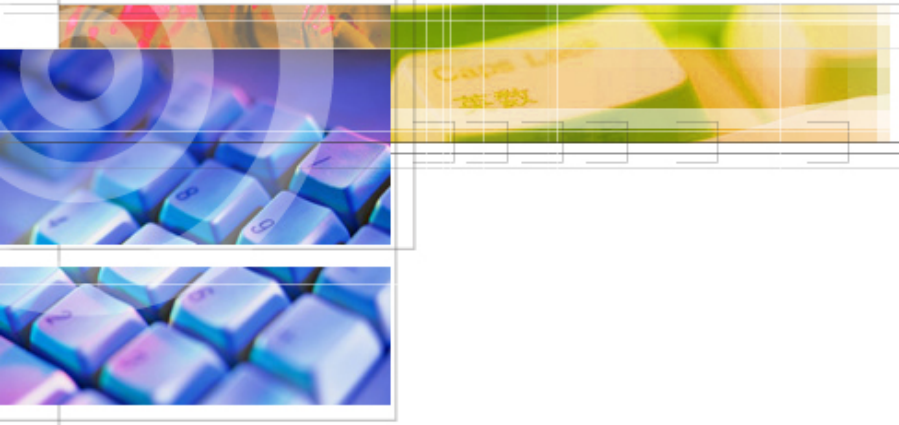


嵌入式Linux软件设计

Embedded Linux Software Design

华中科技大学电信学院

鄢舒

E-mail: yan0shu@gmail.com

参考书目

- [1] 孙天泽，袁文菊，张海峰，嵌入式设计及Linux驱动开发指南——基于ARM 9处理器，北京：电子工业出版社，2005
- [2] 孙琼等,嵌入式linux应用开发详解,北京：人民邮电出版社，2006
- [3] Alessandro Rubini and Jonathan Corbet, Linux Device Drivers, 2nd Edition, O'Reilly, 2002

必备知识和要求

- 1.熟悉C语言和基本程序设计方法；
（C程序设计、数据结构、操作系统原理）
- 2.了解基本软硬件调试方法；
（微机原理实验或单片机实验）
- 3.熟练掌握搜索相关网络资源的技巧；
- 4.熟悉UNIX/Linux操作系统的基本操作；
- 5.基本了解嵌入式Linux系统或Bootload代码的移植过程。

教学形式与考核方法

- 以实验为主，结合课堂讲解基本知识和课后阅读参考文献为辅的教学形式。

（要特别重视实际动手操作和学会从网络上搜索解答问题的技巧）

- 考核方法为按单人或两人一组的方式完成指定或自选的课题，并提交研究报告。

（实验结果和实验过程并重，研究报告要重视对经验的总结特别是失败的经历）

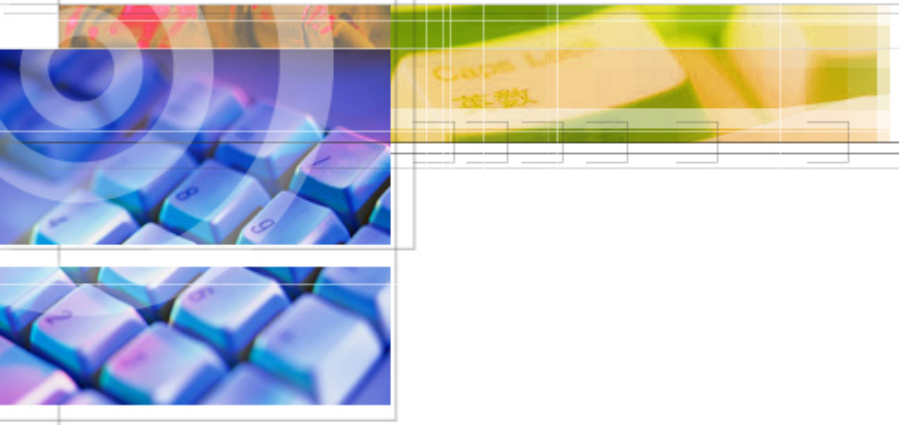
注意事项

- 实验过程中出现异常现象(包括糊味、冒烟或芯片异常发烫等)请立即关闭实验设备电源再检查问题；
- 请勿热插拔串口、并口等计算机端口；
- 下课离开前请整理好实验设备和桌椅。

✓本课程资源地址：

http://pan.baidu.com/s/1i53y8DZ 密码: kixm

（含课件、电子书和文档）



第一讲 嵌入式Linux系统基础

华中科技大学电信学院
鄢舒
E-mail: yan0shu@gmail.com

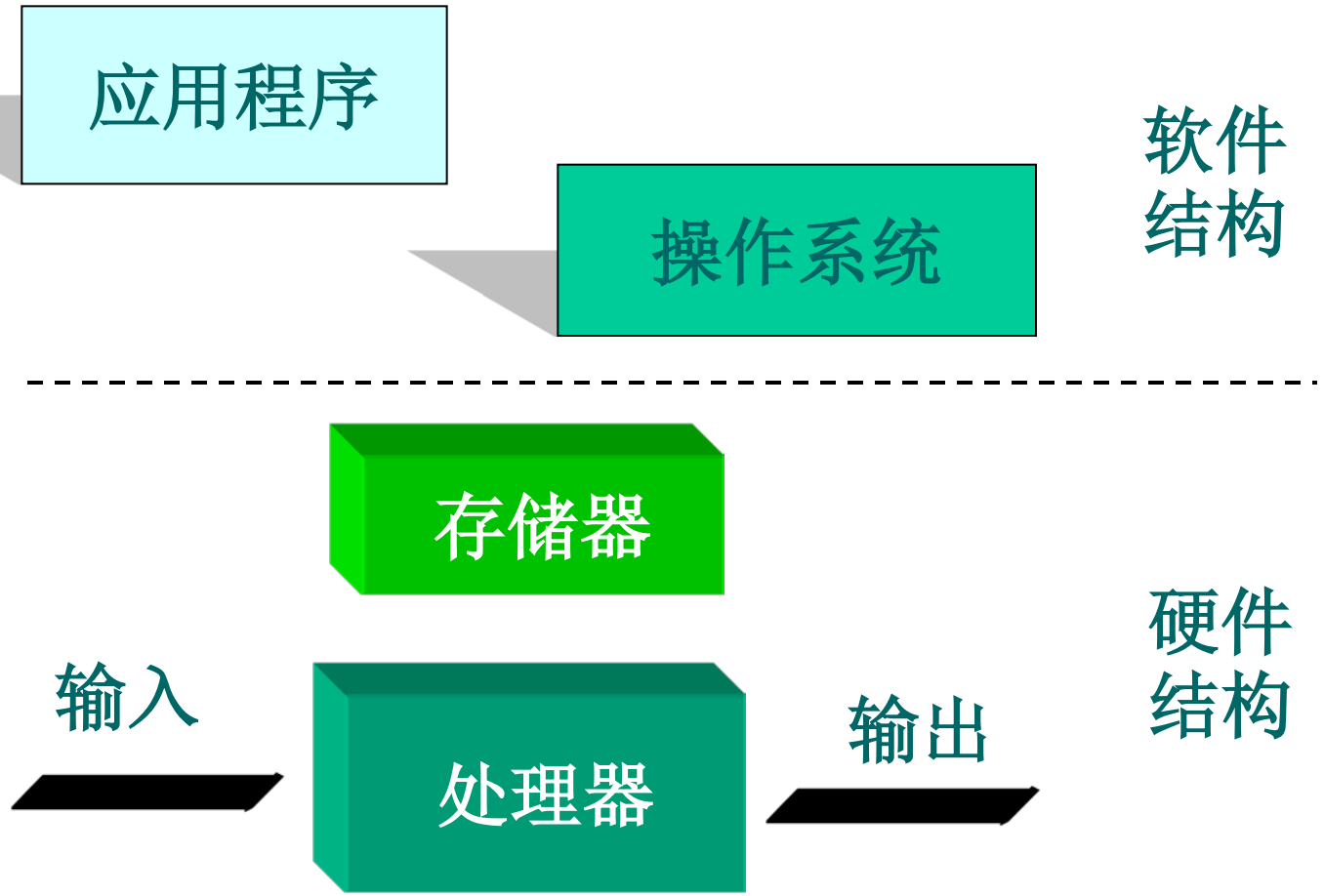
内容提纲(1/8)

- 1. 嵌入式系统概述
- 2. 嵌入式系统软件开发
- 3. 嵌入式操作系统介绍
- 4. 嵌入式Linux操作系统
- 5. Linux程序设计基础
- 6. GNU make入门
- 7. GDB简介
- 8. 实验系统HUDAX-ARM9-2410简介

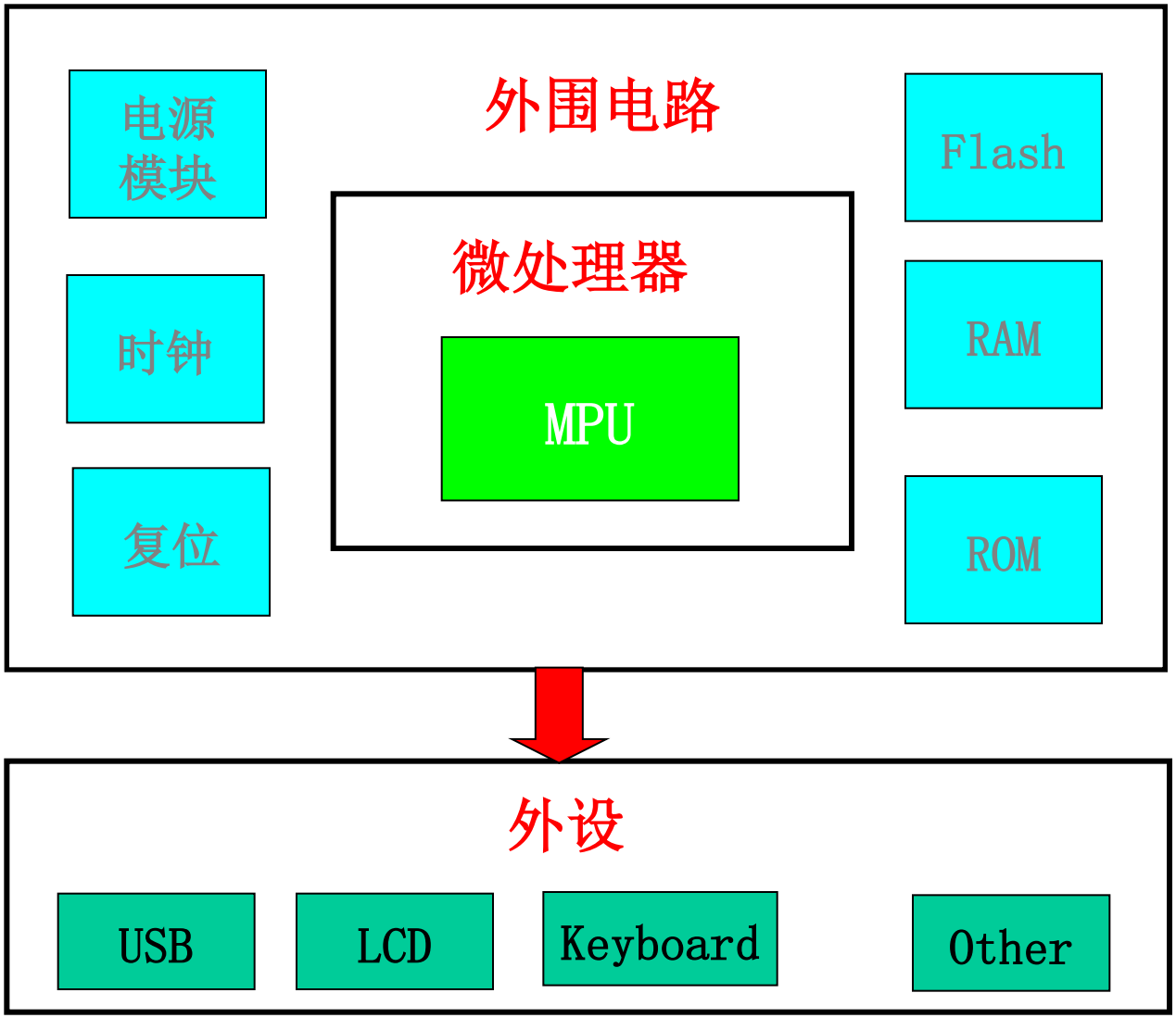
1.1 嵌入式系统的定义

- 嵌入式系统(Embedded system)是指：
  - 以应用为中心、以计算机技术为基础、软件硬件可裁剪、功能、可靠性、成本、体积、功耗严格要求的专用计算机系统。
  - 嵌入式系统一般由嵌入式微处理器、外围硬件设备、嵌入式操作系统以及用户应用程序四个部分组成，用于实现对其他设备的控制、监视或管理等目标。

