



# 嵌入式系统设计

## 第6课 嵌入式软件开发环境和调试技术

王总辉

zhwang@zju.edu.cn

<http://course.zju.edu.cn/>

---

# 提 纲



- 启动和初始化
- Linux常用命令和工具
- 宿主机-目标机开发模式
- 宿主机-目标机开发环境
- 远程调试原理
- GDB调试
- 内核调试



# 嵌入式软件开发环境构建

---

- 启动和初始化
- Linux常用命令和工具
- 宿主机-目标机开发模式
- 宿主机-目标机开发环境



# 内核启动和初始化进程 (X86 Linux为例)

## □ 开机自检

## □ 核心引导阶段

- bootsect阶段
- setup阶段
- head. S阶段
- main. c阶段

## □ init 进程和 inittab 引导脚本

- 确定用户登录模式
- 执行内容/etc/rc.d/rc. sysinit
- 启动内核的外挂模块及各运行级的脚本

# 开机自检

- 在刚开机时，根据X86CUP的特性，代码段寄存器的值为全1，指令计数器的值为全0，即CS=FFFF、IP=0000。
- 这时CPU根据CS和IP 的值执行FFFF0H处的指令，FFFF0H处的指令一般总是一个JMP指令，这个地址通常是ROM BIOS 的入口地址。
- 接着，ROM BIOS 进行开机自检，如检查内存，键盘等。在自检过程中，ROM BIOS会在上位内存中进行扫描，看看是否存在合法的设备控制卡ROM BIOS（如:SCSI卡上的ROM），如果有，就执行其中的一些初始化代码。
- 最后，ROM BIOS 读取磁盘上的第一个扇区并将这个扇区的内存装入内存。

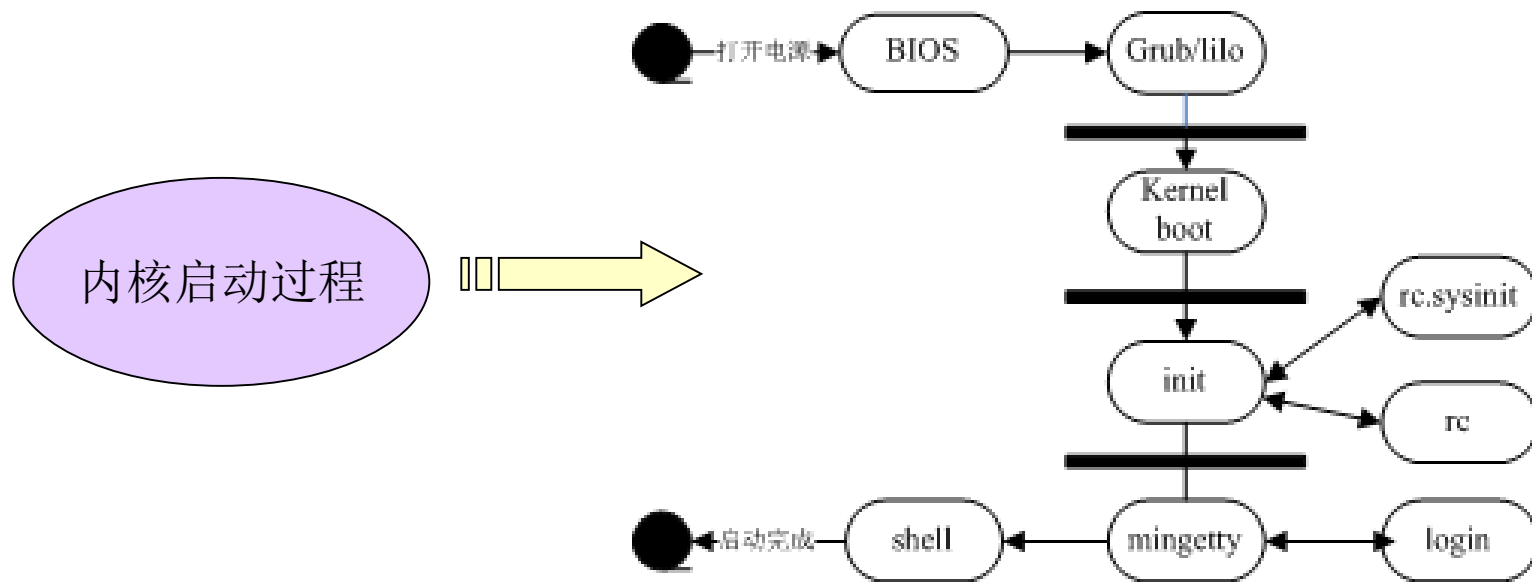


# 核心引导阶段

- 在Grub或者Lilo等引导程序成功完成引导Linux系统的任务之后，Linux就可从它们手中接管了CPU的控制权。在这个过程中主要用到该目录下的这几个文件：bootsect. S、setup. S以及compressed目录下head. S等。
- Linux的内核通常是压缩过后的，包括如上述提到的那几个重要的汇编程序，它们都是在压缩内核vmlinuz中的。因为Linux中提供的内核包含了众多驱动和功能，因而比较大，所以采用压缩内核可以节省大量的空间。

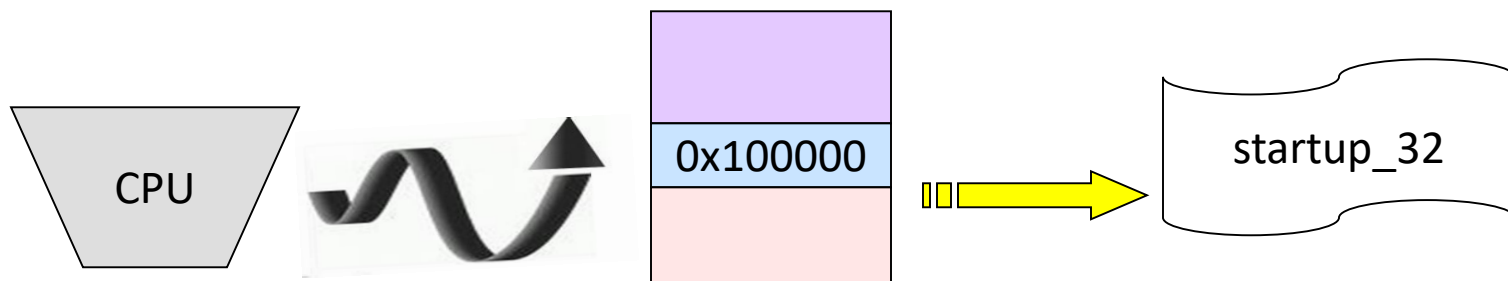
# 核心引导阶段—bootsect阶段

- 当grub读入vmlinuz后，会根据bootsect，把它自身和setup程序段读到不大于0x90000开始的内存里，然后grub会跳过bootsect那512bytes的程序段，直接运行setup里的第一条指令。就是说bzImage里bootsect的程序没有再被执行了，而bootsect.S在完成了指令搬移之后就退出了。之后执行权就转到了setup.S的程序中。



# 核心引导阶段—setup阶段

- setup. S的主要功能就是利用ROM BIOS中断读取机器系统数据，并将系统参数保存到0x90000~0x901FF开始的内存中位置。
- 此外，setup. S还将video. S中的代码包含进来，检测和设置显示器和显示模式。
- 最后，它还会设置CPU的控制寄存器CR0，从而进入32位保护模式运行，并跳转到绝对地址为0x100000，同时执行startup\_32。







# 核心引导阶段—head.S阶段

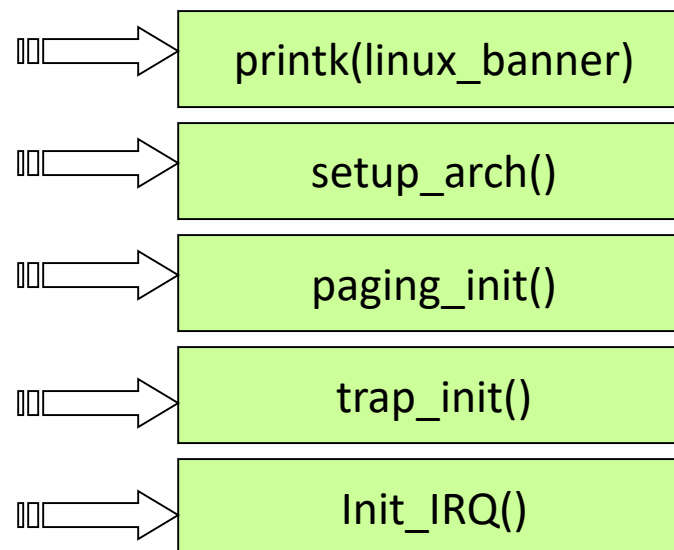
- 当运行到head.S时，系统已经运行在保护模式，而head.S完成了一个重要任务就是将内核解压。就如本节前面提到的，内核是通过压缩的方式放在内存中的，head.S通过调用misc.c中定义的decompress\_kernel()函数，将内核vmlinuz解压到0X100000的。
- 接下来head.S程序完成寄存器、分页表的初始化工作，但要注意的，这个head.S程序与完成解压缩工作的head.S程序是不同的，它在源代码中的位置是arch/i386/kernel/head.S程序与完成解压缩工作的head.S程序是不同的。
- 在完成初始化之后，head.S就跳转到start\_kernel()函数中去。



