

嵌入式Linux开发环境搭建

本章学习目标

- 掌握嵌入式交叉编译环境的搭建；
- 掌握制作交叉工具链的方法；
- 了解 NFS、TFTP、SSH 等服务的配置方法；
- 掌握嵌入式宿主机开发环境的安装、配置和使用。

3.1 交叉开发模式

3.1.1 嵌入式交叉开发模式

在安装有 Windows 操作系统或者 Linux 操作系统的个人计算机上开发软件时,程序的编辑、编译、链接、调试以及最终的发布运行都是在个人计算机上完成的。而对于嵌入式系统,由于其硬件的特殊性,比如体积小、存储空间小、处理器的处理能力低等因素,一般不能直接使用普通个人计算机上的操作系统和开发工具,所以就要为特定的目标板定制操作系统,而定制的过程都是在普通的个人计算机环境下进行的,人们把这种在个人计算机环境下开发编译出来的软件,放到特定目标板上运行的开发模式叫作交叉开发模式。

市面上的 TQ2440、Mini2440、S5PV210、IMX6 等开发板都是目标板,最初的目标板里面什么程序都没有烧写和安装。以嵌入式 Linux 为例,需要给目标板安装开机引导程序 Bootloader、内核 Kernel 以及文件系统 rootfs,这三部分是最基本的,可能还会有其他一些内容,比如开机 Logo、参数配置、设备树、安全数据等一些定制化的内容。这里 Bootloader 的作用是对硬件进行初始化,加载内核,完成系统启动。系统启动后可以挂载文件系统、运行应用程序等。应用程序存放在文件系统下某个目录里。对于嵌入式 Linux 系统从启动到应用程序运行这一系列过程,与个人计算机的启动过程是相对应的。下面通过一张表格与个人计算机软件做个比较,如表 3-1 所示。

表 3-1 个人计算机系统与嵌入式系统比较

个人计算机系统	嵌入式系统
BIOS	Bootloader(U-boot 等)
操作系统(Windows 7/Windows 10)	操作系统(Linux)
本地磁盘(C 盘、D 盘等)	Flash 存储器(文件系统)

对于嵌入式 Linux 开发,一般可分为如下 3 个步骤,且所有开发过程都是在个人计算机上进行的。

1) Bootloader 开发

通常 Bootloader 就类似于个人计算机的 BIOS,这是硬件上的第一个程序(相对开发者而言),通常只能通过硬件提供的一些调试接口,比如 JTAG 接口,将编译好的镜像文件通过 JTAG 工具烧到目标板上对应的存储区域。随着技术的进步,现在很多芯片的初始化程序烧写也变得简单了,直接通过串口、网口或 USB 接口将程序烧到芯片(比如 Flash 芯片)上。有时也可以把程序事先烧写到存储卡 SD、U 盘等外部存储设备上,然后插到目标板上,上电后从外接存储介质启动。

2) Linux 内核开发

内核编译好后,一般不再需要上述 JTAG 工具直接烧写,由于 Linux 内核镜像文件比较大,JTAG 烧写的速度比较慢。通常可以在 Bootloader 程序中增加一些功能,比如通过 USB 或网线将镜像(编译后的 Linux,通常称为镜像文件)烧写到目标板的存储器中。

3) 制作文件系统

文件系统的大小也很大,有的文件系统可能会有几十兆字节或几百兆字节,所以一般也不会用 JTAG 直接烧写,通常也是通过 Bootloader 去烧写。除此之外,在开发调试时,为了避免频繁烧写系统,可以使用操作系统提供的 NFS 网络文件系统,这样就可以远程挂载文件系统,将要调试的程序放在计算机上对应的文件系统目录下就可以。

下面用一张图简单描述主机与目标板之间的交叉开发模式,如图 3-1 所示。



图 3-1 交叉开发模式

3.1.2 硬件需求

(1) 宿主机计算机要求。

嵌入式 Linux 系统开发,对开发主机的性能不是特别要求,台式机或笔记本计算机均可。现在的计算机配置都很高,不存在不满足需求的问题。原则上需要有一个串口且具有上网功能。

