

嵌入式与移动开发系列

**NITE** 国家信息技术紧缺人才培养工程  
National Information Technology Education Project  
国家信息技术紧缺人才培养工程系列丛书

众多专家、厂商联合推荐 • 业界权威培训机构的经验总结

# 嵌入式Linux系统开发 标准教程 (第2版)

华清远见嵌入式培训中心 编著

提供36小时嵌入式专家讲座视频和教学课件

## Embed Linux System Development



光盘内容  
本书源代码  
本书配套PPT  
嵌入式专家讲座视频

 **人民邮电出版社**  
POSTS & TELECOM PRESS



## 第 4 章 嵌入式交叉开发环境

### 本章目标

本章内容包括嵌入式交叉开发环境的概念和配置，以及应用程序交叉开发和调试的方法。交叉开发环境是嵌入式 Linux 开发的基础，后续的开发过程几乎都是基于交叉开发环境的。因此，理解和掌握本章内容会大大方便嵌入式 Linux 开发。

- 交叉开发环境介绍
- 建立交叉开发环境
- 交叉调试应用程序

## 4.1 交叉开发环境介绍

本节将介绍交叉开发模型以及相关概念，为后面具体配置交叉开发环境做好概念上的准备。

### 4.1.1 交叉开发概念模型

嵌入式系统是专用计算机系统，它对系统的功能、可靠性、成本、体积、功耗等某些方面有严格的要求。例如：PDA 需要通过电池供电，需要尽可能降低功耗；网络交换机，不需要键盘显示等外围设备；还有大部分嵌入式设备没有磁盘等大容量存储设备。

电信服务器也属于嵌入式系统范畴，尽管配置了显示器、键盘、鼠标等计算机外设，但是它更注重系统的可靠性，而不是用户界面的可操作性。

由于嵌入式系统硬件上的特殊性，一般不能安装发行版的 Linux 系统。例如 Flash 存储空间很小，没有足够的空间安装；或者处理器很特殊，也没有发行版的 Linux 系统可用。所以需要专门为特定的目标板定制 Linux 操作系统，这必然需要相应的开发环境。于是人们想到了交叉开发模式。交叉开发模型如图 4.1 所示。

图 4.1 中 TARGET 就是目标板，HOST 是开发主机。在开发主机上，可以安装开发工具，编辑、编译目标板的 Linux 引导程序、内核和文件系统，然后在目标板上运行。通常这种在主机环境下开发，在目标板上运行的开发模式叫作交叉开发。

在交叉开发环境下，开发主机也是工作站，可以给开发者提供开发工具；同时也是一台服务器，可以配置启动各种网络服务。

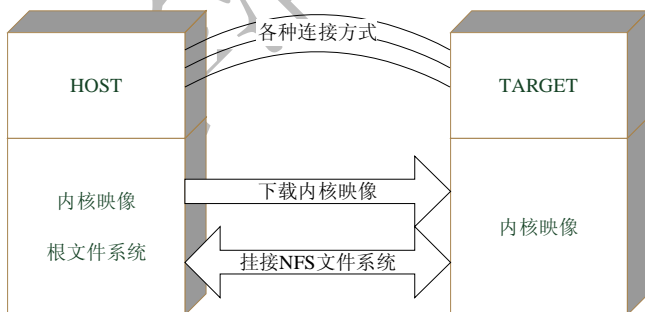


图 4.1 交叉开发模型

在 PC 主机上，Linux 已经成为优秀的计算机操作系统。各种 Linux 发行版本，可以直接在 PC 上安装，功能十分强大。它不仅能够支持各种处理器和外围设备接口，而且提供了图形化的用户交互界面和丰富的开发环境，更重要的是 Linux 系统性能稳定。它为开发者提供了以下功能。

- n 非常稳定的多任务操作系统。
- n 丰富的设备驱动程序支持和网络工具。
- n 强大的 Shell。
- n 本地编译器。

- n 编辑器。
- n 图形化的用户界面。

Redhat Linux 9 版本对计算机要求的最低配置如下。

- n CPU 主频 400MHz 以上。
- n 内存 128MB。
- n 硬盘 1.3GB。

推荐配置如下。

- n CPU 主频 1.0GHz 以上。
- n 内存 256MB。
- n 硬盘 5GB。

采用目前主流的计算机配置，完全能够满足推荐配置。无论 Linux 图形界面响应，还是程序编译，速度都很快，操作起来就很流畅。这对于嵌入式 Linux 开发者来说，可以大大提高开发效率。

对于交叉开发方式，一方面开发者可以在熟悉的主机环境下进行程序开发；另一方面又可以真实地在目标板上运行调试程序，可以避免受到目标板硬件的限制。这种开发方式贯穿嵌入式 Linux 系统开发的全过程。

要建立交叉开发方式，需要主机与目标板之间建立连接，才能实现远程通信、传输文件等功能。这依赖于不同连接方式。

#### 4.1.2 目标板与主机之间的连接

目标板和主机之间通常可以使用串口、以太网接口、USB 接口以及 JTAG 接口等连接方式。下面分别介绍这些通信接口的特点。

##### (1) 串行通信接口。

串行通信接口常用 9 针串口(DB9)和 25 针串口(DB25)，通信距离较近时(<12m)，可以用电缆线直接连接标准 RS232C 端口；如果距离较远，就采用 RS422 或者 RS485 接口，需附加调制解调器 (Modem)。其中最常用的是三线制接法，即地、接收数据和发送数据三脚相连，直接用 RS232C 相连，PC 机上一般带有 2 个 9 针串口。串口常用信号引脚如图 4.1 所示。

表 4.1 串口常用信号引脚

引脚功能	缩写	DB9 引脚号	DB25 引脚号
数据载波检测	DCD	1	8
接收数据	RXD	2	3
发送数据	TXD	3	2
数据终端准备	DTR	4	20
信号地	GND	5	7
数据设备准备好	DSR	6	6
请求发送	RTS	7	4

清除发送	CTS	8	5
振铃指示	DELL	9	22

通过串口可以作为控制台，向目标板发送命令，显示信息；也可以通过串口传送文件；还可以通过串口调试内核及程序。串口的设备驱动实现也比较简单。

缺点是通讯速率慢，不适合大数据量传输。

### (2) 以太网接口。

以太网以其高度灵活，相对简单，易于实现的特点，成为当今最重要的一种局域网建网技术。虽然其他网络技术也曾经被认为可以取代以太网的地位，但是绝大多数的网络管理人员仍然把以太网作为首选的网络解决方案。

