原 《嵌入式》复习资料公共版

2013年10月06日 21:25:12 阅读数：3898

嵌入式 期末复习资料，把word文件摘录到博客中。许多内容很有用。

第一章

根据 IEEE （国际电气和电子工程师协会）的定义：

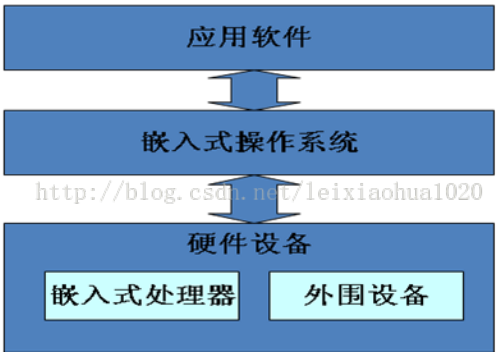
v 嵌入式系统是“用于控制、监视或者辅助某个设备、机器或工厂运作的装置”

国内普遍认同的定义是：
嵌入式系统是以应用为中心，以计算机技术为基础，并且软硬件可剪裁，适用于应用系统对功能、可靠性、成本、体积、功耗有严格要求的专用计算机系统。

简单的讲，就是 **嵌入** 到对象体重的 **专用计算机系统** 。

嵌入式系统的体系结构

硬件组成：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应用程序 4 个部分组成。



软件系统组成：硬件设备、 Bootloader 、内核、文件系统、应用程序

嵌入式系统的特点

嵌入性：嵌入式系统要嵌入到对象体系中

对象性：对象学科体系与技术支持

智能性：微处理器（专用计算机）内核

目前嵌入式操作系统主要有：

Linux ，μ C/OS ， WindowsCE ， Vxworks ， PalmOS ， Symbian,

商用型

Vxworks, Nucleux ， PalmOS, Symbian, WindowsCE

免费型

Linux, μ C/OS

Linux 系统的主要特点： Linux 系统是一个多用户、多任务的操作系统

v 开放的源码，丰富的软件资源

v 功能强大的内核，性能高效、稳定多任务

v 支持多种体系结构

v 完善的网络通信、图形、文件管理机制

v 支持大量的周边硬件设备

v 大小、功能都可定制

v 良好的开发环境，不断发展的开发工具集

v 软件开发者的广泛支持

v 价格低廉

嵌入式处理器

v 嵌入式微处理器（ Embedded Microprocessor Unit ）

v 嵌入式微控制器（ Microcontroller Unit ）

v 嵌入式 DSP 处理器（ Embedded Digital Signal Processor ）

v 嵌入式片上系统（ System on Chip ）

目前主要的嵌入式处理器类型有 Am186/88 、 386EX 、 SC-400 、 PowerPC 、 Xscale 、 68000 、 MIPS 、 ARM/StrongARM 系列等

第二章

Linux 命令格式

— 指令 [选项] [参数]

— 选项前需使用 - 或者 -- 。大部分指令在选项后要求一个以上的参数。

— 指令、选项、参数中间用空格隔开，不论几个空格， shell 认为是 1 个空格。

— 指令太长的时候，使用 \ 符号跳转回车，是指令连续到下一行。 \ 符号后可直接回车，但不能接空格符。

—

设备管理

在 linux 系统中，以 **文件** 方式访问设备

eg

v **mount - 挂载文件系统**

mount /dev/cdrom /mnt/cdrom

v **umount - 卸载**

umount /mnt/cdrom

v passwd - 更改密码

v who - 显示登录的所有用户

v ls 一 查看文件

v **cp 一 拷贝文件**

v mv 一 移动或重命名文件

v **rm 一 删除文件**

v touch 一 创建空文件或更新文件时间

v **cat 一 显示文本文件内容**

v more 一 分页显示

v less 一 分页显示，可向前翻页

v cd 一 改变当前路径

v **pwd 一 察看当前完整路径**

v **mkdir 一 创立新目录**

v **rmdir 一 删除空目录**

v . 一代表当前层目录

v .. 一代表上层目录

v / 一根目录

v ~ 一代表自己的根目录

v ~user 一代表 user 这个用户的根目录

打包命令：(注意：打包和压缩是两个不同的概念)