注: 笔记本计算机可以通过 USB 转串口实现串口功能,即购买一个 USB 转串口的转接头即可。

(2) 目标板要求。

基于 ARMv4、v5、v6、v7、v8 架构的处理器都可以用来学习 ARM 技术,比如 ARM7、ARM9、Cortex-A8、Cortex-A15、Cortex-A57 等,本书基于 Cortex-A8,选用的是三星 S5PV210 芯片进行讲解,书中例子都在 TQ210 开发板上调试通过。针对 ARM9 S3C2440 开发板,由于开发过程、开发原理与 S5PV210 类似,所以相关的知识不再重复介绍,只在本书配套的公众号上以视频形式做介绍。另外,对于其他厂家的 ARM 开发板,原理都是一样的,不同的只是寄存器的配置,所以本书的例子也同样适用于这些开发板,只需要稍做修改即可移植到开发板上使用。另外,在学习的过程中如果没有用于实践的开发板,只要能坚持将本书的例子自己写一遍,或参考本书的例子移植更高版本的 Bootloader 和 Linux 系统,并且能编译通过,那对学习嵌入式 Linux 系统开发都是很有帮助的。

3.2 软件环境搭建与配置

3.2.1 宿主机 Linux 操作系统的安装

因为本书讲的是嵌入式 Linux 系统开发,所有的程序开发都要在 Linux 操作系统上进行,



必须要在计算机上安装 Linux 操作系统。而一般开发人员都习惯在 Windows 操作系统上操作,主要是 Windows 的图形操作界面很友好。所以为了既可以使用 Windows 操作系统,又可以开发基于 Linux 的程序,我们就需要在 Windows 上安装一个虚拟机实现一机双系统,这样就不影响我们正常使用 Windows 操作系统,在后面介绍了 SSH 这个服务之后,我们完全可以把装在虚拟机上的 Linux 系统当成 Windows 上的一个“开发工具”来使用。

虚拟机(Virtual Machine)指通过软件模拟的具有完整硬件系统功能的、运行在一个完全隔离环境中的完整计算机系统。流行的虚拟机有 VMware、VirtualBox 和 Virtual PC 等。

Linux 操作系统我们选用 64 位的 Ubuntu18.04 LTS,这是一个稳定的版本,Ubuntu 的资源也比较丰富,可以从 Ubuntu 官方网站下载。虚拟机选择 VirtualBox-6.0.8,它是基于 GPL 协议的开源虚拟机工具。

注: Ubuntu 版本选择,建议使用本书推荐的版本,选用其他版本可能会遇到软件版本不一致等细微差异。关于 VirtualBox 的版本,读者也可以选择其他高版本,本书使用的版本也是比较新的了。

1. 在 Windows 上安装虚拟机

VirtualBox 从官网下载后,先解压,然后直接运行 exe 安装文件进行安装。对于这个软件的具体安装过程比较简单,本书不做重点介绍,一般默认安装在计算机 C 盘下,如果读者不希望安装在 C 盘下,可以手动指定安装路径。下面重点讲解怎么创建一个虚拟机。

创建过程以作者的环境为例,计算机是 8GB 内存、1TB 机械硬盘(如是固态硬盘性能更佳),Windows 7 64 位操作系统(支持 Windows 10),所以在创建虚拟机时,配置 2GB 内存(从系统使用流畅度考虑,建议虚拟机内存大小为实际内存的四分之一,如果计算机本身内存不是很大,给虚拟机分配 1/2 即可),硬盘 80GB(通常不小于 60GB 即可)。下面仅列出关键步骤,其他步骤比较简单,根据提示操作即可。

(1) 检查计算机的 BIOS 是否开启虚拟化功能,如图 3-2 所示。



图 3-2 开启虚拟化

不同牌子的计算机,进入 BIOS 的快捷键是不同的,但方式都一样,在开机前按住键盘上 F1~F12 功能键中的某一个,进去后找到图 3-2 所示的 Virtualization 字样的选项,单击进去开启该项功能即可。