以太网 IEEE 802.3 通常使用专门的网络接口卡或通过系统主电路板上的电路实现。以太网使用收发器与网络媒体进行连接。收发器可以完成多种物理层功能，其中包括对网络碰撞进行检测。收发器可以作为独立的设备通过电缆与终端站连接，也可以直接被集成到终端站的网卡当中。

以太网采用广播机制，所有与网络连接的工作站都可以看到网络上传递的数据。通过查看包含在帧中的目标地址，确定是否进行接收或放弃。如果证明数据确实是发给自己的，工作站将会接收数据并传递给高层协议进行处理。

以太网采用 CSMA/CD 媒体访问机制，任何工作站都可以在任何时间访问网络。在发送数据之前，工作站首先需要侦听网络是否空闲，如果网络上没有任何数据传送，工作站就会把所要发送的信息投放到网络当中。否则，工作站只能等待网络下一次出现空闲的时候再进行数据的发送。

作为一种基于竞争机制的网络环境，以太网允许任何一台网络设备在网络空闲时发送信息。因为没有任何集中式的管理措施，所以非常有可能出现多台工作站同时检测到网络处于空闲状态，进而同时向网络发送数据的情况。这时，发出的信息会相互碰撞而导致损坏。工作站必须等待一段时间之后，重新发送数据。补偿算法用来决定发生碰撞后，工作站应当在何时重新发送数据帧。

网络接口一般采用 RJ-45 标准插头，PC 机上一般都配置 10M/100M 以太网卡，实现局域网连接。通过以太网连接和网络协议，可以实现快速的数据通讯和文件传输。

缺点是驱动程序实现比较麻烦，好在以太网接口的设备驱动也很多。

### (3) USB 接口。

USB (Universal Serial Bus) 接口支持热插拔，具有即插即用的优点，最多可连接 127 台外设，所以 USB 接口已经成为 PC 外设的标准接口。USB 有两个规范，即 USB 1.1 和 USB 2.0。

USB 1.1 是较早的 USB 规范，其高速方式的传输速率为 12Mbps，低速方式的传输速率为 1.5Mbps。

USB 2.0 规范则是最新的 USB 规范。它的传输速率达到了 480Mbps，足以满足大多数外设的速率要求。USB 2.0 中的“增强主机控制器接口”(EHCI)定义了一个与 USB 1.1 相兼容的架构。所有支持 USB 1.1 的设备都可以直接在 USB 2.0 的接口上使用而不必担心兼容性问题，而且像 USB 线、插头等附件也都可以直接使用。

USB 的设备支持热插拔，通讯速率也很快。

缺点是 USB 设备区分主从端，两端分别要有不同的驱动程序支持。

#### (4) JTAG 等接口。

JTAG 技术是一种嵌入式调试技术，它在芯片内部封装了专门的测试电路测试接口（TAP, Test Access Port），通过 JTAG 测试工具对芯片的核进行测试。它是联合测试行动小组（JTAG, Joint Test Action Group）定义的一种国际标准测试协议，主要用于芯片内部测试及对系统进行仿真、调试。

目前大多数比较复杂的器件都支持 JTAG 协议，如 ARM、DSP、FPGA 器件等。标准的 JTAG 接口是 4 线：TMS、TCK、TDI、TDO，分别为测试模式选择、测试时钟、测试数据输入和测试数据输出。

JTAG 接口的时钟一般在 1MHz~16MHz 之间，所以传输速率可以很快。但是实际的数据传输速度要取决于仿真器与主机端的通讯速度和传输软件。

另外还有 EJTAG（Extended JTAG）和 BDM（Background Debug Mode）接口定义，分别在 MIPS 芯片和 PowerPC 5xx/8xx 芯片上设计应用。这些接口的电气性能不同，但是功能大体上是相似的。

### 4.1.3 文件传输

主机端编译的 Linux 内核影像必须有至少一种方式下载到目标板上执行。通常是目标板的引导程序负责把主机端的影像文件下载到内存中。根据不同的连接方式，可以有多种文件传输方式，每一种方式都需要相应的传输软件和协议。

#### (1) 串口传输方式。

主机端通过 kermit、minicom 或者 Windows 超级终端等工具都可以通过串口发送文件。当然发送之前需要配置好数据传输率和传输协议，目标板端也要做好接收准备。通常波特率可以配置成 115200bit/s, 8 位数据位, 不带校验位。传输协议可以是 Kermit、Xmodem、Ymodem、Zmodem 等。

#### (2) 网络传输方式。

网络传输方式一般采用 TFTP（Trivial File Transport Protocol）协议。TFTP 协议是一种简单的网络传输协议，是基于 UDP 传输的，没有传输控制，所以对于大文件的传输是不可靠的。不过正好适合目标板的引导程序，因为协议简单，功能容易实现。当然，使用 TFTP 传输之前，需要驱动目标板以太网接口并且配置 IP 地址。

#### (3) USB 接口传输方式。

通常分主从设备端，主机端为主设备端，目标板端为从设备端。主机端需要安装驱动程序，识别从设备后，可以传输数据。USB 2.0 标准的数据传输速率非常快。

#### (4) JTAG 接口传输方式。

JTAG 仿真器跟主机之间的连接通常是串口、并口、以太网接口或者 USB 接口。传输速率也受到主机连接方式的限制，这取决于仿真器硬件的接口配置。

采用并口连接方式的仿真器最简单，也叫作 JTAG 电缆（CABLE），价格也最便宜。性能好的仿真器一般会采用以太网接口或者 USB 接口通信。

#### (5) 移动存储设备。

如果目标板上有软盘、CDROM、USB 盘等移动存储介质，就可以制作启动盘或者复制到目标板上，从而引导启动。移动存储设备一般在 X86 平台上比较普遍。

#### 4.1.4 网络文件系统

网络文件系统（NFS，Network File System）最早是 SUN 开发的一种文件系统。NFS 允许一个系统在网络上共享目录和文件。通过使用 NFS，用户和程序可以像访问本地文件一样访问远端系统上的文件，这极大地简化了信息共享。

Linux 系统支持 NFS，并且可以配置启动 NFS 网络服务。

NFS 文件系统的优点如下。

(1) 本地工作站使用更少的磁盘空间，因为通常的数据可以存放在一台机器上而且可以通过网络访问到。

(2) 用户可以通过网络访问共享目录，而不必在计算机上为每个用户都创建工作目录。

(3) 软驱、CDROM 等存储设备可以在网络上面共享使用。这可以减少整个网络上的移动介质设备的数量。

(4) NFS 至少有一台服务器和一台（或者更多）客户机两个主要部分。客户机远程访问存放在服务器上的数据。需要配置启动 NFS 等相关服务。

网络文件系统的优点正好适合嵌入式 Linux 系统开发。目标板没有足够的存储空间，Linux 内核挂接网络根文件系统可以避免使用本地存储介质，快速建立 Linux 系统。这样可以方便地运行和调试应用程序。