v tar ： 使用 tar 程序打出来的包都是以 .tar 结尾的。

v 使用 tar 命令可以把一大堆的文件和目录全部打包成一个文件，这对于备份文件或者将几个文件组合为一个文件以便于传输非常有用，语法如下：

v # tar [选项]f targetfile.tar 文件 / 目录

v 注：选项后面的 f 是必须的，通常用来指定包的文件名。

压缩命令：

v gzip/gunzip ： 目前常用的压缩工具，压缩效率较高

v bzip2/bunzip2: 对 gzip 进行升级的第 2 代压缩工具

v compress/uncompress ：早期压缩程序

v 由 tar 程序打包，并经过 gzip 程序的压缩 .tar.gz

v 由 tar 程序打包，并经过 bzip2 程序的压缩 .tar.bz2

tar 命令选项：

— c: 创建新的打包文件

— z: 将 tar 与 gzip 同时使用

— j: 将 tar 与 bzip2 同时使用

— v: verbose 的简写，捆绑或解开时查看内容

— f: 对普通文件操作，指定包文件名

— r: 往 tar 文件中增添其它文件

— Z ：将 tar 与 compress 同时使用

— t ：确认 tar 文件的内容，列出包文件的所有内容

— x ：从 tar 包文件中恢复所有文件，事实上是一个解包的过程。

— **tar -xvf < 压缩文件 .tar> ： 解包文件**

— **tar -cvf < 压缩文件 .tar> < 未压缩文件或目录 > ：建立打包文件**

— tar -tvf < 压缩文件 .tar> ：查看内容

网络相关命令

— **ifconfig**

— **ping**

— **ftp**

— **telnet**

— **eg ： ifconfig eth0192.168.1.1 设置 IP 地址为 192.168.1.1**

— **ifconfig eth0 down 关闭网络接口 eth0**

/ 目录也称为根目录，位于 linux 文件系统目录结构的顶层，是系统中的唯一分区

/bin ：该目录中含有常用的命令文件，该目录不能包含子目录

/dev ：该目录中包含大部分的设备文件

/home ：该目录中包含系统上各个用户的主目录，子目录名称即为各用户名

/sbin ： 该目录中保存系统管理员使用的管理程序或者 root 用户使用的命令文件

（ eg ifconfig 等等）

/boot ：该目录中存放系统的内核文件和引导装载程序文件

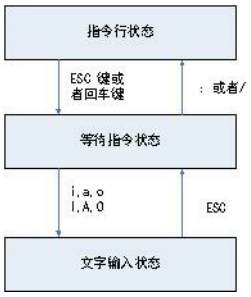
/etc ： 该目录中保存系统中的大部分配置文件

/mnt ：为某些设备提供的默认挂载点

第三章

Vi 的模式

— **vi 有 3 种状态：一般模式（等待指令状态），编辑模式（文字输入状态），命令行模 式（指令行状态）**



— 启动 vi 后进入 **一般模式** ，是 **等待指令状态** ，在这种状态下，可以上下左右移动光标，也可以用删除字符、删除整行、复制、粘贴来处理文件内容，但无法编辑。

— 等待指令状态下输入 i ， a ， o ， r 进入 **编辑模式(即文字输入状态)** 。通常在 linux 下,按了上述字母后,画面的左下方会出现 INSERT 或者 REPLACE 字样，这时可以输入文字。文字输入状态下按 ESC 键可退出编辑模式，进入一般模式。

— 在一般模式下（等待指令状态下），输入：或者 / 进入 **指令行状态** ,将光标移动到最后一行,可以搜寻数据,读取、存盘、退出 vi, 显示行号，大量字符替换等操作。

vi-- 退出命令

— <:q> 不保存退出

— <:q!> 不保存强制性退出

— <:w> 保存编辑

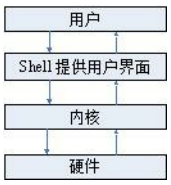
— <:w filename> 存入文件 filename 中

— <:w! filename> 强制性存入文件 filename 中

— <:wq>(<:x>) 保存并退出

第四章

Shell 的基本概念



— 对于操作系统而言， Shell 搭起了用户与操作系统的桥梁，提供了基本的操作界面，让用户可以下达各种命令，在系统中进行操作，产生彼此间的依赖关系。

— Shell 是一个命令语言解释器,拥有自己内建的 Shell 命令集,也能被系统中的其他程序调用。用户在提示符下输入的命令都被 Shell 解释后传给 linux 核心。

Shell 的种类

— Bourne Shell(sh)

— ***Bourne Again Shell(Bash)***

Red HatLinux 9.0 中默认使用的 shell

— C Shell （ Csh ）

— Tcsh

— Korn Shell(ksh)

Shell 中的特殊字符

— 1. 通配符 * ，?， []

— 2. 引号 “” , ‘’ , ``

— 3. 注释符 #

— 4. 输入输出重定向 < ,<<,>,>>

— 5. 管道符： "|"

通配符

— 通配符又称多义符。在描述文件时，有时在文件名部分用到一些通配符，以加强命令的功能。在 Linux 系统中有以下基本的通配符：

— **？：表示该位置可以是一个任意的单个字符。**

— **＊：表示该位置可以是若干个任意字符。**

— [charset] ：可替代 charset 集中的任何单个字符

— 例如 [cChH] ：表示在文件的该位置中可出现任意单个的 c 或 h 字符的大小写形式。

— 另外，通配符集还能描述介于字符对之间的所有字符。如 “[a-z]” 就可以代替任意小写字母，而 [a-zA-Z] 则可替代任意字母。注意可替代的字符包括 a 到 z 和 A 到 Z 字符对之间的所有字符。

