嵌入式系统原理

Principle of Embedded System

合肥工业大学.计算机与信息学院

2023年5月

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

本章主要讲述

6.1 Bootloader过程

6.2 嵌入式操作系统简介

6.3 Linux终端命令

6.4 Shell编程

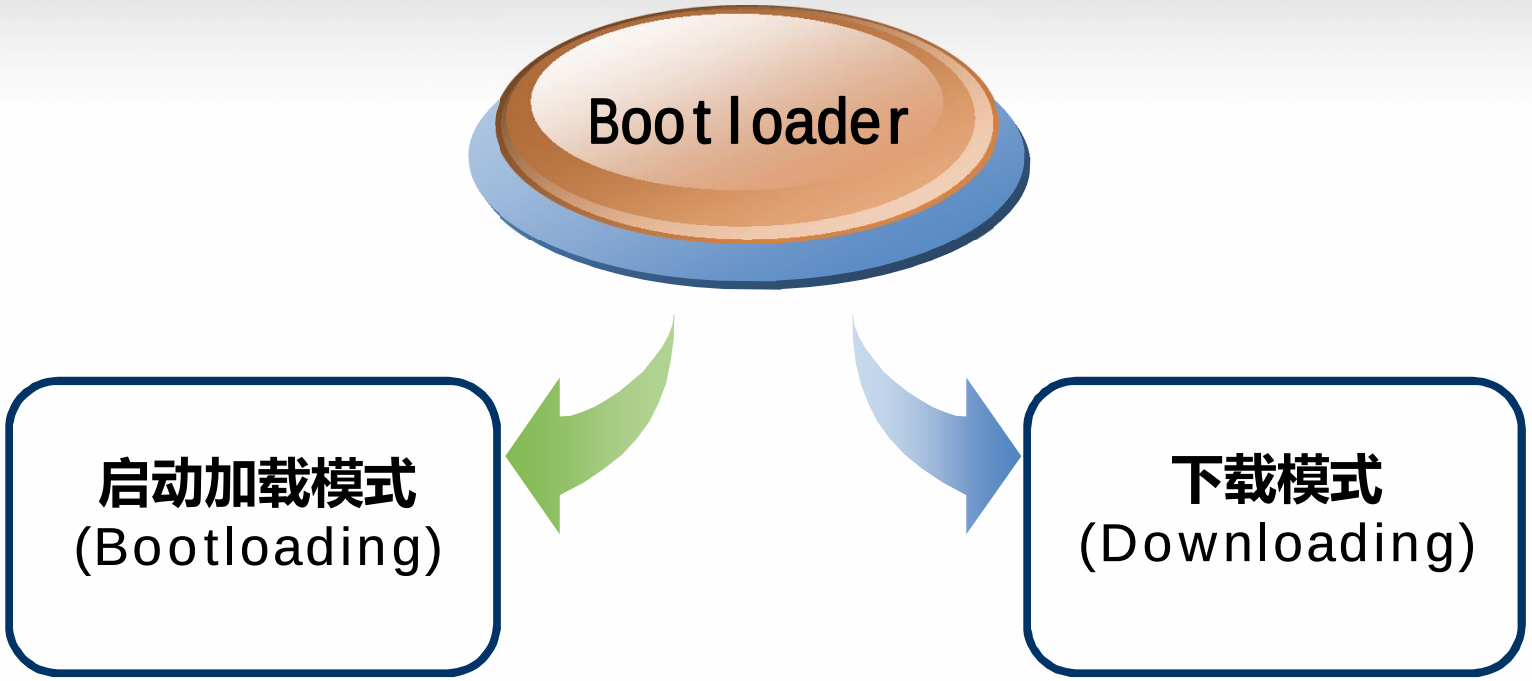
6.5 Linux编程基础

Bootloader概念

- **Bootloader** 就是在操作系统内核运行之前运行的 一段小程序。
- 通过这段小程序，可以初始化硬件设备、建立内存空间的映射图，从而将系统的软硬件环境带到一个合适的状态，以便为最终调用操作系统内核准备好正确的环境。

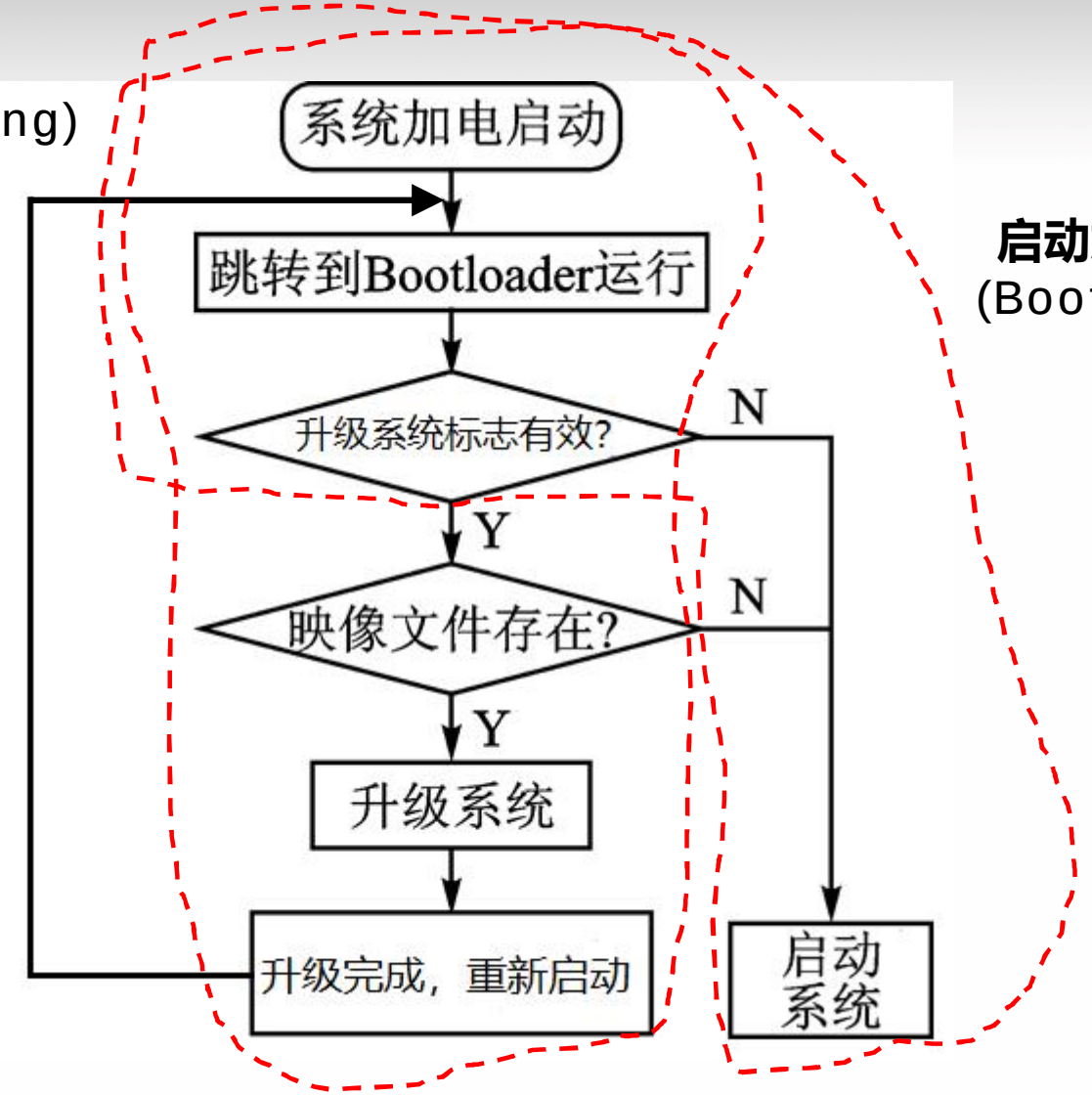
- **Boot Loader**是严重地依赖于硬件而实现的， 包括**CPU**、嵌入式板级设备的配置等。
- 依赖于处理器架构： ARM、MIPS、DSP、x86 etc
- 依赖于具体的板级配置 ：不同厂家的芯片、不同的内存空间

Bootloader的运行模式



Bootloader的运行模式

下载模式
(Downloading)



启动加载模式
(Bootloading)

启动加载模式(Bootloading)

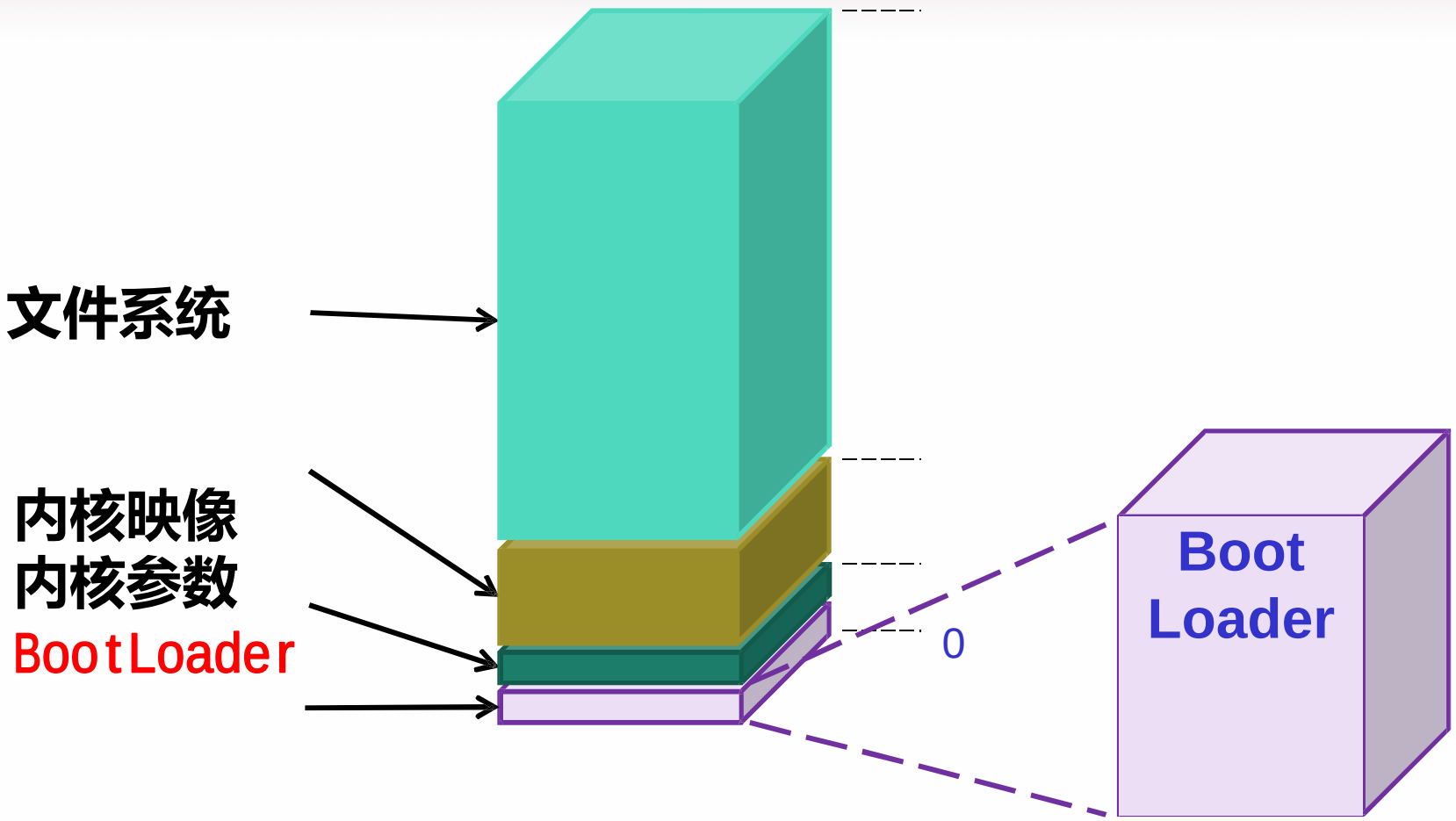
- 自主（**Autonomous**）模式，是**BootLoader** 的正常工作模式
- 流程：
  - 从目标机某个固态存储设备上将**OS**加载到 **RAM**
  - 准备好内核运行所需的环境和参数
  - 在**RAM**运行操作系统内核

下载模式(Downloading)

