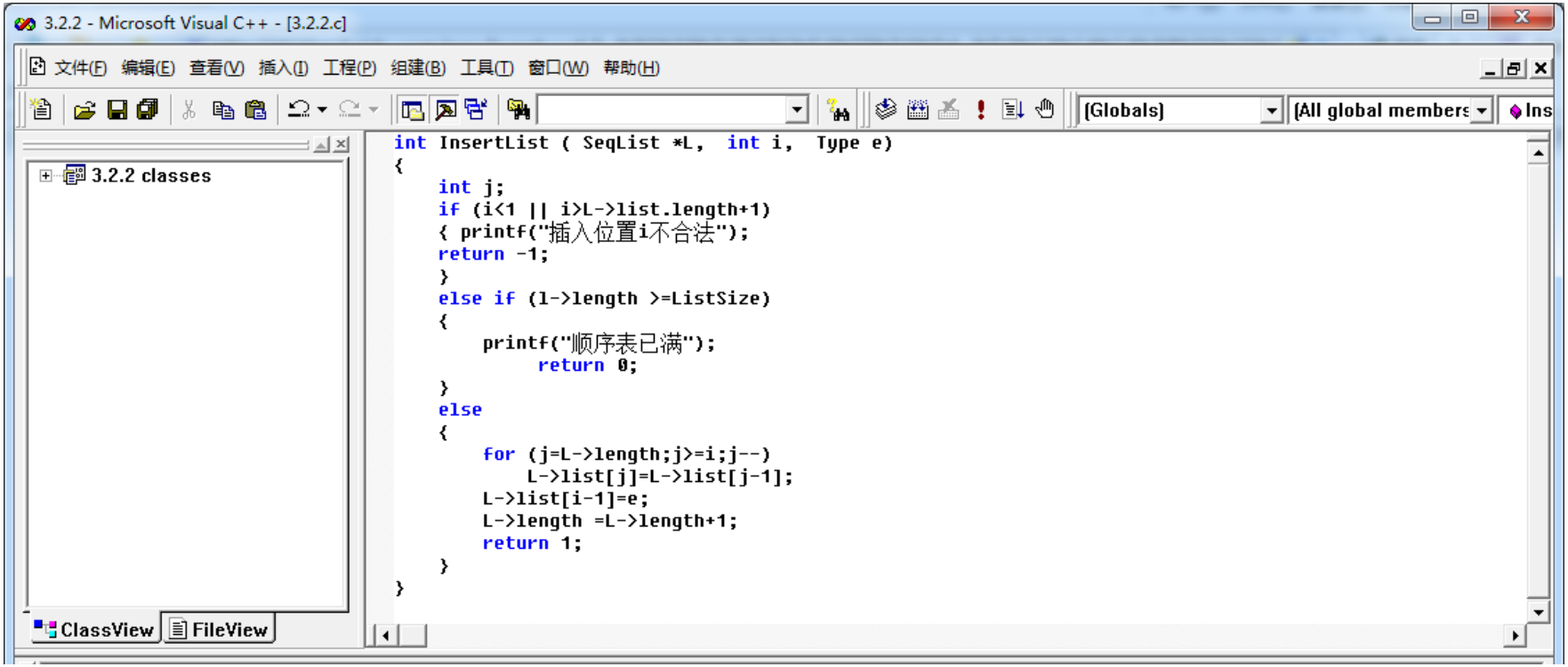
6.1 嵌入式Linux软件设计概述

6.2 Linux简介

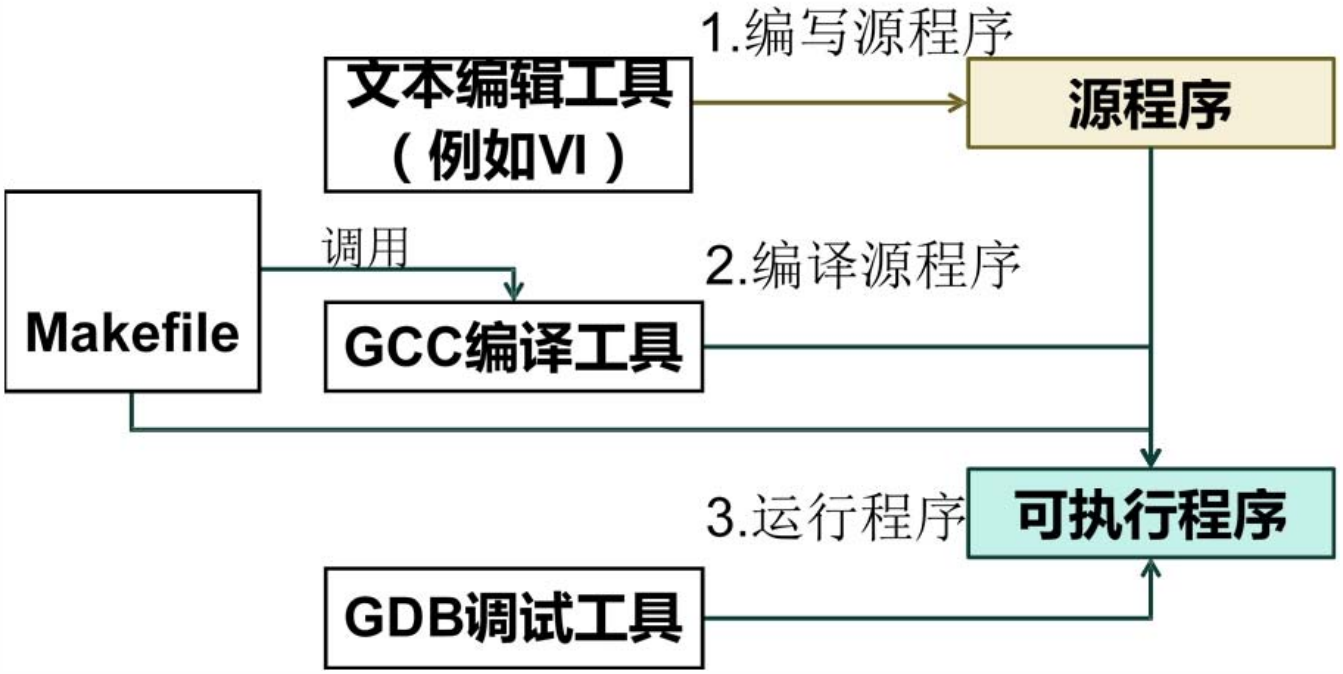
6.3 Linux终端命令

6.4 Linux编程基础

Windows下编的程序能否直接在Linux下编译运行？



- VI编辑器
- GCC编译器