引号

— 在 shell 中引号分为三种： **单引号，双引号和反引号。**

— 由单引号括起来的字符都作为普通字符出现。特殊字符用单引号括起来以后，也会失去原有意义，而只作为普通字符解释。例如：

— string='$PATH'

— echo $string

— $PATH

双引号

— 由双引号括起来的字符，除 $ 、 \ 、'和”这几个字符仍是特殊字符并保留其特殊功能外，其余字符仍作为普通字符对待。

— 例如，我们假定 PATH 的值为 .:/usr/bin:/bin ，输入如下命令：

— TestString ＝” $PATH\”\$PATH”

— echo $TestString

— .:/usr/bin:/ bin”$PATH

反引号

— 反引号（ ` ）这个字符所对应的键一般位于键盘的左上角，不要将其同单引号(')混淆。反引号括起来的字符串被 shell 解释为命令行，在执行时， shell 首先执行该命令行，并以它的标准输出结果取代整个反引号（包括两个反引号）部分。例如：

— pwd

— /home/xyz

— string=”current directory is`pwd`”

— echo $string

— current directour is /home/xyz

— 注：在反引号之间的命令行中可以使用 shell 的特殊字符。 Shell 为得到 `` 中的结果，它实际上要去执行  `` 中的指定的命令。执行时，命令中的特殊字符如 $ “ ？等又将具有特殊含义。例如：

— ls

— note readme.txt Notice Unix.dir

— TestString="`echo $HOME``ls[nN]*`"

— echo $ TestString

— /home/yxz note Notice

注释符

— 在 shell 编程中经常要对某些正文行进行注释，以增加程序的可读性。

— 在 shell 中以字符 # 开头的正文行表示注释行

标准输入 / 输出和重定向

— 执行一个 shell 命令行时通常会自动打开三个标准文件，即标准输入文件（ stdin ），通常对应终端的键盘；标准输出文件（ stdout ）和标准错误输出文件（ stderr ），这两个文件都对应终端的屏幕。进程将从标准输入文件中得到输入数据，将正常输出数据输出到标准输出文件，而将错误信息送到标准错误文件中。

— 使用 <,>,>>,<< 符号进行重定向

— 输入重定向一般形式为：命令 < 文件名

wc< /etc/passwd

— here 文档重定向

wc<< delim

>this is ……

>that is ….

…….

>delim

4 17 98

— 输出重定向 >

ls > directory.out

— 输出追加重定向 >>

ls*.doc >> directory.out

— 错误输出重定向 2> 2>>

ls/usr/tmp 2>err.file

— &> 将标准输出和错误输出同时送到同一文件

ls/usr/tmp &>output.file

重定向的用途

1 、当屏幕输出的信息很重要，我们需要保存的时候。

2 、背景程序的输出，不希望它干扰正常的输出

3 、一些命令，我们已经知道可能的错误信息，希望扔掉。

4 、错误信息与正确信息需要分开输出。

管道

— 管道可以把一系列命令连接起来，这意味着第一个命令的输出会作为第二个命令的输入通过管道传给第二个命令，第二个命令的输出又会作为第三个命令的输入，以此类推。显示在屏幕上的是管道行中最后一个命令的输出（如果命令行中未使用输出重定向）。

— 通过使用管道符“ |” 来建立一个管道行。

例如：

— $ ls /usr/bin|wc -w

命令替换

— 命令替换是将一个命令的输出作为另外一个命令的参数

— command1 `command2`

例如：

— $ cd `pwd`

bash 变量的设定

— echo $variable

— 显示变量值，可用 echo 指令。要分辨是否为变量，在变量前加一 $ 符号。例如

— echo $PATH

— echo $MAIL

— echo $HOME

$ ？	最近执行的命令的返回的值
$$	存储 shell 程序的 pid （本进程的进程号）
$!	最近后台进程号
$#	Shell 自变量个数， $1 表示第一个自变量

循环命令（ for,while,until,shift)

— for 语句有两种语法，第 1 种适合于

— for var in list

— do

— commands

— done

— 对在 list 中的每 1 项， for 语句都执行 1 次。 list 可以包含几个由空格隔开的变量,也可以是直接输入的几个值。每执行 1 次， var 都被赋予 list 中的当前值，直到最后 1 个为止。