## 4.2 安装交叉编译工具

基于上述硬件环境配置的需求，接下来一步步构建这个交叉开发环境。首先要安装交叉编译工具链。

### 4.2.1 获取交叉开发工具链

Linux 使用 GNU 的工具，社区的开发者已经编译出了常用体系结构的工具链，从因特网上可以下载。我们可以下载这些工具，建立交叉开发环境，也可以自己动手编译新的工具链，那就请仔细阅读第 5 章的内容。

对于 ARM 体系结构的编译器，也有不少站点提供下载。免费提供的工具链包括 binutils 和 gcc，但是都不提供 gdb 交叉调试器。社区主要维护 Linux 内核的发布，文件系统也很少。这里介绍几个 ARM Linux 的免费站点。

(1) <http://arm.linux.org.uk>

这个站点是 ARM Linux 的官方站点，Linux 2.4 内核发布过很多针对 ARM 平台的补丁。有许多 ARM/XSCALE 开发者维护这个站点，也可以下载到 ARM/XSCALE 开发常用的工具链。

ARM Linux 工具链下载的 HTTP 和 FTP 地址如下。

<http://ftp.arm.linux.org.uk/pub/armlinux/toolchain/>

<ftp://ftp.arm.linux.org.uk/pub/linux/arm/toolchain/>

(2) <http://www.handhelds.org>

HANDHELDS 是手持设备的开发网站。因为 ARM/XSCALE 处理器在手持设备上应用广泛，所以也有很多 ARM Linux 开发的资源。

这里的 ARM Linux 工具链版本比其他网站相对高一些，下载链接地址如下。

<http://www.handhelds.org/download/projects/toolchain/>

(3) <http://linux.omap.com>

这是 OMAP Linux 网站，从 TI 公司网站可以链接过来。TI 公司基于 ARM926E 核发布了一系列 OMAP 处理器，具有低功耗，智能电源管理的特点，适合移动手持设备的应用。这个站点专门为 OMAP 平台提供 Linux BSP。

这里的 ARM 工具链下载连接地址如下。

<http://linux.omap.com/pub/toolchain>

(4) <http://www.mvista.com>

这是 Montavista 公司的主页网址，浏览 Professional 3.1 的版本的产品介绍，会发现可以免费注册部分平台的预览版（Previewkit）。注册成功以后，通过 Email 得到 Montavista 提供的下载网址和软件安装密码。

Montavista Linux 能够支持各种体系结构的开发板，只对部分硬件平台提供预览版。预览版是 Montavista 免费提供给客户的软件，包括交叉编译器、内核以及很小的一个文件系统，可以用来学习建立嵌入式 Linux 交叉开发环境。

Montavista Linux 的发行版包含完整的交叉开发工具链、内核和文件系统，还有集成开发环境。Montavista Linux 发行版对内核、应用程序和库、以及工具都已经作过完整的测试，对产品提供技术支持等。

另外，还有其他 Linux 发行商的开发包。有些半导体商也会为他们的处理器提供板级开发包（BSP，Board Support Packages）。

## 4.2.2 主机安装工具链

下载的工具链有不同的包装格式，RPM 的格式就很常用，也有把工具链直接压缩成 tar 包的。

对于 RPM 的格式，可以通过 rpm 命令把软件包安装到主机上。可是这些工具安装到哪里去了呢？RPM 包安装的时候都会有缺省的安装目录，可以通过 rpm 命令来查询。这个命令是 Redhat Linux 上的常用命令，可以参考第 3 章的内容。

对于 tar 包，可以使用 tar 命令解压的。问题是解压出来的工具应该放在什么路径下？因为 GCC 编译器的运行是依赖于其他工具和库，通常不能把这些工具放在任意目录下。只好向下载的站点求教，一般通过相关的 README 或者说明文档可以得到具体的安装路径。

另外，通过 gcc 命令也可以得到安装的路径。以 ARM Linux 站点提供的 cross-3.3.2.tar.bz2 包为例说明。解压 cross-3.3.2.tar.bz2 后，查看 GCC 版本号，可以得到一些信息。

```
$ tar -jxvf cross-3.3.2.tar.bz2
$ ./3.3.2/bin/arm-linux-gcc -v
Reading specs from ./3.3.2/bin/./lib/gcc-lib/arm-linux/3.3.2/specs
```

```
Configured with: ../gcc-3.3.2/configure --target=arm-linux
--with-cpu=strongarm1100 --prefix=/usr/local/arm/3.3.2 i686-pc-linux-gnu
--with-headers=/work/kernel.h3900/include --enable-threads=threads
--enable-shared --enable-static --enable-languages=c,c++
Thread model: posix
gcc version 3.3.2
```

从上面打印的版本信息中可以看到“--prefix=/usr/local/arm/3.3.2”，这就是 GCC 安装的路径。它是在 GCC 编译前通过 prefix 选项配置的。

所以，这个工具链应该安装的路径是：/usr/local/arm/3.3.2。

```
$ mkdir -p /usr/local/arm
$ mv ./2.95.3 /usr/local/arm/
```

然后，在环境变量 PATH 中添加路径，就可以直接使用 arm-linux-gcc 命令了。

```
$ export PATH=$PATH:/usr/local/arm/3.3.2/bin
```

## 4.3 主机开发环境配置

### 4.3.1 主机环境配置

主机端安装 Linux 操作系统的时候，只要磁盘有足够空间，最好是完全安装。因为漏装了有些软件工具，会使得开发很不方便。当然，安装完了以后再来安装需要的软件包也可以，最好是直接使用 rpm 命令来安装对应的包。

接下来就是主机 Linux 环境配置。首先要确认主机的网络接口驱动成功，并且配置网络接口的 IP 地址。可以通过 ifconfig 命令查看所有网络接口，还可以配置网口的 IP 地址。

```
$ ifconfig -a
eth0    Link encap:Ethernet HWaddr 00:0E:A6:B4:56:E6
        inet      addr:    192.168.254.1      Bcast:    192.168.254.255
Mask:255.255.255.0
        UP BROADCAST MULTICAST MTU:1500 Metric:1
        RX packets:0 errors:0 dropped:0 overruns:0 frame:0
        TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen: 0
        RX bytes:0 (0.0 b) TX bytes:0 (0.0 b)
        Interrupt:0 Base address:0xa000

lo      Link encap:Local Loopback
        inet addr:127.0.0.1 Mask:255.0.0.0
```

```
UP LOOPBACK RUNNING MTU:16436 Metric:1
RX packets:241 errors:0 dropped:0 overruns:0 frame:0
TX packets:241 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:15950 (15.5 Kb) TX bytes:15950 (15.5 Kb)
$ ifconfig eth0 192.168.254.1
```