# 核心引导阶段—main.c阶段

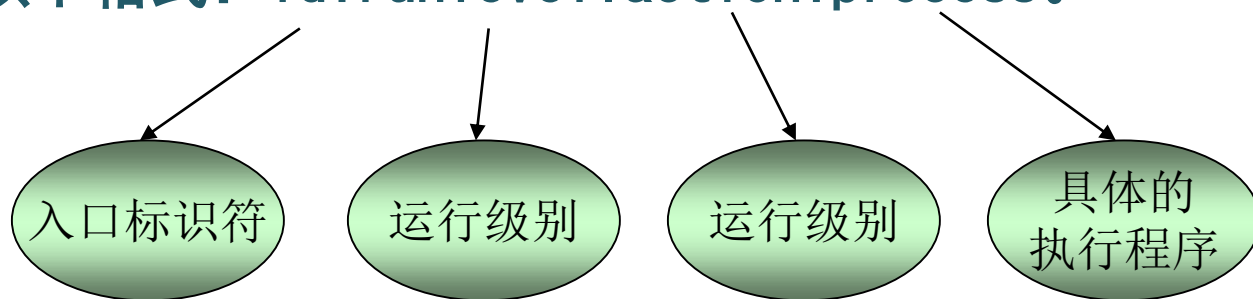
□ `start_kernel()` 是“`init/main.c`”中定义的函数，`start_kernel()` 调用了一系列初始化函数，进行内核的初始化工作。要注意的是，在初始化之前系统中断是被屏蔽的，另外内核也处于被锁定状态，以保证只有一个CPU用于Linux系统的启动。在`start_kernel()`的最后，调用了`init()`函数

- ◆ 输出 Linux 版本信息
- ◆ 设置与体系结构相关的环境
- ◆ 页表结构初始化
- ◆ 使用“`arch/alpha/kernel/entry.S`”中的入口点设置系统自陷入口
- ◆ 使用 `alpha_mv` 结构和 `entry.S` 入口初始化系统 IRQ



# init 进程和 inittab 引导脚本

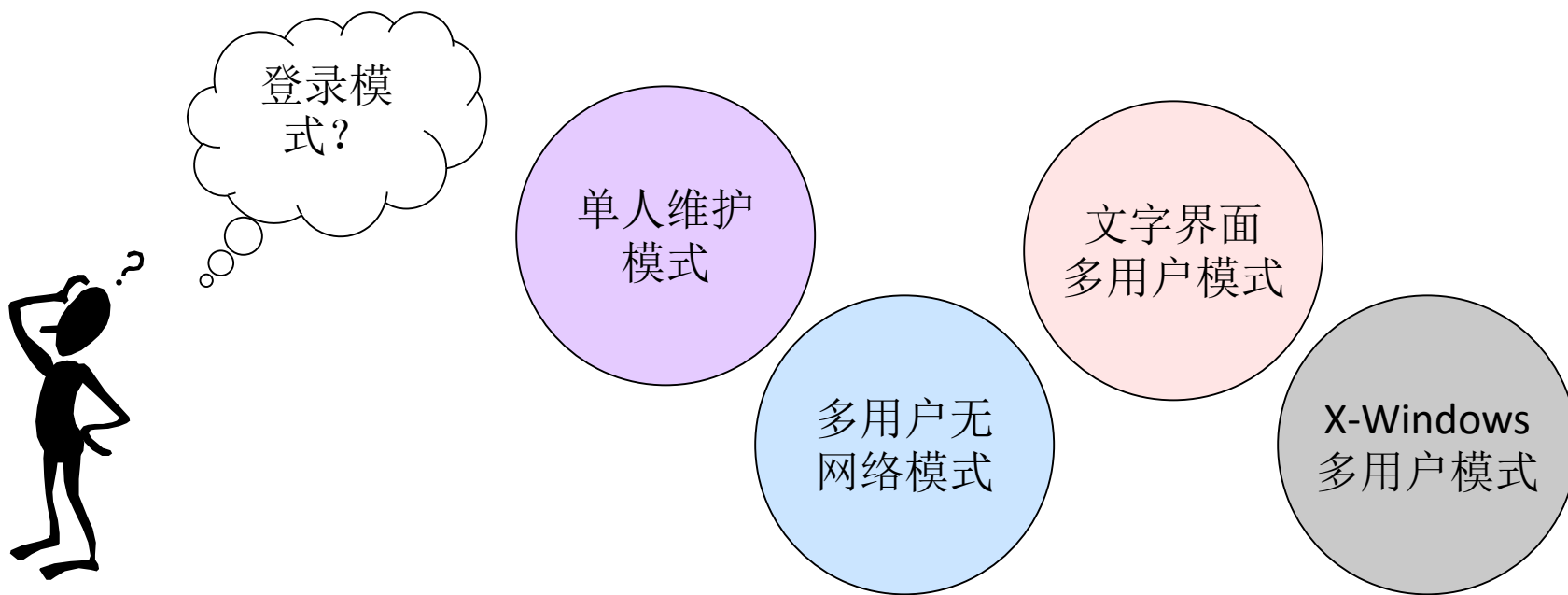
- ◆ init 进程是系统所有进程的起点，内核在完成内核引导以后，即在本线程（进程）空间内加载 init 程序，它的进程号是 1。
- ◆ inittab 是以行为单位的描述性（非执行性）文本，每一个指令行都具有以下格式：id:runlevel:action:process。



- ◆ id 一般要求 4 个字符以内，对于 getty 或其他 login 程序项，要求 id 与 tty 的编号相同，否则 getty 程序将不能正常工作。
- ◆ runlevel 是 init 所处于的运行级别的标识，一般使用 0—6 以及 S 或 s。
- ◆ initdefault 是一个特殊的 action 值，用于标识缺省的启动级别  
sysinit、boot、bootwait 等 action 将在系统启动时无条件运行。

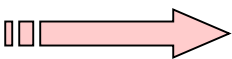

# 确定用户登录模式

- 在“/etc/inittab”中列出了登录模式，其中的单人维护模式是类似于Windows中的“安全模式”，在这种情况下，系统不加载复杂的模式从而使系统能够正常启动。其中本系统中默认的为5，也就是X-Windows多用户模式。



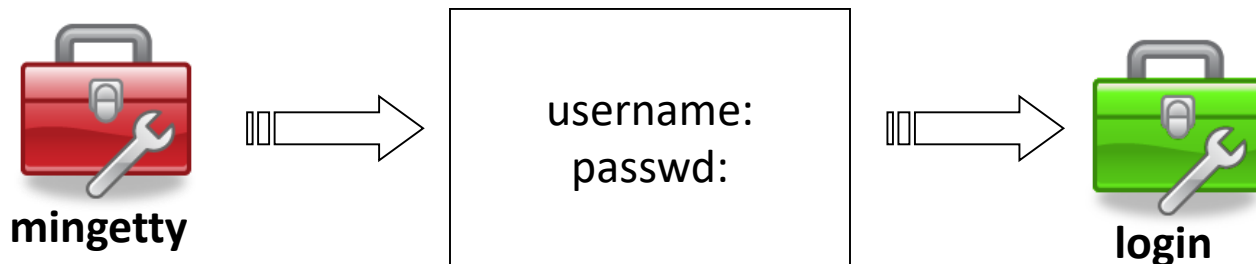


# 执行内容/etc/rc.d/rc.sysinit

- rc.sysinit 中最常见的动作就是：
  - 激活交换分区
  - 检查磁盘
  - 加载硬件模块
- 如果没有其他 boot、bootwait 动作
  - 运行级别3  /etc/rc.d/rc 将会得到执行
  - 命令行数3  /etc/rc.d/rc3.d/目录下的文件
- rc3.d 下的文件都是指向/etc/rc.d/init.d/目录下各个 Shell 脚本的符号连接。

# getty 和 login

- ◆ `mingetty` 程序能打开终端，设置模式。同时它会显示一个文本登录界面，这个界面就是经常看到的登录界面，在这个登录界面中会提示用户名，而用户输入的用户名将作为参数传给 `login` 程序来验证用户的身份。
- ◆ `login` 程序在 `getty` 的同一个进程空间中运行，接受 `getty` 传来的 用户名参数作为登录的用户名。
- ◆ 当用户登录通过了这些检查后，`login` 将搜索 `/etc/passwd` 文件用于匹配密码、设置主目录和加载 `shell`
- ◆ 在设置好 `shell` 的 `uid`、`gid`，以及 `TERM`，`PATH` 等环境变量以后，进程加载 `shell`，`login` 的任务也就完成了





# 等待用户登录

- ◆ 系统初始化完毕后，INIT 为每一个虚拟控制台和终端启动一个GETTY进程。GETTY进程负责接受和检验用户的登录要求。
- ◆ 至此，Linux系统的启动工作全部完成。不同核心版本的Linux 的启动过程有一定的差异，不同发行版本的Linux 的启动也可能稍有不同，但基本过程是类似的。另外，在“BOOT:”后，利用“Linux single”命令可以迫使Linux 进入单用户模式，除不要求用户登录和不启动虚拟终端以外，启动过程的其它部分也基本类似。