- Make工程管理器
- GDB调试器



1.编写源代码

■VI编辑器

[root@localhost chap2]# vi hello.c

2.编译源程序

■GCC——GNU Compiler Collection

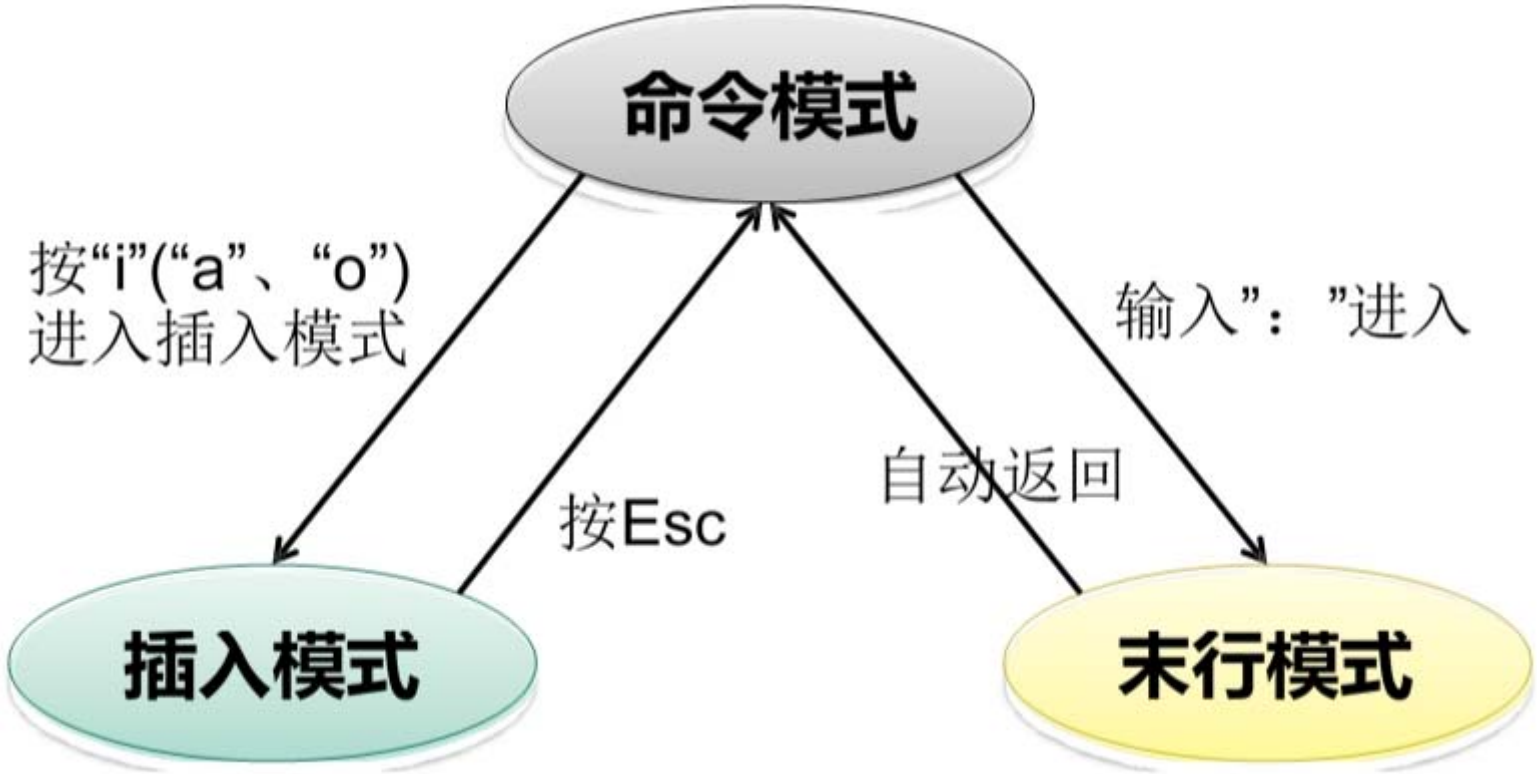
[root@localhost chap2]# gcc –o hello hello.c