(2) 启动 VirtualBox,选择“新建(N)”,如图 3-3 所示。

(3) 自定义虚拟计算机名称为 u1864book,选择安装路径为 E:\VirtualBox,操作系统类型为 Linux,操作系统版本为 Ubuntu (64-bit),以及内存大小设置、硬盘容量设置等,设置完成后,单击“创建”按钮,即完成虚拟机创建,如图 3-4 所示。



图 3-3 启动 VirtualBox



图 3-4 虚拟机设置

2. 在虚拟机上安装 Linux 操作系统

本书以 64 位 Ubuntu 18.04 LTS 版作为范例,读者如果对 Ubuntu 系统的使用比较熟悉,也可以选择其他版本的 Ubuntu,这里没有强制要求,但建议选择 LTS 版本,可以从官方获得软件支持(服务期 5 年)。下面只列举关键步骤,其他都比较简单,读者可按提示操作即可。

注: 在虚拟操作系统 Ubuntu 18.04 中,可以按 Ctrl+Alt 组合键释放当前鼠标光标,回到 Windows 操作系统中。

(1) 打开上面创建好的虚拟机 u1864book,选择“启动(T)”,然后在启动盘界面选择下载好的 64 位 Ubuntu 18.04 镜像文件,如图 3-5 所示。

(2) 选择 Install Ubuntu,选择键盘布局类型(默认即可),如图 3-6 所示。

(3) 将在线下载升级选项的钩去掉,否则会边安装边升级,需要很长时间,本书使用默认



图 3-5 启动虚拟机

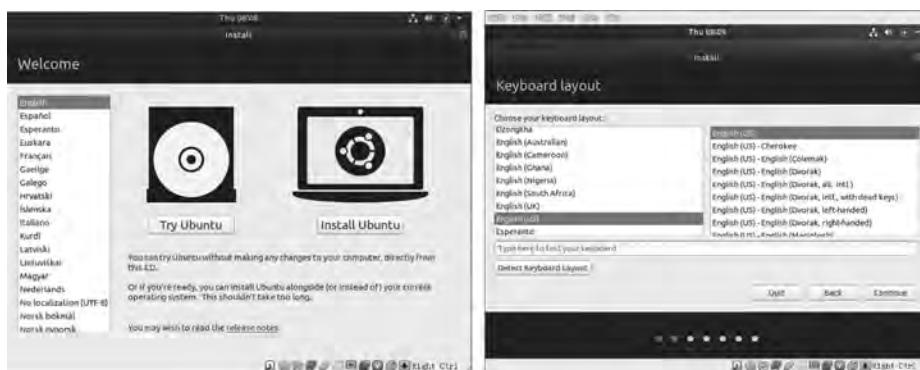


图 3-6 键盘类型选择

的系统即可,无须升级。接下来的一步很关键,要按照图 3-7 所示选择 Something else,如果选择 Erase disk and install Ubuntu 会格式化硬盘,所以通常在虚拟机上安装 Ubuntu 都不会选择这一项,需要特别注意。