# 嵌入式软件开发环境构建

---

- 启动和初始化
- Linux常用命令和工具
- 宿主机-目标机开发模式
- 宿主机-目标机开发环境





# Linux常用命令和工具

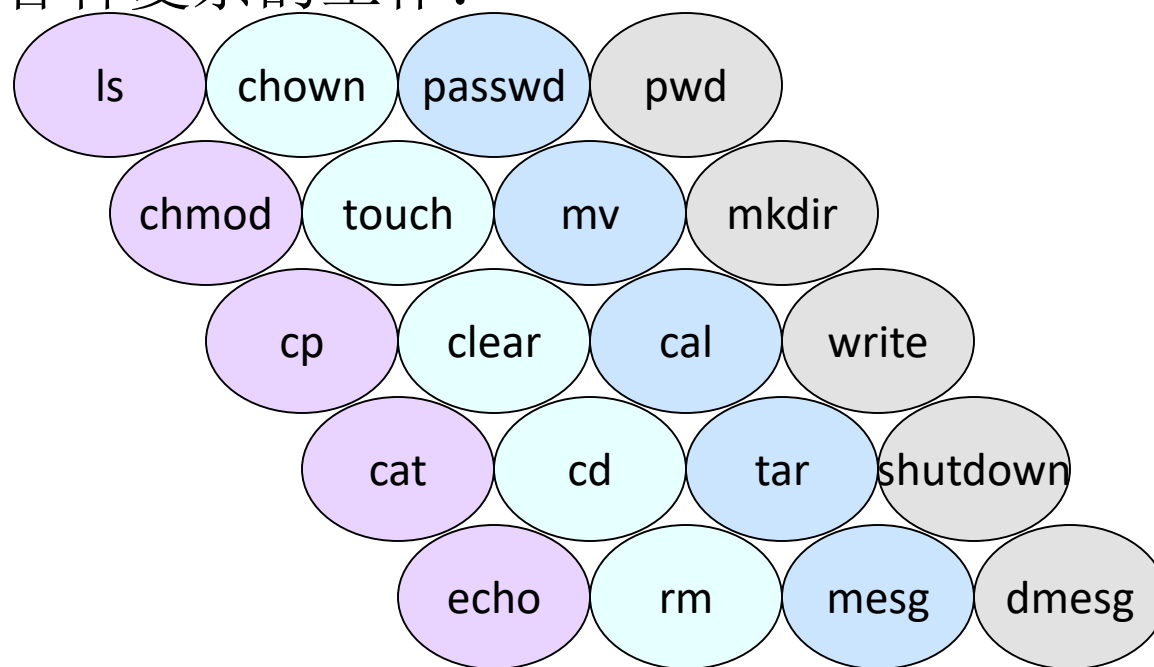
- 常用的Linux命令及使用
- Linux 下的编辑器 vi
- Linux 下的 Shell
  - Shell 简介
  - Shell 的发展历史
  - Shell 的使用
  - Shell 的功能

# 常用的Linux命令及使用

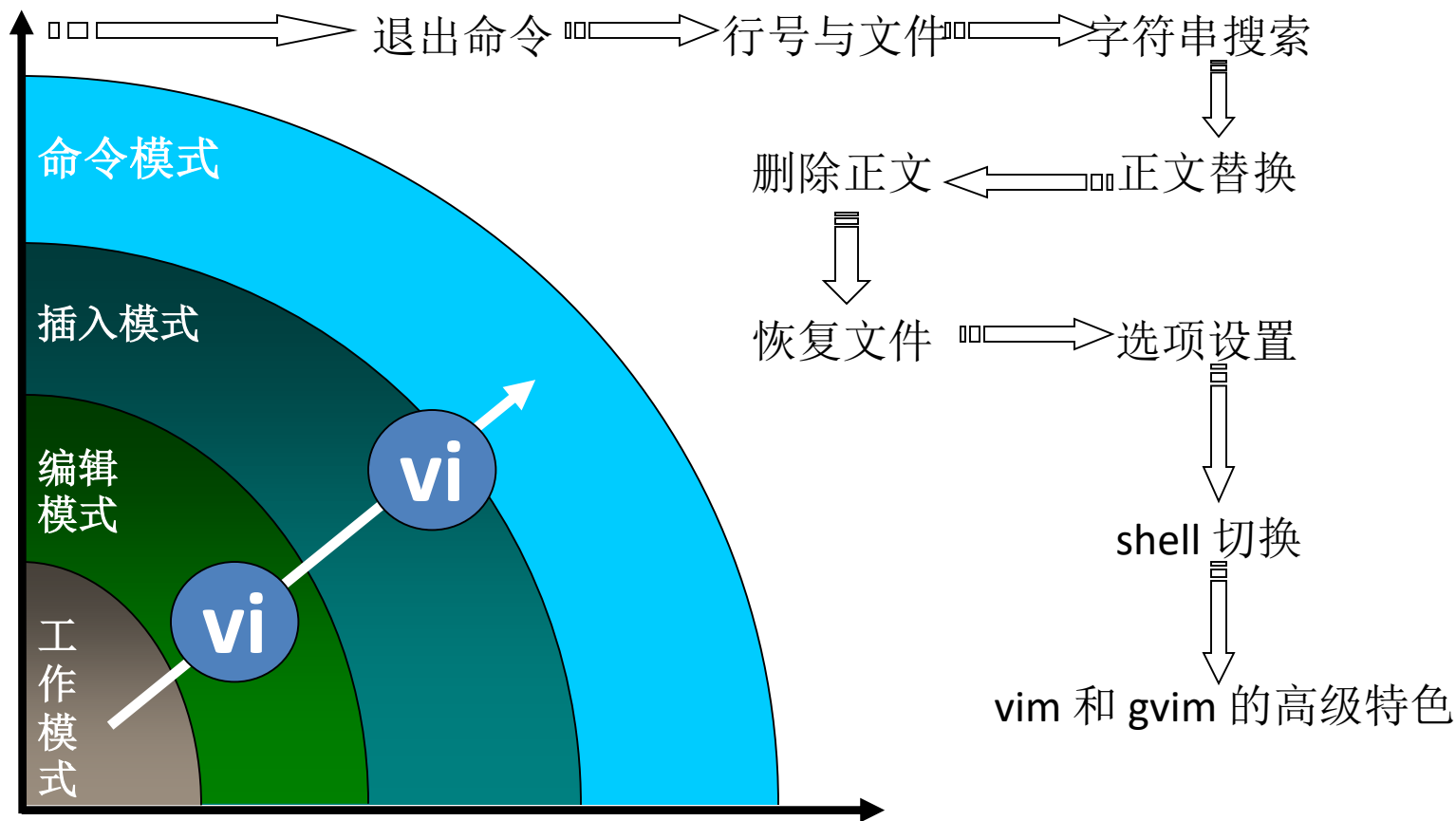
■ Linux 的系统管理主要在控制终端下进行，通过使用 命令行的方式进行管理。

■ Linux 的文件命令可以完成各种复杂的工作：

- ◆ 对目录进行复制
- ◆ 目录移动
- ◆ 目录链接
- ◆ 搜索和查找文件
- ◆ 搜索和查找目录
- ◆ 阅读文件内容
- ◆ 显示文件内容
- ◆ 打印文件内容

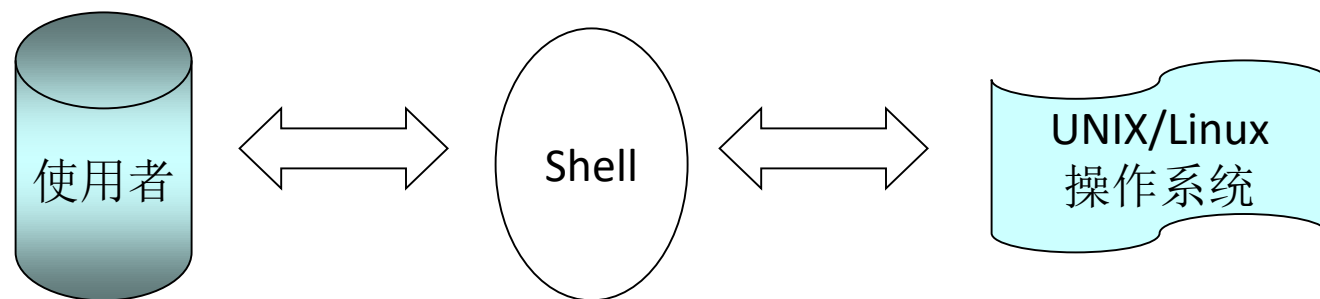


# Linux 下的编辑器 vi



# Shell 简介

- ◆ Shell 是一种具备特殊功能的程序，它是介于使用者和 UNIX/Linux 操作系统核心程序（kernel）之间的一个接口。
- ◆ 在系统起动的时候，核心程序会被加载内存，负责管理系统的工作，直到系统关闭为止。它建立并控制着处理程序，管理内存、档案系统、通讯等等。  
而其它的程序，包括 shell 程序，都存放在磁盘中。核心程序将它们加载内存，执行它们，并且在它们中止后清理系统。
- ◆ 一个 shell 命令档很像是 DOS 下的批次档（如 Autoexec.bat）：它把一连串的 UNIX 命令存入一个档案，然后执行该档。