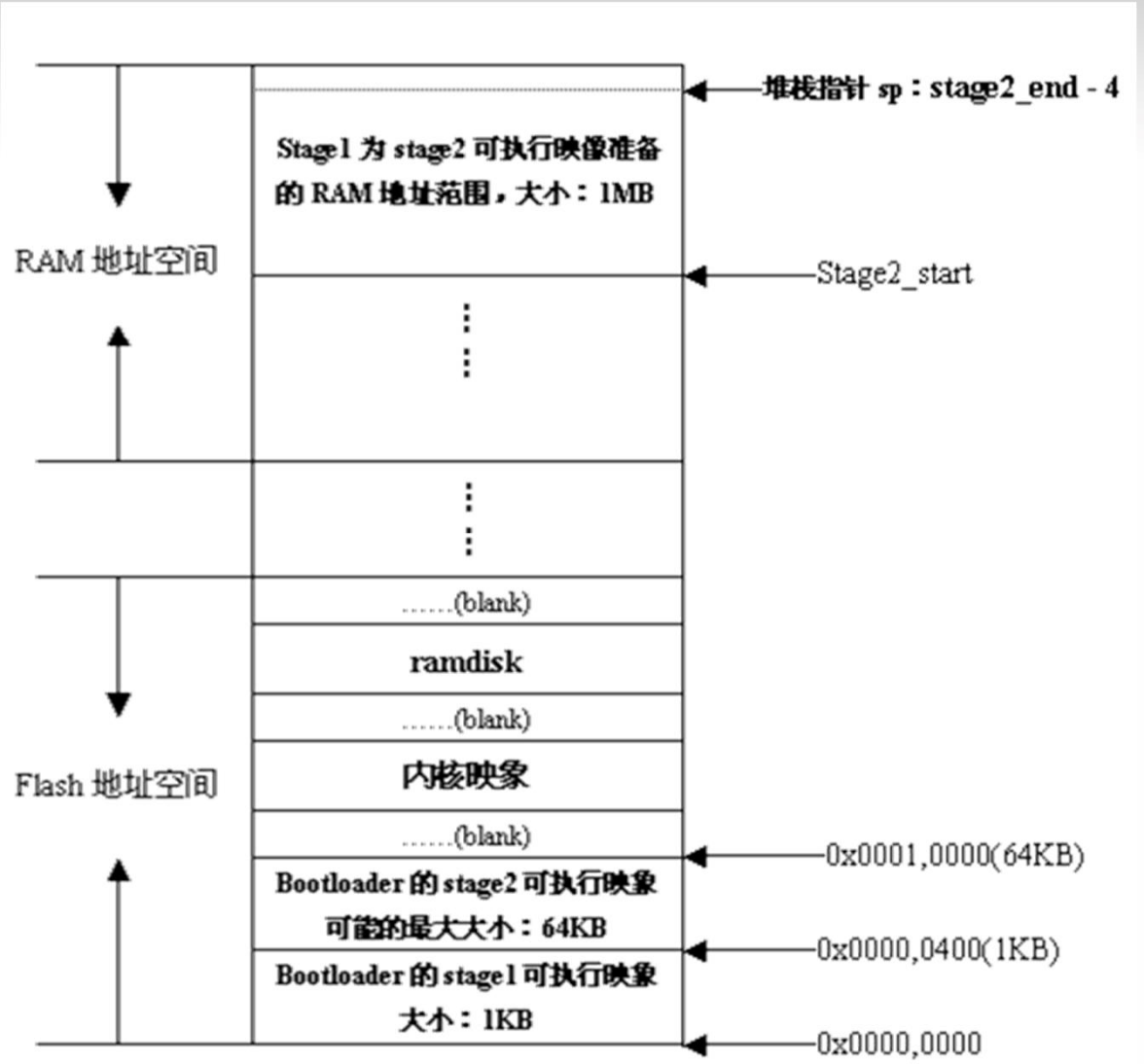
- 用于调试和版本修改：目标机上的 **Boot Loader** 将通过串口连接或网络连接等通信手段从主机下载文件到目标机的 **RAM** 中，然后再被 **Boot Loader** 写到目标机上的**FLASH** 类固态存储设备中
- 所谓的“刷机”就是使用这种模式
- 工作于这种模式下的 **Boot Loader**需要提供 一个简单的命令行接口

  常用命令**tftp**、**update**

- 固态存储设备的典型空间分配结构



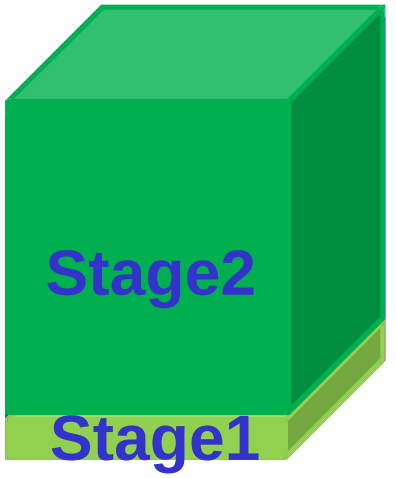
Bootloader 运行过程



BootLoader通常分为 stage1 和 stage2 两大部分

- Stage1——依赖于CPU体系结构的代码，例如设备初始化代码等。通常都用汇编语言来实现， 以达到短小精悍的目的
- Stage2——通常用C语言来实现，可以实现更复杂的功能，而且代码会具有更好的可读性和可移植性

分级载入机制



Stage1具体过程

1. **基本的硬件初始化**，目的是为 stage2 的执行以及随后的 kernel 的执行准备好一些基本的硬件环境

(1) 屏蔽所有的中断

- 在 BootLoader 的执行全过程中可以不必响应任何中断
- 中断屏蔽通过写 CPU 的中断屏蔽寄存器或状态寄存器来完成

(2) 设置CPU的速度和时钟频率

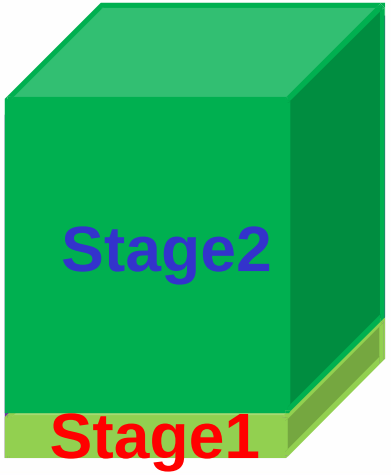
(3) RAM初始化

- 设置内存控制器的功能寄存器及各内存控制寄存器等

(4)初始化 LED

- 目的是表明系统的状态是 OK 还是 Error

(5)关闭 CPU 内部指令 / 数据 cache

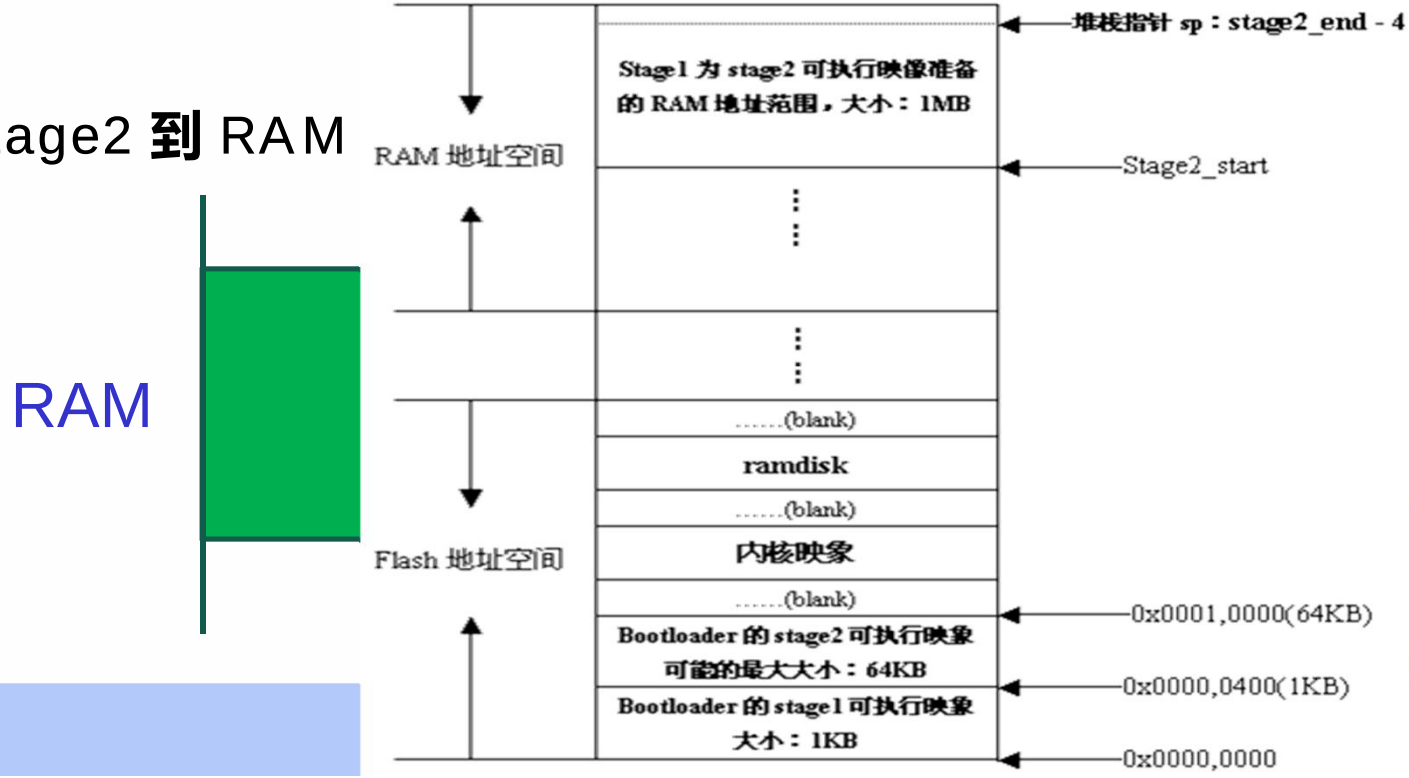


Stage1具体过程

2.为加载 stage2 准备 RAM 空间

为了获得更快的执行速度，通常把 stage2 加载到 RAM 空间中来执行，必须为加载 Boot Loader 的 stage2 准备好一 段可用的 RAM 空间范围。

3.拷贝 stage2 到 RAM



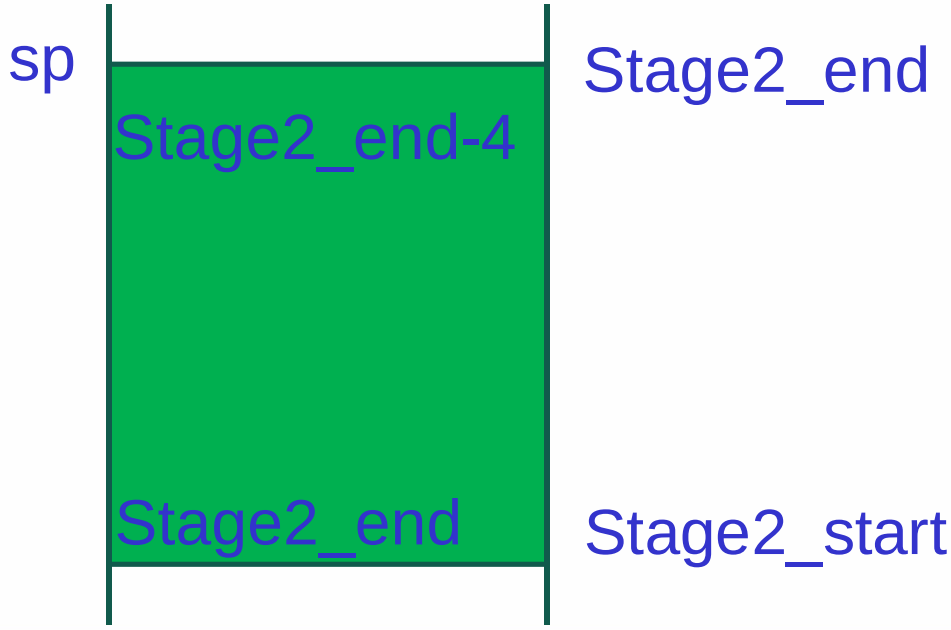
Stage1具体过程

4.设置堆栈指针 sp为执行 C 语言代码作好准备

- sp 指向stage2的RAM空间的最顶端
- 这里的堆栈向下生长， sp指向stage2_end-4
- ARM支持全部4种堆栈方式——满递增、空递增、满递减、空递减