也可以通过 Red Hat Linux 9 的图形配置界面来配置，启动配置窗口的命令为 redhat-config-network。图 4.2 所示就是网络设备配置的图形窗口。



图 4.2 网络设备配置窗口

然后把交叉开发工具链的路径添加到环境变量 **PATH** 中，这样可以方便地在 **Bash** 或者 **Makefile** 中使用这些工具。通常可以在环境变量的配置文件有 3 个，分别在不同的范围生效。

`/etc/profile` 是系统启动过程执行的一个脚本，对所有用户都生效。

`~/.bash_profile` 是用户的脚本，在用户登录时生效。

`~/.bashrc` 也是用户的脚本，在 `~/.bash_profile` 中调用生效。

把环境变量配置的命令添加到其中一个文件中即可。

### 4.3.2 串口控制台工具

串行通信接口很适合作为控制台，在各种操作系统上一般都有现成的控制台程序可以使用。Windows 操作系统有超级终端（Hyperterminal）工具；Linux/UNIX 操作系统有 minicom 等工具。

无论什么操作系统和通信工具，都可以作为串口控制台。如果在 Windows 平台上运行 Linux 虚拟机，这个串口通信软件可以任选一种。



超级终端是 Windows 系统的串口通信工具，完全图形化的界面，操作非常简单。使用超级终端也要配置相应的连接。

建立一个超级终端的连接，需要为其配置如图 4.3 所示的参数。主要是串口号、通信速率和是否流控。每建立一个配置可以保存下来。

Linux 系统通常使用 minicom 串口通信工具。由于 minicom 不是图形窗口的工具，操作起来要麻烦一些。使用 minicom 串口终端之前，需要先配置参数。

minicom 的配置界面是菜单方式。在 Shell 下执行“minicom -s”命令，出现如图 4.4 所示的配置菜单。注意 minicom 程序要访问串口设备，需要以 root 的权限操作。

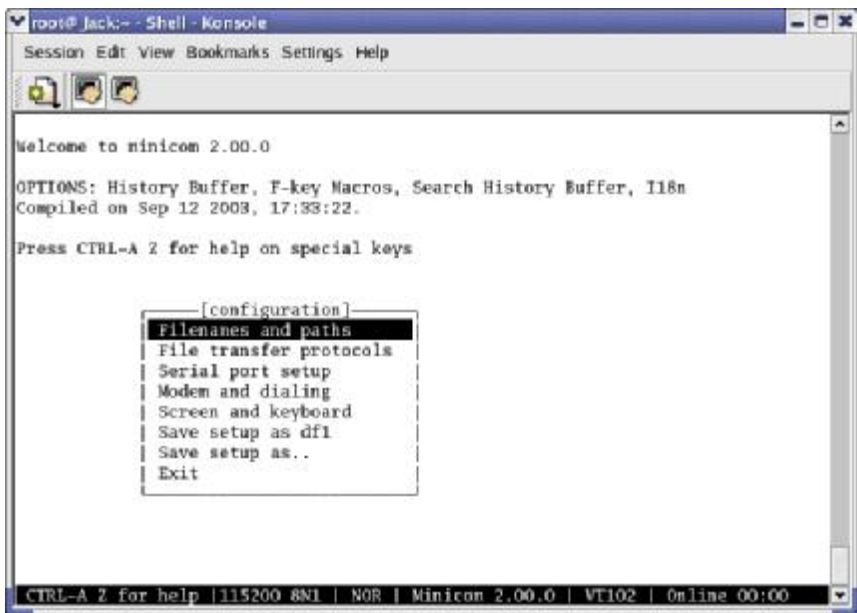


图 4.4 minicom 配置界面

图 4.4 菜单中，可以先通过光标移动键选中菜单项，再敲回车进入子菜单项。

选择“Serial port setup”菜单项，根据目标板的串口通信参数设置。这些配置项都有快捷键（用大写字母显示），可以通过相应的按键选择进入子项。串口配置参数如图 4.5 所示。

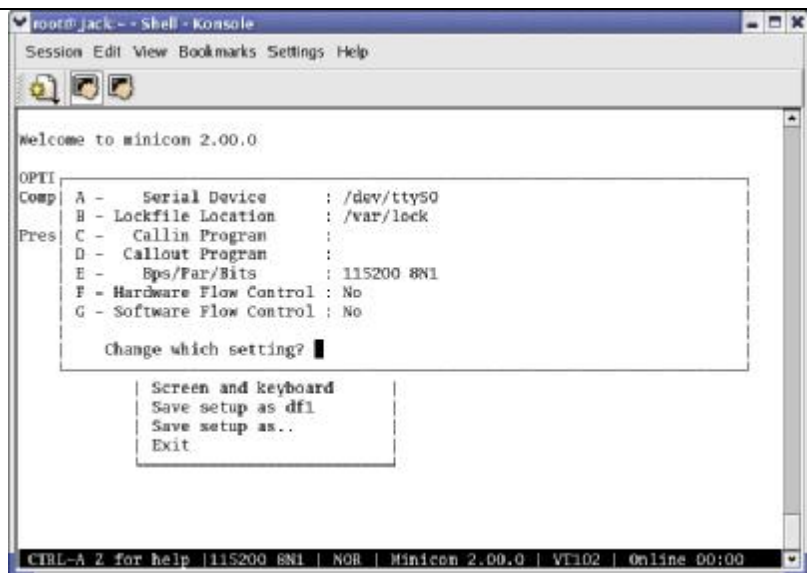


图 4.5 minicom 串口参数配置界面

敲【A】键，可以进入并且修改要使用的串口设备，例如：/dev/ttyS0 是串口 1，/dev/ttyS1 是串口 2。修改完一项，按【ESC】键返回准备选择其他配置项。通常串口通讯速率和硬件流控也要设置，这些项在配置时提供可选的参数值。

参数设置完成后，敲回车键返回如图 4.4 所示的主配置菜单。这时可以保存配置参数。移动光标选择“Save as dfl”菜单项，敲回车保存为缺省设置。

最后移动光标选择“Exit from Minicom”退出。

minicom 的配置参数缺省保存在/etc/minirc.dfl 文件中，内容如下。

```
#/etc/minirc.dfl
# Machine-generated file - use "minicom -s" to change parameters.
pr port          /dev/ttyS0
pu baudrate     115200
pu minit
pu mreset
pu mhangup
pu rtscts       No
```