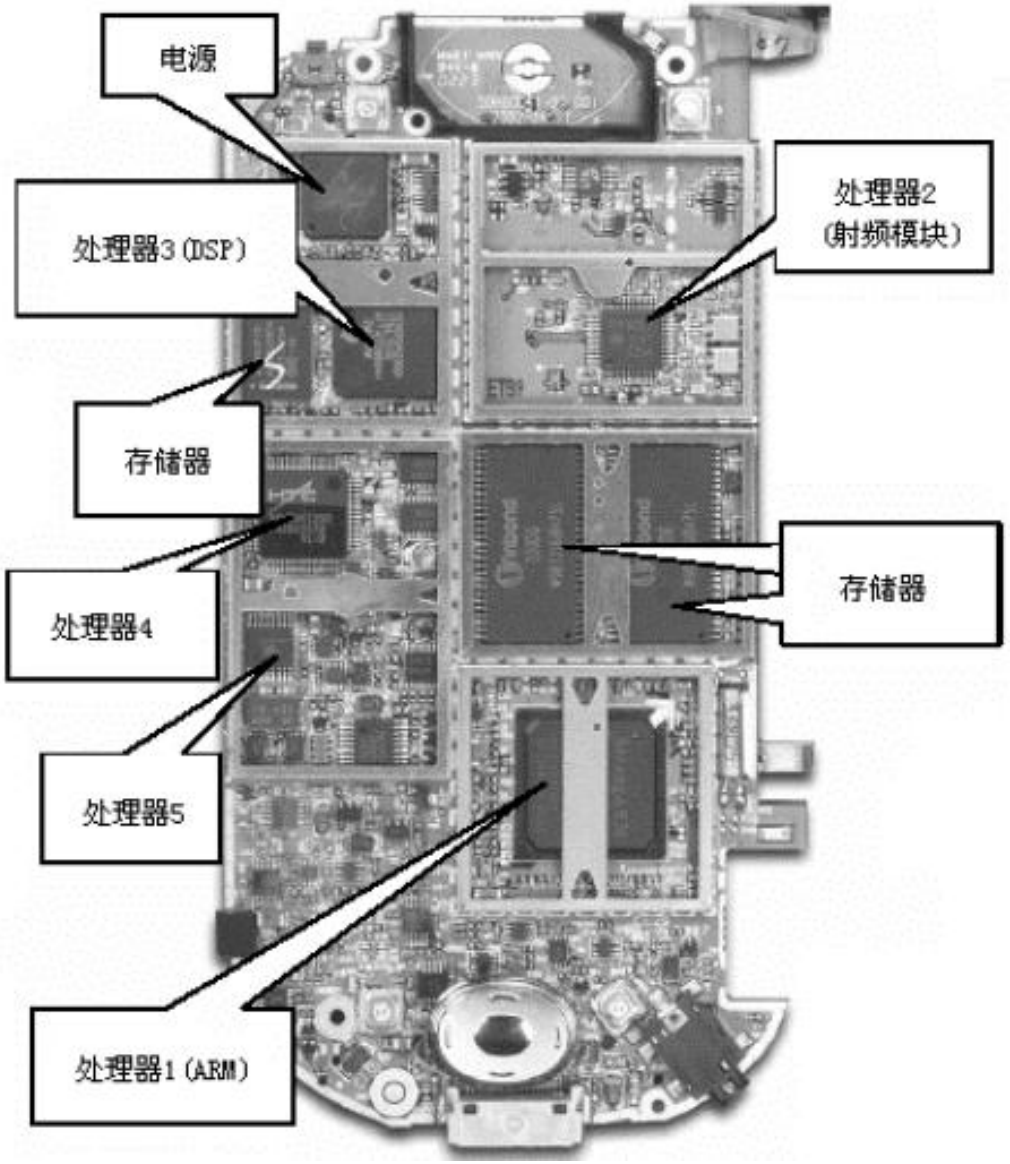
1.2 嵌入式系统的架构



1.3 典型的嵌入式系统的基本硬件组成



1.4 实际的嵌入式系统



1.5 嵌入式系统的发展趋势

- 嵌入式应用软件的开发需要强大的开发工具和操作系统的支持
  - 采用实时多任务编程技术和交叉开发工具来实现复杂的控制功能
  - 简化应用程序设计，保障软件质量和缩短开发周期
- 联网成为必然趋势
- 实现小尺寸、微功耗和底成本
- 提供精巧的多媒体人机界面

内容提纲(2/8)

- 1. 嵌入式系统概述
- 2. 嵌入式系统软件开发
- 3. 嵌入式操作系统介绍
- 4. 嵌入式Linux操作系统
- 5. Linux程序设计基础
- 6. GNU make入门
- 7. GDB简介
- 8. 实验系统HUDAX-ARM9-2410简介

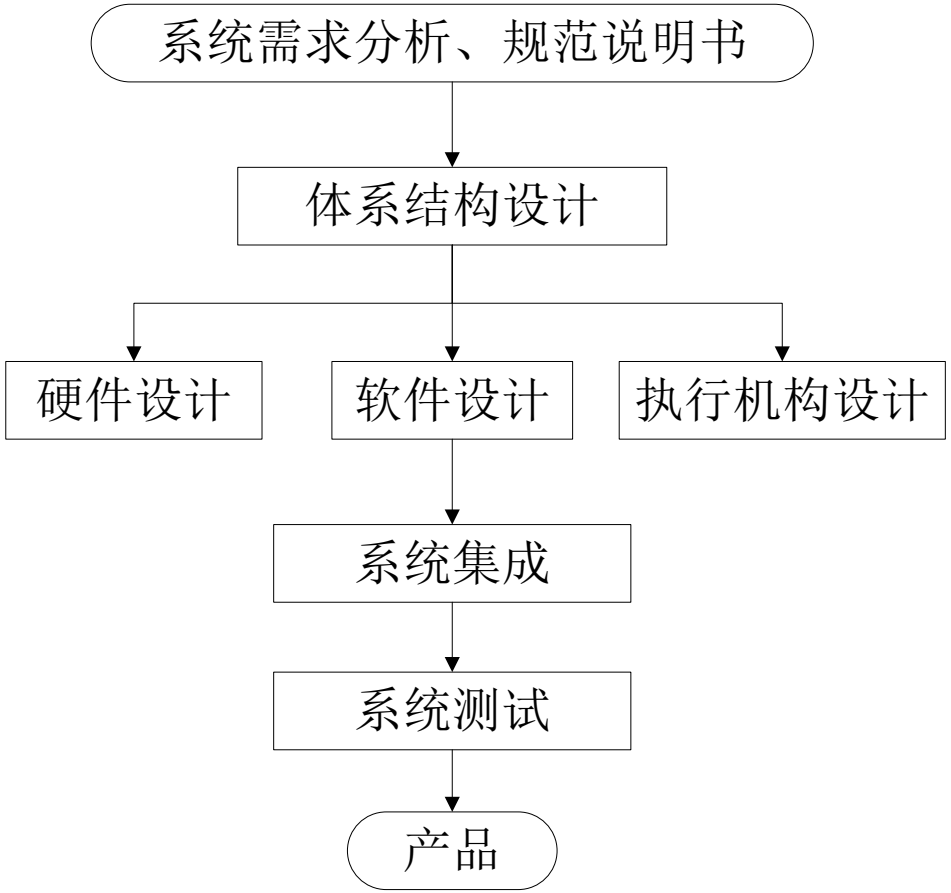
2.1 嵌入式系统的软件特征

- 软件固化：提高执行速度和系统的可靠性
- 软件代码高质量和高可靠性
- 嵌入式操作系统具备实时处理能力

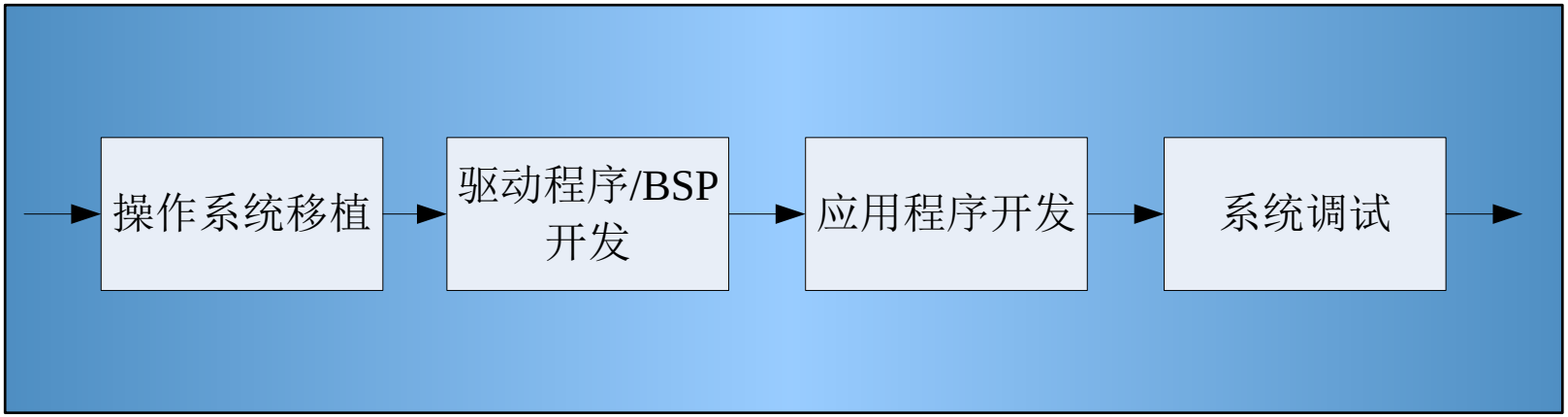
2.2 与普通软件开发的区别

- 是否需要操作系统
- 程序编译和程序执行是在两个平台
  存在host端和target端，需要交叉编译
- 输入/输出的界面不同
- 可利用的资源非常有限
- 常常要和硬件打交道