3. 运行程序

4. 调试程序

[root@localhost chap2]#./hello

VI编辑器模式



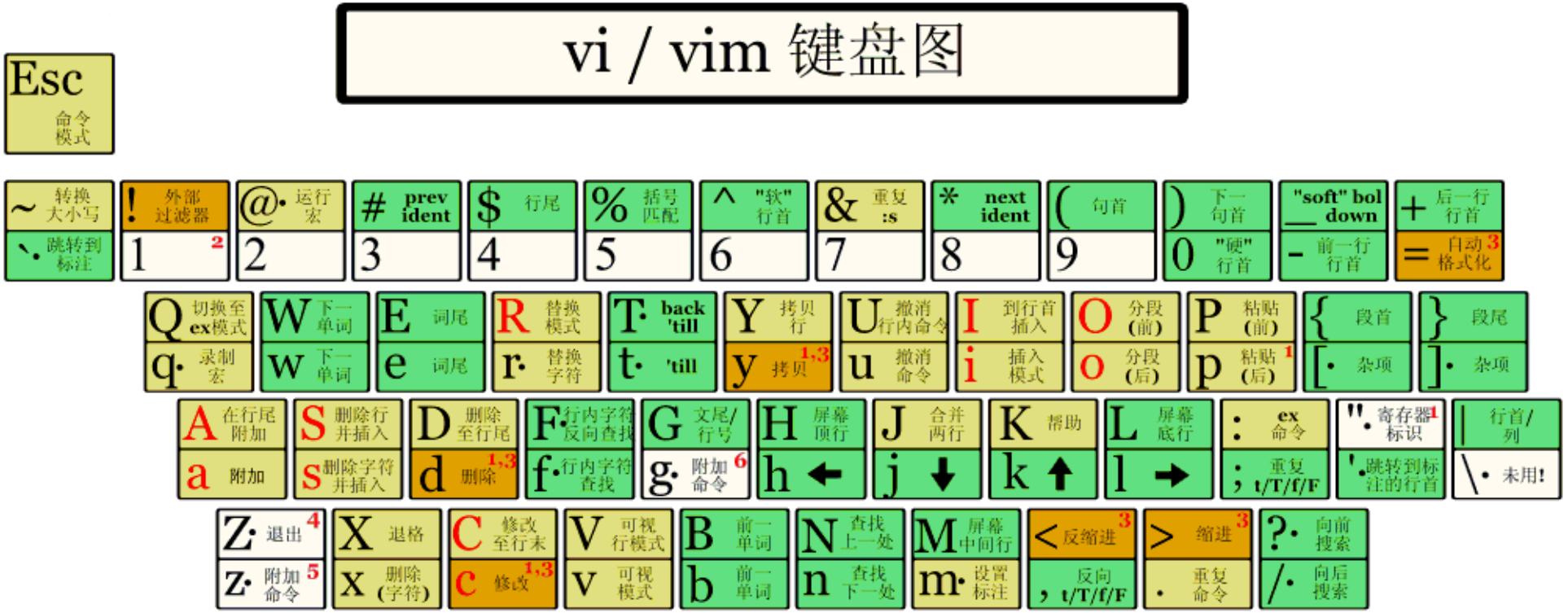
VI末行模式

在命令行模式下输入冒号“:”，就可以进入底行模式

- 1、文件的保存和退出

:w　　保存
:q　　退出
:w!　　强制保存
:q!　　强制退出
:wq（或x）　保存退出
:wq!（或x！）强制保存退出

VI/VIM键盘图


vi / vim 键盘图
Esc 命令模式
~ 转换大小写
` 跳转到标注
! 外部过滤器
1
@ 运行宏
2
prev ident
3
$ 行尾
4
% 括号匹配
5
^ "软"行首
6
& 重复 :s
7
* next ident
8
(句首
9
) 下一句首
0 "硬"行首
"soft" bol
down
- 前一行行首
+ 后一行行首
= 自动格式化
Q 切换至ex模式
q 录制宏
W 下一单词
w 下一单词
E 词尾
e 词尾
R 替换模式
r 替换字符
T back 'till
t 'till
Y 拷贝行
y 拷贝
U 撤消行内命令
u 撤消命令
I 到行首插入
i 插入模式
O 分段(前)
o 分段(后)
P 粘贴(前)
p 粘贴(后)
{ 段首
[杂项
} 段尾
] 杂项
A 在行尾附加
a 附加
S 删除行并插入
s 删除字符并插入
D 删除至行尾
d 删除
F 行内字符反向查找
f 行内字符查找
G 文尾/行号
g 附加命令
H 屏幕顶行
h
J 合并两行
j
K 帮助
k
L 屏幕底行
l
: ex命令
; 重复 t/T/f/F
" 寄存器 标识
' 跳转到标注的行首
| 行首/列
\ 未用!
Z 退出
z 附加命令
X 退格
x 删除(字符)
C 修改至行末
c 修改
V 可视行模式
v 可视模式
B 前一单词
b 前一单词
N 查找上一处
n 查找下一处
M 屏幕中间行
m 设置标注
< 反缩进
, 反向 t/T/f/F
> 缩进
. 重复命令
? 向前搜索
/ 向后搜索

- 名称：
  - **GNU project C and C++ Compiler**
  - **GNU Compiler Collection**
- 管理与维护
  - **GNU项目**
- 对C/C++编译的控制
  - 预处理（**Preprocessing**）
  - 编译（**Compilation**）
  - 汇编（**Assembly**）
  - 链接（**Linking)**

- 基本使用格式
  - $ gcc [选项] <文件名>
- 常用选项及含义

gcc常用选项	
选项	含义
-o *file*	将经过gcc处理过的结果存为文件*file*，这个结果文件可能是预处理文件、汇编文件、目标文件或者最终的可执行文件。 假设被处理的源文件为source.suffix，如果这个选项被省略了，那么生成的可执行文件默认名称为a.out；目标文件默认名为source.o；汇编文件默认名为source.s；生成的预处理文件则发送到标准输出设备。

gcc常用选项	
选项	含义
-c	仅对源文件进行编译，不链接生成可执行文件。在对源文件进行查错时，或只需产生目标文件时可以使用该选项。
-g[gdb]	在可执行文件中加入调试信息，方便进行程序的调试。如果使用中括号中的选项，表示加入**gdb**扩展的调试信息，方便使用**gdb**来进行调试
-O[0、1、2、3]	对生成的代码使用优化，中括号中的部分为优化级别，缺省的情况为**2**级优化，**0**为不进行优化。注意，采用更高级的优化并不一定得到效率更高的代码。
-D*name[=definition]*	将名为***name***的宏定义为***definition***，如果中括号中的部分缺省，则宏被定义为**1**

gcc常用选项	
选项	含义
-I*dir*	在编译源程序时增加一个搜索头文件的额外目录——***dir***，即**include**增加一个搜索的额外目录。
-L*dir*	在编译源文件时增加一个搜索库文件的额外目录——***dir***
-l*library*	在编译链接文件时增加一个额外的库，库名为***library*.a**
-w	禁止所有警告
-W*warning*	允许产生***warning***类型的警告，***warning***可以是：**main**、**unused**等很多取值，最常用是**-Wall**，表示产生所有警告。如果***warning***取值为**error**，其含义是将所有警告作为错误（**error**），即出现警告就停止编译。

gcc文件扩展名规范		
扩展名	类型	可进行的操作方式
.c	c语言源程序	预处理、编译、汇编、链接
.C，.cc，.cp，.cpp，.c++，.cxx	c++语言源程序	预处理、编译、汇编、链接
.i	预处理后的c语言源程序	编译、汇编、链接
.ii	预处理后的c++语言源程序	编译、汇编、链接
.s	预处理后的汇编程序	汇编、链接
.S	未预处理的汇编程序	预处理、汇编、链接
.h	头文件	不进行任何操作
.o	目标文件	链接

- 使用gcc编译代码

  - 源代码

```
    源程序——hello.c
#include <stdio.h>
int main(void)
{
        printf("hello gcc!\r\n");
        return 0;
}
```

- 生成预处理文件
- 命令
  - **$gcc –E hello.c –o hello.i**

预处理文件hello.i的部分内容

```
......
extern void funlockfile (FILE *__stream) ;
# 679 "/usr/include/stdio.h" 3

# 2 "hello.c" 2

int main(void)
{
        printf("hello gcc!\n");
        return 0;
}
```

- 生成汇编文件
  - 命令
    - **$gcc –S hello.c –o hello.s**

文件hello.s的部分内容

```
......
main:
        pushl   %ebp
        movl    %esp, %ebp
.........
        addl    $16, %esp
        movl    $0, %eax
        leave
        ret
      ″
......
```

- 生成二进制文件
  - 生成目标文件
    - 命令：
      - **$gcc –c hello.c –o hello.o**
  - 生成可执行文件
    - 命令：
      - **$gcc hello.c –o hello**
      - 运行程序
      - **$./hello**
        hello gcc!

• 编译多个文件

greeting.h

```
#ifndef _GREETING_H
#define _GREETING_H
void greeting (char * name);
#endif
```

greeting.c

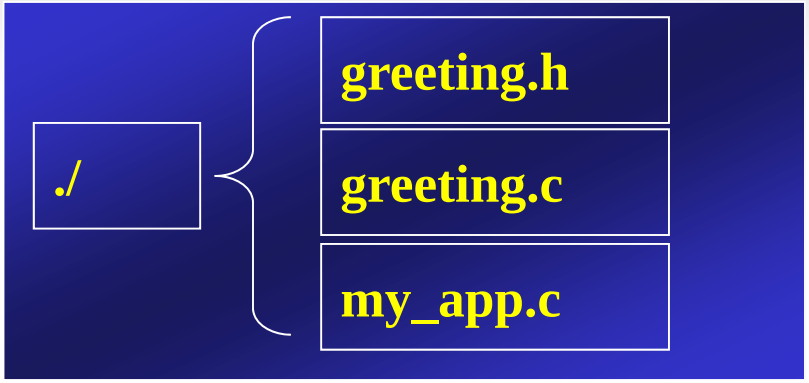
```
#include <stdio.h>
#include "greeting.h"
void greeting (char * name)
{
printf("Hello %s!\r\n",name);
}
```

my_app.c

```
#include <stdio.h>
#include "greeting.h"
#define N 10
int main(void)
{
char name[N];
printf("Your Name,Please:");
scanf("%s",name);
greeting(name);
return 0;
}
```