再启动 minicom 的时候，直接在 Shell 下执行 minicom 命令，就可以进入 minicom 控制台。

当运行在 minicom 控制台下面时，通过组合键可以进入 minicom 菜单。组合键的用法是：先按【Ctrl+A】组合键，再敲入一个命令键。其中主要的几个命令键如下。

【Z】命令键是显示所有的命令并进入命令主菜单，如图 4.6 所示。

【X】命令键退出 minicom，会提示确认退出；

【ESC】键退出命令主菜单，返回到控制台。

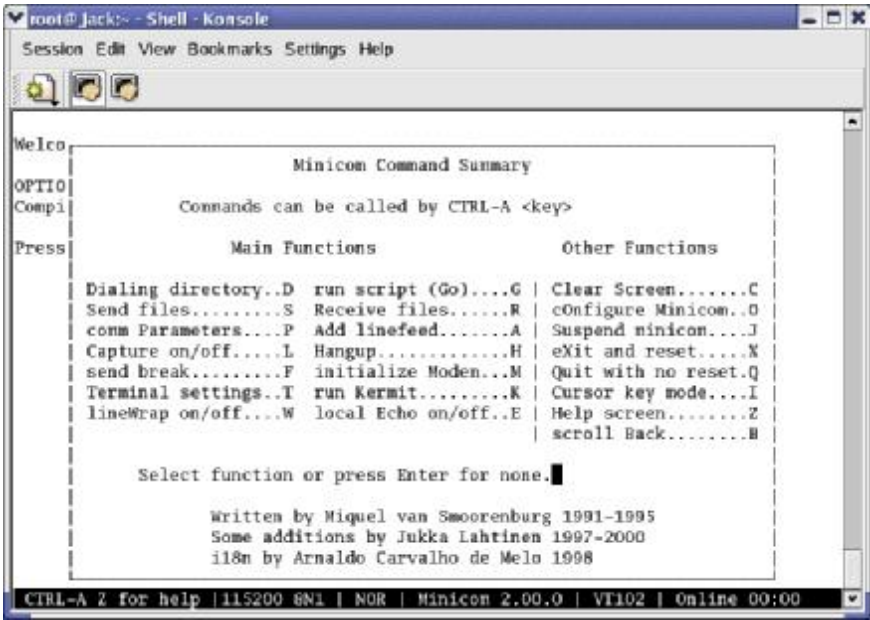


图 4.6 minicom 命令主菜单

### 4.3.3 DHCP 服务

目标板的 Bootloader 或者内核都需要分配 IP 地址。这可以通过动态主机配置协议 (DHCP Dynamic Host Configuration Protocol) 或者 BOOTP 协议实现。

BOOTP 协议可以给计算机分配 IP 地址并且通过网络获取映像文件的路径; DHCP 则是向后兼容 BOOTP 的协议拓展。

Linux 操作系统的主机一般包含 dhcpd 的软件包, 可以配置 DHCP 服务。配置服务的操作需要 root 用户的权限。

首先要确认主机上已经安装所有必需的软件包, 创建相关文件。确认 /var/lib/dhcp/dhcpd.

leases 已经存在。如果这个文件不存在, 可以手工创建目录和文件。

DHCP 服务的配置文件是 /etc/dhcpd.conf, 通过 “man dhcpd.conf” 命令可以查看配置手册。手册详细说明了 dhcpd.conf 几种配置语句的用法。以下一个很好的例子。

```

# /etc/dhcpd.conf
allow bootp;
ddns-update-style none;
subnet 192.168.1.0 netmask 255.255.255.0 {
    group {
        host mytarget {
            hardware ethernet 00:01:EC:E0:0A:0B;
            fixed-address 192.168.1.100;
            filename "zImage";
            option root-path "/usr/local/arm/3.3.2/rootfs";

```

```

    }
}
}

```

上面的配置文件中，为指定的目标板配置了相关网络参数。其中有些参数含义如下。

(1) `host` 指定目标板网络名称为“`mytarget`”，在直接使用 IP 地址的局域网没有什么影响。`mytarget` 可以是目标板的网络名称。

(2) `hardware ethernet` 对应目标板以太网接口的 MAC 地址。这个需要在目标板网络接口打开以后获取相关参数，然后修改配置。

(3) `fixed-address` 是给目标板分配的 IP 地址。通常是指定的一个 IP 地址。

(4) `filename` 是映像文件的名称。目标板的 Bootloader 可以通过 BOOTP 协议获取映像文件，然后下载到目标板内存。这一项并不是所有目标板必要的。

(5) `root-path` 是网络文件系统的路径，目标板可以根据这个路径挂接 NFS 根文件系统。

(6) `subnet` 和 `netmask` 分别是子网和掩码，IP 地址的配置需要在这个网段。

配置好 `dhcpd.conf` 文件以后，就可以启动 `dhcpd` 守候进程了。可以通过图形化的服务配置界面。命令行的方式也很简洁，执行启动命令：

```
$ /etc/init.d/dhcpd start
```

每次修改 `dhcpd.conf` 文件以后，都需要重启 `dhcpd` 服务。执行下列命令。

```
$ /etc/init.d/dhcpd restart
```

如果希望系统每次重启都自动启动这项服务，可以使用 `chkconfig` 命令打开配置。

```
$ chkconfig dhcpd on
```

这样 DHCP 服务就设置完成了。在使用过程中，可能需要经常修改配置文件并且重启 `dhcpd` 服务。

网络服务的启动和停止也可以通过图形化窗口来配置，在 Red Hat Linux 9 系统上可以执行“`redhat-config-network`”命令，弹出图 4.7 所示配置窗口。

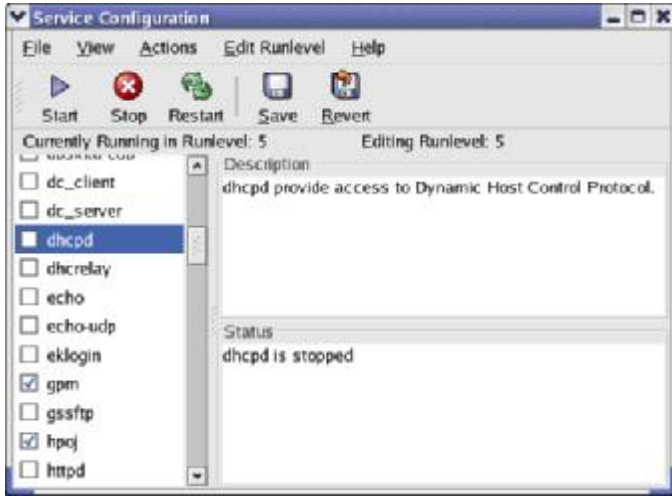


图 4.7 系统服务配置窗口

华清远见

### 4.3.4 TFTP 服务

TFTP 协议是简单的文件传输协议，所以实现简单，使用方便，正好适合目标板 Bootloader 使用。但是文件传输是基于 UDP 的，文件传输（特别是大文件）是不可靠的。