2.3 嵌入式系统设计的一般流程



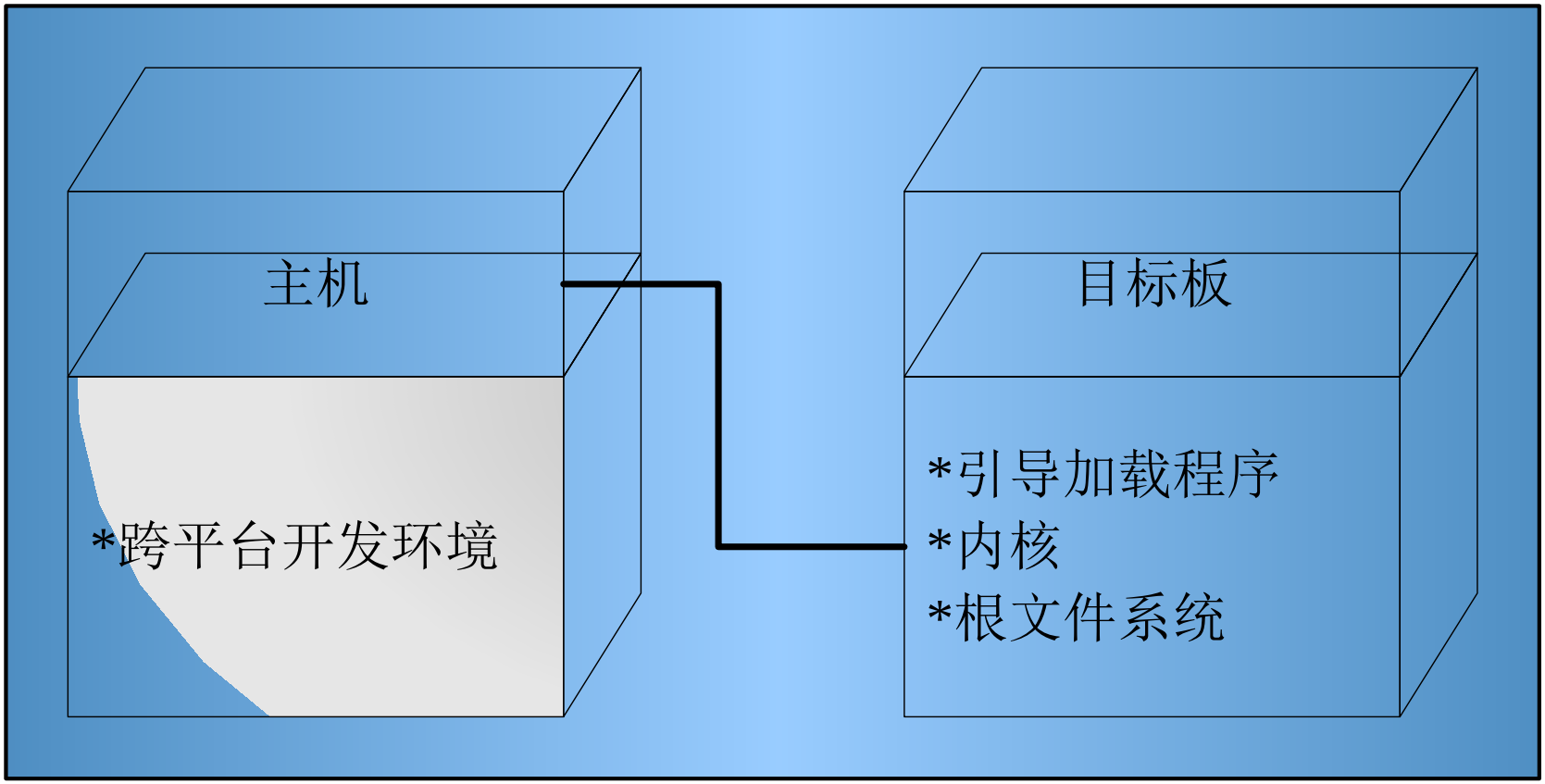
2.4 嵌入式软件开发流程



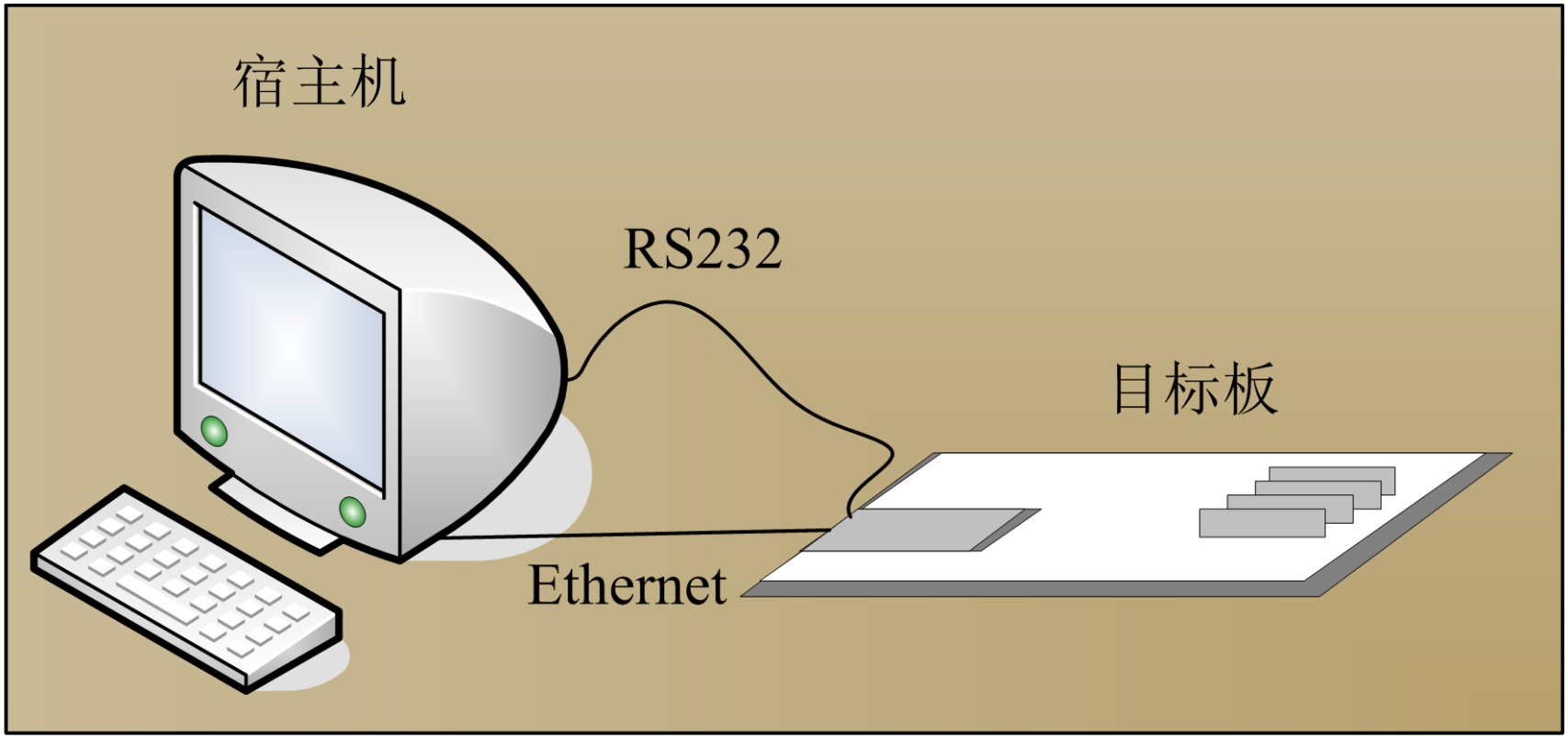
2.5 嵌入式系统软件的开发环境

- 嵌入式系统的一个特点在于其开发的特殊性与困难性。
  - 开发机器 != 执行机器
  - 开发环境 != 执行环境
- 专门的开发环境与开发工具
  - Linux系统配套的gcc,gdb等开发工具
  - VxWorks系统配套的Tornado集成开发工具

2.6 连接式开发环境(1/3)



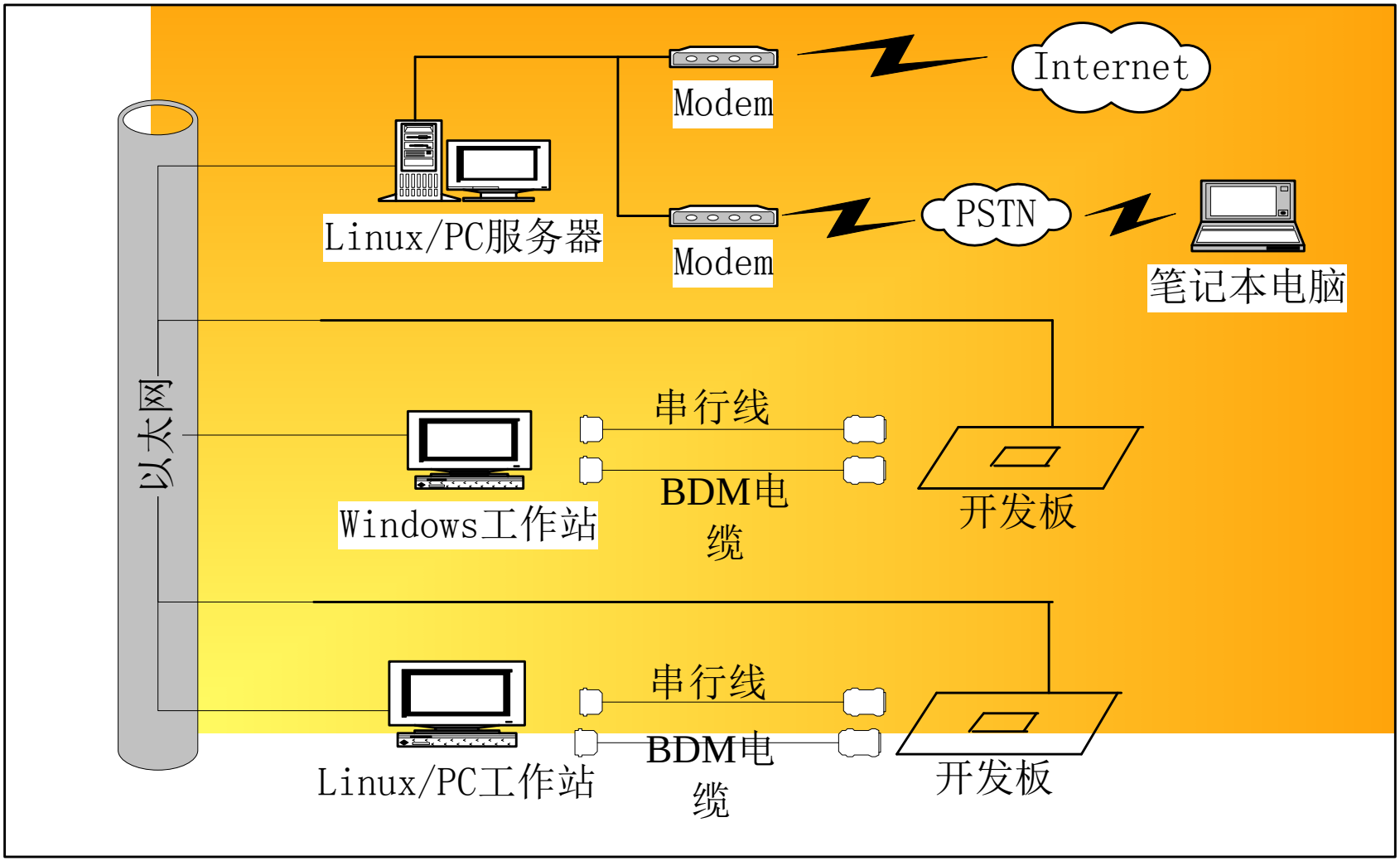
2.6 连接式开发环境(2/3)



2.6 连接式开发环境(3/3)

- 宿主机
  - 开发机器（编辑器、编译器、调试器、….）
- 目标机
  - 程序运行的机器
- 交叉编译是指宿主机和目标机是不同的系统
- Ethernet连接用来下载可执行文件、内核、根文件系统等大型对象。
- RS232连接用来进行调试。

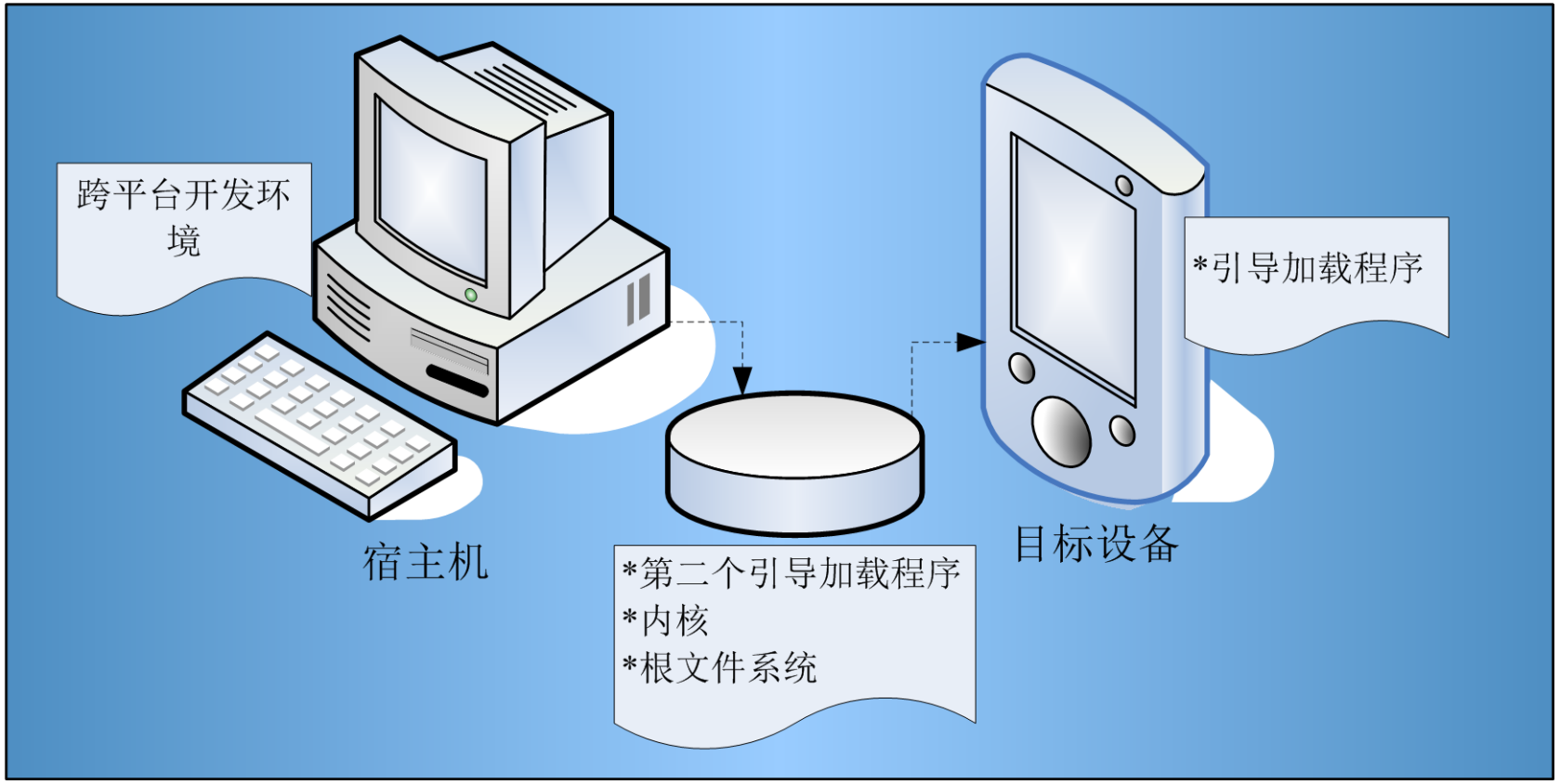
2.7 大型连接式开发环境



2.8 可抽换存储装置开发环境(1/2)