5.跳转到

stage2 的C 入口点



Stage2具体过程

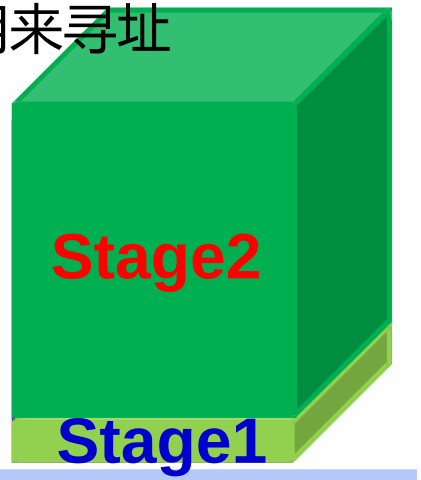
1.初始化本阶段要使用到的硬件设备

- 初始化至少一个串口，以便和终端用户进行I/O 输出信息
- 初始化计时器等
- 设备初始化完成后，可以输出一些打印信息， 程序名字字符串、版本号等。

2.检测系统的内存映射（memory map）

是指在整个 4GB 物理地址空间中 有哪些地址范围被分配用来寻址系统的 RAM 单元

- 不同CPU有不同的内存映射，必须准确识别



Stage2具体过程

3.加载内核映像和根文件系统映像

(1) 规划内存占用的布局

内核映像所占用的内存范围；根文件系统所占用的内存范围

(2)从 Flash 上拷贝内核映像和根文件系统

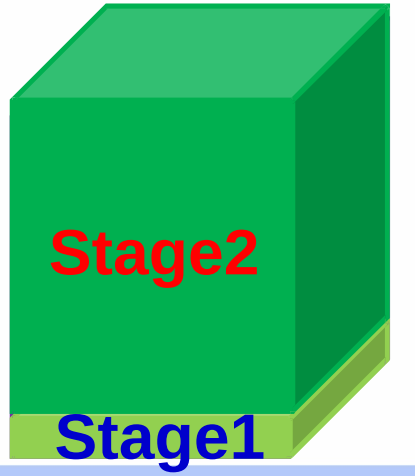
4.设置内核的启动参数

将内核映像和根文件系统映像拷贝到 RAM 空 间中后，就可以准备启动 Linux 内核了

- 在调用内核之前，需要设置 Linux 内核的启动参数

5.调用内核

- 直接跳转到内核的第一条指令处



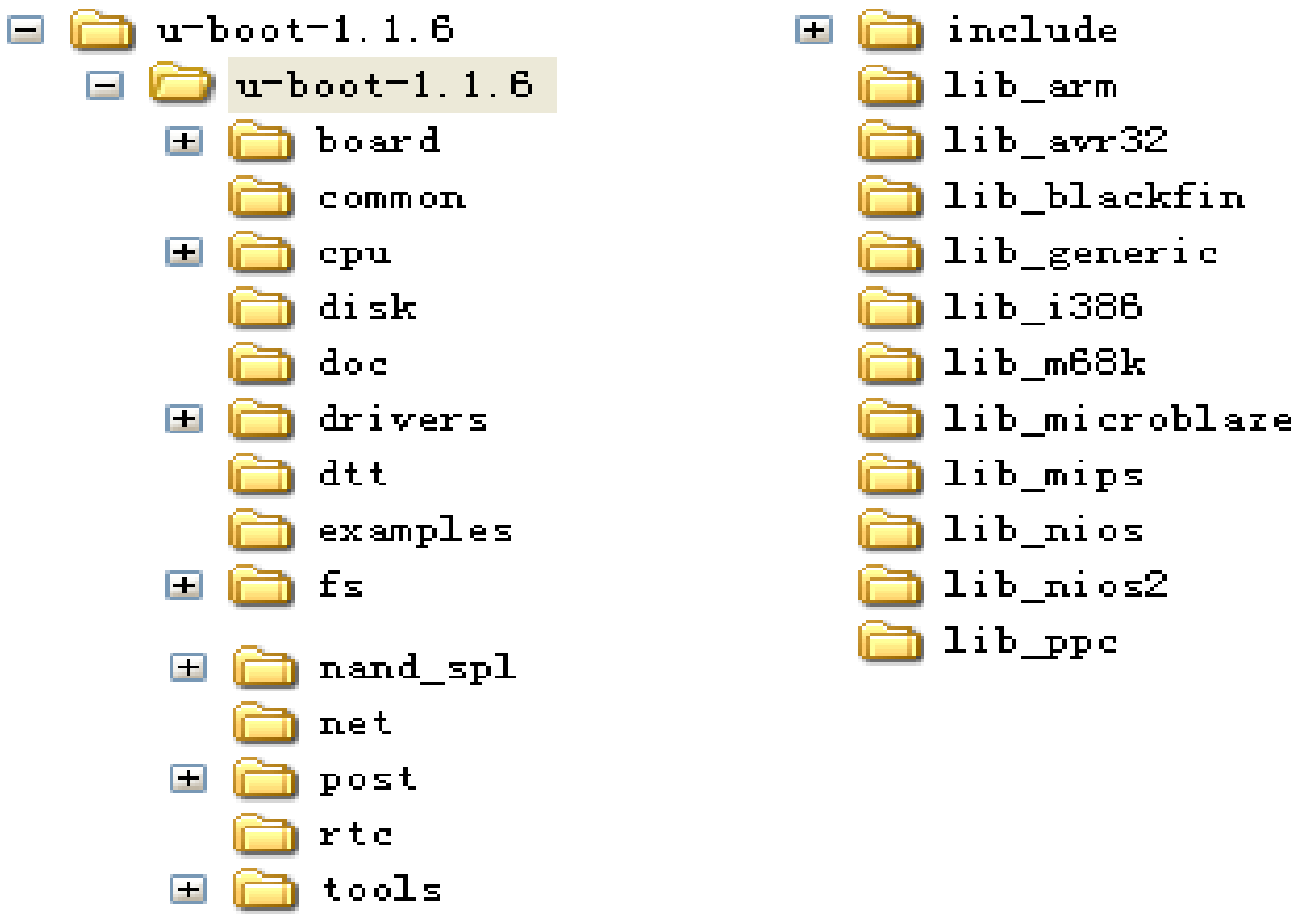
德国DENX软件工程中心U-boot

- ✓ **系统引导：支持**NFS**挂载、**RAMDISK(**压缩或 非压缩)形式的根文件系统。**
- ✓ **支持目标板环境参数多种存储方式，**FLASH、NVRAM、EEPROM
- ✓ **设备驱动:串口、**SDRAM、FLASH、**以太网、**LCD、NVRAM、EEPROM、**键盘、**USB、PCMCIA、PCI、RTC**等驱动支持。**
- ✓ **上电自检功能：**SDRAM、FLASH**大小自动检 测；**SDRAM**故障检测；**CPU**型号。**

U-Boot顶层目录下有多个子目录，分别存放和管理不同的源程序，这些目录中所存放的文件有其规则，可分为3类

- ✓ 第1类目录与处理器体系结构或者开发板硬件直接相关；
- ✓ 第2类目录是一些通用的函数或者驱动程序；
- ✓ 第3类目录是U-Boot的应用程序、工具或者文 档。

常用Bootloader简介



常用Bootloader简介

Bootloader除了u-boot，还有redboot，lilo等。Vivi 是韩国mizi公司专门为三星s3c2440芯片设计的Bootloader。

Vivi也可以分为2个阶段，阶段1的代码在arch/s3c2440/head.S中，阶段2的代码从init/main.c的main函数开始。

```
vivi-+-arch-+-s3c2440
|-Documentation
|-drivers-+-serial
|         `-mtd-+-maps
|               |-nor
|               `-nand
|-include-+-platform
|         |-mtd
|         `-proc
|-init
|-lib-+-priv_data
|-scripts-+-lxdialog
|-test
|-util
```

本章主要讲述

6.1 Bootloader过程

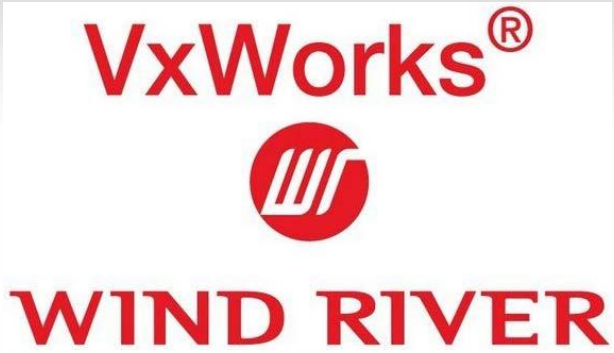
6.2 嵌入式操作系统简介

6.3 Linux终端命令

6.4 Shell编程

6.5 Linux编程基础

嵌入式操作系统

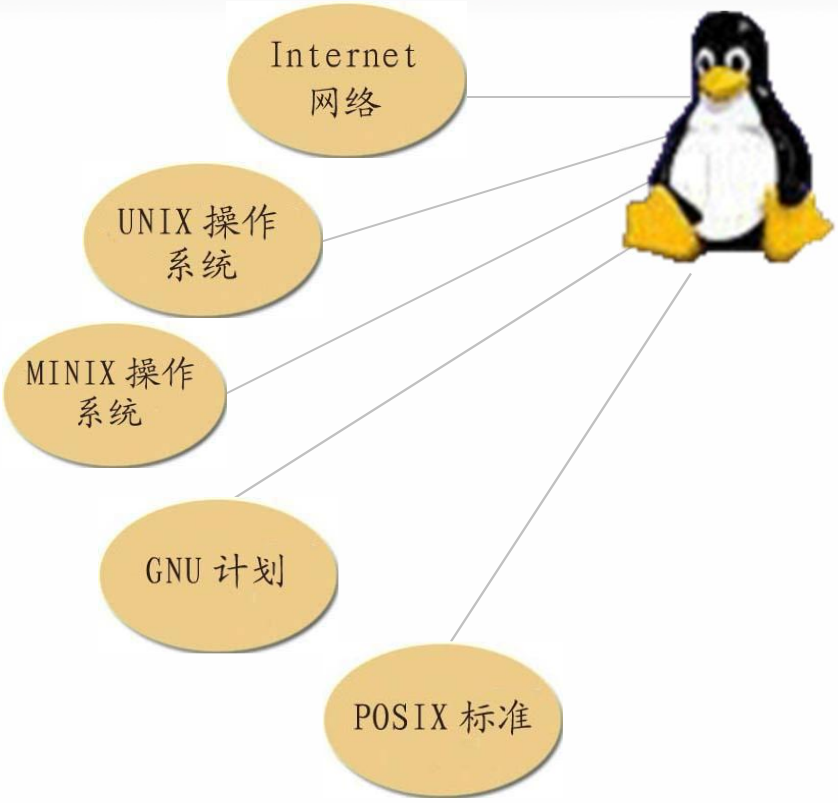


完全开源，软件资源丰富，高性能，稳定

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

Linux诞生

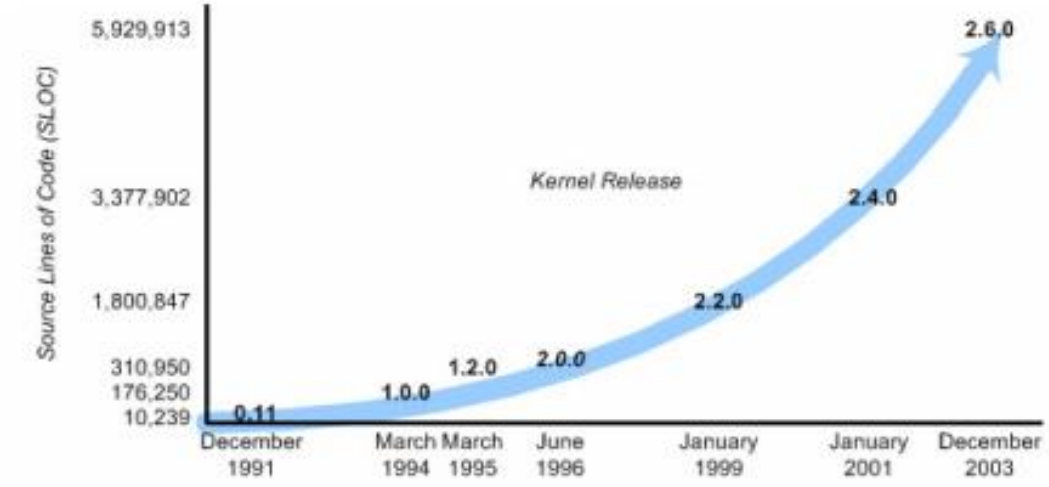


- UNIX 操作系统是美国贝尔实验室于1969年夏在DEC PDP-7 小型计算机上开发的一个分时操作系统
- MINIX 系统是由 Andrew S. Tanenbaum（AST）1987 年开发的，主要用于学生学习操作系统原理
- 如果没有 Internet 网，没有遍布全世界的无数计算机黑客通过网络的无私奉献，那么 Linux 绝对不可能发展到现在的水平。