TFTP 服务在 Linux 系统上有客户端和服务端两个软件包。配置 TFTP 服务，必须先安装好。

TFTP 服务也可以通过图形化的配置窗口来启动。当然操作过程需要 root 权限。缺省的情况下，把/tftpboot 目录作为输出文件的根目录。

另外，还可以手工修改 TFTP 配置文件，定制 TFTP 服务。通过命令行的方式启动 TFTP 服务。配置文件/etc/xinetd.d/tftp 内容如下。

```
# /etc/xinetd.d/tftp
# default: off
service tftp
{
    disable          =   yes
    socket_type      =   dgram
    protocol         =   udp
    wait             =   yes
    user             =   root
    server           =   /usr/sbin/in.tftpd
    server_args      =   -s /tftpboot
    per_source      =   11
    cps              =   100 2
    flags            =   IPv4
}
```

其中，disable 是指关闭还是打开 TFTP 服务。如果要打开服务，把 yes 改成 no。server 是指定服务器程序为/usr/sbin/in.tftpd。server\_args 则指定输出文件的根目录为/tftpboot，文件必须放到/tftpboot 目录下才能被输出。

修改配置以后，还需要执行下列命令使 xinetd 重新启动 TFTP 服务。

```
$ /etc/init.d/xinetd restart
```

### 4.3.5 NFS 服务

NFS 服务的主要任务是把本地的一个目录通过网络输出，其他计算机可以远程地挂接这个目录并且访问文件。

NFS 服务有自己的协议和端口号，但是在文件传输或者其他相关信息传递的时候，NFS 则使用远程过程调用（RPC，Remote Procedure Call）协议。

RPC 负责管理端口号的对应与服务相关的工作。NFS 本身的服务并没有提供文件

《嵌入式 Linux 系统开发标准教程》——第 4 章、嵌入式交叉开发环境传递的协议，它通过 RPC 的功能负责。因此，还需要系统启动 portmap 服务。

NFS 服务通过一系列工具来配置文件输出，配置文件是/etc/exports。配置文件的语法格式如下。

共享目录 主机名称 1 或 IP1(参数 1, 参数 2) 主机名称 2 或 IP2(参数 3, 参数 4)

“共享目录”是主机上要向外输出的一个目录。

“主机名称或者 IP”则是允许按照指定权限访问这个共享目录的远程主机。

“参数”则定义了各种访问权限。

exports 配置文件参数说明如表 4.2 所示。

表 4.2 exports 配置文件参数说明

参 数	含 义
rw	具有可擦写的权限
ro	具有只读的权限
no_root_squash	如果登录共享目录的使用者是 root 的话，那么他对于这个目录具有 root 的权限
root_squash	如果登录共享目录的使用者是 root 的话，那么他的权限将被限制为匿名使用 通常他的 UID 与 GID 都会变成 nobody
all_squash	不论登录共享目录的使用者是什么身份，他的权限将被限制为匿名使用者
anonuid	前面关于*_squash 提到的匿名使用者的 UID 设定值，通常为 nobody。这里可 设定 UID 值，并且 UID 也必须/etc/passwd 中设置
anongid	与上面的 anonuid 类似，只是 GID 变成 group ID
sync	文件同步写入到内存和硬盘当中
async	文件会先暂存在内存，而不是直接写入硬盘

举例说明如下。

(1) /usr/local/arm/3.3.2/rootfs \*(rw, no\_root\_squash)

表示输出/usr/local/arm/3.3.2/rootfs 目录，并且所有的 IP 都可以访问。

(2) /home/public 192.168.0.\*(rw)

表示输出/home/public 目录，只允许 192.168.0.\*网段的 IP 访问。

(3) /home/test 192.168.1.100(rw)

表示输出/home/test 目录，并且只允许 192.168.1.100 访问。

(4) /home/linux \*.linux.org (rw, all\_squash, anonuid=40, anongid=40)

表示输出/home/linux 目录，并且允许\*.linux.org 主机登录。在/home/linux 下面写文件时，文件的用户变成 UID 为 40 的使用者。

编辑修改好/etc/exports 这个配置文件，就可以启动服务 portmap 和 NFS 服务了。常用系统启动脚本来启动服务。

```
$ /etc/rc.d/init.d/portmap start
```

```
$ /etc/rc.d/init.d/nfs start
```

也可以通过 service 命令来启动。

```
$ service nfs start
$ service portmap start
```

启动完成后，可以查看/var/log/messages，确认是否正确激活服务。

如果只修改了/etc/exports 文件，并不总是要重启 NFS 服务。可以使用 exportfs 工具重新读取/etc/exports，就可以加载输出的目录。

exportfs 工具的使用语法如下。

```
exportfs [-aruv]
```

- a: 全部挂载（或卸载）/etc/exports 的设置。
- r: 重新挂载/etc/exports 的设置，更新/etc/exports 和/var/lib/nfs/xtab 里面的内容。
- u: 卸载某一个目录。
- v: 在输出的时候，把共享目录显示出来。

在 NFS 已经启动的情况下，如果又修改了/etc/exports 文件，可以执行命令：

```
$ exportfs -ra
```

系统日志文件/var/lib/nfs/xtab 中可以查看共享目录访问权限，不过只有已经被挂载的目录才会出现在日志文件中。

远程计算机作为 NFS 客户端，可以简单通过 mount 命令挂载这个目录使用。例如：

```
$ mount -t nfs 192.168.1.1:/home/test /mnt
```

这条命令就是把 192.168.1.1 主机上的/home/test 目录作为 NFS 文件系统挂载到 /mnt 目录下。如果系统每次启动的时候都要挂载，可以在 fstab 中添加相应一行配置。

如果希望 NFS 服务在每次系统引导时都要启动，可以通过 chkconfig 打开这个选项。

```
$ /sbin/chkconfig nfs on
```

## 4.4 启动目标板

### 4.4.1 系统引导过程

在各种体系结构平台上，多数内核映像都采用压缩格式（MIPS 平台例外，它的映像采用非压缩格式）。Linux 系统的一般启动过程通常划分为内核引导、内核启动和应用程序启动 3 个阶段，如图 4.8 所示。

第一阶段是目标板硬件初始化，解压内核映像，再跳转到内核映像入口。这部分的工作一般由目标板的引导程序和内核映像的自引导程序完成。不同体系结构的目标板引导的方式和程序都有差异。