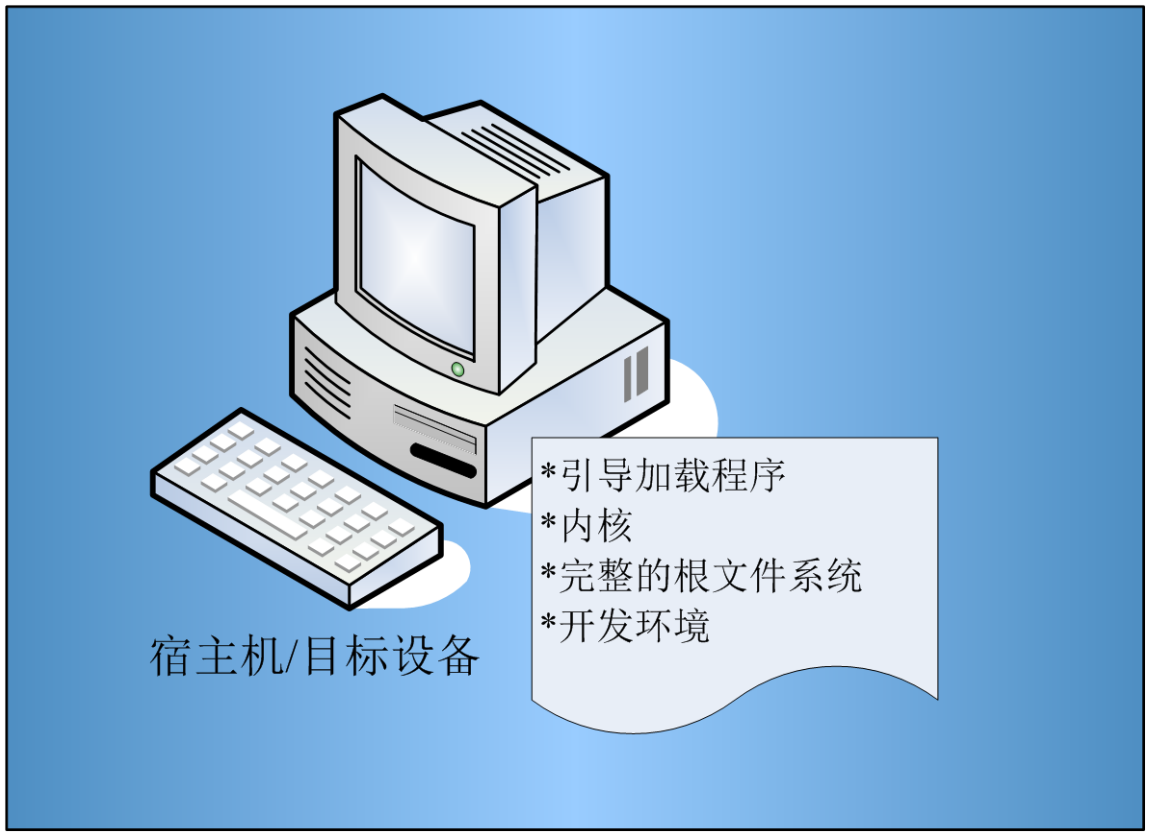
- 主机和目标板之间没有实际连接。
- 主机先将数据写入存储装置，然后将存储装置转接到目标板，并用该存储装置引导。
- 目标板只包含了最起码的引导加载程序，其他组件存放在抽换式存储媒体上。

2.8 可抽换存储装置开发环境(2/2)



2.9 独立式开发环境

- 目标板是一个独立的开发系统，包含了引导、操作以及开发额外软件所必须的软件。
- 非常适合以PC为主的高级嵌入式系统开发应用。



2.10 嵌入式系统开发工具

- 编译器
- 链接器
- 定址器
- 软件仿真
- 调试工具
- 硬件调试器
  - ✓ICE（In-Circuit Emulator）
  - ✓ICD（In-Circuit Debugger）
- 集成开发环境（IDE）

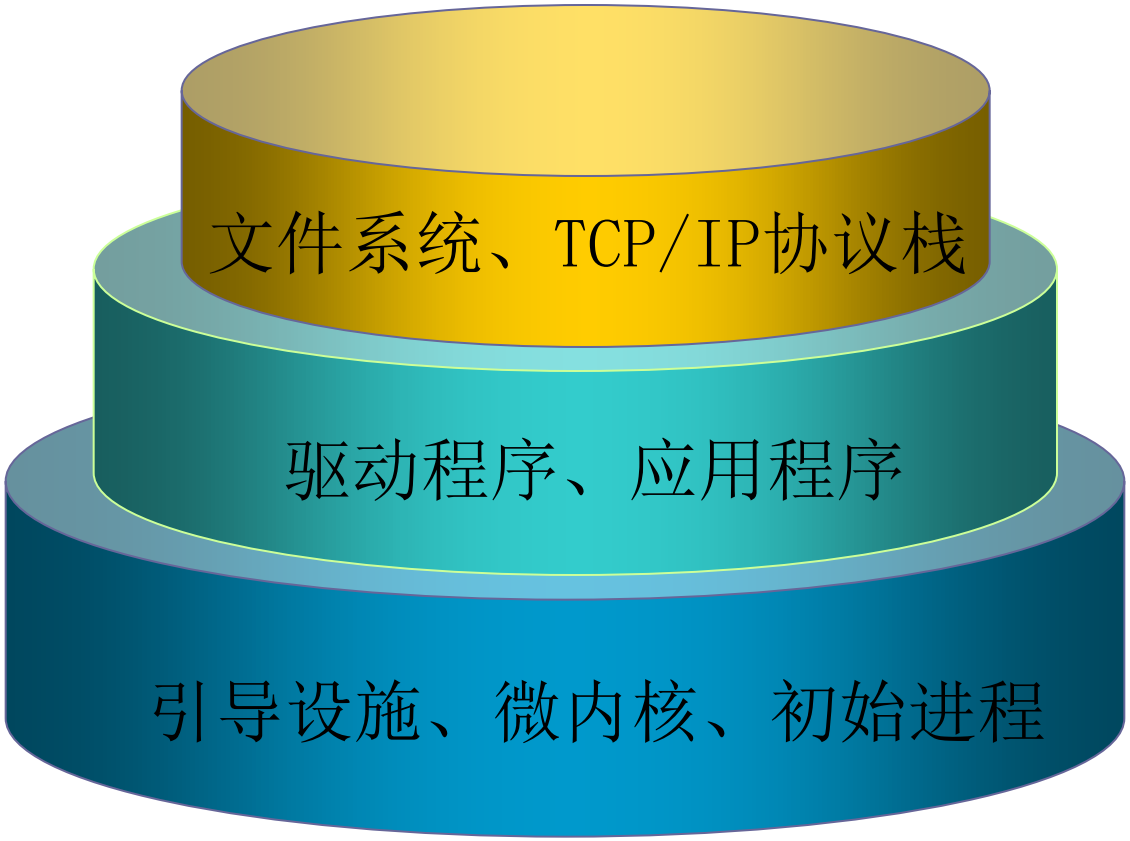
2.11 常见嵌入式软件开发工具

- 典型的商业开发工具及供应商
  - ARM公司的SDT、ADS和Realview
  - Windriver公司的Tornado for VxWorks
  - MontaVisa公司的HardHat Linux
  - Microsoft公司的Platform Builder和eMbedded Visual C++
- 免费开发工具
  - GNU系列开发工具
    - ✓从源代码开始
    - ✓已编译好的二进制工具
  - Wiggler调试器

内容提纲(3/8)

- 1. 嵌入式系统概述
- 2. 嵌入式系统软件开发
- 3. 嵌入式操作系统介绍
- 4. 嵌入式Linux操作系统
- 5. Linux程序设计基础
- 6. GNU make入门
- 7. GDB简介
- 8. 实验系统HUDAX-ARM9-2410简介

3.1 嵌入式操作系统的组成



3.2 嵌入式操作系统的分类

- 多任务操作系统

  并行性、调度性和无序性
- 分时操作系统

  并行性、独立性、交互性和及时性
- 实时操作系统（IEEE实时UNIX分委会的定义）
  - ✓ 异步事件响应
  - ✓ 切换时间和中断延迟时间确定
  - ✓ 抢占式调度
  - ✓ 内存锁定
  - ✓ 优先级中断和调度
  - ✓ 连续文件
  - ✓ 同步

3.3 对实时系统的一些误解

- 误解1：嵌入式系统经常就是实时性系统。其实，很多嵌入式系统并不需要实时性。

- 误解2：实时系统处理速度很快
  - 实时性是一个相对的概念。

3.4 实时系统的特点

- 在实时系统中，系统的正确性不仅仅依赖于计算的逻辑结果而且依赖于结果产生的时间
- 实时操作系统必须在指定的时间内对外部或内部的事件进行响应和处理
- 需要高效的中断处理能力来处理异步事件和高效的I/O能力来处理有严格时间限制的数据收发应用

3.5 实时操作系统的性能参数

- 系统响应时间(System response time)

  系统发出处理要求到系统给出应答信号的时间。

- 任务切换时间(Context-switching time)

  是任务之间切换而使用的时间。

- 中断延迟(Interrupt latency)

  是计算机接收到中断信号到操作系统作出响应，并转入中断服务程序的时间。

3.6 实时操作系统的基本功能

- 任务管理
  （多任务和基于优先级的任务调度）
- 任务间同步和通信
  （信号量和共享内存等）
- 存储器优化管理（含ROM的管理）
- 中断管理服务

3.7 嵌入式操作系统的特点

- 体积小
- 运行时间长
- 故障重启
- 低功耗
- 价格便宜
- 动态加载

3.8 典型的嵌入式操作系统

- 嵌入式Linux
- VxWorks
- Windows CE
- Palm OS
- Symbian OS
- μC/OS-II
- …

内容提纲(4/8)

- 1. 嵌入式系统概述
- 2. 嵌入式系统软件开发
- 3. 嵌入式操作系统介绍
- 4. 嵌入式Linux操作系统
- 5. Linux程序设计基础
- 6. GNU make入门
- 7. GDB简介
- 8. 实验系统HUDAX-ARM9-2410简介

4.1 GNU/Linux操作系统

GNU代表GNU's Not Unix。它既是一个操作系统，也是一种规范。

Linux是一个内核, 然而一个完整的操作系统不仅仅是内核而已。

Linux是一套免费使用和自由传播的类Unix操作系统。我们通常所说的Linux，指的是GNU/Linux，即采用Linux内核的GNU操作系统。

Linux的理念 “自由、开源、团结互助”