Linux版本历史

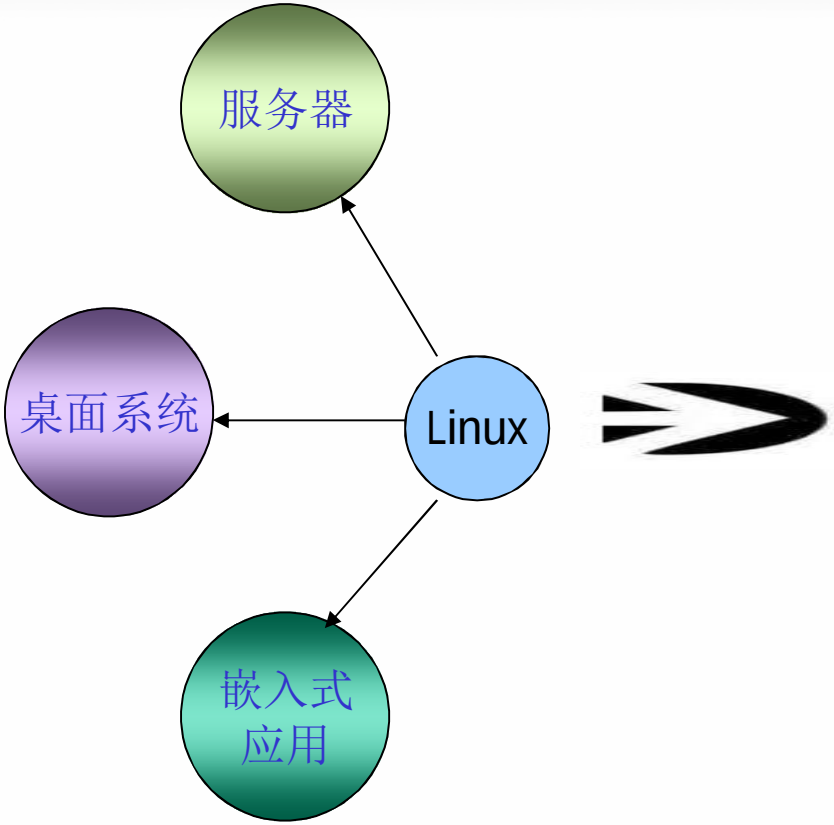
- **从 Linux 诞生开始，Linux 内核就从来没有停止过升级，从0.02 版本到 1999年具有里程碑意义的2.2 版本，一直到我们现在看到的 x.x.xx版本。**

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。**

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**

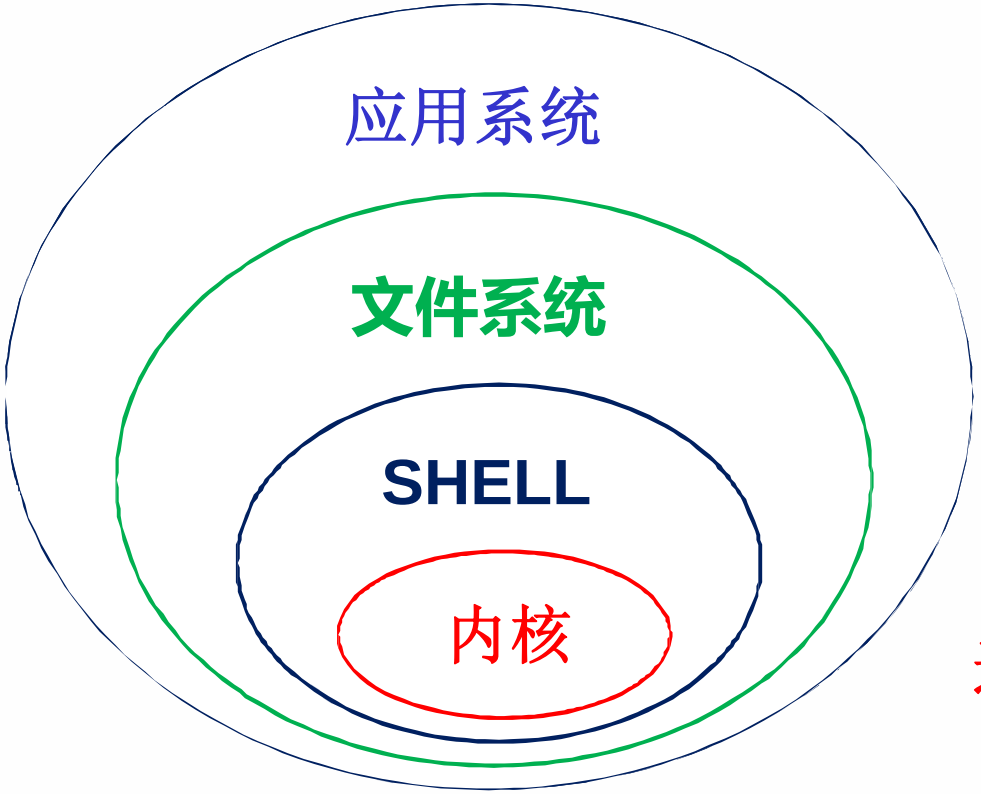
✓ 模拟出硬件设备，然后在该硬件设备基础上安装系统

✓ 安装Vmware/Virtual Box等虚拟机软件

✓ 建立虚拟机

✓ 在虚拟机下安装Linux

Linux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

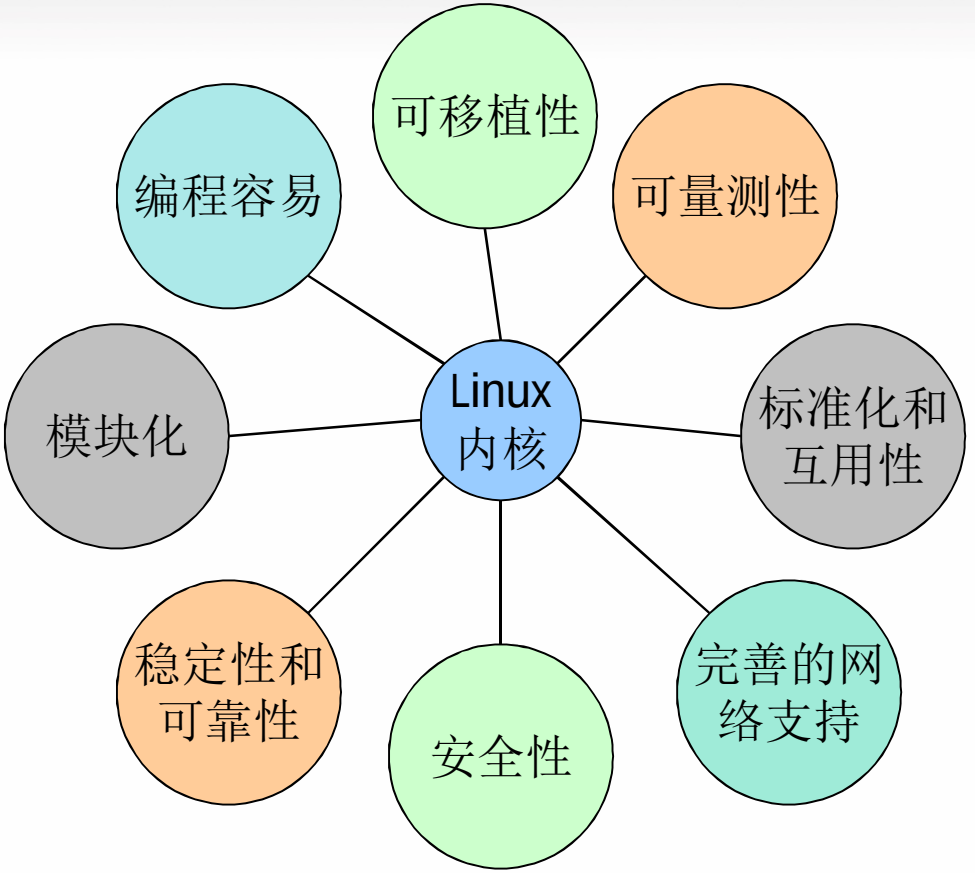
用户与内核交互的一种接口，可输入命令等

运行程序和管理硬件设备的核心

Linux内核任务与特点

■ 内核的任务

- **用于应用程序执行的流程管理。**
- **内存和I / O管理。**
- **系统调用控制——内核的核心行为。**
- **借助设备驱动程序进行设备管理。**