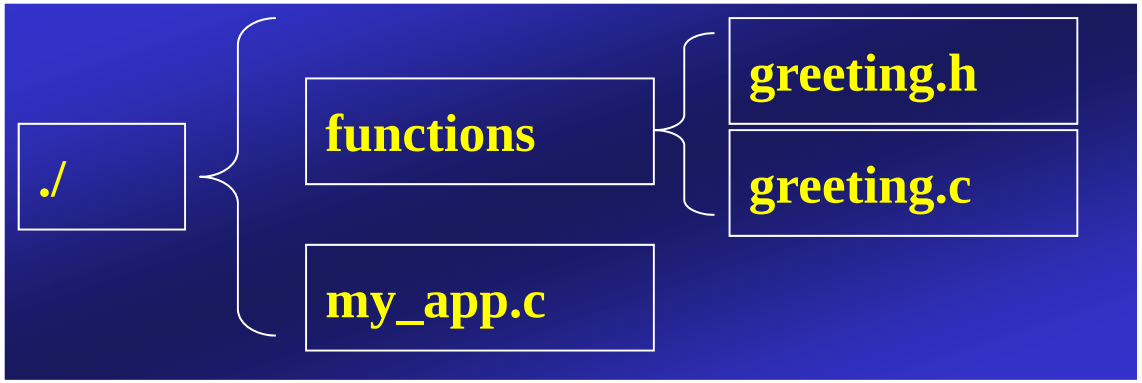
编译多个文件

- 目录结构(1)
  - 编译命令



$ gcc *my_app.c greeting.c* –o my_app

- 目录结构(2)
  - 编译方式(1)



$ gcc my_app.c *functions/greeting.c* –o my_app *-I function*

编译多个文件

- 目录结构(2)
  - 编译方式(2)
    - 分步编译
    - 命令：
      - 1、$gcc -*c* my_app.c -Ifunctions
      - 2、$gcc -*c* functions/greeting.c
      - 3、$gcc *my_app.o greeting.o* –o my_app
    - 思路：
      - 编译每一个.c文件，得到.o的目标文件；
      - 将每一个.o的目标文件链接成一个可执行的文件。

使用make工具

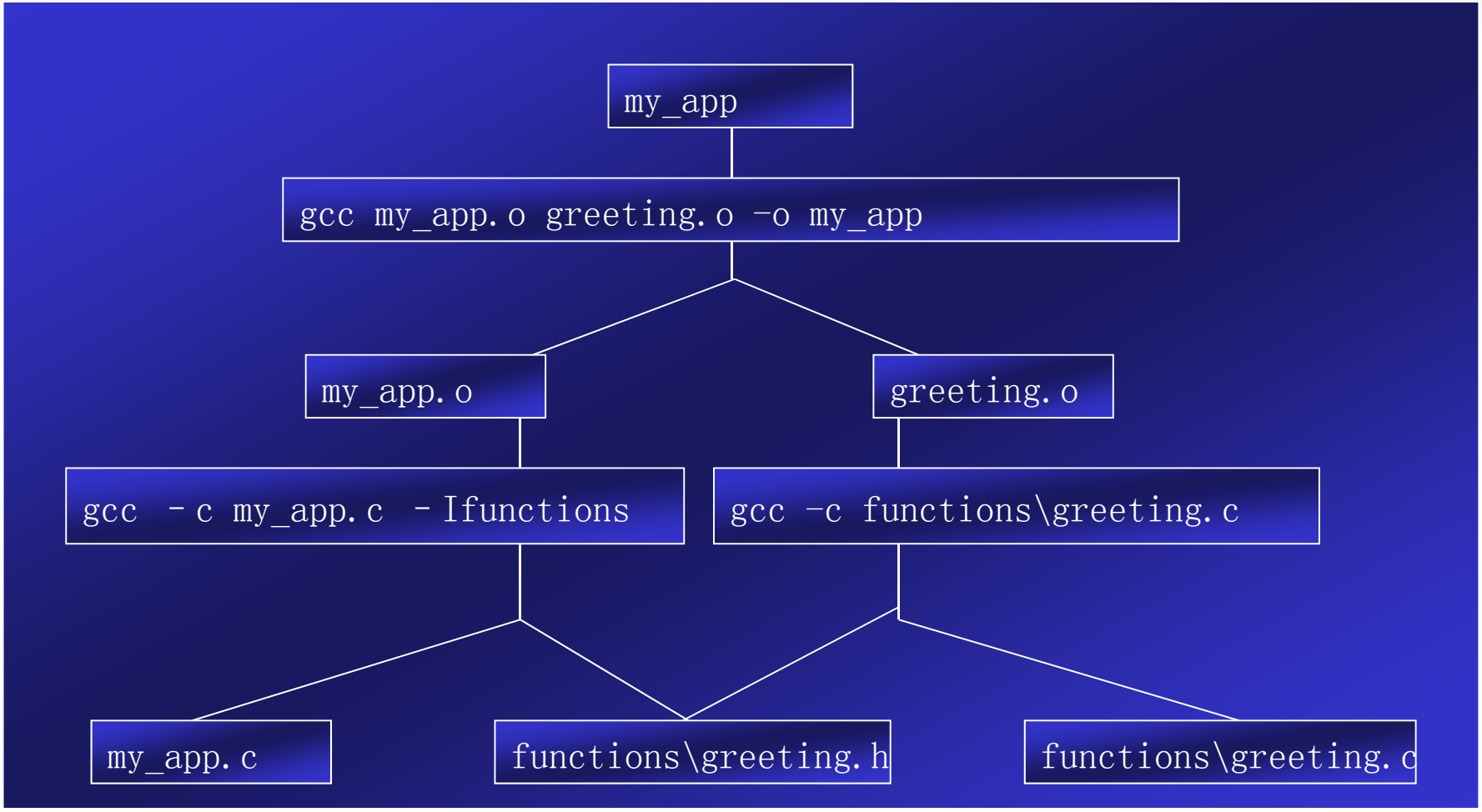
- 基本格式：
- 目标：欲生成的目标文件
- 依赖项：生成目标需要的文件
- 原理：
  - 判断依赖项是否为最新，否则，生成新的目标
- **make工具的使用格式：**
  - **make [[命令选项] [命令参数]]**
  - 通常使用**make**就可以了，**make**会寻找**Makefile**作为编译指导文件。