4.2 Linux操作系统的版本

- Linux最早是Linus Torvalds于1991年在芬兰赫尔辛基大学原创开发的，并在GNU的GPL（General Public License）原则下发行。
- Linux的版本号又分为两部分：内核（Kernel）版本和发行（Distribution）版本。内核版本的序号由3部分数字构成，其形式如下：

  major.minor.patchlevel

  如：2.4.18　2.6.14

4.3 Linux操作系统的常见自由发行版(1/5)

4.3 Linux操作系统的常见自由发行版(2/5)

Debian最早由Ian Murdock于1993年创建。可以算是迄今为止，最遵循GNU规范的Linux系统。

Debian是Linux发行版当中比较自由的一种，由位于世界各地上千名的自愿者不断开发和维护。它不属于任何的商业公司，完全由开源社区所有。Debian GNU/Linux不单是个操作系统，它也包含多过15490个软件包，它们是一些经已编译的软件，并包装成一个容易安装的格式。dpkg是Debian系列特有的软件包管理工具，它被誉为所有Linux软件包管理工具（比如RPM）最强大的！配合apt-get，在Debian上安装、升级、删除和管理软件变得异常容易。

4.3 Linux操作系统的常见自由发行版(3/5)

Ubuntu是一个南非的民族观念，着眼于人们之间的忠诚和联系。该词来自于祖鲁语和科萨语。Ubuntu精神的大意是“人道待人”（对他人仁慈）。另一种翻译可以是：“天下共享的信念，连接起每个人”。

Ubuntu项目完全遵从开源软件开发的原则；并且鼓励人们使用、完善并传播开源软件。也就是说Ubuntu目前是并将永远是免费的。Ubuntu是基于Debian之上，旨在创建一个可以为桌面和服务器提供一个最新且一贯的Linux系统。Ubuntu囊括了大量精挑细选自Debian发行版的软件包，同时保留了Debian强大的软件包管理系统，以便简易的安装或彻底的删除程序。与大多数发行版附带数量巨大的可用可不用的软件不同，Ubuntu的软件包清单只包含那些高质量的重要应用程序。

4.3 Linux操作系统的常见自由发行版(4/5)

Slackware由Patrick Volkerding创建于1992年，算起来应当是历史最悠久的Linux发行版。

优点：非常稳定、安全，高度坚持UNIX的规范

缺点：所有配置通过编辑文件进行，自动硬件检测能力差

软件包管理系统：Slackware Package Management (TGZ)

Fedora项目是由Red Hat赞助，由开源社区与Red Hat工程师合作开发的项目统称。

全世界的Linux用户最熟悉、最耳闻能详的发行版想必就是Red Hat了。正统的Red Hat版本早已停止技术支持，最后一版是Red Hat 9.0。目前Red Hat分为两个系列：由Red Hat公司提供收费技术支持和更新的Red Hat Enterprise Linux，以及由社区开发的免费的Fedora Core。

4.3 Linux操作系统的常见自由发行版(5/5)

德国最著名的Linux发行版

快速、设计干净而有弹性，是一个现代模式的发行版。

还有更多。。。

4.4 Linux的特点

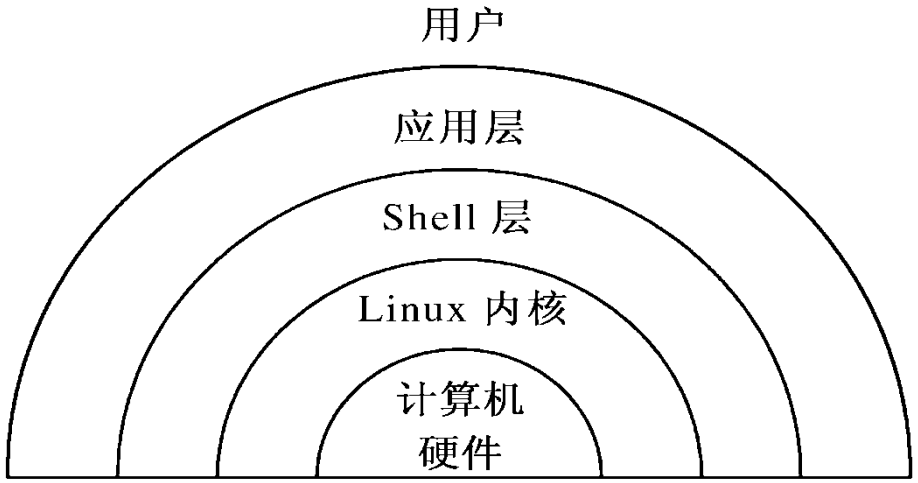
1. 多任务、多用户
2. 支持多种文件系统
3. 采用虚拟内存管理技术
4. 良好的可移植性
5. 设备独立性
6. 丰富的网络功能
7. 提供全部源代码

4.5 Linux的功能

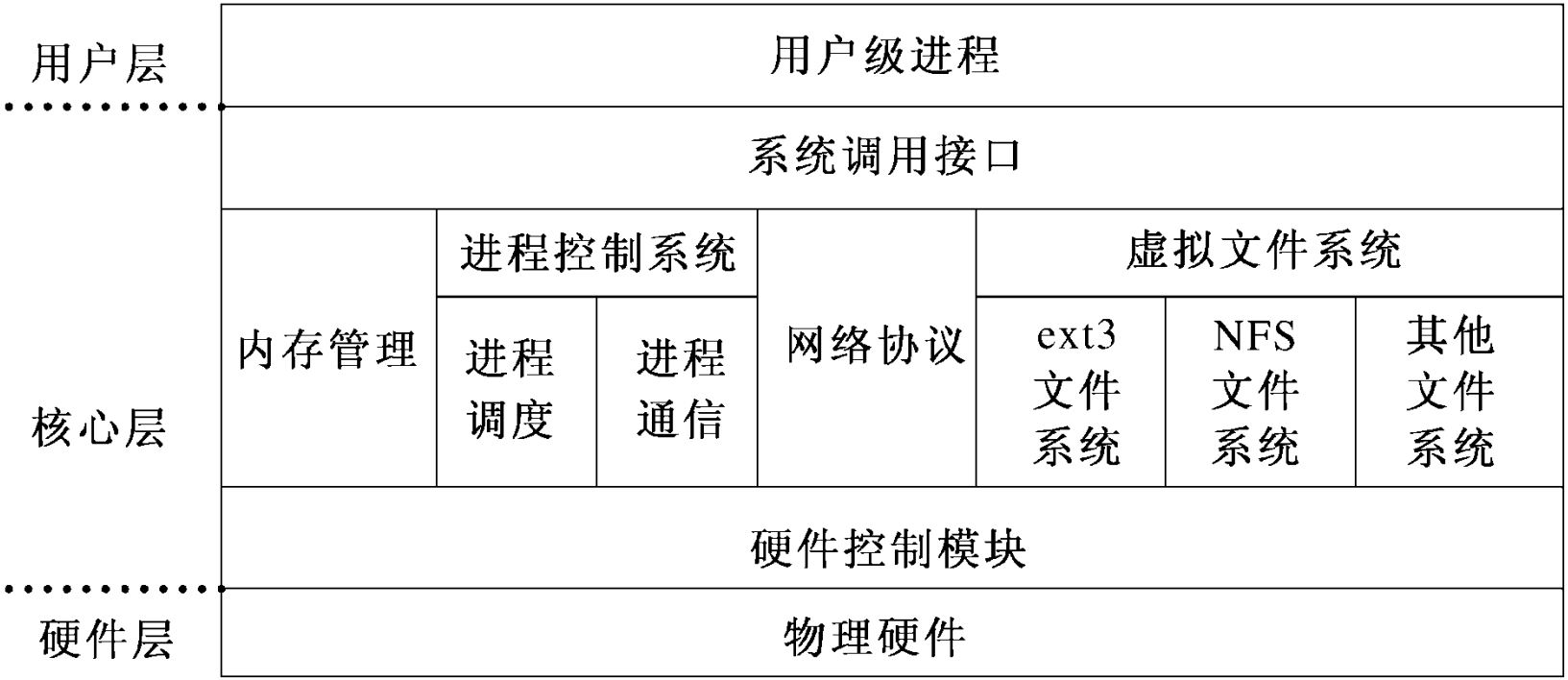
1. 稳定的核心
2. 丰富的应用软件
3. X Window系统
4. 多重启动
5. 网络功能
6. 软件开发工具

4.6 Linux内核体系结构

- 与UNIX系统相似，Linux系统大致可分为3层：靠近硬件的底层是内核，即Linux操作系统的常驻内存部分；中间层是内核之外的Shell层，亦即操作系统的系统程序部分；最高层是应用层，即用户程序部分，包括各种文本处理程序、语言编译程序及游戏程序等。Linux的系统结构如图所示。



4.7 Linux系统的核心框图



4.8 嵌入式Linux操作系统

- 符合POSIX 1003.1标准
- 支持多用户访问和多任务编程
- 采用页式存储管理
- 支持动态链接
- 支持多种文件系统
- 支持TCP/IP、PPP等网络功能
- 支持多种体系结构
- 采用微内核，其大小功能可以定制

4.9 嵌入式Linux的特点(1/3)

	专用嵌入式实时操作系统	嵌入式 Linux 操作系统
版权费	每生产一件产品需交纳一份版权费	免费
购买费用	数十万元（RMB）	免费
技术支持	由开发商独家提供有限的技术支持	全世界的自由软件开发者提供支持
网络特性	另加数十万元（RMB）购买	免费且性能优异
软件移值	难（因为是封闭系统）	易，代码开放（有许多应用软件支持）
应用产品开发周期	长，因为可参考的代码有限	短，新产品上市迅速，因为有许多公开的代码可以参考和移植
实时性能	好	须改进，可用 PT_Linux 等模块弥补
稳定性	较好	较好，但在高性能系统中须改进

4.9 嵌入式Linux的特点(2/3)

- 开放源码，丰富的软件资源，广泛的软件开发者的支持，价格低廉，结构灵活，适用面广。
- 精简的内核，性能高、稳定，多任务。
- 适用于不同的CPU，支持多种体系结构，如X86、ARM、MIPS、ALPHA、SPARC等。
- 能够提供完善的嵌入式GUI以及嵌入式XWindow。
- 提供嵌入式浏览器、邮件程序、MP3播放器、MPEG播放器、记事本等应用程序。

4.9 嵌入式Linux的特点(3/3)

- 提供完整的开发工具和SDK，同时提供PC上的开发版本。
- 用户可定制，可提供图形化的定制和配置工具。
- 常用嵌入式芯片的驱动集，支持大量的周边硬件设备，驱动丰富。
- 针对嵌入式的存储方案，提供实时版本和完善的嵌入式解决方案。
- 完善的中文支持，强大的技术支持，完整的文档。

4.10 嵌入式Linux的优势

- Linux系统是层次结构且内核完全开放
- 强大的网络支持功能
- Linux具备一整套工具链，容易自行建立嵌入式系统的开发环境和交叉运行环境，并且可以跨越嵌入式系统开发中仿真工具的障碍
- Linux具有广泛的硬件支持特性

内容提纲(5/8)

- 1.嵌入式系统概述
- 2.嵌入式系统软件开发
- 3.嵌入式操作系统介绍
- 4.嵌入式Linux操作系统
- 5. Linux程序设计基础
- 6. GNU make入门
- 7. GDB简介
- 8. 实验系统HUDAX-ARM9-2410简介

5.1 GNU编程风格(1/2)

1. 函数返回类型说明和函数名分两行放置，函数起始字符和函数开头左花括号放到最左边。
2. 尽量不要让两个不同优先级的操作符出现在相同的对齐方式中，应该附加额外的括号使得代码缩进可以表示出嵌套。
3. 按照规定方式排版do-while语句：
4. 每个程序都应该以一段简短的说明其功能的注释开头。
5. 请为每个函数书写注释，说明函数是做什么的，需要哪些入口参数，参数可能值的含义和用途。如果用了非常见的、非标准的东西，或者可能导致函数不能工作的任何可能的值，应该进行特殊说明。如果存在重要的返回值，也需要说明。

5.1 GNU编程风格(2/2)

6. 不要声明多个变量时跨行，每一行都以一个新的声明开头。
7. 当一个if中嵌套了另一个if-else时，应用花括号把if-else括起来。
8. 要在同一个声明中同时说明结构标识和变量或者结构标识和类型定义(typedef)。先定义变量，再使用。
9. 尽量避免在if的条件中进行赋值。
10. 请在名字中使用下划线以分割单词，尽量使用小写；把大写字母留给宏和枚举常量，以及根据统一惯例使用的前缀。例如，应该使用类似ignore_space_change_flag的名字；不要使用类似iCantReadThis的名字。