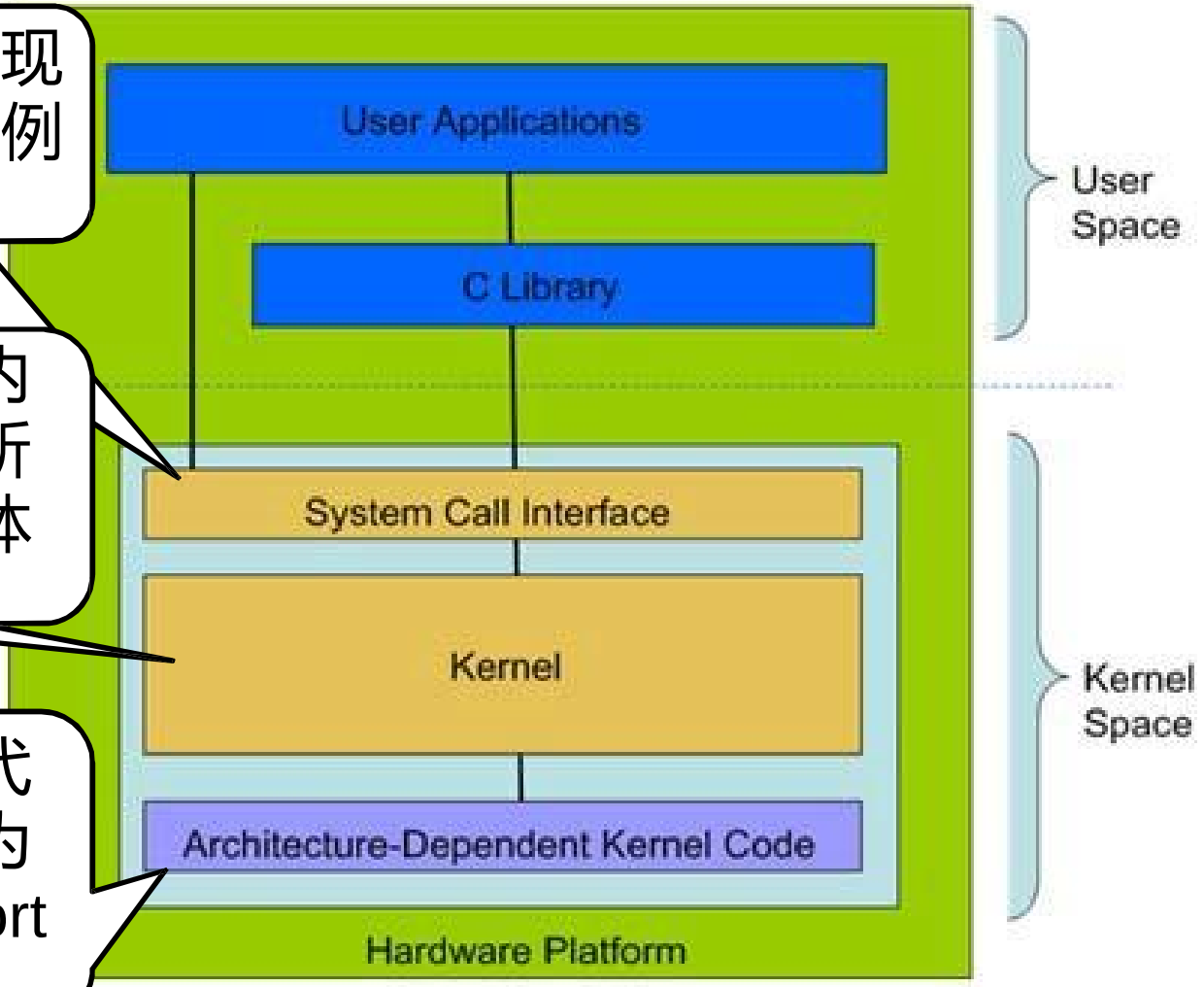


Linux内核结构

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

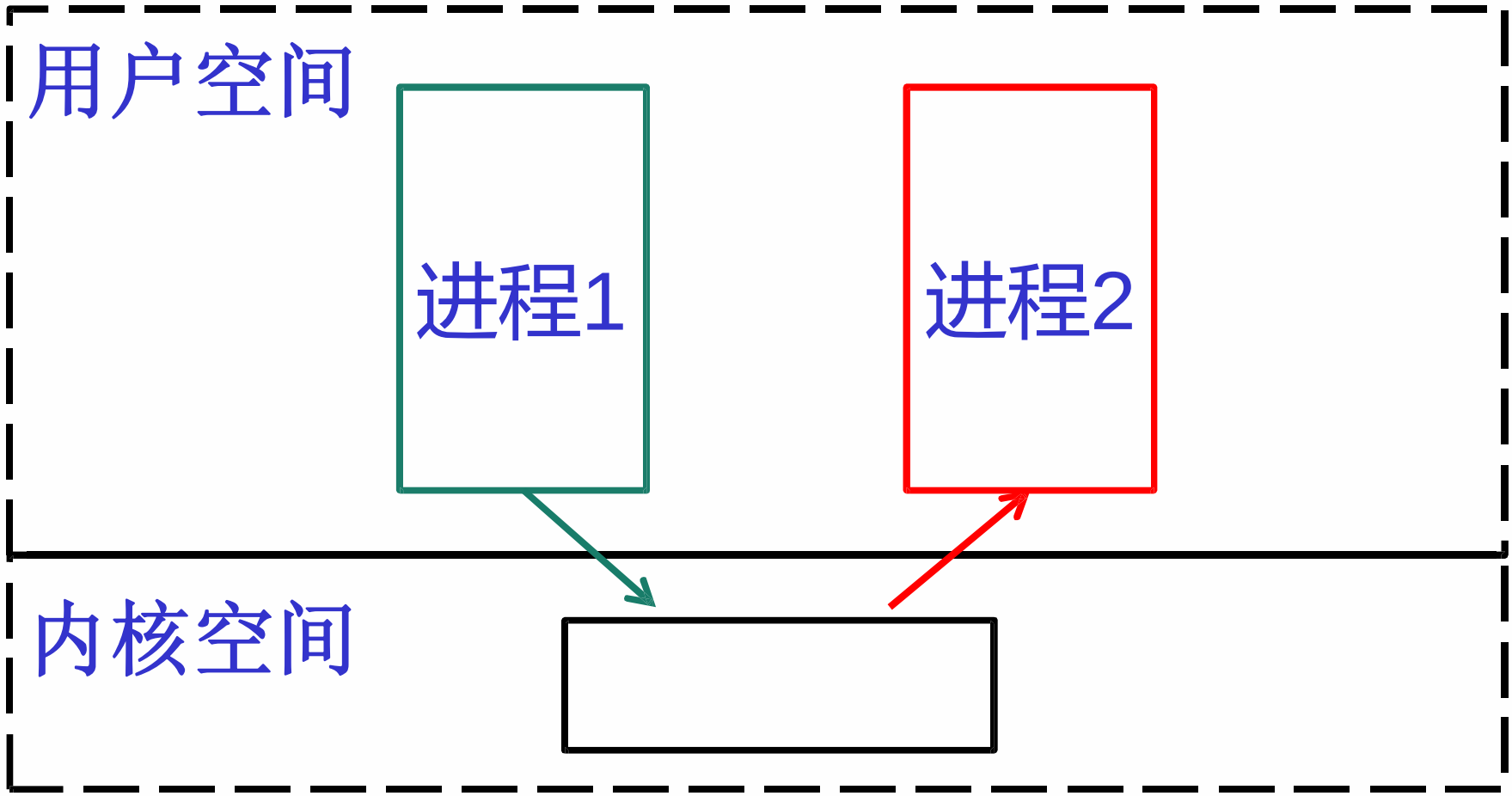
进程调度

- 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

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。

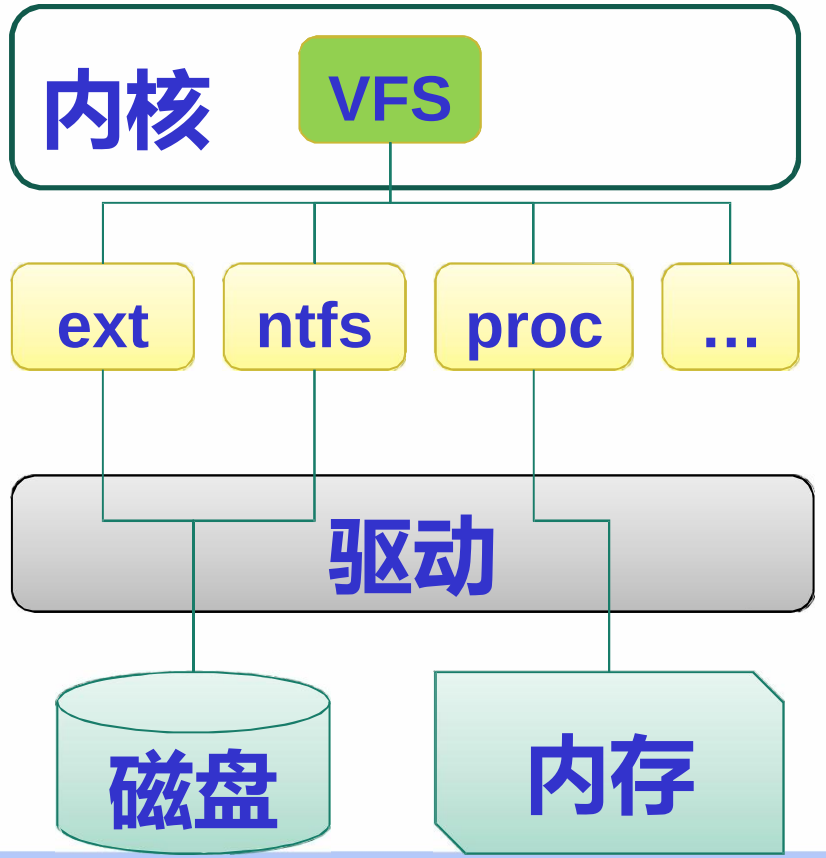
虚拟文件系统

- **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



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

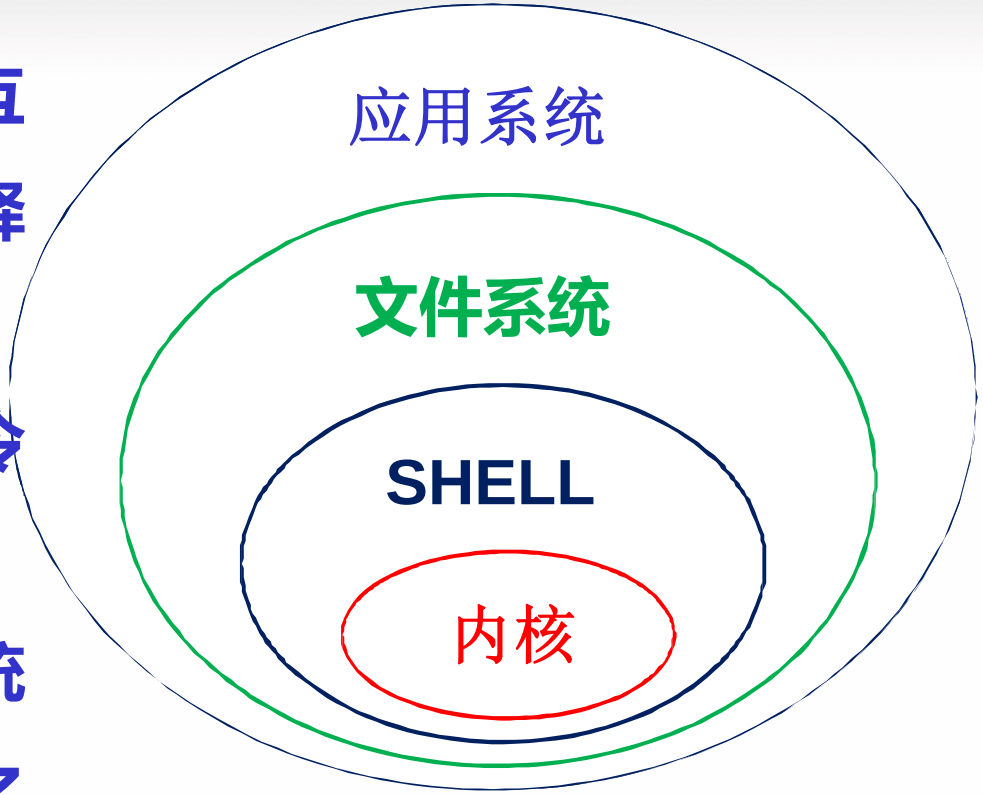
(7)内核的编译，将裁剪好的内核进行编译，生成二进制映像文件

(8) 将生成的二进制映像文件，烧写到目标平台

内核烧写

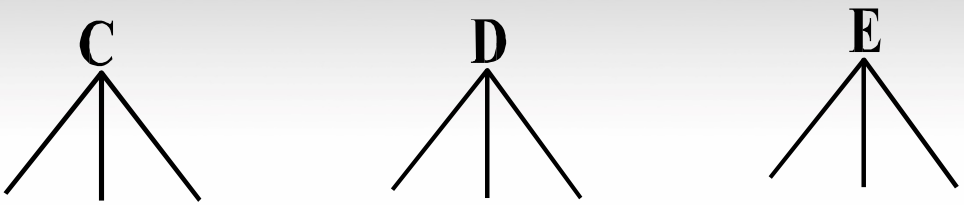
- **通过串口**
  - ✓ minicom
  - ✓ **与**bootloader**的烧写类似**
- **通过网络**
  - ✓ Uboot**下使用**tftp**烧写**
  - ✓ **通过**JTAG**接口**

- 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。

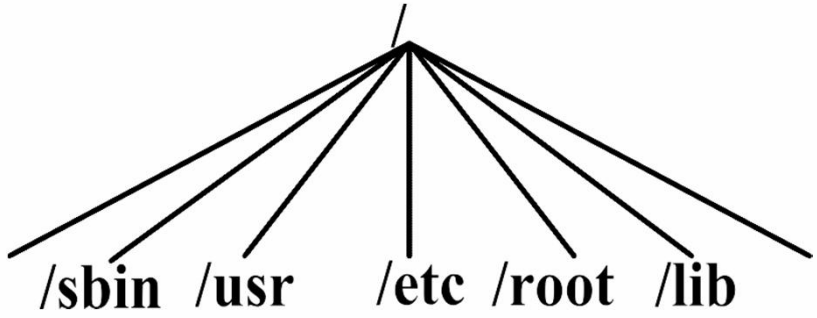


- 操作系统中负责管理和存储文件信息的软件机构称为文件管理系统，简称文件系统
- 文件系统由三部分组成：与文件管理有关的软件、 被管理的文件以及实施文件管理所需的数据结构
- **Linux**支持许多不同的文件系统