— for 语句中的 in 语句，可以使用通配符

for 第 2 种形式，适合数字，类似 C 语言

— **for ((初始值；终止值；步长))**

— **do**

— **commands**

— **done**

Shell 举例

```
[plain]
1.   #!/bin/bash
2.  # This is a simple test.
3.  cd /root
4.  echo “x=$2"
5.  x=46
6.  y=${x}y
7.  echo “y=$y"
8.  z=4*6
9.  echo “z=$z"
10. d=`pwd`
11. echo d=$d
12. for i
13. do
14.   echo $i > file1
15. done
16. cat file1
17. read i1 < file1
18. echo i1=$i1
19. # end
```

— **运行 bash test.sh 12 3 4**

— **结果**

— **x=2**

— **y=46y**

— **z=4*6**

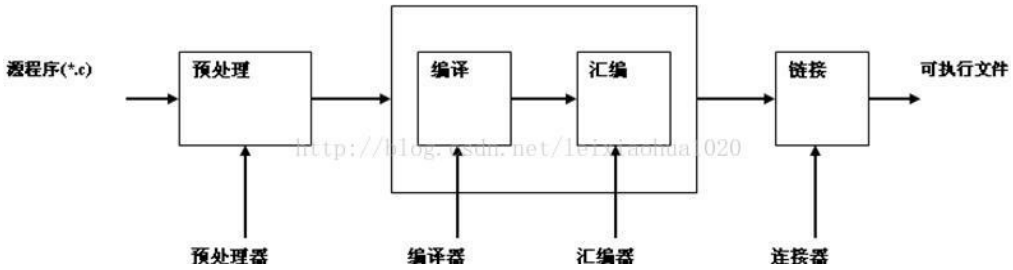
— **d=/root**

— **4**

— **i1=4**

第五章

GCC 简介



v GCC 是 GNU CompilerCollection 的简称。可以编译多种语言编写的程序。

v 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 **预处理、编译、汇编** 和 **链接** 四个阶段，每个阶段分别调用不同的工具进行处理，如下图

GCC 编译器

— 1 、 gcc **–E** hello.c –o hello.i(**经预处理的 C 源文件**)

— 2 、 gcc **–S** hello.i –o hello.s(**汇编源文件**)

— 3 、 gc **c –c** hello.s –o hello.o(**二进制目标代码**)

— 4 、 gcc hello.o **–o** hello(**链接后的目标代码**)

— 5 、 **./hello** (执行 hello 的 bash 脚本文件)

Gdb 调试器

— 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。

— 键入 gdb 回车即进入 GDB 调试器。

ex ： useradd qq

passwd qq

gcc g h.c –o h

Makefile

v **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。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。**

v 而且 makefile 就像一个 shell 脚本一样，可以执行操作系统的命令。

v 除非特别指定，否则 make 的工作目录就是当前目录。 target 是需要创建的二进制文件或目标文件；

dependency 是在创建 target 时需要用到的一个或多 个文件的列表；

command 命令序列是创建 target 文件所需要执行的 步骤，比如编译命令；

v Makefile 规则的一般形式如下：

target ： dependency dependency

(tab) <command>

补充：

为什么使用交叉编译

采用交叉编译的主要原因：多数嵌入式目标系统不能提供足够的资源供编译过程使用，因而只好将编译工具转移到高性能的主机中进行。

因为目的平台上不允许或不能够安装我们所需要的编译器，而我们有需要这个编译器的某些特征；有时又是因为目的平台还没有建立，连操作系统都没有，根本谈不上运行什么编译器。

交叉编译工具链安装：

1. 下载源文件、补丁和建立编译的目录

2. 建立内核头文件

3. 建立二进制工具（ binutils ）

4. 建立初始编译器（ bootstrap gcc ）

5. 建立 c 库（ glibc ）

6. 建立全套编译器（ full gcc ）

第六章

Bootloader 的概念

简单地说，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，我们可以初始化硬件设备、建立内存空间映射图，从而将系统的软硬件环境带到一个合适状态，以便为最终调用操作系统内核准备好正确的环境。

Bootloader 的两种模式

启动加载（ loading ）模式：

又称（ Automomous ）这种模式也称为"自主"(Autonomous)模式。也即 Boot Loader 从目标机上的某个固态存储设备上将操作系统加载到 RAM 中运行,整个过程并没有用户的介入。这种模式是 Boot Loader 的正常工作模式,因此在嵌入式产品发布的时侯,Boot Loader 显然必须工作在这种模式下。

下载（ downloading ）模式：

在这种模式下目标机上的 BootLoader 将通过串口连接或者网络连接等通信手段从主机下载文件，比如：下载应用程序、数据文件、内核映像等。 从主机下载的文件通常首先被 BootLoader 保存到目标记得 RAM 中然后再被 BootLoader 写到目标机上的固态存储设备中。 BootLoader 的这种模式通常在系统更新时使用。