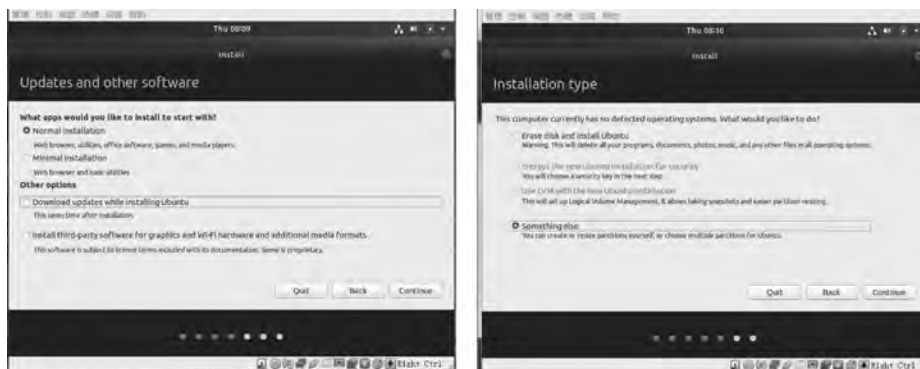


图 3-7 安装类型选择

(4) 给虚拟机上的硬盘分区,通常只需要分两个区就可以:一个主分区和一个交换分区 swap。交换分区的大小一般是虚拟机内存的 2 倍,所以本书交互分区是 4GB(4096MB),剩下的空间都留给主分区。“/dev/sda”可以看作虚拟机硬盘的名称,双击它弹出一个对话框提示是否要分区,单击 Continue 按钮继续操作即可,这样就创建了一个 free space 的空分区,双击它就会弹出具体分区配置界面,默认第一个分区是主分区,所以只需要将挂载点 Mount point 修改为“/”(斜杠符号是 Linux 系统的根结点,或者叫根目录,文件系统的内容都是挂载在这个目录下面,后续讲文件系统再做详细介绍),大小 Size 设置为 81803(这是总分区大小减去 swap 分区大小后的 Size: $85899 - 4096 = 81803$),最后单击 OK 按钮完成主分区创建。接下来再创建交换分区,需要注意的是 Use as 这里选择 swap area,其他默认即可,具体如图 3-8 所示。

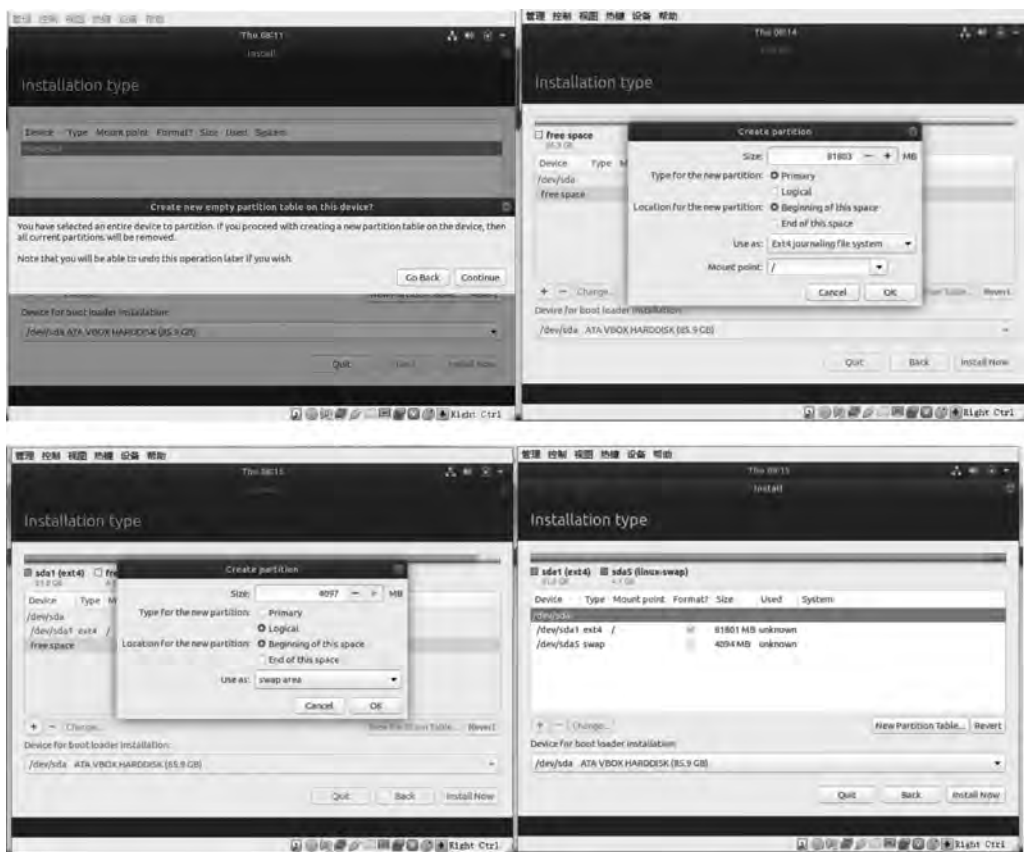


图 3-8 虚拟机硬盘分区

(5) 接着单击 Install Now 按钮会弹出一个硬盘分区确认提示框,这里直接单击 Continue 按钮开始 Ubuntu 系统的安装,首先提示选择时区,默认选择 Shanghai 即可,单击 Continue 按钮进入用户账户和计算机名的创建,创建完成后单击 Continue 按钮开始安装系统,等待大约 30 分钟(安装时长由计算机性能决定)弹出重启系统提示界面,即完成 Ubuntu 系统的安装,具体如图 3-9 所示。