✓ 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*

本章主要讲述

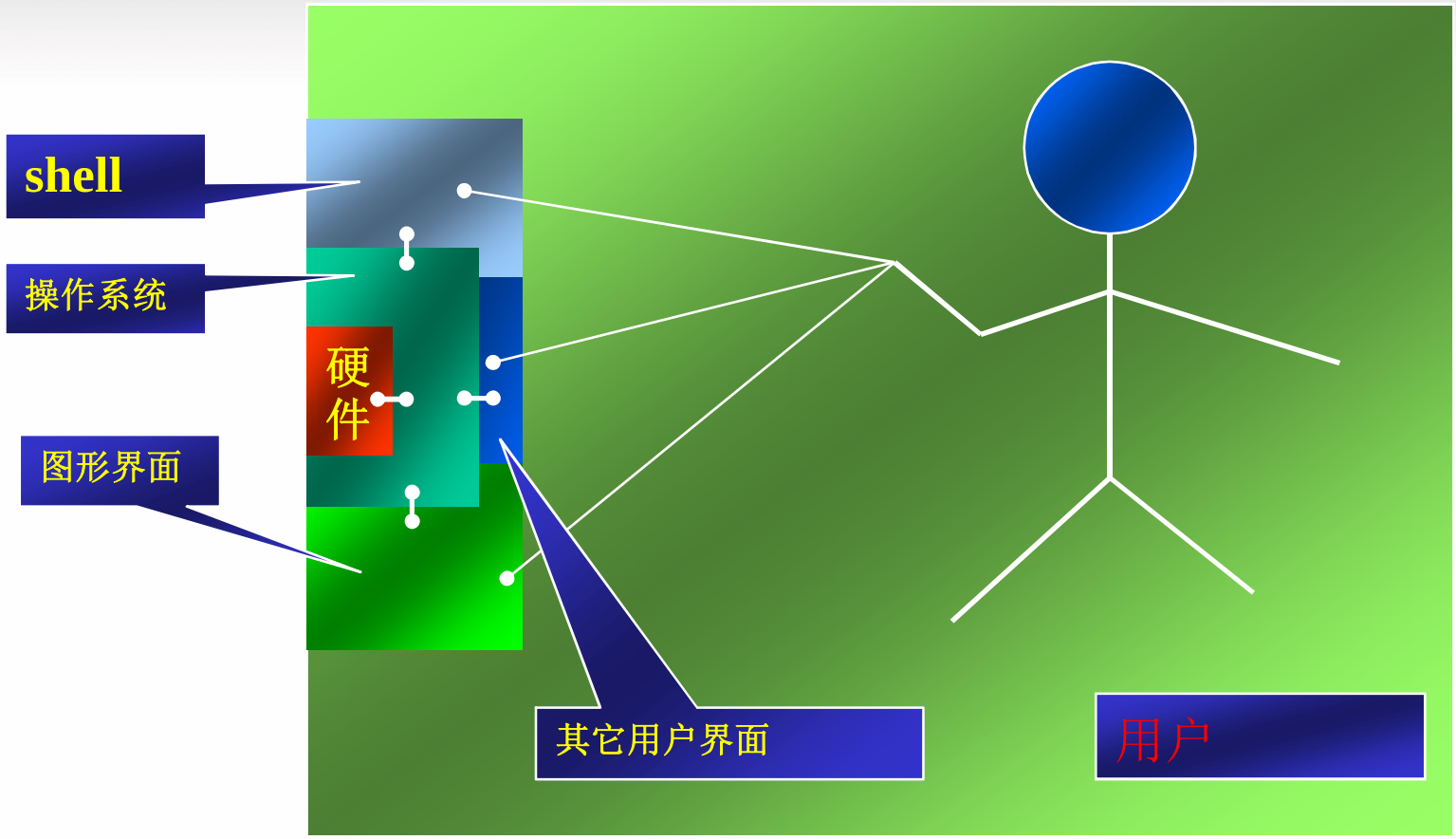
6.1 Bootloader过程

6.2 嵌入式操作系统简介

6.3 Linux终端命令

6.4 Shell编程

6.5 Linux编程基础


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 。

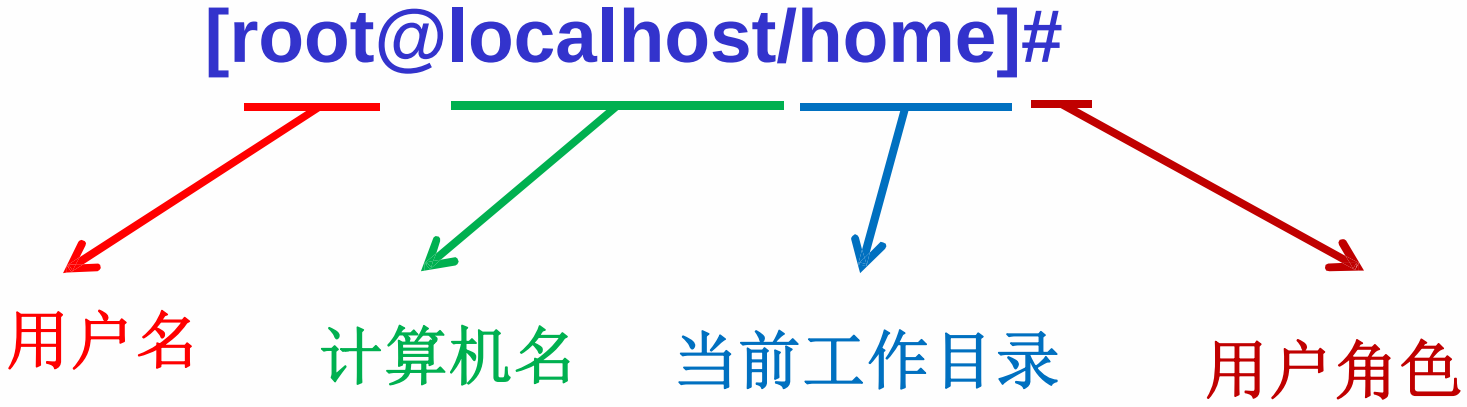
■ Shell的主要功能

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

解压缩

■ unzip

- 格式 :unzip [选项] 文件.zip
- 范例
- 1. unzip tmp.zip
- 解压tmp.zip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 Bootloader过程

6.2 嵌入式操作系统简介

6.3 Linux终端命令

6.4 Shell编程

6.5 Linux编程基础

Shell程序的特点及用途

- shell程序可以认为是将shell命令按照控制结构组织到一个文本文件中，批量的交给shell去执行
- 不同的shell解释器使用不同的shell命令语法
- shell程序解释执行，不生成可以执行的二进制文件
- 可以帮助用户完成特定的任务，提高使用、维护系统的效率
- 了解shell程序可以更好的配置和使用linux

Shell程序举例

greeting.sh		解释
1	#!/bin/bash	以 #! 开始，其后为使用的shell
2	#a Simple shell Script Example	以 # 开始，其后为程序注释
3	#a Function	同上
4	function say_hello()	以 functin 开始，定义函数
5	{	函数开始
6	echo "Enter Your Name,Please. :"	echo命令输出字符串
7	read name	读入用户的输入到变量name
8	echo "Hello $name"	输出
9	}	函数结束
10	echo "Programme Starts Here....."	程序开始的第一条命令，输出提示信息
11	say_hello	调用函数
12	echo "Programme Ends."	输出提示，提示程序结束

程序编译和运行过程

- 一般步骤：
  - 编辑文件
  - 保存文件
  - 将文件赋予可以执行的权限
  - 运行及排错
- 常用到的命令：
  - vi，编辑、保存文件
  - ls -l 查看文件权限
  - chmod 改变程序执行权限
  - 直接键入文件名运行文件

Shell程序过程


```
[tom@localhost ~]$ ll
总用量 20
-rw-r--r--  1 root root    0
drwxr-xr-x  2 tom  tom  4096
-rw-rw-r--  1 tom  tom   210  6月 29 22:58 greeting.sh
[tom@localhost ~]$ chmod +x greeting.sh
[tom@localhost ~]$ ll
总用量 20
-rw-r--r--  1 root root    0  6月 20 19:32 abc.txt
drwxr-xr-x  2 tom  tom  4096  6月 19 01:23 Desktop
-rwxrwxr-x  1 tom  tom   210  6月 29 22:58 greeting.sh
[tom@localhost ~]$ ./greeting.sh
Programme Starts Here.....
Enter Your Name,Please.   :
tom
Hello tom
Programme Ends.
[tom@localhost ~]$
```

查看权限

查看权限，初始状态无执行（x）权限

增加可执行（x）的权限

查看权限，已经具备执行（x）权限

运行程序

程序运行过程输出

一般结构

- shell类型
- 函数
- 主过程

```
#!/bin/bash
function fun1(){
}
......
funciton funn(){
}
..........
.
.
.
.
.
.
```

shell类型

函数定义

主过程

- Shell编程中，使用变量无需事先声明
- 变量的赋值与引用：
  - ✓ 赋值：变量名=变量值（注意：不能留空格）
  - ✓ 引用：$var引用var变量
- Shell变量有几种类型
  - ✓ 用户自定义变量
  - ✓ 环境变量
  - ✓ 位置参数变量
  - ✓ 专用参数变量

用户自定义变量

- 由用户自己定义、修改和使用，Shell的默认赋值是字符串赋值

```
var=1
var=$var+1
echo  $var              #打印的结果是什么呢?
```

- 为了达到想要的效果有以下几种表达方式
  - ✓ let "var+=1"
  - ✓ var=$[$var+1]
  - ✓ var=`expr $var + 1`#注意加号两边的空格

变量的引用

- **格式:**

  $变量名，或者${变量名}

  变量名为一个字符用方式一，变量名多于一个字符建议用第2种方式

- **例子:**

```
a=1
abc="hello"
echo $a
echo ${abc}
```

用户自定义变量

- $ aa=Hello
- $echo $aa
- Hello

- $ aa= “ Yes Sir ”
- $echo $aa
- Yes Sir

- $ aa=7+5
- $echo $aa
- 7+5

注：等号两边不能有空格，如果字符串两边有空格，必须加用引号

Shell常用内置命令

- echo：在屏幕上显示出由arg指定的字串
  - 命令格式：**echo arg**
- export
  - 命令格式：**export变量[=变量值]**
  - **Shell可以用export把它的变量向下带入子 Shell，从而让子进程继承父进程中的环境变量**
  - **不带任何变量名的export语句将显示出当前所 有的export变量**

- read：从键盘输入内容为变量赋值
  - **read [-p "信息"] [var1 var2 ...]**
  - 若省略变量名，则将输入的内容存入**REPLY**变量
- readonly：不能被清除或重新赋值的变量
  - **readonly variable**

■ 双引号

- 双引号内的字符，除$、`和\仍保留其特殊功能外，其余字符均作为普通字符对待
- $表示变量替换
- 倒引号表示命令替换
- \为转义字符

- echo “Dir is `pwd` and logname is $LOGNAME”
- names="Zhangsan Lisi Wangwu"

■ 单引号

由单引号括起来的字符都作为普通字符出现， 即使是$、`和\

```
echo      'The time is `date` , the file is  $HOME/abc '
The time is ` date ` ,  the file is $HOME/abc
```

■ 倒引号

倒引号括起来的字符串被shell解释为命令行， 在执行时，Shell会先执行该命令行，并以它的 标准输出结果取代整个倒引号部分。在前面示例中已经见过。

- echo “Dir is `pwd` and logname is $LOGNAME”
- names="Zhangsan Lisi Wangwu"

- $[]：可以接受不同基数的数字的表达式

  echo $[10+1] （输出：11）

  echo “$[2+3],$HOME” （输出：5,/root）

  echo $[2<<3],$[8>>1] （输出：16,4）

  echo $[2>3],$[3>2] （输出：0,1 表达式为false时输出0，为true时输出1）

- 字符表达式：直接书写，采用单引号，双引号引起来。

  echo " $HOME，That is your root directory." （输出：/root，That is your root directory.）

  echo ' $HOME，That is your root directory. ' （输出：$HOME，That is your root directory.）

  单引号和双引号的区别在于：单引号是原样显示，双引号则显示出变量的值。

控制结构

根据某个条件的判断结果，改变程序执行的路径。可以简单的将控制结构分为分支和循环两种 。

- 常见分支结构：
  - if
  - case
- 常见循环结构：
  - for
  - while
  - until

- **if分支**
  - 格式：

```
if 条件1
then
命令
[elif 条件2
    then
命令]
[else
命令]
fi
```

  - 说明：
    - 中括号中的部分可省略；
    - 当条件为真（0）时执行then后面的语句，否则执行else后面的语句；
    - 以fi作为if结构的结束。

- if分支

- #!/bin/bash

  #if.sh

  if ["10" -lt "12"] #注意：if和[之间，[和"10"之间，"12"和]都有空格，如果不加空格，会出现语法错误

  then

  echo "Yes,10 is less than 12"

  fi

- **case分支**
  - 格式：

```
case 条件 in
模式1)
        命令1
        ; ;
[模式2）
        命令2
        ; ;
..............
模式n）
        命令n
        ; ; ]
esac
```

  - 说明：
    - “条件”可以是变量、表达式、shell命令等；
    - “模式”为条件的值，并且一个“模式”可以匹配多种值，不同值之间用竖线（|）联结；
    - 一个模式要用双分号（；；）作为结束；
    - 以逆序的case命令（esac）表示case分支语句的结束

- **#!/bin/bash**
  #case.sh
  echo -n "Enter a start or stop:"
  read ANS
  case $ANS in
  start)
  echo "You select start"
  ;;
  stop)
  echo "You select stop"
  ;;
  ***)**
  echo "`basename $0`: You select is not between start and stop" >&2
  #注意：>和&2之间没有空格,>&2 表示将显示输出到标准输出（一般是屏幕）上
  exit;
  ;;
  esac

- **for循环**
- 格式

```
for 变量 [in 列表]
do
        命令（通常用到循环变量）
done
```

– 说明：

- “列表”为存储了一系列值的列表，随着循环的进行，变量从列表中的第一个值依次取到最后一个值；
- do和done之间的命令通常为根据变量进行处理的一系列命令，这些命令每次循环都执行一次；
- 如果中括号中的部分省略掉，Bash则认为是“in $@”，即执行该程序时通过命令行传给程序的所有参数的列表。

```
#!/bin/sh

for foo in bar fud 43
do
  echo $foo
done
exit 0
```

That results in the following output:

```
bar
fud
43
```

- **while循环与until循环**
  - 格式：

```
while/until   条件
do
        命令
done
```

- 说明：
  - while循环中，只要条件为真，就执行do和done之间的循环命令；
  - until循环中，只要条件不为真，就执行do和done之间的循环命令，或者说，在until循环中，一直执行do和done之间的循环命令，直到条件为真；
  - 避免生成死循环。

```
#!/bin/sh

echo "Enter password"
read trythis

while [ "$trythis" != "secret" ]; do
  echo "Sorry, try again"
  read trythis
done
exit 0
```

An example of the output from this script is as follows:

```
Enter password
password
Sorry, try again
secret
$
```

- 函数

- 格式：
  - 定义：

```
[function] 函数名（）
{
        命令
}
```

  - 引用：

```
函数名 [ 参数1 参数2 ...参数n ]
```

- 说明：
  - 中括号中的部分可以省略；
  - 如果在函数内部需要使用传递给函数的参数，一般用$0、$1、.....、$n，以及$#、$*、$@这些特殊变量：
    - $0为执行脚本的文件名；
    - $1是传递给函数的第1个参数；
    - $#为传递给函数的参数个数；
    - $*和$@为传递给函数的所有参数

```
#!/bin/sh

foo() {
    echo "Function foo is executing"
}

echo "script starting"
foo
echo "script ended"

exit 0
```

Running the script will output the following:

```
script starting
Function foo is executing
script ending
```