Bootloader 启动过程

Stage1 ：

v 基本的硬件初始化

v 为加载 stage2 准备 RAM 空间

v 拷贝 stage2 到 RAM 中

v 设置堆栈指针 sp

v 跳到 stage2 的入口点

Stage2 ：

v 初始化本阶段要使用的硬件设备

v 检测系统的内存映射

v 加载内核映像和文件系统映像

v 设置内核的启动参数

v 调用内核

第七章

Linux 内核功能大致分为以下几部分：

1. 进程管理

2. 内存管理

3. 文件系统

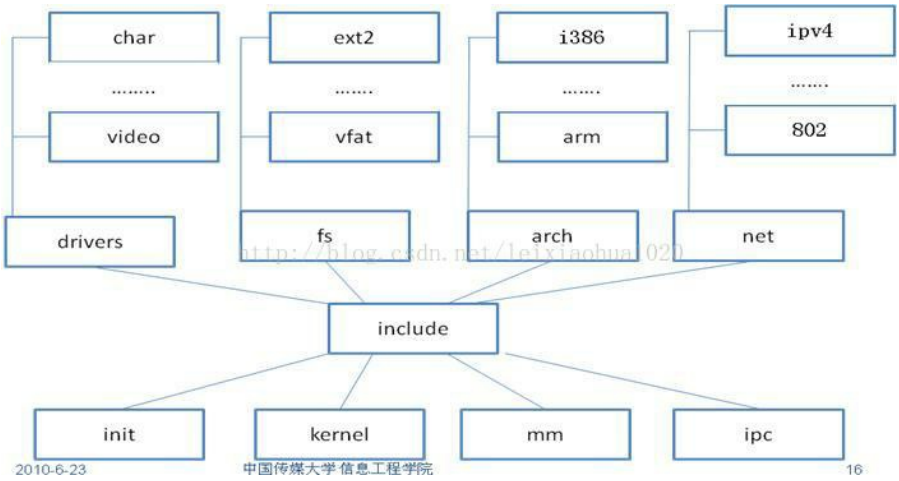
4. 设备控制

5. 网络功能

Linux 内核结构

一般在 Linux 系统中的 /usr/src/Linux-*.*.* 目录下就是内核源代码。

..* 代表内核版本



/arch 子目录包含了所有硬件结构特定的内核代码。如 i386 ， arm,alpha 等

• /drivers 子目录包含了内核中所有的设备驱动程序，如 usb ， sound 等

• /fs 子目录包含了所有的文件系统的代码。如 ntfs,ext3,jffs2 等

• /include 子目录包含了建立内核代码时所需要的大部分库文件，这个模块利用其他模块重建内核。该目录也包括了不同平台需要的库文件。如 asm -arm 是 arm 平台需要的库文件。

• /init 子目录包含了内核的初始化代码，内核从此处开始工作

• /ipc 子目录包含了进程间通信代码

• /kernel 子目录包含了主内核代码

• /mm 子目录包含了所有内存管理代码

• /net 子目录包含了和网络相关的代码。如 atm,ipv6 等

Linux 内核结构 — 版本号

Linux 版本号格式为“ x.yy.zz ”

x: 介于 0 到 9 之间，表示主版本号，它的变化意味着内核在设计或实现上有重大改变

yy: 介于 0 到 99 之间，表示次版本号，一方面表示版本的变迁，一方面标志着版本的种类，即发行版还是开发版。 **yy 是偶数表示一个相对稳定的、已经发行的版本。 yy 是奇数表示还在开发中的测试版本**

zz: 表示内核增加内容不是很多，改动不是很大时的变迁

Linux 操作系统移植

• arch 目录存放着和体系结构相关部分的内核代码。

• 我们的平台是 ARM 的，所以我们只关心 arm 目录

• 实际上在编译内核时，建立了一个 arch 链接，直接指向 **arch/arm**

第八章

什么是文件系统

在 Linux 系统中所有内容都被表示为文件，组织文件的各种方法成为文件系统

v **Flash 主要分为 NOR 和 NAND 两类**

› 1 ）读取 NOR Flash 可以读取它任意随机地址的值。同时它和 SDRAM 一样可以直接运行装载在 NOR FLASH 的代码。

› 2 ） NAND Flash 的读取是以一次读取一块的形式来进行的，通常是一次读取 512 个字节。 NAND Flash 比较容易出现坏位。一般情况下，不能直接运行 NAND Flash 上的代码