第二阶段是内核的初始化，初始化设备驱动，挂载根文件系统。这里是 Linux 内

《嵌入式 Linux 系统开发标准教程》——第 4 章、嵌入式交叉开发环境  
核通用的启动函数入口。所有体系结构的目标板都顺序调用统一的函数，尽管有些函数的代码实现是跟体系结构相关的。

第三阶段是执行用户空间的 `init` 程序，完成系统初始化、启动相关服务和管理用户登录等工作。这个阶段可以提供给用户交互界面，例如：`Shell` 命令行或者图形化的窗口界面。也可以自动执行应用程序。

在 Linux 系统启动过程中，有两个关键点。一个是内核映像的解压启动；另一个是根文件系统的挂接。

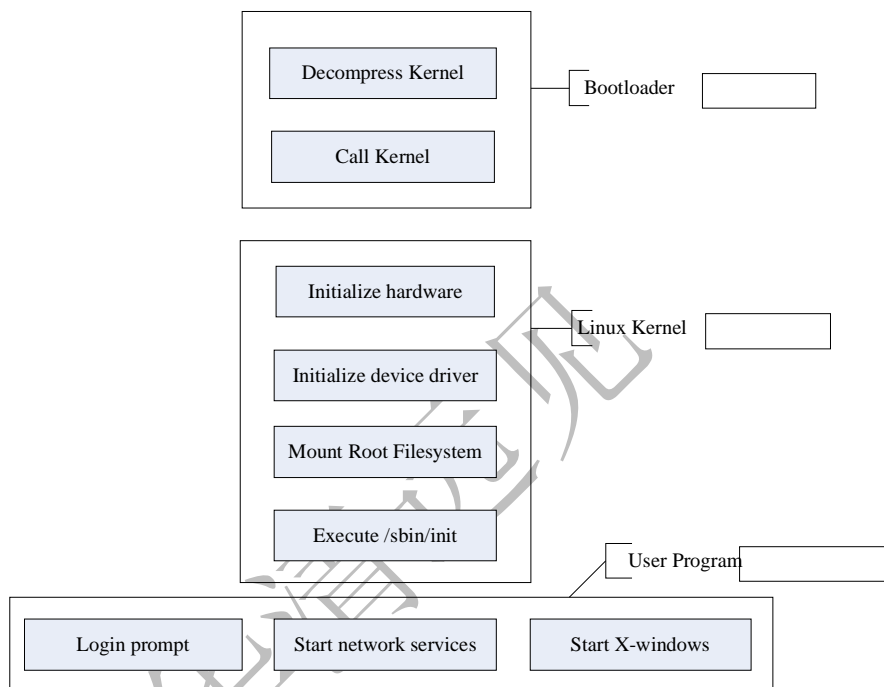


图 4.8 Linux 系统启动过程

#### 4.4.2 内核解压启动

目标板处理器上电或者复位后，首先执行引导程序（`Bootloader`），初始化内存等硬件，然后把压缩的内核映像加载到内存中，最后跳转到内核映像入口执行。这样就控制权完全交给内核映像了。

接下来内核映像继续执行，完成自解压或者重定位，然后跳转到解压后的内核代码入口。这部分主要是 Linux 内核的自引导程序，又叫作 `Linux bootloader`，包含在内核源代码中。这部分引导代码相对简单，不可能替代目标板上的 `Bootloader`。

目标板的 `Bootloader` 具有加载内核映像的功能。在嵌入式 Linux 开发中，经常用到网络加载的方式，就是通过 `TFTP` 协议把内核映像加载到目标板内存。那么目标板的 `Bootloader` 还应该能够驱动网络接口，配置 IP 地址。不同的 `Bootloader` 还有一系列命令进行配置。对于 `U-boot` 的使用和代码分析请参考第 6 章。

这里以 ARM 开发板的 `U-boot` 为例说明网络加载启动内核映像。

U-Boot 1.1.2 (Dec 25 2005 - 14:47:21)

```
U-Boot code: 33F80000 -> 33F98680 BSS: -> 33F9C7C0
RAM Configuration:
Bank #0: 30000000 64 MB
Flash: 2 MB
*** Warning - bad CRC, using default environment
In: serial
Out: serial
Err: serial
=> tftp 30008000 zImage #加载压缩的内核映像 zImage 到内存 0x30008000
TFTP from server 192.168.1.1; our IP address is 192.168.1.100
Filename 'zImage'.
Load address: 0x30008000
Loading:
#####
done
Bytes transferred = 1048584 (100008 hex)
=> go 30008000 #跳转到压缩的内核映像入口
## Starting application at 0x30008000 ... #内核映像自解压启动
Uncompressing
Linux.....
done, booting the kernel.
version 2.6.14 (root@newasus) (gcc version 3.3.2) #5 Sun Dec 25 15:40:42
5CPU: ARM920Tid(wb) [41129200] revision 0 (ARMv4T) #开始执行内核初始化函数
Machine: SMDK2410
Warning: bad configuration page, trying to continue
Memory policy: ECC disabled, Data cache writeback
CPU S3C2410A (id 0x32410002)
S3C2410: core 200.000 MHz, memory 100.000 MHz, peripheral 50.000 MHz
S3C2410 Clocks, (c) 2004 Simtec Electronics
CLOCK: Slow mode (1.500 MHz), fast, MPLL on, UPLL on
CPU0: D VIVT write-back cache
CPU0: I cache: 16384 bytes, associativity 64, 32 byte lines, 8 sets
CPU0: D cache: 16384 bytes, associativity 64, 32 byte lines, 8 sets
Built 1 zonelists
Kernel command line: root=/dev/nfs nfsroot=192.168.1.1:/usr/local/arm/
3.3.2/rootfs
.....
```

这样内核就在目标板上启动起来了。

### 4.4.3 挂载根文件系统

因为文件和应用程序都要存储在文件系统中，所以 Linux 离不开文件系统。在内核启动到最后，必须挂载一个根文件系统。从文件系统的目录下找到 `init` 程序，启动 `init` 进程。

在交叉开发环境中，通常采用 NFS 文件系统。在内核启动过程可以挂载 NFS 根文件系统。这种方式将极大地方便嵌入式 Linux 交叉开发，在第 4.5 节可以很好地体会到这一点。