# Shell 的发展历史

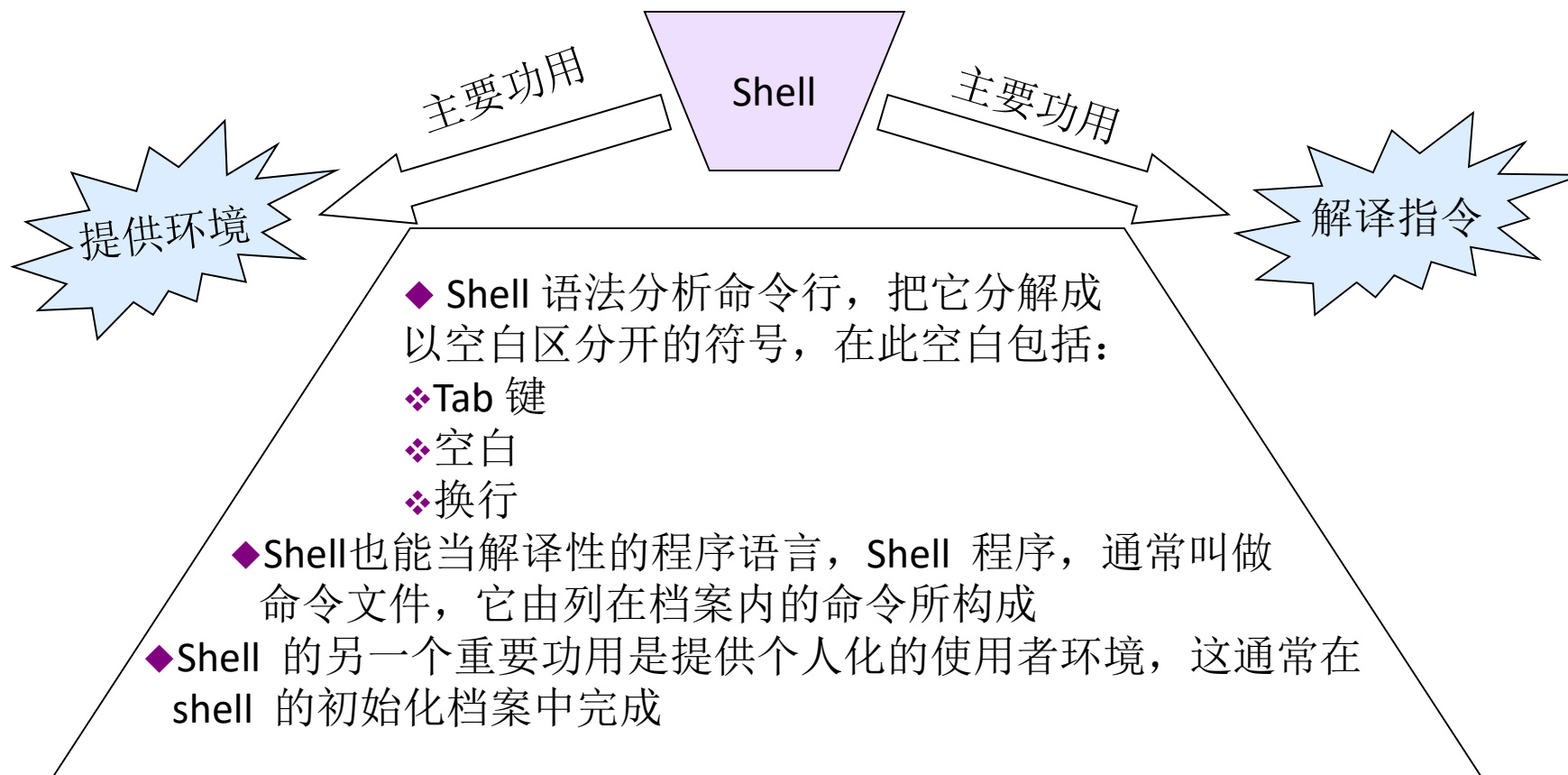
- ◆ C shell 是柏克莱大学所开发的，且加入了一些新特性：
  - ◆ 命令列历程
  - ◆ 别名
  - ◆ 内建算术
  - ◆ 档名完成
  - ◆ 工作控制
- ◆ C shell 提示符号的默认值是%

- ◆ Bourne shell 是标准的 UNIX shell
- ◆ 以前常被用来做为管理系统之用
- ◆ 大部份的系统管理命令文件，rc start、stop 与 shutdown 都是Bourne shell的命令档
- ◆ 在单一使用者模式下以 root 登入时它常被系统管理者使用



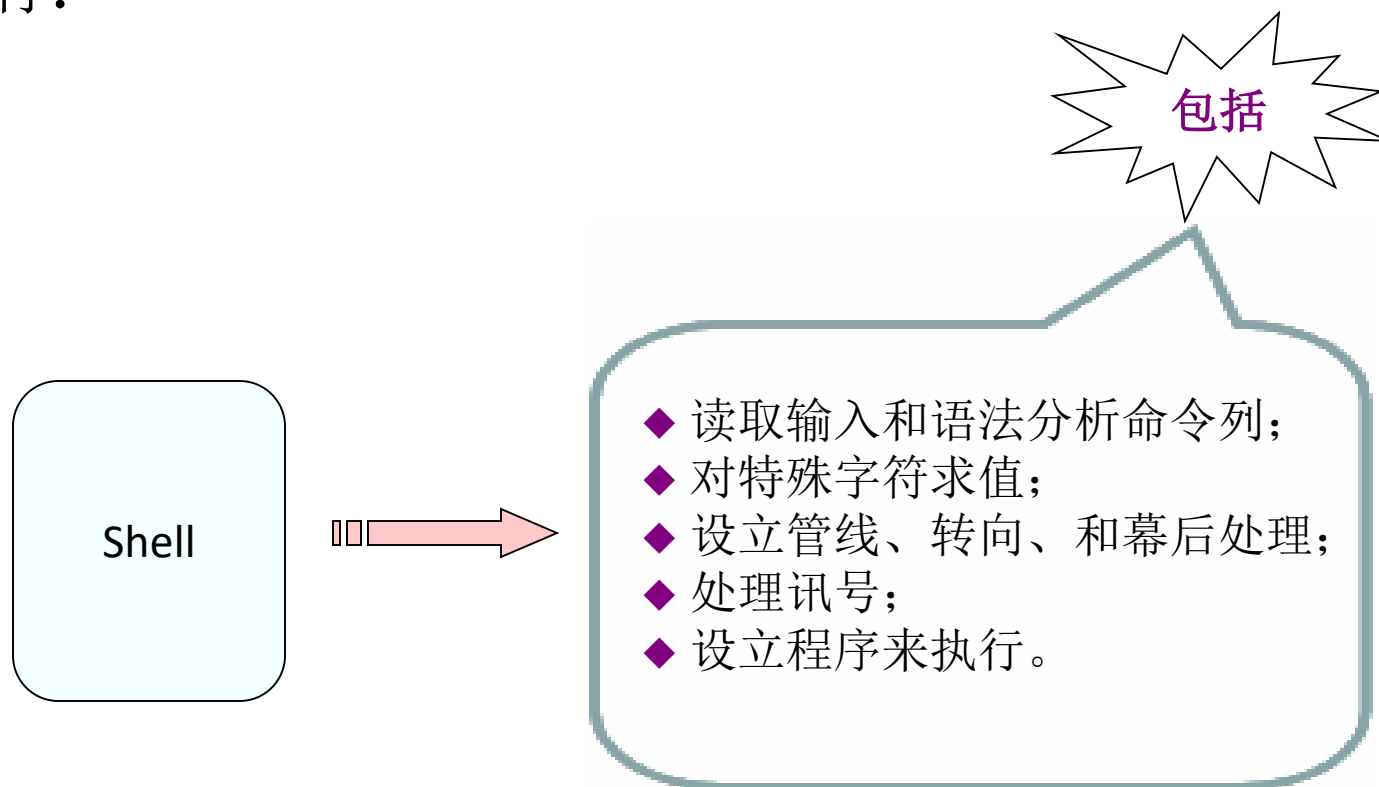
- ◆ Korn shell 是 Bourne shell 的超集，由 AT&T 的 David Korn 所开发。
- ◆ 它增加了一些特色，比 C shell 更为先进。
- ◆ Korn shell 的特色包括了可编辑的历程、别名、函式、正规表达式、万用字符、内建算术、工作控制、共作处理和特殊的除错功能。