› 3 ） NOR 的读取速度比 NAND 稍快一些。

› 4 ） NAND 的写入和擦除速度比 NOR 快很多。

› 5 ）容量和成本： NAND Flash 容量大，成本低。

› 6 ）寿命（耐用性）： NAND 闪存每个块的最大擦写次数是 100 万次，而 NOR 的擦写次数是 10 万次。

虚拟文件系统 VFS ：

在 UNIX 系统中，文件系统是最基本的资源。在系统内核和文件系统之间制定一个标准的接口而实现的，不同文件结构之间可以通过该接口方便地交换数据。 Linux 正是使用这种方式，在系统内核和文件系统之间提供了一种标准接口—— VFS （ virtual file system ，虚拟文件系统）。

 Linux 下的文件系统，由虚拟文件系统和实际的文件系统两个层次组成。目前常用的有 EXT2 、 CRAMFS 、 JFFS2 、 NFTL 、 NFS 和 RAM 磁盘文件系统等。

（1） cramfs 文件系统

cramfs 文件系统，是一个压缩式的文件系统，不需要一次性地将文件系统中的所有内容解压缩到内存中，而是在系统需要访问数据时，马上计算出该数据在 cramfs 中的位置，将其实时解压缩到内存中，系统到内存中读取数据。解压缩和解压缩后的数据存储位置是由 cramfs 文件系统进行维护的。

（2） jffs /jffs2

考虑到多数系统需要能够读 / 写的文件系统，可以使用 MTDdriver 的 JFFS 和 JFFS2 日志式文件在 Flash 头部建立根文件系统（ Root Filesystem ）。日志式文件系统可以免受系统突然掉电的危险，并且在下一次系统引导时不需要文件系统的检查。由于 JFFS 和 JFFS2 文件格式是特别为 Flash 存储器设计的，二者都具一种称为“损耗平衡”的特点，也就是说 Flash 的所有被擦写的单元都保持相同的擦写次数。利用这些特有保护措施， Flash 的使用周期得到相当大的提升。 JFFS2 使用压缩的文件格式，为 Flash 节省了大量的存储空间，它更优于 JFFS 格式在系统中使用。

（3）Romfs （ ROM file system)

传统型的 Romfs 文件系统是一种简单的、紧凑的、只读的文件系统，不支持动态擦写保存，按顺序存放数据，因而支持应用程序以 XIP(eXecute In Place ，片内运行 ）方式运行，在系统运行时，节省 RAM 空间。 uCLinux 系统通常采用 Romfs 文件系统。

（ 4 ） YAFFS

YAFFS ， Yet Another Flash File System ，是一种类似于 JFFS/JFFS2 的专门为 Flash 设计的嵌入式文件系统。

 与 JFFS 相比，它减少了一些功能，因此速度更快、占用内存更少。

（ 5 ） ext3 文件系统

ext3 文件系统是开放源代码的日志文件系统，继承了 ext2 文件系统的特性，可以称为 ext2 文件系统的日志文件系统升级版。

 Linux 主流的文件系统是 ext2, 从 2.4.15 开始，已经开始支持 ext3, 需要在编译时选择支持 ext3 文件系统。

（ 6 ） xfs 文件系统

xfs 是 SGI 发布的日志文件系统，目前 linux 2.4 支持 xfs 文件系统。

功能强大，日志提供了高效率的恢复功能。

大容量， 64 位文件系统，高带宽。

尽量减小了日志记录对文件系统操作事务的影响。

3. 网络文件系统 NFS (Network File System)

 NFS 是由 Sun 开发并发展起来的一项在不同机器、不同操作系统之间通过网络共享文件的技术。在嵌入式 Linux 系统的开发调试阶段，可以利用该技术在主机上建立基于 NFS 的根文件系统，挂载到嵌入式设备，可以很方便地修改根文件系统的内容。

Linux 的根文件系统

v 每台机器都有根文件系统，它包含系统引导和 mount 其他文件系统所必需的文件。还应该包括修复损坏系统、恢复备份等的工具

v 根文件系统一般比较小，根目录一般不含任何文件，除了可能的标准的系统引导映象，通常叫 /vmlinux 。所有其他文件在根文件系统的子目录中。

v 根文件系统损坏一般意味着系统无法引导。

v Linux 文件系统层次结构是树形结构，根是树的最高一层，所有的分支都是从它派生出来的。

v 所有的目录都可以分布在不同的磁盘或分区上；但是因为可以把磁盘或分区装载到目录树中，用户并不必知道文件处在哪个磁盘驱动器上，他们只要知道文件在目录树中的位置就可以了。

只有超级用户才能存取根文件系统，因为从根文件系统出发每个文件都是可以存取的。当系统启动的时候，第一个装载的文件系统就是根文件系统（ / ）。系统的核心包含了根文件系统的信息。