本章主要讲述

6.1 Bootloader过程

6.2 嵌入式操作系统简介

6.3 Linux终端命令

6.4 Shell编程

6.5 Linux编程基础

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

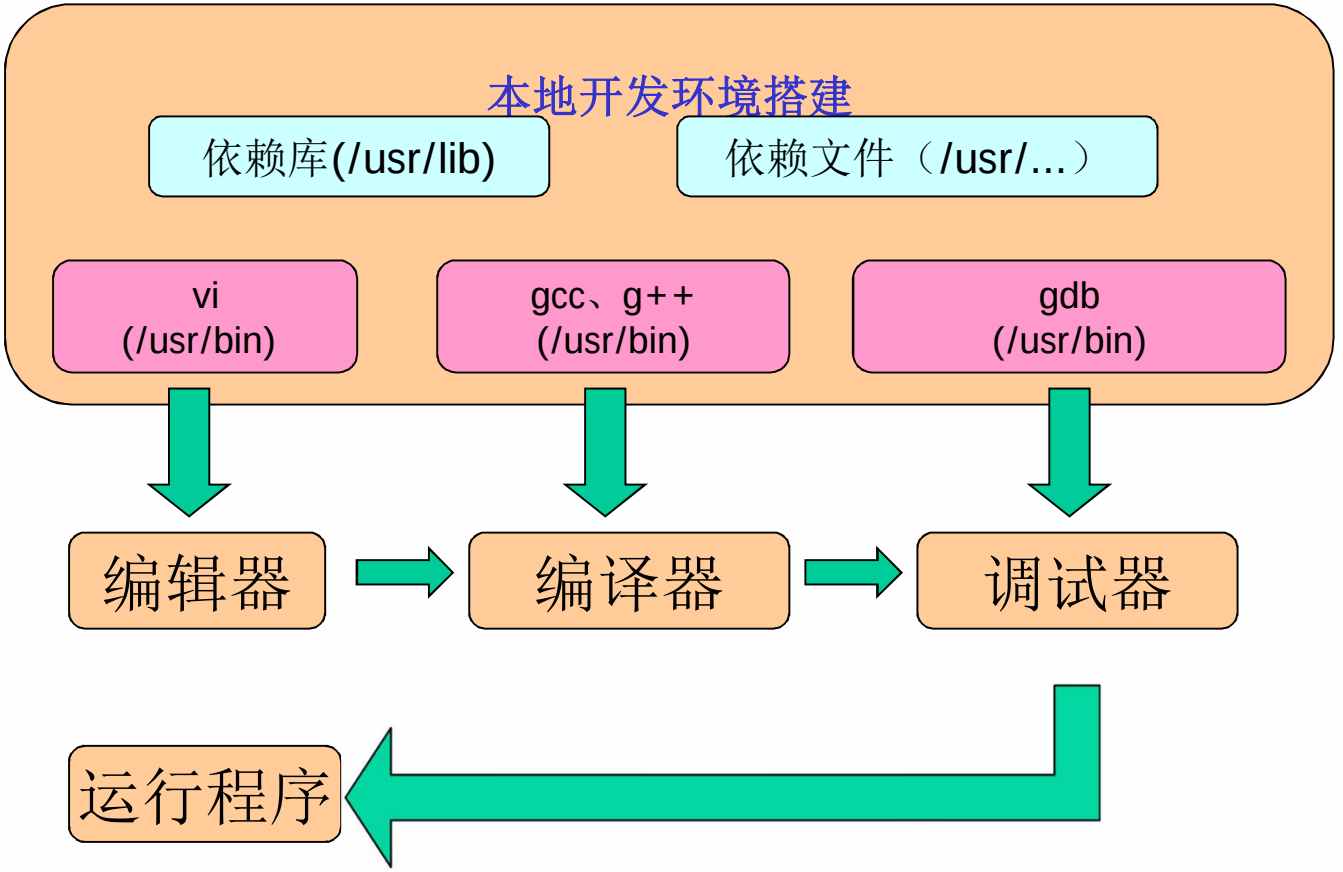
gcc debug.c -o debug -g

3、程序运行

./debug

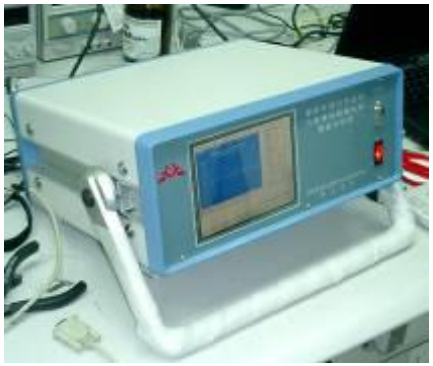
4、程序调试

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

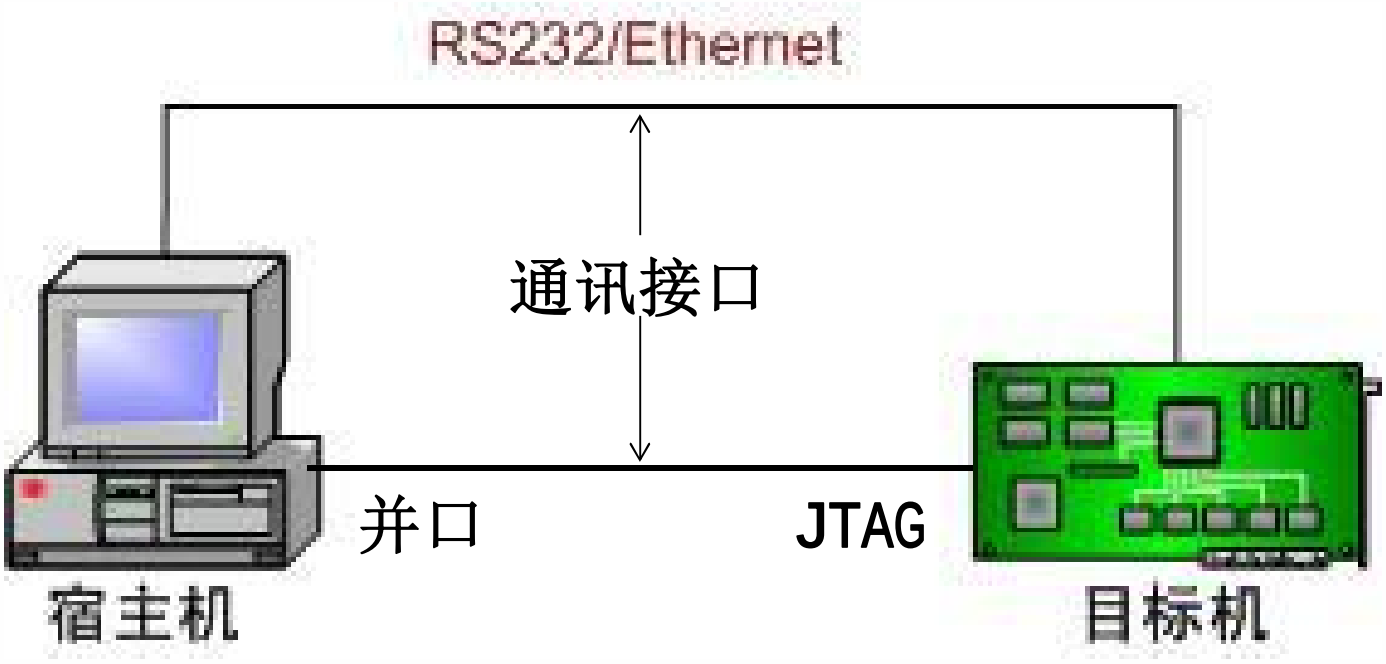
嵌入式系统不具备自举开发能力

- 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

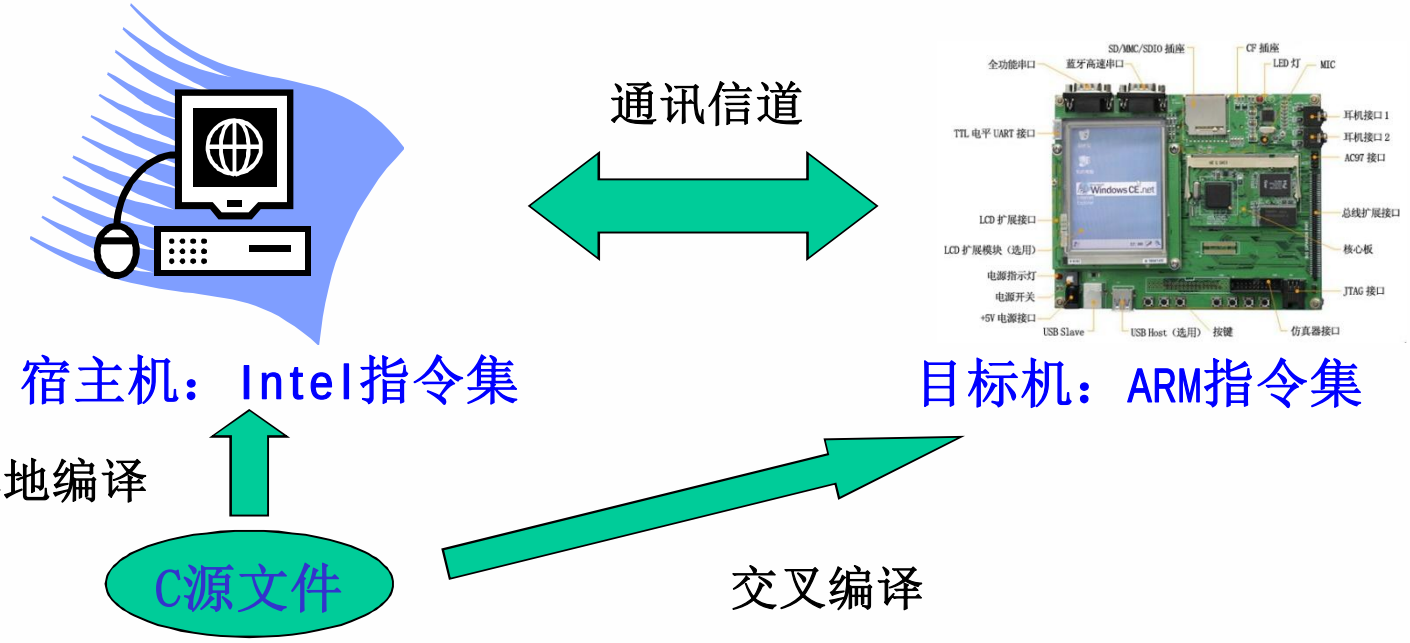
目标机：资源受限

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。

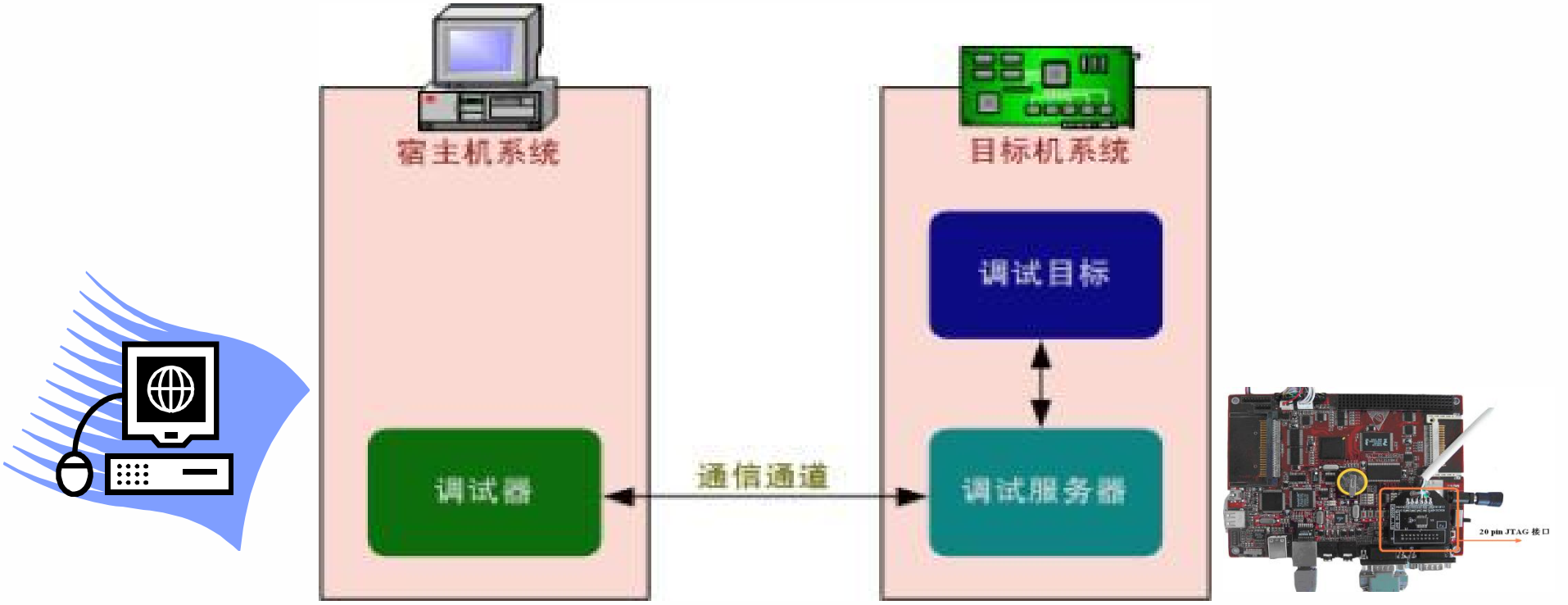
交叉编译与本地编译

- 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

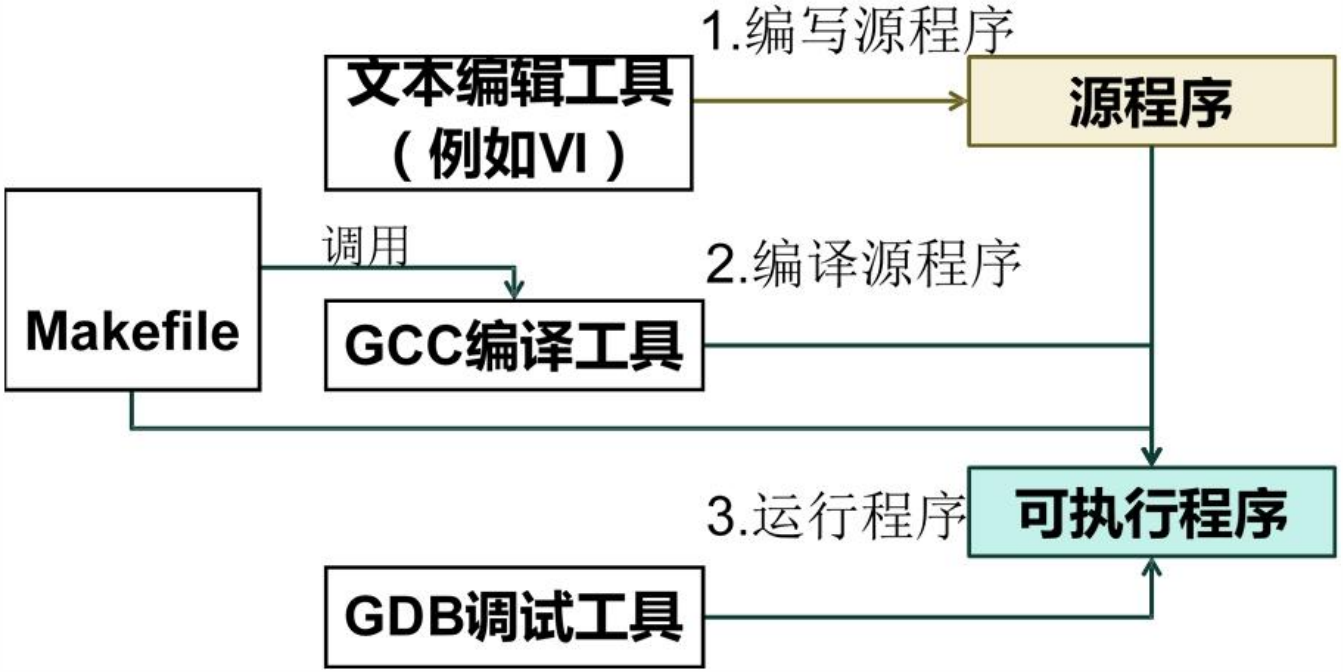


- 基于ARM体系结构的gcc交叉开发环境中，arm-linux-gcc是交叉编译器，arm-linux-ld是交叉链接器

- 一般而言，嵌入式软件需要交叉调试。



- 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

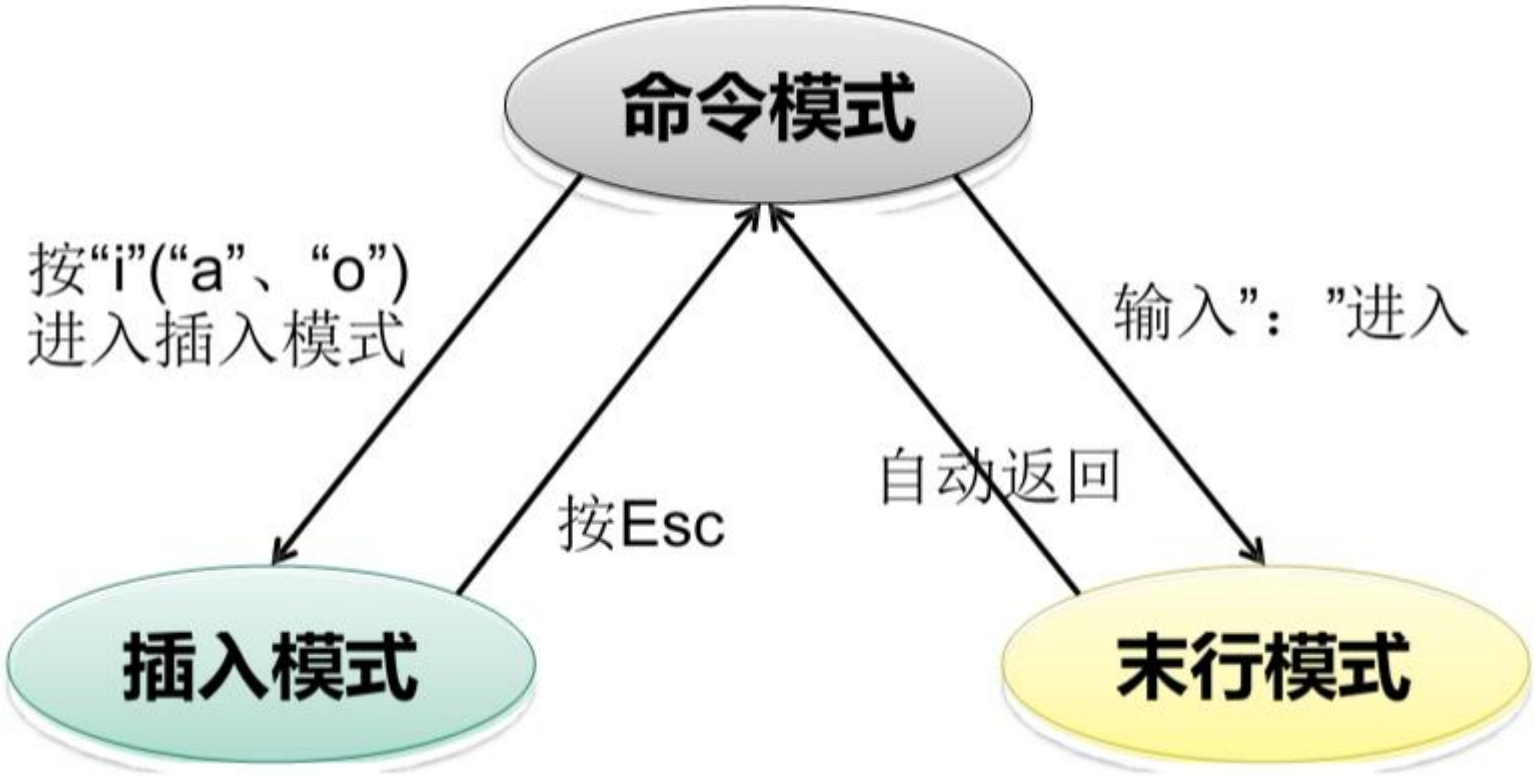