# Shell 的使用



# Shell 的功能

- 为了确保任何提示符号下输入的命令都能够适当地执行。shell 担任的工作包括有：





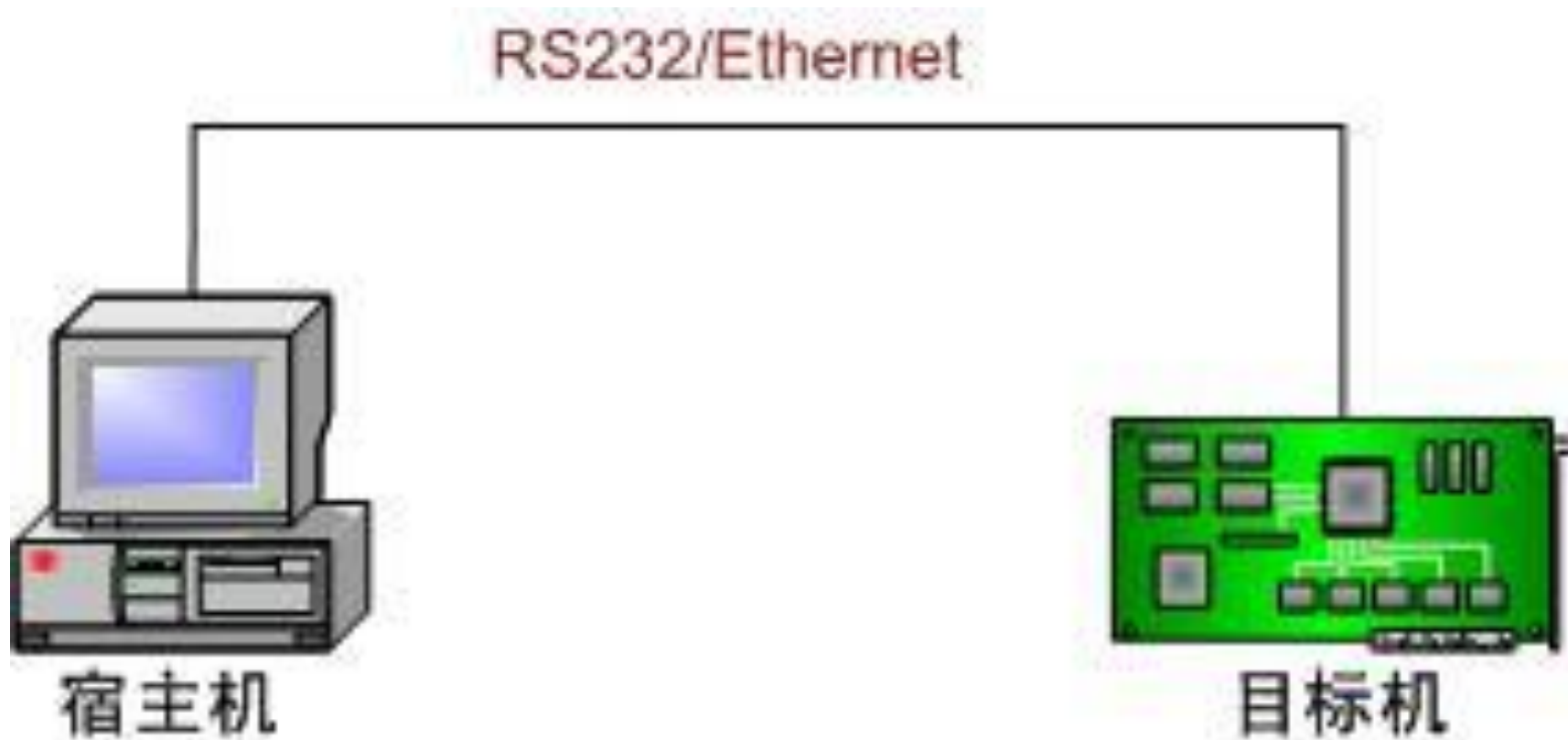
# 嵌入式软件开发环境构建

---

- 启动和初始化
- Linux常用命令和工具
- 宿主机-目标机开发模式
- 宿主机-目标机开发环境



# 宿主机-目标机开发模式





# 嵌入式软件开发环境构建

---

- 启动和初始化
- Linux常用命令和工具
- 宿主机-目标机开发模式
- 宿主机-目标机开发环境



# 宿主机环境

---

□ BOOTP协议

□ TFTP协议

□ 交叉编译



# BOOTP协议

---

- BOOTP服务的全称是BootStrap Protocol
- 使用TCP/IP网络协议中的UDP67/68两个通讯端口
- 常用到的DHCP服务就是从BOOTP服务扩展而来的



# BOOTP启动过程 (1)

- 第一步，在目标板由BootLoader启动BOOTP，此时目标板还没有IP地址，它就用广播形式以IP地址0.0.0.0向网络中发出IP地址查询的请求，这个请求帧中包含了客户机的网卡MAC地址等信息。



## BOOTP启动过程 (2)

- 第二步，主机平台运行BootP服务的服务器接收到的这个请求帧，根据这帧中的MAC地址在Bootptab启动数据库中查找这个MAC的记录，如果没有此MAC的记录则不响应这个请求；如果有就将FOUND帧发送回目标板。FOUND帧中包含的主要信息有目标板的IP地址、服务器的IP地址、硬件类型、网关IP地址、目标板MAC地址和启动映像文件名。



# BOOTP启动过程 (3)

---

- 第三步，目标板就根据FOUND帧中的信息通过TFTP服务器下载启动映像文件。

# TFTP协议

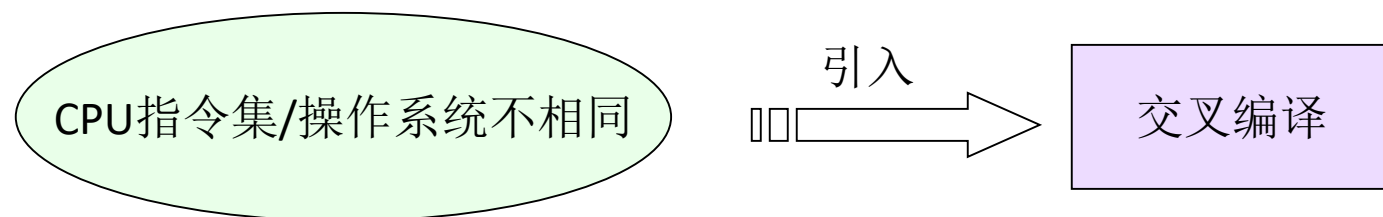


- TFTP服务的全称是Trivial File Transfer Protocol
- TFTP可以看成是一个简化了的FTP
- TFTP在安装时一定要设立一个单独的目录作为TFTP服务的根目录，以减少安全隐患