第九章

Linux 中所有设备被抽象出来，都看成文件

设备的读写和普通文件一样

v Linux 内核有三种类型的设备

· **字符设备** char device ：指存取时没有缓存的设备

· **块设备** block device ：读写都有缓存支持，且必须可以随机存取

· **网络设备** network device ：在 Linux 里做专门的处理

- 内核还使用了一个 **主设备号** 和一个 **次设备号** 来唯一标识设备。
- 主设备号表明某一类设备，标识了设备所对应的驱动程序。
- 次设备号仅由驱动程序解释，一般用于识别在若干可能的硬件设备中， I/O 请求所涉及到的那个设备。
- 如系统中 IDE 硬盘的主设备号是 3 ，而多个硬盘及各个分区分别赋予次设备号 1 ， 2 ， 3…

设备节点：

- 访问一个设备，需要制定一个设备的标识符。在 linux 系统中，这个标识符一般是位于 /dev 下的文件，称为设备节点。
- ls –l /dev
- 创建设备节点 mknod

- **在 Linux 中将驱动程序嵌入内核有两种方式：**
- **一是静态编译链接进内核**
- **二是先编译成模块，再动态加载进内核。**

采用静态编译链接方式，需要把驱动程序的源代码放在内核源代码目录“ Drivers/” 下，并修改 Makefile 文件。在编译链接内核的时候，驱动程序会作为内核的一部分链接到内核镜像文件中。这种方式会增加内核的大小，还要改动内核的源文件，而且不利于调试。

模块（ module ）在 Linux 中是一种已经编译好的目标文件，它可以被链接进内核从而生成可执行的机器代码。如果采用模块加载的方式，驱动程序首先会被编译成未链接的目标文件，具有 root 权限的用户在需要时可利用 **insmod** 命令将其动态的加载到内核中，而在不需要的时候可利用 **rmmod** 命令卸载该模块。 **lsmod** 列出当前系统中加载的模块。

补充：

网络文件系统 NFS (Network File System)

应用层的一种应用服务，它主要应用于 Linux 和 Linux 系统、 Linux 和 Unix 系统之间的文件或目录的共享，对于用户而言可以通过 NFS 方便的访问远地的文件系统，使之成为本地文件系统的一部分。

1. **s** etup ® system services

去掉 iptables 选中 nfs

2. 要配置 NFS 服务器，在服务器端主要配置

/etc/exports 文件

Vi/etc/exports

/s3c2410_linux/nfs

192.168.2.120(rw,insecure,no_root_squash,no_all_squash)

3. 启动服务

/etc/rc.d/init.d/nfs restart

什么是 minicom ： Linux 下串口通信工具，为嵌入式系统提供人机交互的手段。

什么是 tftp ：简单文件传输协议，利用网络接口下载较大的文件，比串口速率快。

eg 在 Bootloader 下使用 tftp 传输内核镜像文件到目标板的 RAM 存储器件中

配置 vsftp 服务的过程

1. 新建终端

2. 配置 IP 地址，例如 ifconfig eth0 192.168.0.124

3. Setup 进入 systems service

4. 关闭 iptables

5. 开启 vsftpd

6. service vsftpd restart

在嵌入式 linux 开发过程中，主机和目标机的连接经常用到串口线，网线和 JTAG 接口线。它们各起到什么作用？

串口线提供主机和目标板之间的通信和人机交互

网线主要用来传输文件和提供文件共享服务

JTAG 接口是调试、下载程序最基本的工具

Linux 的设备驱动程序按实现的功能大致可以分为如下几个部分：

v 驱动程序的注册与注销、

v 设备的打开与释放、

v 设备的读写操作、

v 设备的控制操作、

v 设备的中断和轮询处理。

完整的嵌入式 Linux 产品的开发流程

1. 嵌入式处理器的选型

2. 选择存储设备，选择与产品相关的其他音视频设备

3. 外围电路设计

4. 选择内核版本

5. 建立交叉开发平台

6. 移植交叉编译工具链

7. 移植 Bootloader

8. 移植系统内核

9. 制作文件系统

10. 编写设备驱动

11. 开发应用程序

12. 交叉调试

第十章

ARM 公司简介

ARM （Advanced RISC Machines）既可以认为是一个公司的名字，也可以认为是对一种微处理器体系结构的通称。 1991 年 ARM 公司成立于英国剑桥，设计了大量高性能、廉价、耗能低的 RISC （精简指令集）处理器。