目标：依赖项列表
（Tab缩进）命令

- Makefile示例

Makefile文件	
1	my_app:greeting.o my_app.o
2	gcc my_app.o greeting.o -o my_app
3	greeting.o:functions\greeting.c functions\greeting.h
4	gcc -c functions\greeting.c
5	my_app.o:my_app.c functions\greeting.h
6	gcc –c my_app.c –I functions

目标的依赖关系


my_app
gcc my_app.o greeting.o -o my_app
my_app.o
greeting.o
gcc -c my_app.c -Ifunctions
gcc -c functions\greeting.c
my_app.c
functions\greeting.h
functions\greeting.c

实用的Makefile

更实用的Makefile文件	
1	OBJS = greeting.o my_app.o
2	CC = gcc
3	CFLAGS = -Wall -O –g
4	my_app:${OBJS}
5	${CC} ${OBJS} -o my_app
6	greeting.o:functions\greeting.c functions\greeting.h
7	${CC} ${CFLAGS} -c functions\greeting.c
8	my_app.o:my_app.c functions\greeting.h
9	${CC} ${CFLAGS} -c my_app.c -Ifunctions

如果一个工程有3个头文件，和8个c文件。

edit : main.o kbd.o command.o display.o insert.o search.o files.o utils.o

- **cc -o edit main.o kbd.o command.o display.o insert.o search.o files.o utils.o**

main.o : main.c defs.h

- **cc -c main.c**

kbd.o : kbd.c defs.h command.h

- **cc -c kbd.c**

command.o : command.c defs.h command.h

- **cc -c command.c**

display.o : display.c defs.h buffer.h

- **cc -c display.c**

insert.o : insert.c defs.h buffer.h

- **cc -c insert.c**

search.o : search.c defs.h buffer.h

- **cc -c search.c**

files.o : files.c defs.h buffer.h command.h

- **cc -c files.c**

utils.o : utils.c defs.h

- **cc -c utils.c**

clean :

- **rm edit main.o kbd.o command.o display.o insert.o search.o files.o utils.o**

- objects = main.o kbd.o command.o display.o insert.o search.o files.o utils.o
- edit : $(objects)
  cc -o $(objects)
- main.o : main.c defs.h
  cc -c main.c
- kbd.o : kbd.c defs.h command.h
  cc -c kbd.c
- command.o : command.c defs.h command.h
  cc -c command.c
- display.o : display.c defs.h buffer.h
  cc -c display.c
- insert.o : insert.c defs.h buffer.h
  cc -c insert.c
- search.o : search.c defs.h buffer.h
  cc -c search.c
- files.o : files.c defs.h buffer.h command.h
  cc -c files.c
- utils.o : utils.c defs.h
  cc -c utils.c
- .PHONY : clean
- clean :
  rm edit $(objects)

- objects = main.o kbd.o command.o display.o insert.o search.o files.o utils.o
  - cc = gcc
- edit : $(objects)
  - cc -o edit $(objects)
- main.o : defs.h
- kbd.o : defs.h command.h
- command.o : defs.h command.h
- display.o : defs.h buffer.h
- insert.o : defs.h buffer.h
- search.o : defs.h buffer.h
- files.o : defs.h buffer.h command.h
- utils.o : defs.h .
- .PHONY : clean
- clean :
  - rm edit $(objects)

- objects = main.o kbd.o command.o display.o \
  insert.o search.o files.o utils.o
- edit : $(objects)
  cc -o edit $(objects)
- $(objects) : defs.h
- kbd.o command.o files.o : command.h
- display.o insert.o search.o files.o : buffer.h
- .PHONY : clean
- clean :
  rm edit $(objects)

伪目标文件

- 调试
  - 静态调试：
    - 在程序编译阶段查错并修正错误；
    - 主要为语法错误：
      - 输入错误；
      - 类型匹配错误；
    - 排错方式：
      - 利用错误、警告信息，并结合源文件环境排错
  - 动态调试：
    - 在程序运行阶段差错并修正错误；
    - 主要错误类型：
      - 算法错误；
      - 输入错误；
    - 排错方式：
      - 利用调试工具定位并修正错误；

调试举例

greeting.h

```
#ifndef _GREETING_H
#define _GREETING_H
void greeting (char * name);
#endif
```

greeting.c

```
#include <stdio.h>
#include "greeting.h"
void greeting (char * name)
{
printf("Hello !\r\n");
}
```

my_app.c

```
1  #include <stdio.h>
2  #include "greeting.h"
3  #define N 10
4  int main(void)
5  {
6  char name[n];
7  printf("Your Name,Please:");
8  scanf("%s",name)
9  greeting(name);
10 /*return 0;*/
11 }
```

调试举例

学院 荣誉 责任

- 分块编译
  - **greeting.c**
    - **$gcc *-g -Wall* -c functions/greeting.c**

      -g：将调试信息加入到编译的目标文件中；

      -Wall：将编译过程中的所有级别的警告都打印出来；
    - 无错误
  - **my_app.c**
    - **$gcc *-g -Wall* -c my_app.c -Ifunctions**
    - 参数含义同上
    - 错误信息：

调试举例

- 错误信息：

```
my_app.c: In function `main':
my_app.c:6: `n' undeclared (first use in this function)
my_app.c:6: (Each undeclared identifier is reported only once
my_app.c:6: for each function it appears in.)
my_app.c:9: parse error before "greeting"
my_app.c:6: warning: unused variable `name'
```

– 错误记录格式：
  - 文件名：行号：错误描述

调试举例

- 分析、定位错误（警告）：
  - my_app.c的第6行：
    - 描述含义：
      - n是一个没有声明的变量；
    - 分析：
      - 声明数字name时用到了变量n，但变量n在之前没有声明；
    - 改正：
      - 声明一个新变量n；

      或者
      - 将n改为宏N
    - 这里取第2种改正方法。

调试举例

- **my_app.c的第9行：**
  - 描述含义：
    - 在“**greeting**”之前出现解析错误；
  - 分析：
    - **c**中每行程序以；结束，第**9**行**greeting**之前的程序行没有以；结束；
  - 改正：
    - 第**8**行末尾增加“；”
- 重新编译
- 错误信息：

```
my_app.c: In function `main':
my_app.c:11: warning: control reaches end of non-void function
```

调试举例

- 分析、定位错误（警告）：
  - 警告：**my_app.c的11行**
    - 描述含义：
      - 控制以非空函数结束；
    - 分析：
      - **main**函数返回类型为**int**，源程序没有以**return** 整数形式结束；
    - 改正：
      - 将**main**改为返回**void**类型；

      或者：
      - 在**main**程序后增加**return** 返回语句；
      - 采用第**2**种解决方式；
- 重新编译，无错误或警告信息，完成静态调试