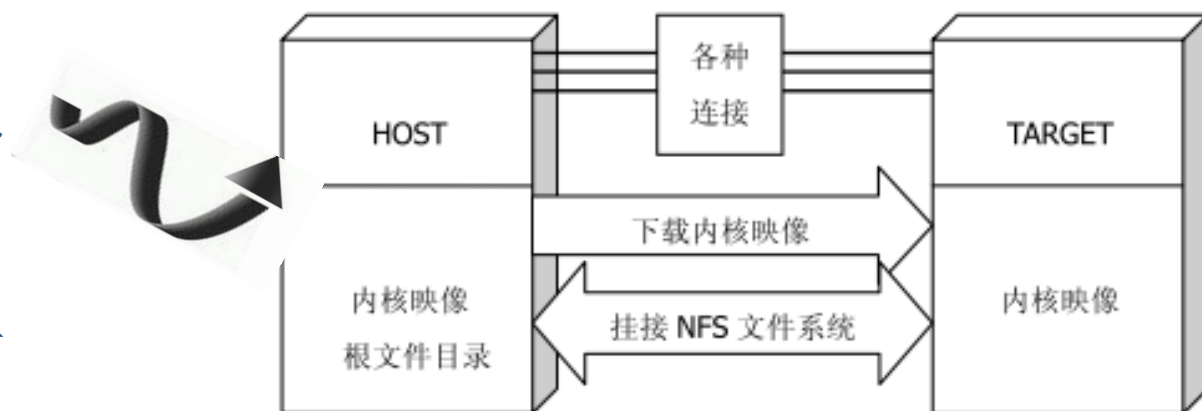


# 宿主机与交叉编译

- 交叉编译是嵌入式开发过程中的一项重要技术，简单地说，就是在一个平台上生成另一个平台上的可执行代码。

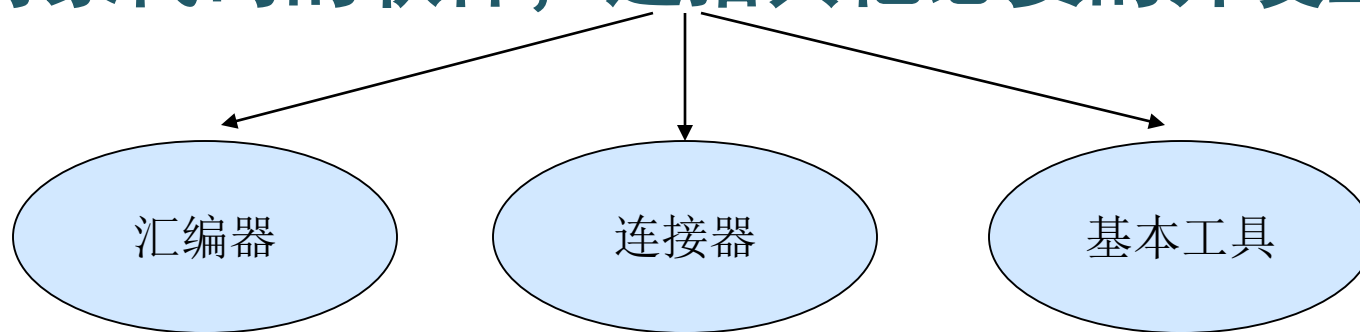


◆ 交叉编译的主要特征是某机器中执行的程序代码不是由本机编译生成，而是由另一台机器编译生成，一般把前者称为目标机，后者称为主机

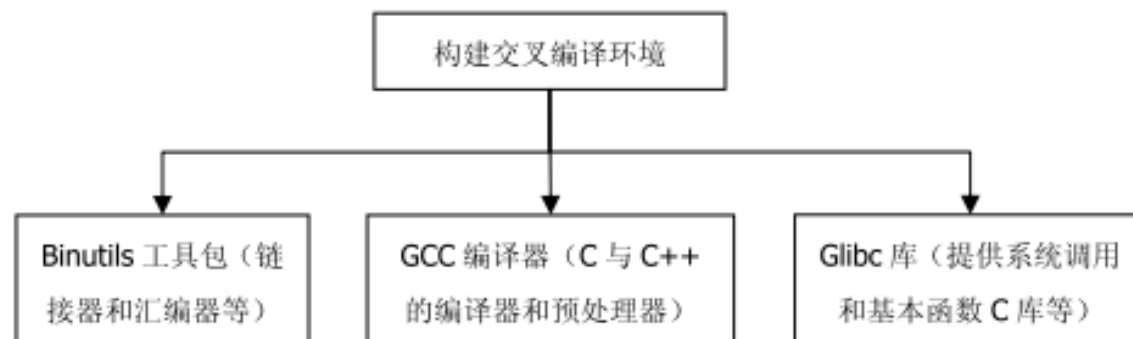


# 交叉编译器及交叉编译环境的组成

- 当提到交叉编译器时，不仅仅是指将一种编程语言的代码转换成对象代码的软件，还指其他必要的开发工具：



Linux下的交叉编译环境





# 目标机环境

## □ JTAG接口简介Joint Test Action Group

- 边界扫描测试，主要用于芯片内部测试。
- 接JTAG下载线或接其他JTAG仿真器。

## □ 其他调试工具

- **示波器**：板级硬件设计及调试，最原始的工具。有两类：模拟示波器和数据示波器
- **逻辑分析仪**：对微处理器总线的数据解码，观察总线发生的传输事务。有两类：状态分析仪和时序分析仪。
- **ICE在线仿真器**：是一种用于替代目标机上的CPU的设备。

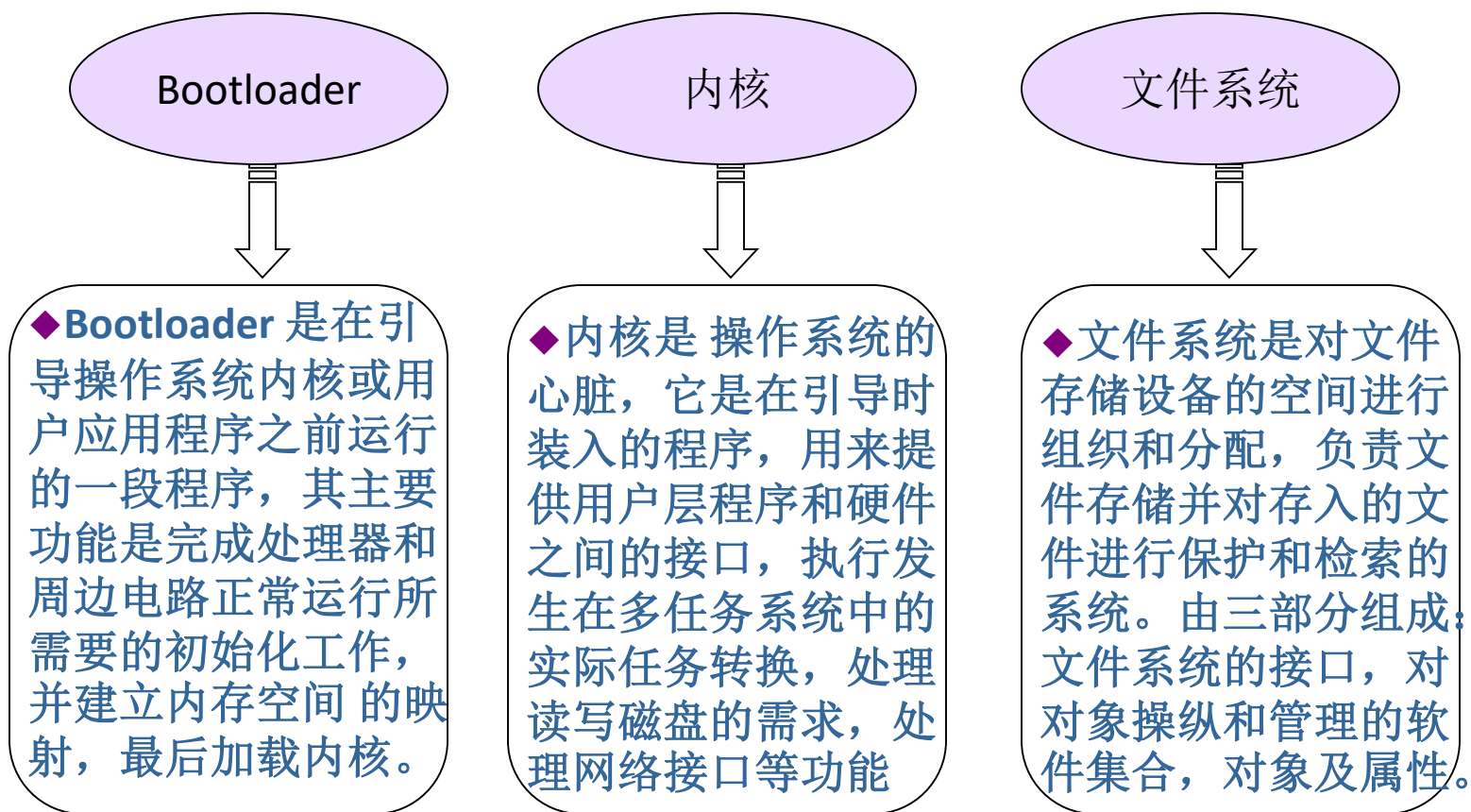


# Bootloader、内核、文件系统

## □ 存储地址空间关系



# Bootloader、内核、文件系统





# 嵌入式软件调试技术

---

□ 远程调试原理

□ GDB调试

□ 内核调试



# 远程调试的工作原理

---

- 通用的桌面操作系统与嵌入式操作系统在调试环境中存在明显的差别
- 远程调试，调试器运行于通用桌面操作系统的应用程序，被调试的程序则运行于基于特定硬件平台的嵌入式操作系统（目标操作系统）



# 远程调试带来的问题

- 调试器与被调试程序如何通信
- 被调试程序产生异常如何及时通知调试器
- 调试器如何控制、访问被调试程序
- 调试器如何识别有关被调试程序的多任务信息并控制某一特定任务
- 调试器如何处理某些与目标硬件平台相关的信息