公司的特点是只设计芯片，而不生产。它将技术授权给世界上许多著名的半导体、软件和 OEM 厂商，并提供服务。

公司也提供基于 ARM 架构的开发设计技术：软件工具，评估板，调试工具，应用软件；总线架构，外围设备单元等。

ARM 的运行模式

ARM 微处理器支持 **7** 种运行模式，分别为：

v 用户模式（ usr ）： ARM 处理器正常的程序执行状态

v 快速中断模式（ fiq ）：用于高速数据传输或通道处理

v 中断模式（ irq ）：用于通用的中断处理

v 管理模式（ svc ）：操作系统使用的保护模式

v 终止模式（ abt ）：当数据或指令预取终止时进入该模式，可用于虚拟存储及存储保护。

v 系统模式（ sys ）：运行具有特权的操作系统任务。

v 未定义模式（ und ）：当未定义的指令执行时进入该模式，可用于支持硬件协处理器的软件仿真。

ARM 的工作状态

ARM 微处理器的工作状态一般有 **两** 种，并可在两种状态之间切换：

v **ARM 状态** ，此时处理器执行 **32 位** 的字对齐的 ARM 指令

v **Thumb 状态** ，此时处理器执行 **16 位** 的、半字对齐的 Thumb 指令

v ARM 处理器在开始执行代码时，处于 ARM 状态，可以通过软件的方法切换到 Thumb 状态

ARM 的寄存器组织

ARM 微处理器共有 37 个 32 位寄存器，其中 **31** 个为 **通用寄存器** ， **6** 个为 **状态寄存器** 。但是这些寄存器不能被同时访问。

在某种时刻，通用寄存器 R14 ～ R0 、程序计数器 PC 、一个或两个状态寄存器都是可访问的。

v 寄存器 R13 在 ARM 指令中通常用作堆栈指针

v R14 也称作 **子程序连接寄存器** (Subroutine Link Register) 或连接寄存器 LR 。当执行 BL 子程序调用指令时， R14 中得到 R15（程序计数器 PC ）的备份

v R15 用作 **程序计数器**

v R16 用作当前 **程序状态寄存器** (Current Program Status Register,CPSR)

v 每种异常模式下都有一个专用的物理状态寄存器，称为 **备份程序状态寄存器** （ SavedProgram Status Register,SPSR ）

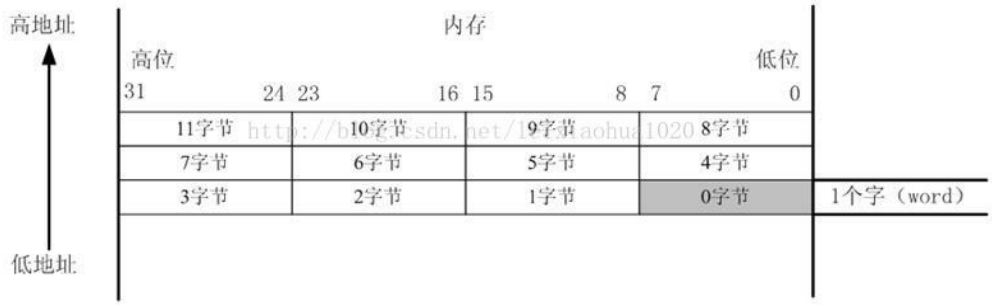
ARM 的存储器格式

ARM 体系结构将存储器看作是从零地址开始的字节的线性组合。从第 0 字节到第 3 字节放置第一个存储的字数据，从第 4 个字节到第 7 个字节放置第二个存储的字数据，依次排列。

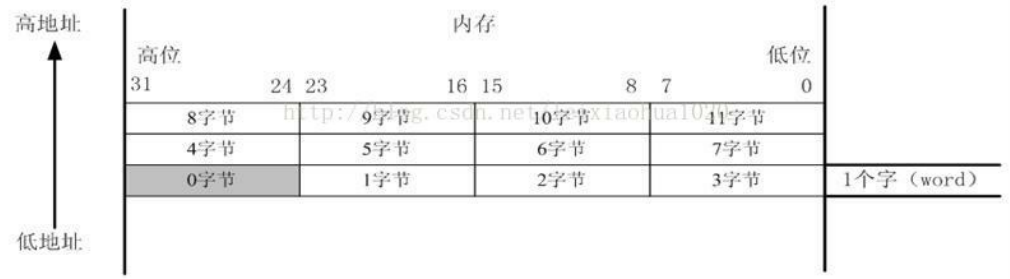
作为 32 位的微处理器， ARM 体系结构所支持的最大寻址空间为 4GB （ 32 位）。

ARM 体系结构可以用两种方法存储字数据，称之为 **大端格式** 和 **小端格式** 。

小端格式。



大端格式



Thumb 指令集

和 ARM 指令集的区别主要体现在 **分支指令、数据处理指令** 和 **寄存器操作指令** 上

ARM 的寻址模式

寻址模式就是处理器指令对内存的访问方式。

ARM 采用的是 Load-store(加载 - 存储) 的内存操作模式，其寻址方式比较简单，包括立即寻址、寄存器寻址、寄存器间接寻址、基址变址寻址、多寄存器寻址、跳转寻址和堆栈寻址

文章标签： 嵌入式 linux内核 arm处理器 嵌入式系统 交叉编译

个人分类： 广播电视工程 纯编程