调试举例

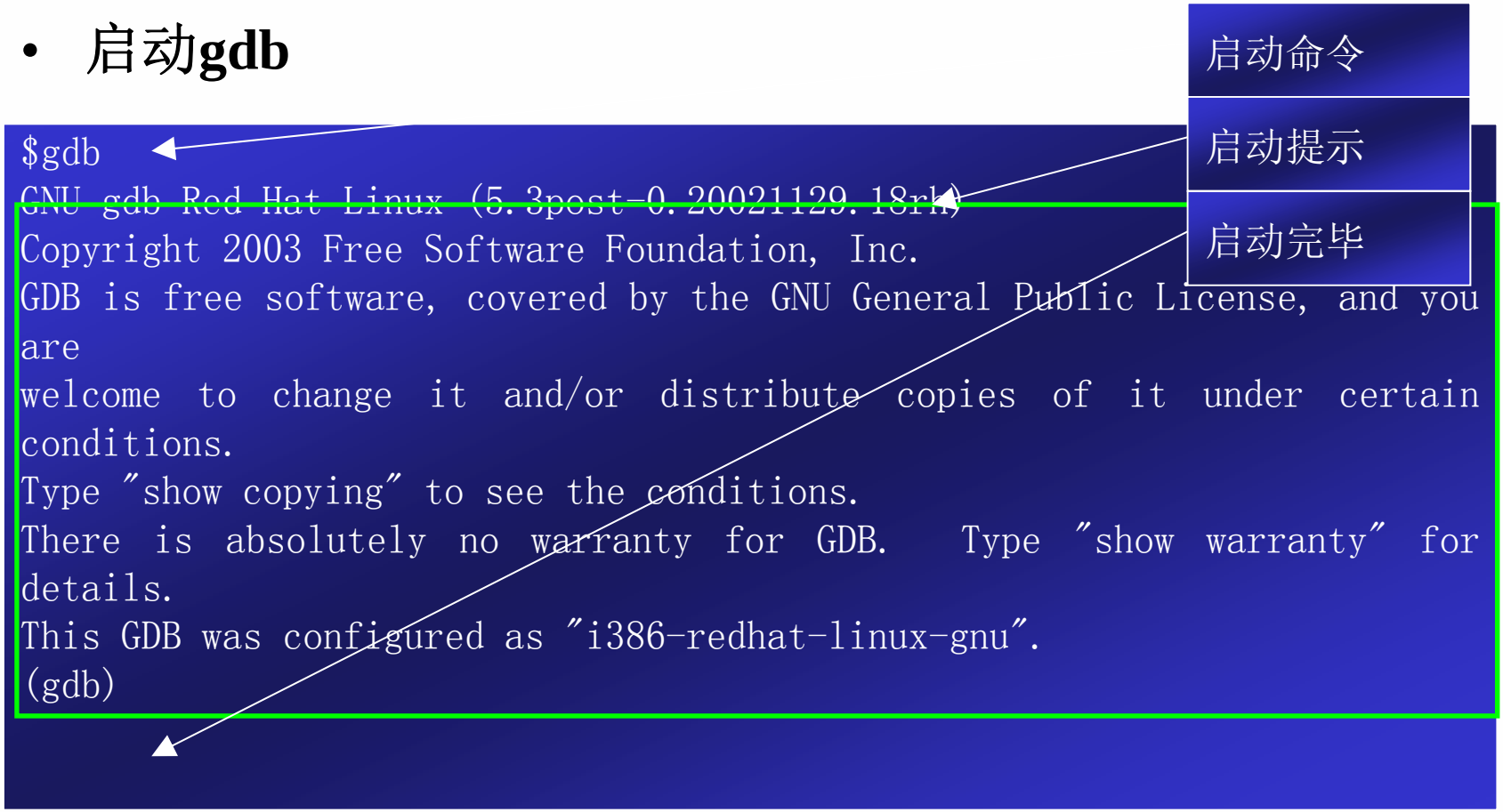
- 静态调试总结
  - 主要为语法错误：
    - 输入错误；
    - 类型匹配错误；
  - 分析信息：
    - 主要来自gcc编译时产生的提示信息
  - 错误警告定位：
    - 不一定在提示信息描述的地方；
    - 综合分析提示信息及提示行的上下文环境，定位并修正错误、警告。
    - 有的警告可以不用修复；

- 常见的动态调试方法：
  - 增加调试语句；
  - 记录程序的执行状况；
  - 观察内存变化；
  - 使用调试工具；
- **GUN Debuger的功能：**
  - 启动程序，设置程序执行的上下文环境；
  - 在指定的条件下停止程序；
  - 程序停止时，检查程序的状态；
  - 在程序运行时，改变程序状态，使其按照改变后的状态继续执行。

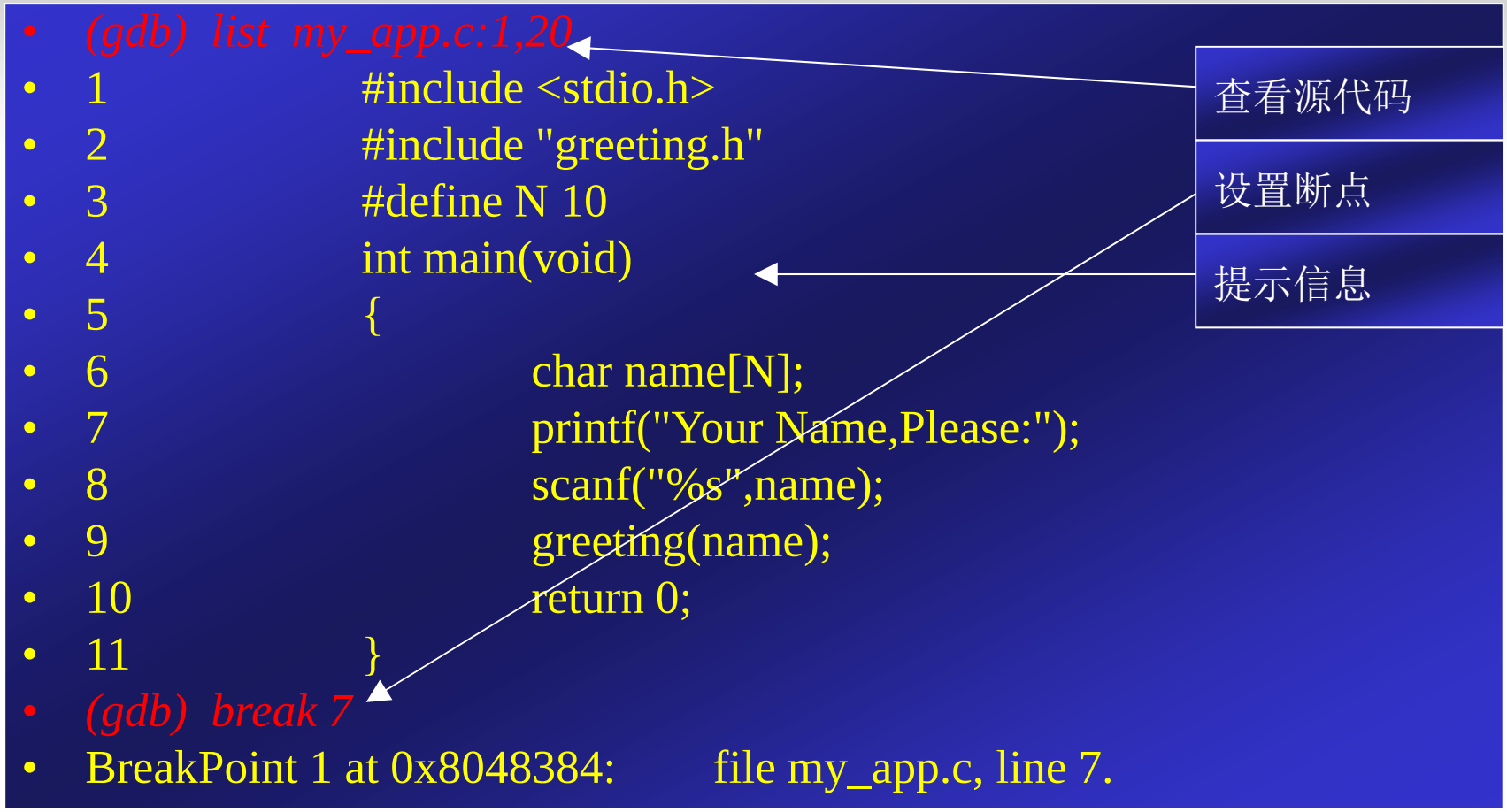
gdb常用的调试命令	
命令	含义
file	指定需要进行调试的程序
step	单步（行）执行，如果遇到函数会进入函数内部
next	单步（行）执行，如果遇到函数不会进入函数内部
run	启动被执行的程序
quit	退出gdb调试环境
print	查看变量或者表达式的值
break	设置断点，程序执行到断点就会暂停起来
shell	执行其后的shell命令
list	查看指定文件或者函数的源代码，并标出行号