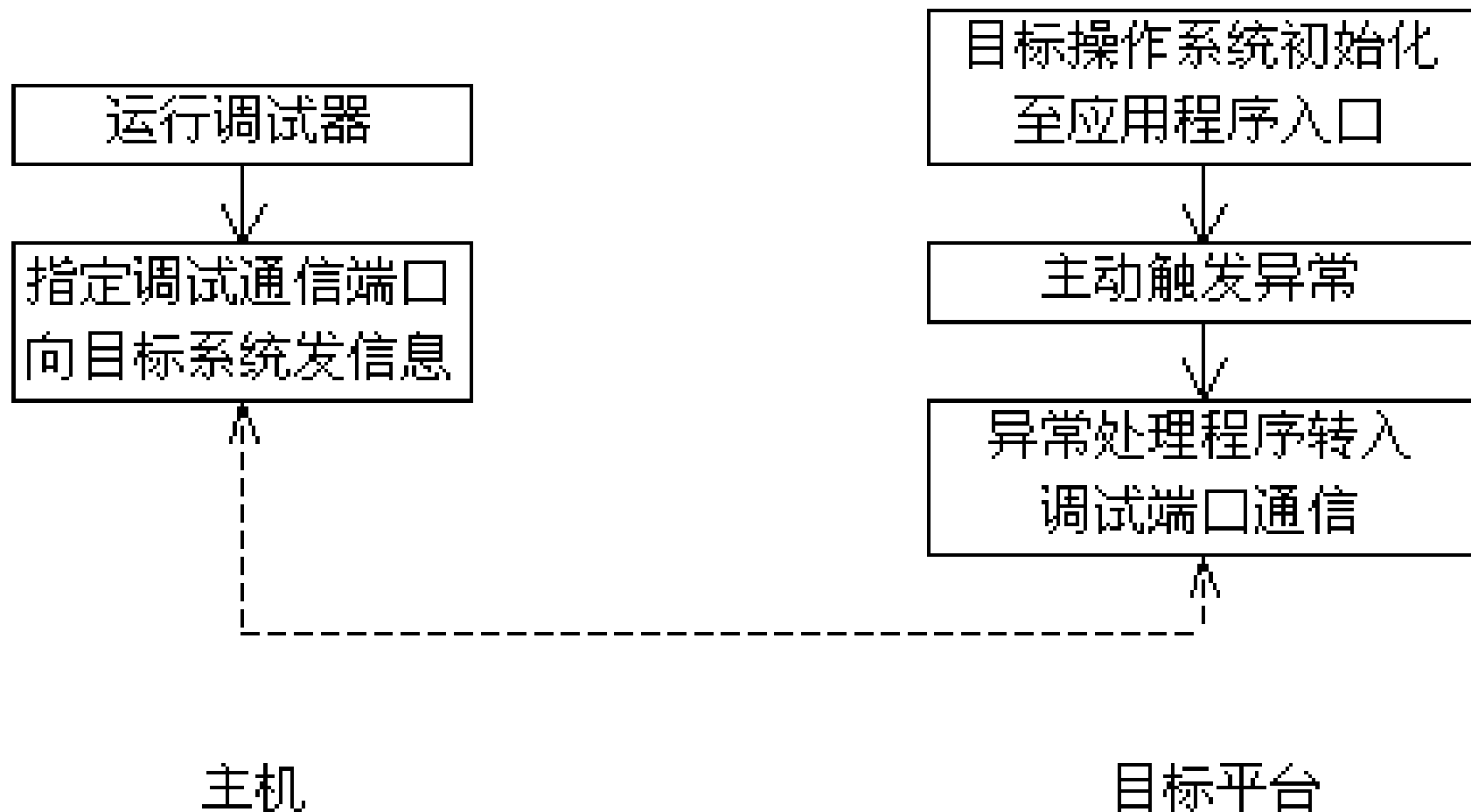




# 插桩 (stub)

- Stub方案是在目标操作系统和调试器内分别加入某些功能模块，二者互通信息来进行调试
- 这一方案需要目标操作系统提供支持远程调试协议的通信模块和多任务调试接口，并改写异常处理的有关部分
- 目标操作系统还需要定义一个设置断点的函数

# 远程调试示意图



## □ GDB是GNU C自带的调试工具

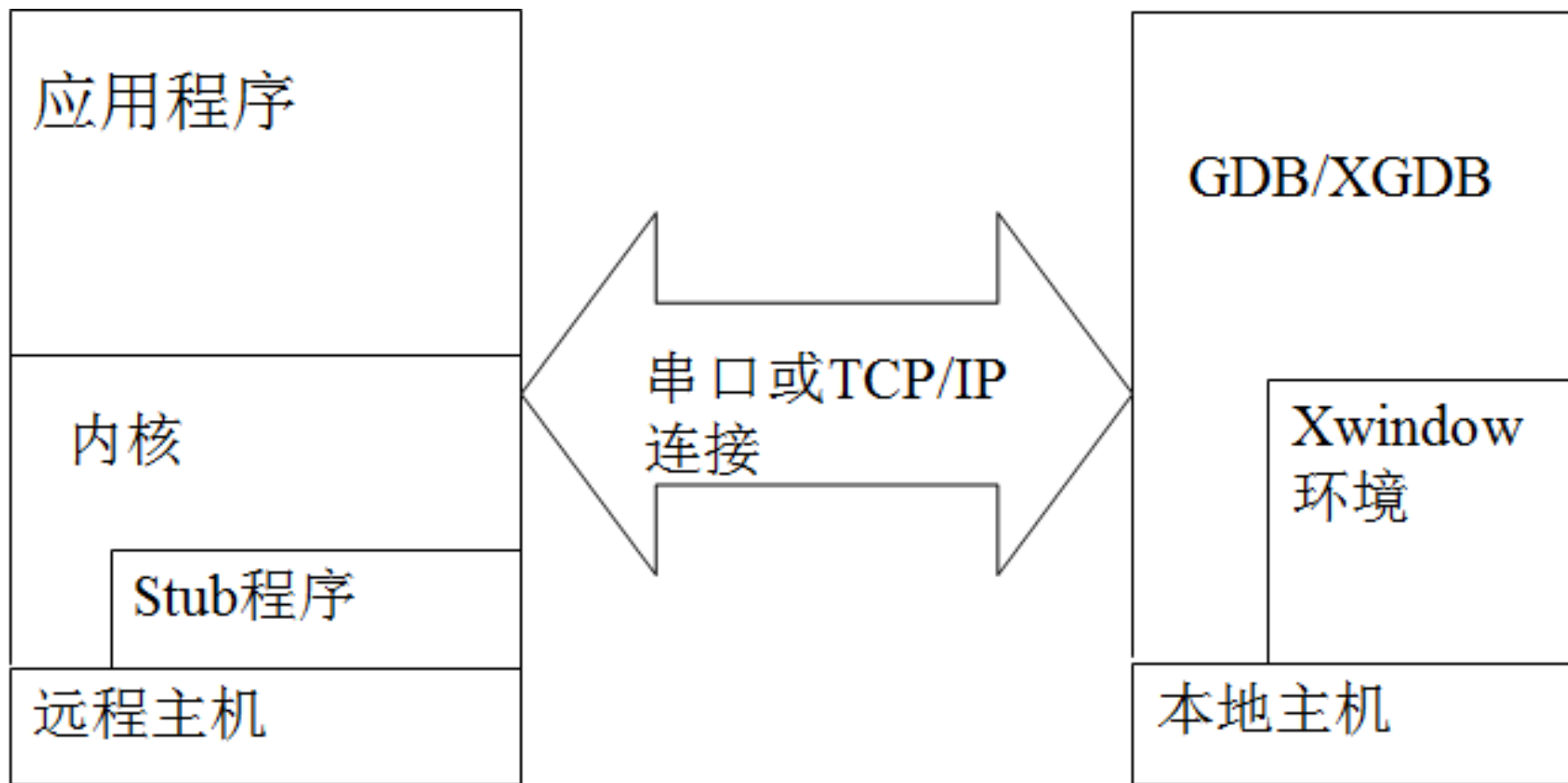
- 运行程序, 可以给程序加上所需的调试任何条件
- 在给定的条件下让程序停止
- 检查程序停止时的运行状态
- 通过改变一些数据, 可以更快地改正程序的错误



# GDB远程调试功能介绍

- 如果需要调试的程序和GDB所运行的环境不同，或者说需要调试的环境上根本无法运行起GDB，就需要使用远程调试功能
- 指定需要调试的远程机器的方法是使用target remote命令
- 在远程机器上，需要实现一个stub文件，在这个文件里面提供串口连接的协议，和传送数据信息的方法

# GDB远程调试环境原理图





# GDB调试

## □ GDB命令

- 补齐功能
- 键入gdb gdb\_test命令来启动GDB并载入程序 gdb\_test, 命令行进入了GDB模式

```
$ gdb gdb_test
```

```
.....
```

```
(gdb)
```



# GDB中的常用命令 (1)

指令	解释
file	载入程序。如file hello。当然，程序的路径名要正确。
quit	退出GDB。也可以输入'C-d'来退出GDB。
run	执行载入后的要调试的程序。可以输入参数。
info	查看程序的信息。可以用help info来查看具体帮助。 info sourc查看当前文件的名称，路径，所使用的程序语言等信息。 info stack 查看调用栈。 info local 查看局部变量信息。 info br br是断点break的缩写，用这条指令，可以得到所设置的所有断点的详细信息。
list	list FUNCTION列出被调试程序某个函数 list LINENUM以当前源文件的某行为中间显示一段源程序 list 接着前一次继续显示 list - 显示前一次之前的源程序 list FILENAME:FUNCTION显示另一个文件的一段程序，



# GDB中的常用命令 (2)

break	最常用和最重要的命令：设置断点。 break FUNCTION在函数入口设置断点 break LINENUM在当前源文件的某一行上设置断点 break FILENAME:LINENUM在另一个源文件的某一行上设置断点 break *ADDRESS在某个地址上设置断点
watch	监视某个表达式或变量，当它被读或被写时让程序断下。格式如下： watch EXPRESSION
set	修改变量值。格式如下： set variable=value
step	单步执行，进入遇到的函数。
next	单步执行，不进入函数调用，即视函数调用为普通语句。
continue	恢复中断的程序执行。
help	通过下面的方法获得帮助，下例为获得list指令。 help list





# 断点与条件断点

## □ GDB中的断点有四种状态

- 有效 (Enabled)
- 禁止 (Disabled)
- 一次有效 (Enabled once)
- 有效后删除 (Enabled for deletion)