5.2 Linux内核编程风格

1. Linux内核缩进风格是8个字符。
2. Linux内核风格采用K&R标准，将开始的大括号放在一行的最后，而将结束的大括号放在一行的第一位。
3. 命名尽量简洁。不应该使用诸如ThisVariableIsATemporaryCounter之类的名字。应该命名为tmp，这样容易书写，也不难理解。但是命名全局变量，就应该用描述性命名方式，例如应该命名“count_active_users()”，而不是“cntusr()”。本地变量应该避免过长。

5.3 编辑器vi的使用

- vi提供了一些功能强大的但容易记忆的命令供用户使用。类似这样的编辑任务在vi中可以轻松高效完成。vi有3种模式：

1. 命令行模式：用户在用vi编辑文件时，最初进入的模式。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。

5.4 vi的编辑命令

命 令	功 能
[N]x	(Expurgate)删除从光标位置开始的连续N个字符（并复制到编辑缓冲区）
[N]dd	(Delete)删除从光标位置开始的连续N行（并复制到编辑缓冲区）
[N]yy	(Yank)复制从光标位置开始的连续N行到编辑缓冲区
p	(Put)从编辑缓冲区复制文本到当前光标位置（即粘贴）
u	(Undo)取消上一次操作（即恢复功能）

5.5 vi的光标命令

命 令	功 能
h	方向键，向左移动光标一个字符的位置，相当于键“←”
j	方向键，向下移动光标到下一行的位置，相当于键“↓”
k	方向键，向上移动光标到上一行的位置，相当于键“↑”
l	方向键，向右移动光标一个字符的位置，相当于键“→”
:N	移动光标到第N行（N待定）
1G	移动光标到文件的第1行
G	移动光标到文件的最后1行
:set number	设置显示行号
:set nonumber	取消显示行号

5.6 vi的文件命令

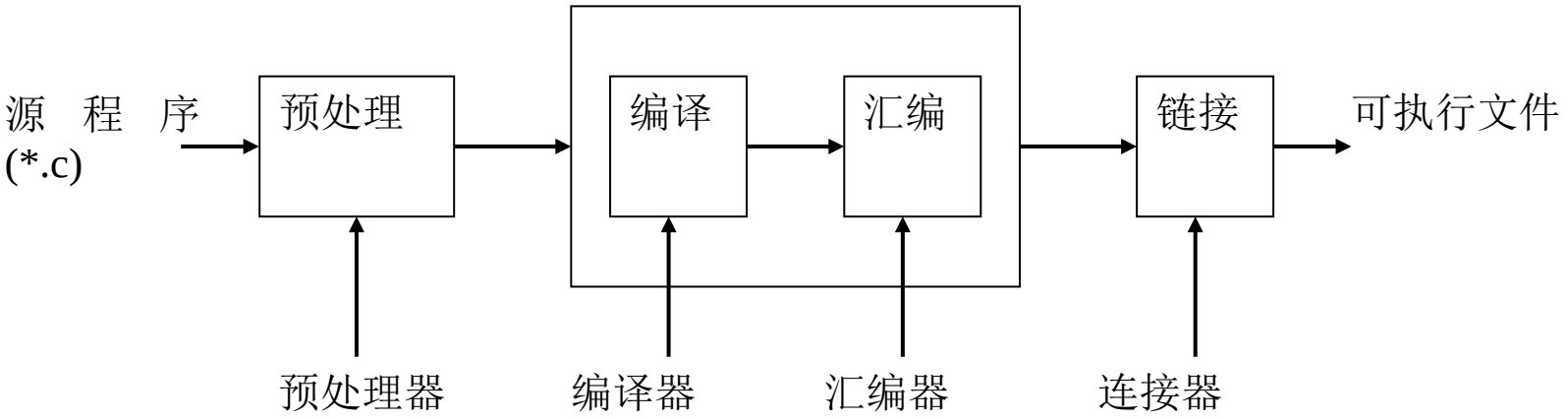
命 令	功 能
:q	(Quit)退出没有被修改的文件 （若文件被修改了而没有保存，则此命令无效）
:q!	强制退出，且不保存修改过的部分
:w	(Write)保存文件，但不退出
:w!	强制保存文件，但不退出
:x	(Exit)保存文件并退出
:x!	强制保存文件并退出
:w File	另存为File给出的文件名，不退出
:w! File	强制另存为File给出的文件名，不退出
:r File	(Read)读入File指定的文件内容插入到光标位置

5.7 vi的状态命令

命 令	功 能
a	(Append)进入编辑状态，从当前光标之后的位置开始插入键盘输入的字符
i	(Insert)进入编辑状态，从当前光标之后的位置开始插入键盘输入的字符
o	(Open)进入编辑状态，并插入一新行，光标移到该新行的行首，以后键盘输入的字符将插入到光标位置
ESC	进入命令状态
:! Command	在vi中执行外部命令Command，按回车键可以返回vi继续工作

5.8 使用GNU cc开发应用程序

- gcc可以使程序员灵活地控制编译过程。编译过程一般可以分为下面四个阶段，每个阶段分别调用不同的工具进行处理。



5.9 gcc的版本信息

- 一般来说，系统安装后就已经安装和设定好了gcc。在shell的提示符下键入gcc -v，屏幕上就会显示出目前正在使用的gcc的版本，同时这可以确定系统所支持的是ELF还是a.out可执行文件格式。

5.10 Linux可执行文件的格式

- Linux系统中可执行文件有两种格式：
  - a.out格式，这种格式用于早期的Linux系统以及 Unix系统的原始格式。a.out来自于Unix C编译程序默认的可执行文件名。当使用共享库时，a.out格式就会发生问题。把a.out格式调整为共享库是一种非常复杂的操作，由于这个原因，一种新的文件格式被引入Unix系统5的第四版本和Solaris系统中。它被称为可执行和连接的格式（ELF）。这种格式很容易实现共享库。
  - ELF格式，已经被Linux系统作为标准的格式采用。gcc编译程序产生的所有的二进制文件都是ELF格式的文件（即使可执行文件的默认名仍然是a.out）。较旧的a.out格式的程序仍然可以运行在支持ELF格式的系统上。

5.11 gcc的使用

- gcc的使用格式如下：
  - $ gcc [options][filenames]
  - 其中filenames为所要编译的程序源文件。
  - 当使用gcc时，gcc会完成预处理、编译、汇编和连接。前三步分别生成目标文件，连接时，把生成的目标文件链接成可执行文件。gcc可以针对支持不同的源程序文件进行不同处理，文件格式以文件的后缀来识别。

5.12 gcc的选项

表 3.7　　Gcc 总体选项列表

后　缀　名	所对应的语言
-c	只是编译不链接，生成目标文件“.o”
-S	只是编译不汇编，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	把输出文件输出到 file 里
-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
-L dir	在库文件的搜索路径列表中添加 dir 目录
-static	链接静态库
-llibrary	连接名为 library 的库文件

5.13 gcc的优化选项

- 当用gcc编译C代码时，它会试着用最少的时间完成编译并且使编译后的代码易于调试. 易于调试意味着编译后的代码与源代码有同样的执行次序，编译后的代码没有经过优化。有很多选项可用于告诉gcc，在耗费更多编译时间和牺牲易调试性的基础上，产生更小更快的可执行文件。这些选项中最典型的是-O和-O2选项。
- -O选项告诉gcc对源代码进行基本优化。这些优化在大多数情况下都会使程序执行的更快。
- -O2选项告诉gcc产生尽可能小和尽可能快的代码。O2选项将使编译的速度比使用O选项时慢。但通常产生的代码执行速度会更快。

5.14 gcc的调试和剖析选项

- GCC 支持数种调试和剖析选项。在这些选项里最常用的是-g和-pg选项。
- -g选项告诉gcc产生能被GNU调试器使用的调试信息以便调试程序。gcc 提供了一个很多其他C编译器里没有的特性，在gcc里能使-g和-O(产生优化代码)连用。这一点非常有用，因为能在与最终产品尽可能相近的情况下调试代码。同时使用这两个选项时必须清楚所写的某些代码已经在优化时被gcc作了改动。
- -pg选项告诉gcc在程序里加入额外的代码，执行时，产生gprof用的剖析信息以显示程序的耗时情况。

内容提纲(6/8)

- 1.嵌入式系统概述
- 2.嵌入式系统软件开发
- 3.嵌入式操作系统介绍
- 4.嵌入式Linux操作系统
- 5. Linux程序设计基础
- 6. GNU make入门
- 7. GDB简介
- 8. 实验系统HUDAX-ARM9-2410简介

6.1 使用GNU make

- 要使用make，必须编写一个叫做makefile的文件，这个文件描述了软件包中文件之间的关系，提供更新每个文件的命令。一般在一个软件包里，通常是可执行文件靠目标文件来更新，目标文件靠编译源文件来更新。
- makefile写好之后，每次改变了某些源文件，只要执行make命令：

  # make

- 所有必要的重新编译将执行。make程序利用makefile中的数据和每个文件的最后修改时间来确定那个文件需要更新，对于需要更新的文件，make程序执行文件makefile中定义的命令来更新。

6.2 makefile文件的基本结构(1/4)

- GNU make的主要功能是读进一个文本文件makefile并根据makefile的内容执行一系列的工作。makefile的默认文件名为GNUmakefile、makefile或Makefile，当然也可以在make的命令行中指定别的文件名。如果不特别指定，make命令在执行时将按顺序查找默认的makefile文件。多数Linux程序员使用第三种文件名Makefile。因为第一个字母是大写，通常被列在一个目录的文件列表的最前面。

6.2 makefile文件的基本结构(2/4)

- Makefile是一个文本形式的数据库文件，其中包含一些规则来告诉make处理哪些文件以及如何处理这些文件。这些规则主要是描述哪些文件（称为target目标文件，不要和编译时产生的目标文件相混淆）是从哪些别的文件（称为dependency依赖文件）中产生的，以及用什么命令（command）来执行这个过程。

6.2 makefile文件的基本结构(3/4)

- 依靠这些信息，make会对磁盘上的文件进行检查，如果目标文件的生成或被改动时的时间（称为该文件时间戳）至少比它的一个依赖文件还旧的话，make就执行相应的命令，以更新目标文件。目标文件不一定是最后的可执行文件，可以是任何一个中间文件并可以作为其他目标文件的依赖文件。

6.2 makefile文件的基本结构(4/4)

- 一个Makefile文件主要含有一系列的规则，每条规则包含以下内容。
- 一个目标（target），即make最终需要创建的文件，如可执行文件和目标文件；目标也可以是要执行的动作，如“clean”。
- 一个或多个依赖文件（dependency）列表，通常是编译目标文件所需要的其他文件。
- 一系列命令(command)，是make执行的动作，通常是把指定的相关文件编译成目标文件的编译命令，每个命令占一行，且每个命令行的起始字符必须为TAB字符。

6.3 一个简单的Makefile的例子(1/7)

```
# 以#开头的为注释行
test：  prog.o code.o
        gcc –o test prog.o code.o
 prog.o：  prog.c prog.h code.h
        gcc –c prog.c –o prog.o
 code.o：  code.c code.h
        gcc –c code.c –o code.o
 clean：
        rm –f *.o
```

6.3 一个简单的Makefile的例子(2/7)

上面的Makefile文件中共定义了四个目标：test、prog.o、code.o和clean。目标从每行的最左边开始写，后面跟一个冒号（：），如果有与这个目标有依赖性的其他目标或文件，把它们列在冒号后面，并以空格隔开。然后另起一行开始写实现这个目标的一组命令。在Makefile中，可使用续行号（\）将一个单独的命令行延续成几行。但要注意在续行号（\）后面不能跟任何字符（包括空格和键）。

6.3 一个简单的Makefile的例子(3/7)

一般情况下，调用make命令可输入：

make target

- target是Makefile文件中定义的目标之一，如果省略target，make就将生成Makefile文件中定义的第一个目标。对于上面Makefile的例子，单独的一个“make”命令等价于：

  # make test

- 因为test是Makefile文件中定义的第一个目标，make首先将其读入，然后从第一行开始执行，把第一个目标test作为它的最终目标，所有后面的目标的更新都会影响到test的更新。第一条规则说明只要文件test的时间戳比文件prog.o或code.o中的任何一个旧，下一行的编译命令将会被执行。

6.3 一个简单的Makefile的例子(4/7)