[root@localhost chap2]# gcc –o hello hello.c

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

- 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

学院 荣誉 责任

greeting.h

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

greeting.c

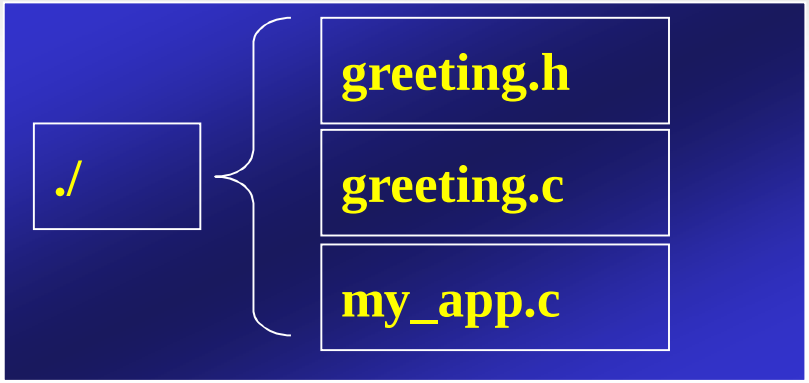
```
#include <stdio.h>
#include "greeting.h"
void greeting (char * name)
{
printf("Hello %s!\r\n",name);
}
```

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;
}
```

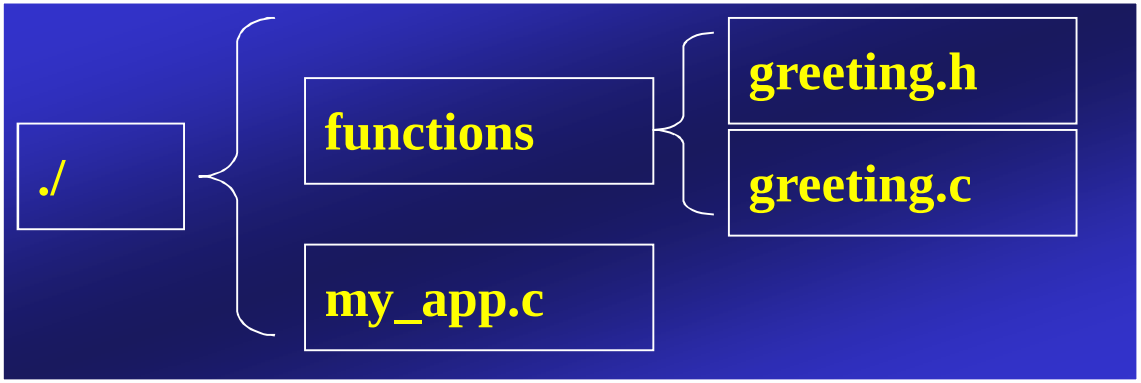
编译多个文件

- 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

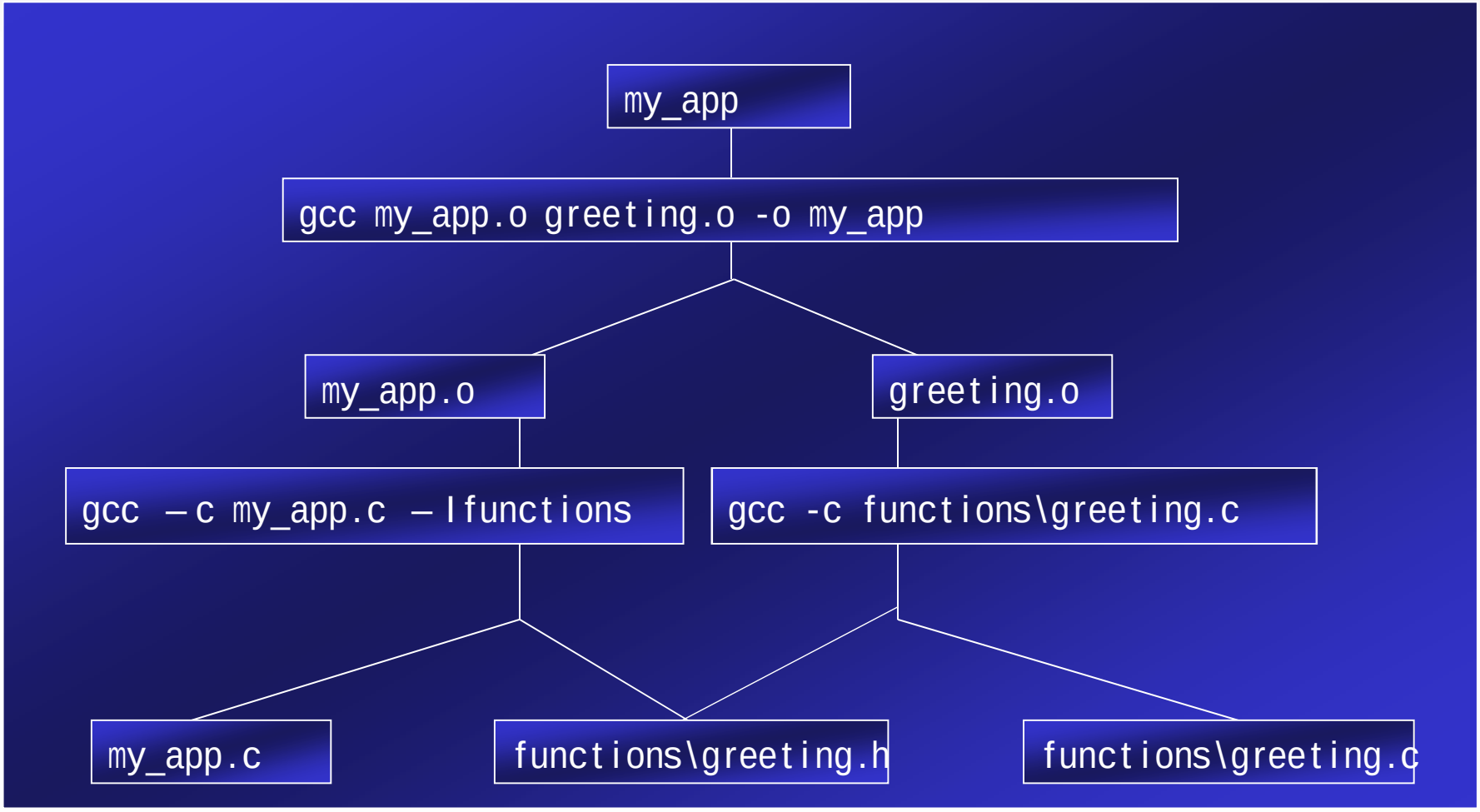
- 基本格式：
- 目标：欲生成的目标文件
- 依赖项：生成目标需要的文件
- 原理：
  - 判断依赖项是否为最新，否则，生成新的目标
- **make工具的使用格式：**
  - **make [[命令选项] [命令参数]]**
  - 通常使用**make**就可以了，**make**会寻找**Makefile**作为编译指导文件。

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**

谢谢各位同学！

Q&A