动态调试举例

- 对静态调试中的例子继续进行动态调试
- 工具:**gdb**
- 启动**gdb**

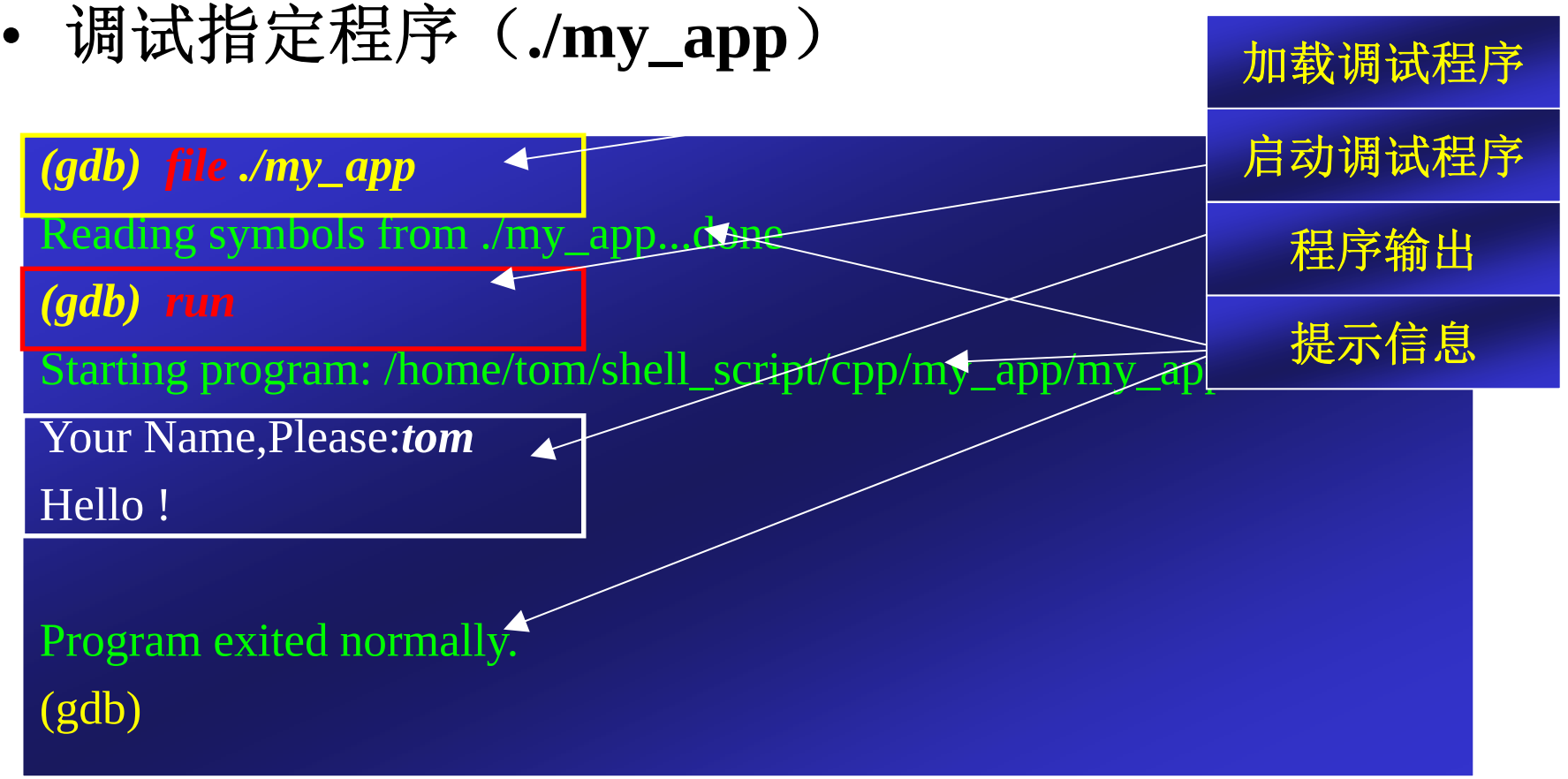


动态调试举例



- 在程序第7行设置断点
  - 命令：*(gdb) break 7*

- 调试指定程序（**./my_app**）

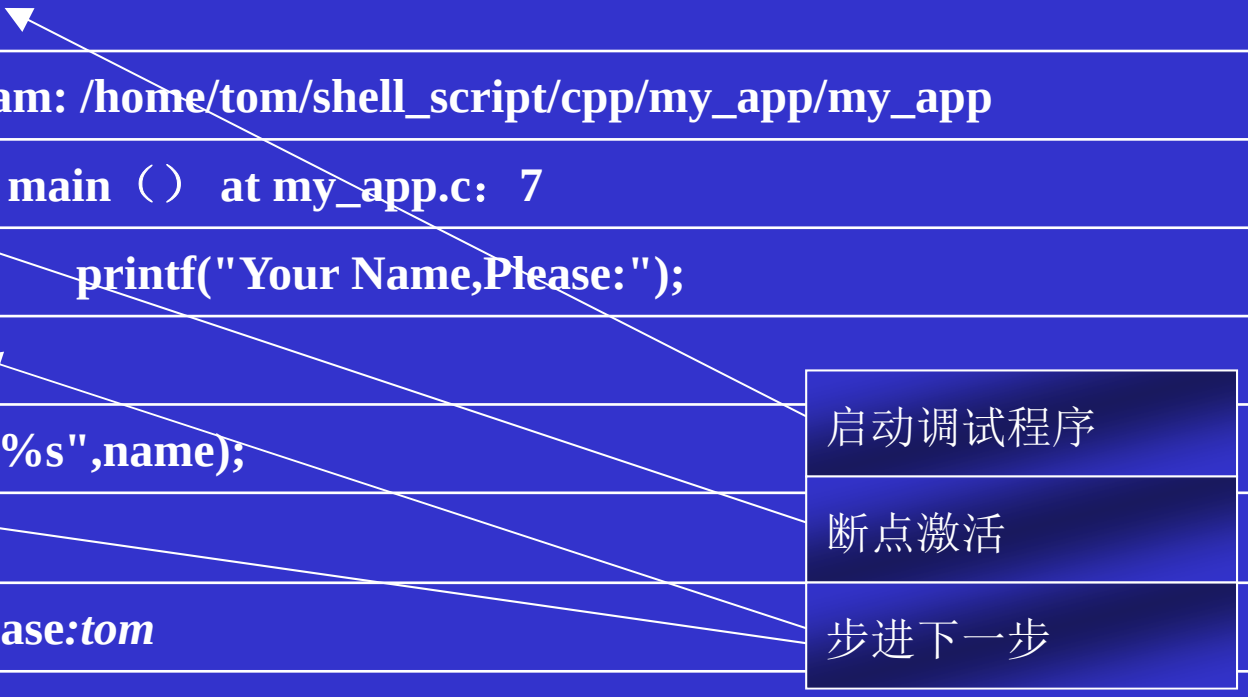


  - 问题：
    - 期望的输出和实际输出不一致

动态调试举例

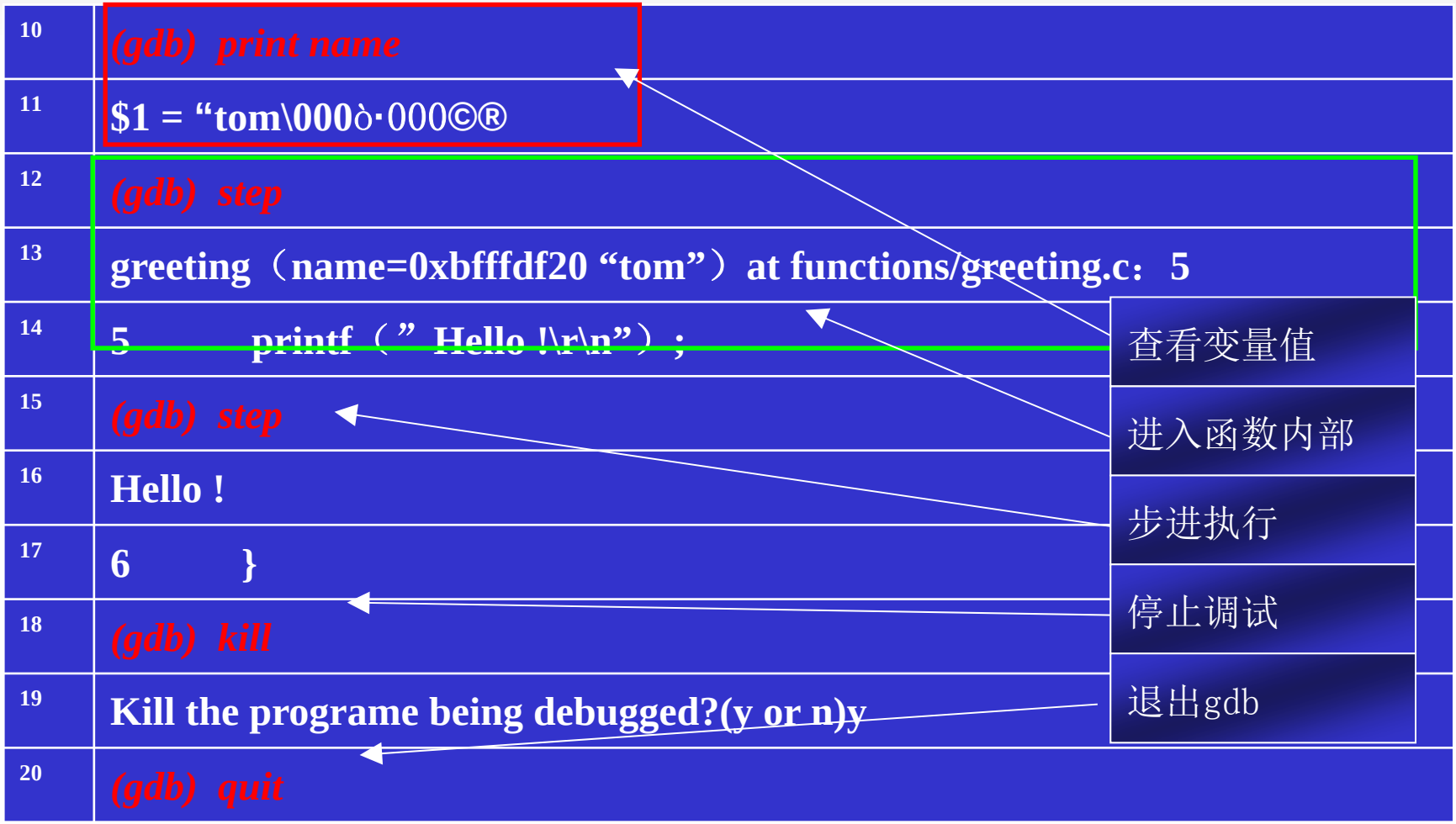
- 错误详细定位

1	*(gdb) run*
2	Starting program: /home/tom/shell_script/cpp/my_app/my_app
3	Breakpoint 1，main（） at my_app.c：7
4	7 printf("Your Name,Please:");
5	*(gdb) next*
6	8 scanf("%s",name);
7	*(gdb) next*
8	Your Name,Please:*tom*
9	9 greeting(name);



动态调试举例

- 错误详细定位



动态调试举例

- 分析:
  - **11行说明name变量被正确赋值（tom）**
  - **13行说明name变量值被正确赋予greeting的参数变量name**
  - **16说明打印出现了错误，即错误出现在函数greeting中；**
- 综合分析
  - 错误出现在**greeting.c**的第**5**行；
  - 原因：
    - 没有输出字符串的格式不对；
- 改正错误

动态调试举例

- 动态调试总结
  - 主要错误类型：
    - 算法错误；
    - 输入错误；
  - 定位方法：
    - 设置断点；
    - 单步步进执行；
    - 查看变量取值变化；
    - 反复执行，逐步缩小错误范围；

谢谢各位同学！

Q&A