- 但是，在检查文件prog.o和code.o的时间戳之前，make会在下面的行中寻找以prog.o和code.o为目标的规则，在第三行中找到了关于prog.o的规则，该文件的依赖文件是prog.c、prog.h和code.h。同样，make会在后面的规则行中继续查找这些依赖文件的规则，如果找不到，则开始检查这些依赖文件的时间戳，如果这些文件中任何一个的时间戳比prog.o的新，make将执行“gcc –c prog.c –o prog.o”命令，更新prog.o文件。

6.3 一个简单的Makefile的例子(5/7)

- 以同样的方法，接下来对文件code.o做类似的检查，依赖文件是code.c和code.h。当make执行完所有这些套嵌的规则后，make将处理最顶层的test规则。如果关于prog.o和code.o的两个规则中的任何一个被执行，至少其中一个.o目标文件就会比test新，那么就要执行test规则中的命令，因此make去执行gcc命令将prog.o和code.o连接成目标文件test。
- 在上面Makefile的例子中，还定义了一个目标clean，它是Makefile中常用的一种专用目标，即删除所有的目标模块。

6.3 一个简单的Makefile的例子(6/7)

- 现在来看一下make做的工作：首先make按顺序读取makefile中的规则，然后检查该规则中的依赖文件与目标文件的时间戳哪个更新，如果目标文件的时间戳比依赖文件还早，就按规则中定义的命令更新目标文件。如果该规则中的依赖文件又是其他规则中的目标文件，那么依照规则链不断执行这个过程，直到Makefile文件的结束，至少可以找到一个不是规则生成的最终依赖文件，获得此文件的时间戳，然后从下到上依照规则链执行目标文件的时间戳比此文件时间戳旧的规则，直到最顶层的规则。

6.3 一个简单的Makefile的例子(7/7)

- 通过以上的分析过程，可以看到make的优点，因为.o目标文件依赖.c源文件，源码文件里一个简单改变都会造成那个文件被重新编译，并根据规则链依次由下到上执行编译过程，直到最终的可执行文件被重新连接。例如，当改变一个头文件的时候，由于所有的依赖关系都在Makefile里，因此不再需要记住依赖此头文件的所有源码文件，make可以自动的重新编译所有那些因依赖这个头文件而改变了的源码文件，如果需要，再进行重新连接。

6.4 Makefile中的变量(1/7)

- Makefile里的变量就像一个环境变量。事实上，环境变量在make中也被解释成make的变量。这些变量对大小写敏感，一般使用大写字母。几乎可以从任何地方引用定义的变量，变量的主要作用如下：
- 保存文件名列表。在前面的例子里，作为依赖文件的一些目标文件名出现在可执行文件的规则中，而在这个规则的命令行里同样包含这些文件并传递给gcc做为命令参数。如果使用一个变量来保存所有的目标文件名，则可以方便地加入新的目标文件而且不易出错。

6.4 Makefile中的变量(2/7)

- 保存可执行命令名，如编译器。在不同的Linux系统中存在着很多相似的编译器系统，这些系统在某些地方会有细微的差别，如果项目被用在一个非gcc的系统里，则必须将所有出现编译器名的地方改成用新的编译器名。但是如果使用一个变量来代替编译器名，那么只需要改变该变量的值。其他所有地方的命令名就都改变了。

6.4 Makefile中的变量(3/7)

- 保存编译器的参数。在很多源代码编译时，gcc需要很长的参数选项，在很多情况下，所有的编译命令使用一组相同的选项，如果把这组选项使用一个变量代表，那么可以把这个变量放在所有引用编译器的地方。当要改变选项的时候，只需改变一次这个变量的内容即可。

6.4 Makefile中的变量(4/7)

- Makefile中的变量是用一个文本串在Makefile中定义的，这个文本串就是变量的值。只要在一行的开始写下这个变量的名字，后面跟一个“＝”号，以及要设定这个变量的值即可定义变量，下面是定义变量的语法：

  VARNAME=string

- 使用时，把变量用括号括起来，并在前面加上$符号，就可以引用变量的值：

  ${VARNAME}

6.4 Makefile中的变量(5/7)

- make解释规则时，VARNAME在等式右端展开为定义它的字符串。变量一般都在Makefile的头部定义。按照惯例，所有的Makefile变量都应该是大写。如果变量的值发生变化，就只需要在一个地方修改，从而简化了Makefile的维护。

6.4 Makefile中的变量(6/7)

- 现在利用变量把前面的Makefile重写一遍：

```
OBJS=prog.o code.o
CC=gcc
test：${ OBJS }
        ${ CC } –o test ${ OBJS }
prog.o：prog.c prog.h code.h
        ${ CC } –c prog.c –o prog.o
code.o：code.c code.h
        ${ CC } –c code.c –o code.o
clean：
        rm –f *.o
```

6.4 Makefile中的变量(7/7)

- 除用户自定义的变量外，make还允许使用环境变量、自动变量和预定义变量。使用环境变量的方法很简单，在make启动时，make读取系统当前已定义的环境变量，并且创建与之同名同值的变量，因此用户可以像在shell中一样在Makefile中方便的引用环境变量。需要注意的是，如果用户在Makefile中定义了同名的变量，用户自定义变量将覆盖同名的环境变量。此外，Makefile中还有一些预定义变量和自动变量，但是看起来并不像自定义变量那样直观。

6.5 Makefile的隐含规则(1/2)

- 在上面的例子中，几个产生目标文件的命令都是从“.c”的C语言源文件和相关文件通过编译产生“.o”目标文件，这也是一般的步骤。实际上，make可以使工作更加自动化，也就是说，make知道一些默认的动作，它有一些称作隐含规则的内置的规则，这些规则告诉make当用户没有完整地给出某些命令的时候，应该怎样执行。

6.5 Makefile的隐含规则(2/2)

- 例如，把生成prog.o和code.o的命令从规则中删除，make将会查找隐含规则，然后会找到并执行一个适当的命令。由于这些命令会使用一些变量，因此可以通过改变这些变量来定制make。象在前面的例子中所定义的那样，make使用变量CC来定义编译器，并且传递变量CFLAGS（编译器参数）、CPPFLAGS（C语言预处理器参数）、TARGET_ARCH（目标机器的结构定义）给编译器，然后加上参数-c，后面跟变量$<（第一个依赖文件名），然后是参数-o加变量$@（目标文件名）。综上所述，一个C编译的具体命令将会是：
- $ {CC} $ {CFLAGS} $ {CPPFLAGS} $ {TARGET_ARCH} –c $< -o $@

6.6 Makefile中的自动变量(1/2)

表 3.15　　Makefile 中常见自动变量

命 令 格 式	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

6.6 Makefile中的自动变量(2/2)

- 在上面的例子中，利用隐含规则，可以简化为：

```
OBJS=prog.o code.o
CC=gcc
test：  ${ OBJS }
        ${ CC } –o $@ $^
 prog.o：  prog.c prog.h code.h
code.o：  code.c code.h
 clean：
        rm –f *.o
```

内容提纲(7/8)

- 1.嵌入式系统概述
- 2.嵌入式系统软件开发
- 3.嵌入式操作系统介绍
- 4.嵌入式Linux操作系统
- 5. Linux程序设计基础
- 6. GNU make入门
- 7. GDB简介
- 8. 实验系统HUDAX-ARM9-2410简介

7.1 调试工具GDB(1/2)

- Linux系统中包含了GNU 调试程序gdb，它是一个用来调试C和 C++ 程序的调试器。可以使程序开发者在程序运行时观察程序的内部结构和内存的使用情况。gdb 所提供的一些功能如下所示：
  - 运行程序，设置所有的能影响程序运行的参数和环境；
  - 控制程序在指定的条件下停止运行；
  - 当程序停止时，可以检查程序的状态；
  - 修改程序的错误，并重新运行程序；
  - 动态监视程序中变量的值；
  - 可以单步执行代码，观察程序的运行状态。

7.1 调试工具GDB(2/2)

- Linux下软件调试工具
- 远端调试
  - 通过串口或网口调试目标设备
  - target命令：指定调试目标和建立连接的GDB命令

```
$ gdb
GNU gdb 2003-09-02-cvs (cygwin-special)
Copyright 2003 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 "i686-pc-cygwin".
(gdb) help target
Connect to a target machine or process.
The first argument is the type or protocol of the target machine.
Remaining arguments are interpreted by the target protocol.  For more
information on the arguments for a particular protocol, type
`help target ' followed by the protocol name.

List of target subcommands:

target async -- Use a remote computer via a serial line
target child -- Win32 child process (started by the "run" command)
target cisco -- Use a remote machine via TCP
target core -- Use a core file as a target
target exec -- Use an executable file as a target
target extended-async -- Use a remote computer via a serial line
target extended-remote -- Use a remote computer via a serial line
target remote -- Use a remote computer via a serial line

Type "help target" followed by target subcommand name for full documentation.
Command name abbreviations are allowed if unambiguous.
(gdb)
```

7.2 关于GDB的几点说明

- gdb的功能非常强大，到目前为止，gdb已能够支持Moduls-2、Chill、Pascal和FORTRAN程序的调试，但是调试这些语言的源程序时有一些功能还不能使用。例如调试FORTRAN程序时还不支持表达式的输入、输出变量或类FORTRAN的词法。
- gdb程序调试的对象是可执行文件，而不是程序的源代码文件。然而，并不是所有的可执行文件都可以用gdb调试。如果要让产生的可执行文件可以用来调试，需在执行gcc指令编译程序时，加上-g参数，指定程序在编译时包含调试信息。调试信息包含程序里的每个变量的类型和在可执行文件里的地址映射以及源代码的行号。gdb 利用这些信息使源代码和机器码相关联。

7.3 gdb的使用方法

- 用下面的方式来运行gdb:

  gdb filename

  - 其中，filename是要调试的可执行文件。用这种方式运行gdb可以直接指定想要调试的程序。这和启动gdb后执行file filename命令效果完全一样。也可以用gdb去检查一个因程序异常终止而产生的core文件，或者与一个正在运行的程序相连。
  - gdb支持很多的命令且能实现不同的功能。这些命令从简单的文件装入到允许你检查所调用的堆栈内容的复杂命令, 下面列出了在使用gdb 调试时会用到的一些命令。

7.4 gdb的基本命令(1/4)

1. file命令：装入想要调试的可执行文件。
2. cd命令：改变工作目录。
3. pwd命令：返回当前工作日录。
4. run命令：执行当前被调试的程序。
5. kill命令：停止正在调试的应用程序。
6. list命令：列出正在调试的应用程序的源代码。
7. break命令：设置断点。

7.4 gdb的基本命令(2/4)

8. Tbreak命令；设置临时断点。它的语法与break相同。区别在于用tbreak设置的断点执行一次之后立即消失。
9. watch命令：设置监视点，监视表达式的变化。
10. awatch命令：设置读写监视点。当要监视的表达式被读或写时将应用程序挂起。它的语法与watch命令相同。
11. rwatch命令：设置读监视点，当监视表达式被读时将程序挂起，等待调试。此命令的语法与watch相同。
12. next命令：执行下一条源代码，但是不进入函数内部。也就是说，将一条函数调用作为一条语句执行。执行这个命令的前提是已经run，开始了代码的执行。

7.4 gdb的基本命令(3/4)

13. step命令：执行下一条源代码，进入函数内部。如果调用了某个函数，会跳到函数所在的代码中等候一步步执行。执行这个命令的前提是已经用run开始执行代码。
14. display命令：在应用程序每次停止运行时显示表达式的值。
15. info break命令：显示当前断点列表，包括每个断点到达的次数。
16. info files命令：显示调试文件的信息。
17. info func命令：显示所有的函数名。
18. info local命令：显示当前函数的所有局部变量的信息。

7.4 gdb的基本命令(4/4)

19. info prog命令：显示调试程序的执行状态。
20. print命令；显示表达式的值。
21. delete命令：删除断点。指定一个断点号码，则删除指定断点。不指定参数则删除所有的断点。
22. shell命令：执行Linux Shell命令。
23. make命令：不退出gdb而重新编译生成可执行文件。
24. quit命令：退出gdb。