## □ 条件断点的设置语句

```
(gdb) break ...if COND
```



# Linux 内核调试

---

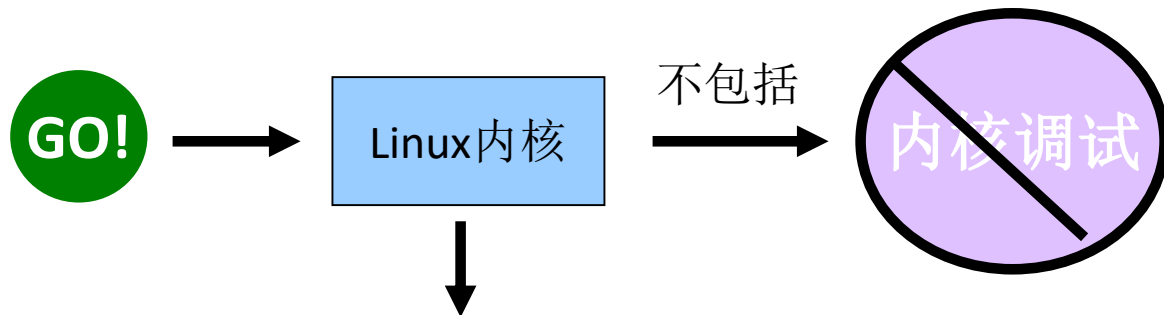
□ Linux内核调试

□ Linux 内核调试——pr intk

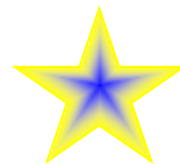
□ Linux 内核调试——GDB

□ Linux 内核调试——KDB

# Linux内核调试



◆调试是软件开发过程中一个必不可少的环节，在内核开发的过程中也不可避免地会面对如何调试内核的问题。



- ◆当内核运行出现错误的时候，首先要明确定义和可靠地重视这个错误现象。
- ◆对于庞大的 Linux 内核软件工程，单靠阅读代码查找问题已经非常困难，需要借助调试技术解决 **BUG**。
- ◆调试内核很难，实际上内核不同于其他软件工程。
- ◆内核的 **BUG** 是多种多样的。



# Linux内核调试步骤

- 需要在Linux内核里面做一些修改，并且提供一个stub文件
- 把stub，串口驱动程序和Linux内核编译连接在一起
- 利用这个核心启动的系统，在需要进行调试的时候，激活程序的断点，等待本地主机的连接
- 然后，就可以进行内核的调试了

# printk、KGDB、KDB

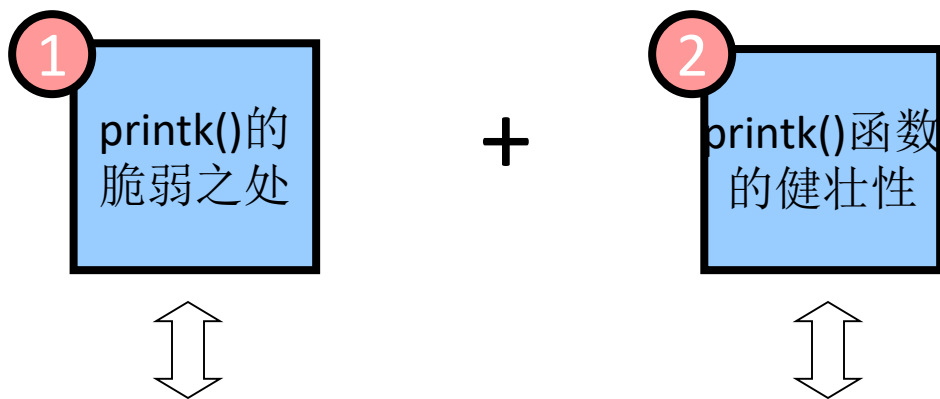


- ◆只有熟悉了内核各部分的代码实现，才能够找到准确的跟踪点
- ◆只有熟悉操作系统的内核机制，才能准确地判断系统运行状态
- ◆调试是无法逃避的任务
- ◆进行调试有很多种方法，比如将消息打印到屏幕上、使用调试器



# Linux 内核调试——printk

- ◆ `printk()` 是调试内核代码时最常用的一种技术。在内核代码中的特定位置加入 `printk()` 调试调用，可以直接把所关心的信息打印到屏幕上
- ◆ `printk` 函数具有极好的健壮性，不受内核运行条件的限制，在系统运行期间 都可以使用。



- ◆ `printk()` 函数的健壮性也有漏洞
- ◆ 在系统启动过程中，在某些地方不能使用它
- ◆ 提供一个 `printk()` 的变体函数
- ◆ 健壮性是 `printk()` 函数最容易被人们接受的一个特质，它可以在中断上下文和进程上下文中调用，它可以在持有锁时调用

- 调试内核代码的时候，则可以用 `printk()` 显示监视信息
- `printk()` 可以指定一个记录级别，在头文件 `<linux/kernel.h>` 中定义了 8 种可用的日志级别字符串：
  - `KERN_EMERG` 用于紧急事件消息，它们一般是系统崩溃之前提示的消息。
  - `KERN_ALERT` 用于需要立即采取行动的情况。



- **KERN\_CRIT** 临界状态，通常涉及严重的硬件或软件操作失败。
- **KERN\_ERR** 用于报告错误状态；设备驱动多用此级来报告来自硬件的问题。
- **KERN\_WARNING** 警告可能出现问题，这类情况通常不会对系统造成严重问题。
- **KERN\_NOTICE** 正常情形的提示。许多与安全相关的状况用这个级别进行汇报。
- **KERN\_INFO** 提示性信息。很多设备驱动启动时，用此级别打印相应的硬件信息。
- **KERN\_DEBUG** 用于调试信息。



- 在标准的Linux系统上，用户空间的守护进程klogd从记录缓冲区中获取内核消息，通过syslogd守护进程将他们保存在系统日志文件中
- 根据日志级别，内核可能会把消息打印到当前控制台上
- 内核在遇到运行错误时，会显示发生错误时处理器的状态

- 大部分错误都是由于 NULL指针的使用或其他不正确的指针值的使用，这些错误通常会导致一个 oops 消息
- oops消息给开发者许多信息，但是往往这些原始信息都是一些十六进制的内存地址，有两个工具可用来将其解析为符号：
  - klogd
  - ksymoops

# Linux 内核调试——KGDB



- ◆ 启动程序，可以按照自定义的要求随心所欲的运行程序
- ◆ 可让被调试的程序在指定的调试的断点处停住
- ◆ 当程序被停住时，可以检查此时程序中所发生的事
- ◆ 动态的改变程序的执行环境



# KGDB分析 (1)

- KGDB适用于两种情况下的调试
  - 开发者需要调试内核
  - 开发者需要调试驱动模块
- 使用KGDB进行内核调试的步骤如下
  - 下载当前使用内核的kgdb patch
  - 在PC主机端安装patch
  - 运行make menuconfig命令，确保kgdb选项中的KERNEL\_HACKING被选上



# KGDB分析 (2)

- 重新编译内核。
- 把新生成的内核镜像zImage拷贝到开发板。
- 在启动开发板上的内核前需要设置：gdb gdbttyS=0 gdbbaud=38400。
- 在PC主机端，使用命令

```
cd /path/to/kernel/source  
gdb vmlinux  
gdb: set remotebaud 38400  
gdb: target remote /dev/ttyS0
```



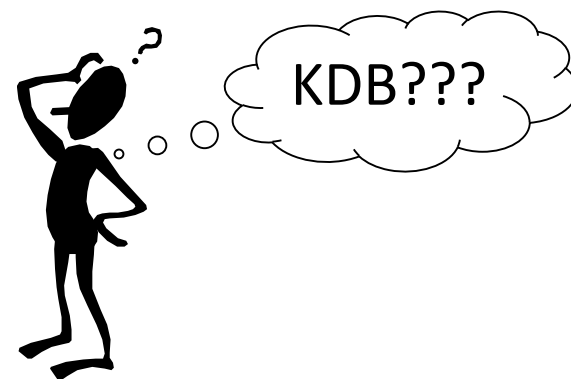
# 内核进入调试状态的路径

## □ 内核进入调试状态有两种方法

- 通过在内核启动的时候向内核传入参数，这时可以调试系统启动过程内核的运行状况
- 在内核完全导入系统正常运行的情况下，通过使用一个gdbstart工具将驱动串口设备，内核的控制权交给本地主机

# Linux 内核调试——KDB

◆KDB 是一个 Linux 系统的内核调试器，它是由 SGI 公司开发的遵循 GPL 许可证的开放源码调试工具。



类别	功能
运行控制类	提供对程序执行的控制
断点设置类	设置断点、清除断点、激活断点、使断点失效
内存操作类	对内存进行显示和修改
堆栈跟踪类	实现对堆栈的跟踪
寄存器类	对寄存器内容进行显示和修改
环境变量类	对 kdb 调试器环境变量进行显示和设置

◆KDB 是一个功能非常强大的工具，它允许进行几个操作，比如内存和寄存器修改、应用断点和堆栈跟踪



□ 谢 谢！