要使目标板挂载 NFS 根文件系统，需要做两方面的工作。一方面是在主机端配置相应的网络服务；另一个方面就是配置目标板的内核选项。

在第 4.3 节中已经详细说明了 TFTP、DHCP 和 NFS 服务的配置，有关主机端的网络服务配置可以参考。这里看一看还需要配置哪些内核选项。

Linux 内核要挂载 NFS 根文件系统，必须具备以下条件。

(1) 以太网接口驱动正常。

这需要配置相应的网络驱动程序。10/100M 以太网接口的驱动一般在菜单项“Network device support”下。

(2) 配置内核启动命令行参数。

实现上述功能，可以通过 DHCP 服务动态配置；也可以通过内核命令行参数指定。配置内核启动命令行参数缺省值的菜单项“Default kernel command line string”。命令行格式如下。

```
root=/dev/nfs rw nfsroot=<nfs_server>:<root_path> ip=<target_ip>
```

<target\_ip>是为目标板指定的 IP 地址；

<nfs\_server>是指定 NFS 服务器的 IP；

<root\_path>是要挂载的 NFS 服务器的目录；

root=/dev/nfs 则指定要挂载 NFS 根文件系统；

rw 表示按照可读/写属性挂载。

(3) 配置内核挂载 NFS 根文件系统。

要使内核挂载 NFS 根文件系统，首先要支持网络协议配置选项，再选择 NFS 文件系统的支持，然后选择 NFS 为根文件系统。

配置编译完内核，下载到目标板上启动。注意要先把开发主机端的服务启动起来。如果目标板的 IP 地址和 NFS 网络路径需要通过 DHCPD 服务获取，可能需要根据目标板以太网接口的 MAC 地址修改 `dhcpd.conf` 配置文件，并重新启动。

网络根文件系统挂载成功以后，目标板就可以登录到 Shell，执行应用程序了。这样，交叉开发环境就建立起来了。

## 4.5 应用程序的远程交叉调试

### 4.5.1 交叉调试的模型

Linux 下 GCC 编译的程序都是采用 GDB 调试的。对于本地调试，要调试的程序和 GDB 运行在同一台主机上。如果目标板没有 gdb 的调试程序，或者没有 gdb 前端的图形化调试界面，像以前那样本地调试就不大可能。对于本地调试，交叉调试的 gdb 运行在开发主机上，而应用程序运行在目标板上。

在目标板上，通过 gdbserver 控制要调试的程序执行，同时与主机的 gdb 远程通信。可以实现交叉调试的功能。这样，gdb 交叉调试运行在主机端，应用程序运行在目标板端。交叉调试模型如图 4.9 所示。

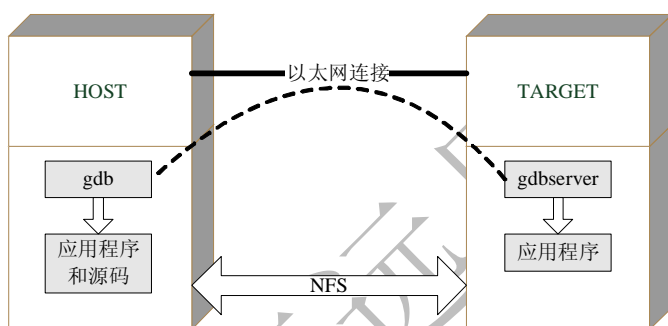


图 4.9 交叉调试模型

### 4.5.2 交叉调试程序实例

举一个简单的例子，来说明交叉调试的过程。

#### 1. 交叉编译

在开发主机上编辑一个 C 程序，再交叉编译，然后在目标板上执行。

(1) 在主机上编辑 hello.c 程序。

```
# include <stdio.h>
main(int argc, char **argv)
{
    int i;
    for ( i=0; i<3; i++)
    {
        printf("Hello i=%d\n", i);
    }
    return 0;
}
```

(2) 交叉编译。

```
$ arm-linux-gcc -o hello hello.c
```

(3) 把可执行程序复制到 NFS 输出的目录下面。

```
$ cp hello /usr/local/arm/3.3.2/rootfs
```

(4) 这时在目标板端也可以访问到同样的程序，执行程序。

```
# ./hello
hello i=1
hello i=2
hello i=3
```

这是一个简单的程序。如果工程包含多个文件，最好使用 Makefile 来管理编译。

## 2. 交叉调试

接下来，还用这个程序，说明交叉调试应用程序过程。交叉调试需要目标板文件中必须有 gdbserver 工具。gdbserver 负责与远程的 gdb 远程通信并且控制本地的应用程序执行。

(1) 编译程序的时候，需要添加-g 编译选项，使程序包含调试信息

```
$ arm-linux-gcc -g -o hello hello.c
```

在主机上要调试的程序必须是带调试信息可执行程序；在目标板上执行的程序则可以使用一个精简过的可执行程序。

(2) 在目标板上，启动 gdbserver，控制程序执行。

```
# gdbserver <host>:2345 hello
```

<host>是主机名称或者 IP 地址。

2345 是网络端口号，服务器在这个端口上等待客户端的连接，这个值可以是任何目标板上可用的端口号。

hello 是调试程序名，还可以添加程序运行的参数。

控制台输出类似下面的显示。

```
Process hello created; pid = 38
```

(3) 在主机端，启动 DDD 和 gdb 调试程序。

```
$ ddd --debugger arm-linux-gdb hello
```

(4) 在 DDD 下窗口的 GDB 控制台下，建立连接。

```
(gdb)target remote <target>:2345
```

<target>是目标板 IP 地址，2345 是端口号，对应 gdbserver 启动时使用的端口号。连接成功，就可以使用 GDB 的命令调试了。

```
Remote debugging using 192.168.1.1:2345
```

(5) 设置断点，执行到断点。

在 main 函数设置断点。单击工具条上的 cont 执行到断点。按照下列命令行的方式同样可以进行调试。

```
(gdb)b main
(gdb)c
```

继续执行到断点，然后就可以单步执行或者设置更多断点调试了。

## 推荐课程： 嵌入式学院-嵌入式 Linux 长期就业班

- 招生简章: <http://www.embedu.org/courses/index.htm>
- 课程内容: <http://www.embedu.org/courses/course1.htm>
- 项目实战: <http://www.embedu.org/courses/project.htm>
- 出版教材: <http://www.embedu.org/courses/course3.htm>
- 实验设备: <http://www.embedu.org/courses/course5.htm>

## 推荐课程： 华清远见-嵌入式 Linux 短期高端培训班

- 嵌入式 Linux 应用开发班:  
<http://www.farsight.com.cn/courses/TS-LinuxApp.htm>
- 嵌入式 Linux 系统开发班:  
<http://www.farsight.com.cn/courses/TS-LinuxEMB.htm>
- 嵌入式 Linux 驱动开发班:  
<http://www.farsight.com.cn/courses/TS-LinuxDriver.htm>