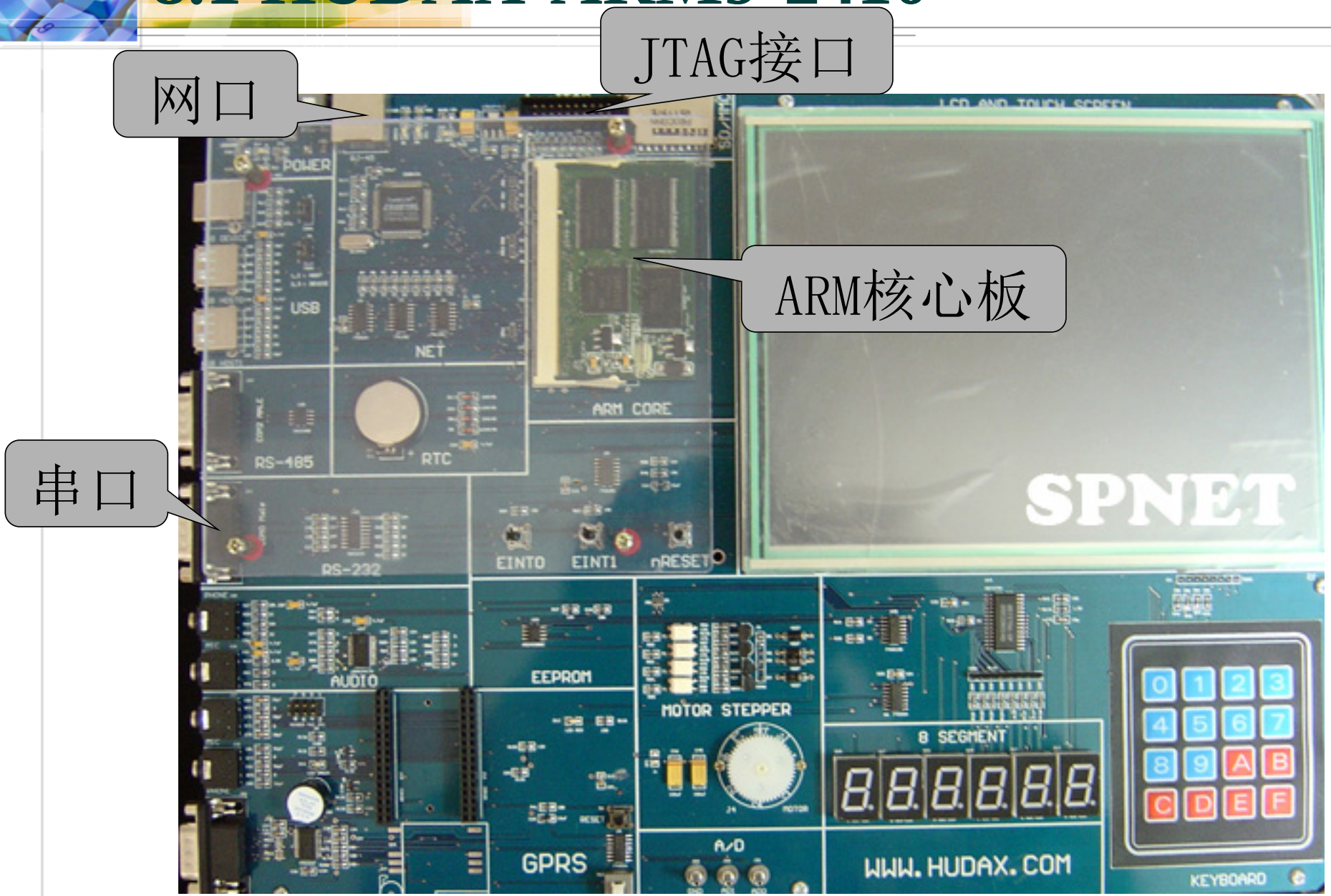
7.5 gdb的常用命令列表

命令	功能
file	装入待调试的可执行文件
kill	中止正在调试的程序
list	列出产生可执行程序的源代码的一部分
next	执行一行源代码但不进入函数内部
step	执行一行源代码而且进入函数内部
run	执行当前被调试的程序
quit	中止 gdb
watch	使用户可以监视一个变量的值，而不管它何时被改变
break	在代码里设置断点，这将使程序执行到这里时被挂起
make	使用户不用退出 gdb 的情况下就可以重新产生可执行文件
shell	使用户不用离开 gdb 的情况下就可以执行 UNIX shell 命令

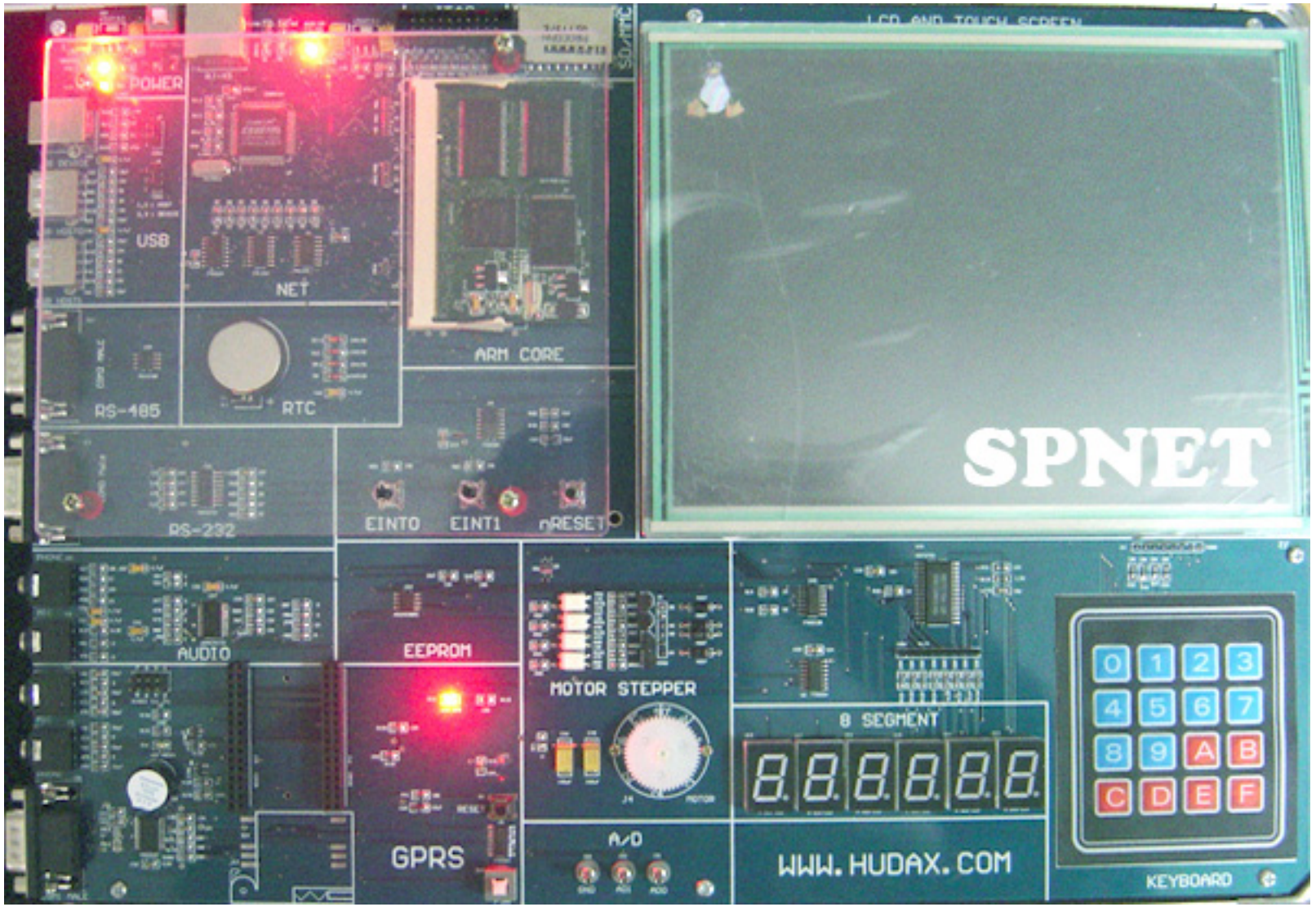
内容提纲(8/8)

- 1.嵌入式系统概述
- 2.嵌入式系统软件开发
- 3.嵌入式操作系统介绍
- 4.嵌入式Linux操作系统
- 5. Linux程序设计基础
- 6. GNU make入门
- 7. GDB简介
- 8. 实验系统HUDAX-ARM9-2410简介

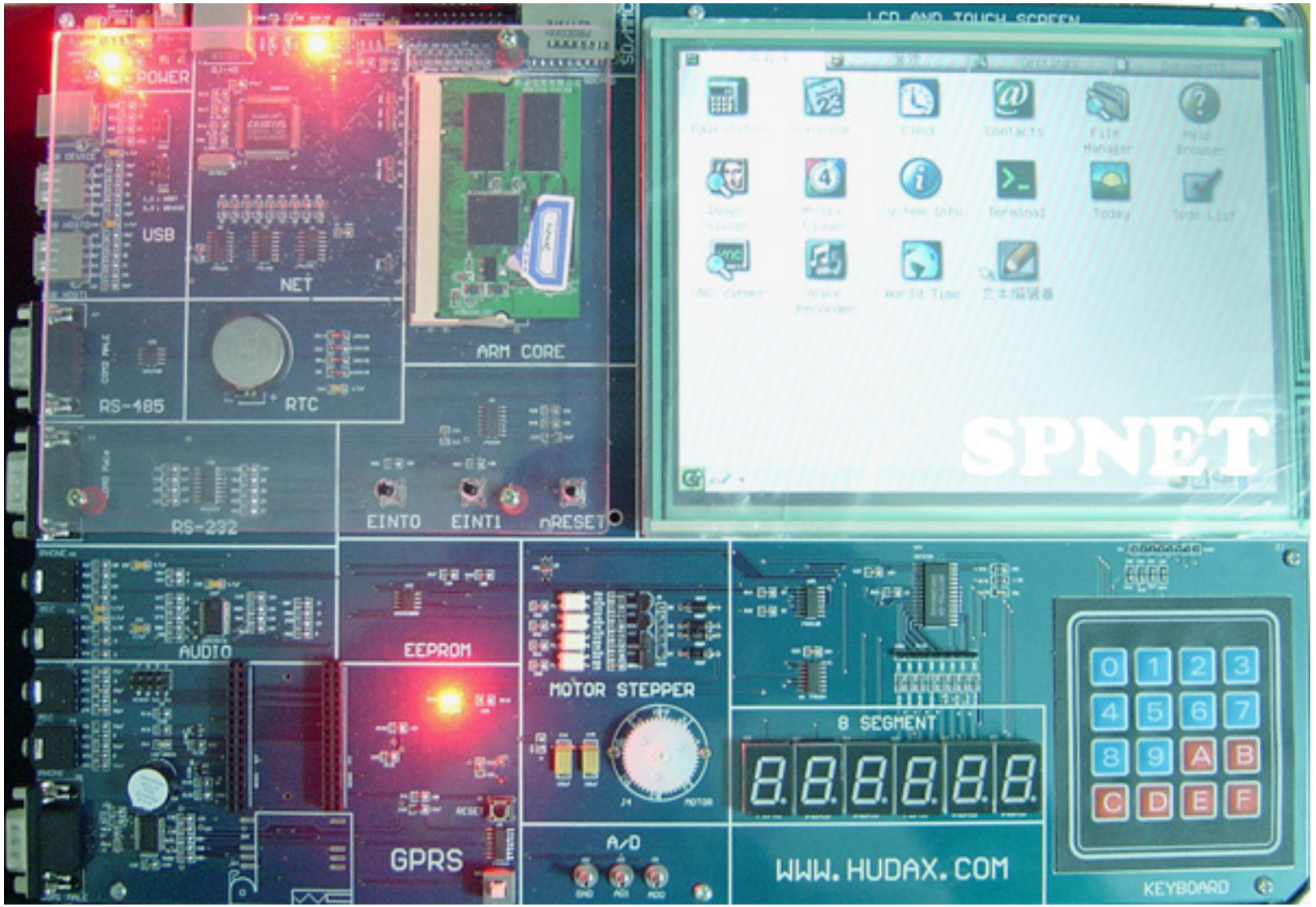
8.1 HUDAX-ARM9-2410



8.2 HUDAX-ARM9-2410系统启动



8.3 HUDAX-ARM9-2410图形界面



课后作业

- 通过阅读参考书籍和网络电子文档熟悉ARM9硬件平台和Linux操作系统的基本知识；
- 安装一种自己喜欢的Linux发行版；
- 熟悉Linux基本操作，包括登录、注销、拷贝、关机等，掌握vi编辑器和minicom终端程序的使用。
- 利用vi编辑器编写一个简单的c源文件；
- 利用gcc编译源文件，并学习gcc的各种选项的用法；
- 学习调试工具gdb的使用，并调试一个简单的程序。