(6) 重启系统输入账号和密码登录 Ubuntu 系统,为了方便对虚拟机上 Ubuntu 系统的操作,比如窗口自动缩放,需要安装 Ubuntu 的增强工具包,具体安装只需要选择虚拟机菜单“设



图 3-9 账户设置与系统安装

备”→“安装增强功能”，即开始自动安装，安装过程中会提示输入用户密码（即登录账户的密码），具体如图 3-10 所示。

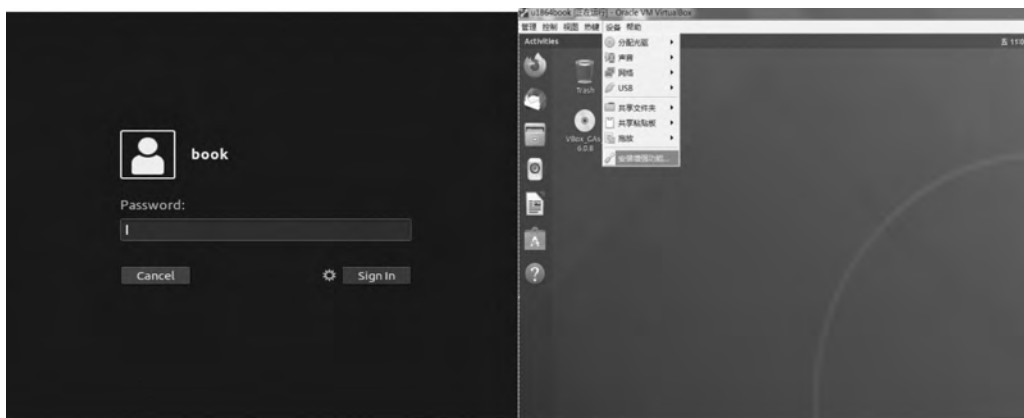


图 3-10 安装增强工具包

3.2.2 配置宿主机 Linux 操作系统

为方便 Windows 操作系统与虚拟机上的 Ubuntu 操作系统（即宿主操作系统，相对于目标板而言）相互访问，以及目标开发板与宿主操作系统之间的通信，需要配置宿主操作系统。



视频讲解

下面主要介绍宿主操作系统网络配置,以及设置网络服务。对于 Windows 操作系统的网络设置,主要是设置网络 IP 地址和网关。如果系统开启了 DHCP 功能,一般都会自动获得 IP 地址,格式都是 192.168.x.x 形式。读者只需确认本机的 IP 地址是什么,然后确定宿主操作系统的 IP 地址与 Windows 系统在同一网段、网关设置与 Windows 操作系统一致即可,本书 Windows 操作系统的 IP 地址为 192.168.1.123,网关为 192.168.1.1。如果所在网络环境管控严格,比如需要代理、防护墙隔离等,可以准备一台交互机(Hub),将开发主机与目标板的网络都接到 Hub 上,以保证开发主机与目标在同一网络环境下。

1. 宿主操作系统网络设置

VirtualBox 虚拟机提供了 3 种网络连接方式:桥接方式(Bridged)、网络地址转译方式(NAT)和主机网络方式(Host-only)。对于嵌入式交叉开发模式,需要将 Windows 操作系统、宿主 Linux 操作系统和目标开发板的 IP 地址设为同一个网段,所以选择桥接方式。具体操作:单击 VirtualBox 的菜单“控制(M)”,选择“设置”弹出如图 3-11 所示的设置页面,连接方式选择“桥接网卡”,界面名称指计算机网卡的名称,对于笔记本电脑可能有两个名称,一个有线网卡,另一个是无线网卡,根据实际使用那一个选择即可,选择好后,单击 OK 按钮保存退出。



图 3-11 宿主机网络设置

设置完桥接方式后,每次启动 Ubuntu 系统都会自动获取一个与 Windows 在同一网段的 IP 地址,宿主机与 Windows 系统即可通信,比如通过 FTP 传输送文件等,即满足本书后续嵌入式系统开发的需要,如需给宿主机设置固定的 IP 地址,可以参考下面的步骤进行配置。不同版本的 Ubuntu 可能网卡设备名称不一样,比如 eth0、enp0s3,以实际系统为准,可以用 ifconfig 命令查看网卡名称(如果没有此命令,用 sudo apt install ifconfig 安装即可),如下所示:

```
book@jxes:~$ sudo apt install ifconfig
```

```
book@jxes:~$ ifconfig
enp0s3: flags = 4163 < UP, BROADCAST, RUNNING, MULTICAST > mtu 1500
    inet 192.168.1.199 netmask 255.255.255.0 broadcast 192.168.1.255
    inet6 fe80::378:7547:155b:2fbc prefixlen 64 scopeid 0x20 < link >
    ether 08:00:27:e1:a8:0d txqueuelen 1000 (Ethernet)
```

(1) 在 Ubuntu 的控制台上打开 `/etc/network/interfaces`, 在里面添加静态 IP 设置。

Ubuntu 控制台实际就是一个命令行工具窗口, 同时按住键盘上的 `Ctrl+Alt+T` 即可快速打开控制台。

```
# 设置静态网卡
auto enp0s3
iface enp0s3 inet static
# 设置 IP
address 192.168.1.199
# 设置子网掩码
netmask 255.255.255.0
# 设置网关
gateway 192.168.1.1
# 设置 dns - nameservers 通过 host 访问 Internet
dns - nameservers 192.168.1.1
dns - nameservers 210.6.10.11
```

注:

- 必须设置 `dns-nameservers`, 否则无法访问 Internet, `dns-nameservers` 后面跟的是网关地址和本地实际的 DNS。
- 如果读者访问 Internet 是通过代理服务器访问的, 上述配置无法访问 Internet, 但不影响嵌入式交叉开发, 目标板、Windows 操作系统和 Ubuntu 三者之间仍可互通。如果要访问外网, 需要安装代理软件并配置, 这里不作介绍。

(2) 在 `/etc/network/interfaces` 设置完后, 需要重启网络服务设置才会生效, 可以用如下方式重启服务。

```
book@jxes:~$ sudo service network-manager restart
```

或者

```
book@jxes:~$ sudo ifdown enp0s3
book@jxes:~$ sudo ifup enp0s3
```

或者直接重启 Ubuntu 操作系统。

注: 配置时需要要有 root 权限, 而“book”是一个普通管理员账户, 所以需要用到 `sudo` 命令 (`sudo` 是 Ubuntu 独有的命令, 用于在普通用户下执行 root 权限) 执行 root 权限下的命令。

(3) 验证宿主机网络是否配置成功。

```
book@jxes:~$ ping www.baidu.com
PING www.a.shifen.com (180.101.49.12) 56(84) bytes of data.
64 bytes from 180.101.49.12 (180.101.49.12): icmp_seq=1 ttl=55 time=239 ms
64 bytes from 180.101.49.12 (180.101.49.12): icmp_seq=2 ttl=55 time=32.4 ms
```

(4) 使用 `netplan` 配置网络。

Ubuntu 从版本 V17 开始引入了 `netplan` 方式配置网络, 一改传统配置的烦琐, 只需要打开 `/etc/netplan/` 目录下的 `yaml` 文件, 在里面配置好后, 执行 `sudo netplan apply` 配置立即

生效。

```
book@jxes:~$ ls /etc/netplan/
01-network-manager-all.yaml
```

配置后的 yaml 文件如下所示：

```
1 # Let NetworkManager manage all devices on this system
2 network:
3     ethernets:
4         enp0s3:
5             dhcp4: no
6             addresses: [192.168.1.199/24]
7             optional: true
8             gateway4: 192.168.1.1
9             nameservers:
10                addresses: [8.8.8.8, 8.8.4.4]
11     version: 2
12     renderer: NetworkManager
```

修改 yaml 配置文件需要注意以下 3 点：

- 子行只能在父行下方,而且要缩进几格,一般缩进 2 格或 4 格即可。
- 缩进不可用 Tab 键,只能以空格方式缩进。
- 同层级的缩进格数要一样。

2. 开启 FTP、SSH 和 NFS 服务

(1) 更新软件源列表和系统。

虽然 Ubuntu 的系统资源比较丰富,但不保证所有软件都事先为我们安装配置好,所以在安装这 3 个服务前先更新 Ubuntu。

更新软件源列表：

```
book@jxes:~$ sudo apt-get update
```

注：apt-get 是 Ubuntu 系统的一个在线安装命令,通过它可以在线安装与升级系统软件以及工具包等,此命令也可直接简写为 apt。

更新系统：

```
book@jxes:~$ sudo apt-get dist-upgrade
book@jxes:~$ sudo apt-get autoremove
```

注：这里更新完成后,在后续开发过程中,如果 Ubuntu 系统有任何更新提示,都可忽略,而且也不建议随便更新系统,以免破坏我们配置好的交叉开发环境。

(2) 安装、设置、启动 FTP 服务。

执行以下命令安装 FTP 服务：

```
book@jxes:~$ sudo apt install vsftpd
```

修改 vsftpd 的配置文件/etc/vsftpd.conf,去掉下面两个选项前面的“#”：

```
# local_enable = YES    # 是否允许本地用户登录
# write_enable = YES    # 是否允许上传文件
```

重启 FTP 服务,使设置生效：

```
book@jxes:~$ sudo /etc/init.d/vsftpd restart
```

(3) 安装、设置、启动 SSH 服务。

执行以下命令安装 SSH server 服务,安装后就会自动运行,无须配置:

```
book@jxes:~$ sudo apt install openssh-server
```

(4) 安装、设置、启动 NFS 服务。

执行以下命令安装 NFS 服务:

```
book@jxes:~$ sudo apt install nfs-kernel-server portmap
```

设置/etc/exports,添加以下内容,后续就可通过网络文件系统访问/opt/nfs_root 目录:

```
/opt/nfs_root * (rw, sync, no_root_squash) (注: no_root_squash 关掉 root 限制)
```

重启 NFS 服务使设置生效:

```
book@jxes:~$ sudo /etc/init.d/nfs-kernel-server restart
```

注: 如果/opt/nfs_root 这个共享目录不存在,重启 NFS 服务会失败:

```
$ sudo /etc/init.d/nfs-kernel-server restart
[...] Restarting nfs-kernel-server (via systemctl): nfs-kernel-server.serviceJob for nfs-server.service canceled.
failed!
```

查看本地对外共享的目录:

```
book@jxes:~$ showmount -e
Export list for jxes: /opt/nfs_root *
```

3.2.3 在宿主机上安装、配置开发环境

本节主要介绍一些常用 C\C++ 开发环境的安装,以及 Linux 下的常用编辑工具,为后面学习嵌入式 C 编程做准备。这些工具在 Ubuntu 18.04 版本都已自带,所以下面重点了解它们的用途。

1. 安装编译环境

安装主要编译工具 gcc、g++ 和 make:

```
book@jxes:~$ sudo apt install build-essential
```

安装语法、词法分析器:

```
book@jxes:~$ sudo apt install bison flex
```

安装 C 函数库的 man 手册:

```
book@jxes:~$ sudo apt install manpages-dev
```

安装 autoconf、automake 用于制作 makefile:

```
book@jxes:~$ sudo apt install autoconf automake
```

安装帮助文档(可选择安装,仅作参考):

```
book@jxes:~$ sudo apt install binutils-doc cpp-doc gcc-doc glibc-doc stl-manual
```

2. 安装编辑工具 vim

执行以下命令安装:

```
qinfen@JXES:~$ sudo apt install vim
```

修改 vim 配置文件/etc/vim/vimrc,在最后添加以下内容,配置完成后下次输入 vi 工具命令就会直接调用 vim 命令:

```
book@jxes:~$ sudo vim /etc/vim/vimrc
book@jxes:~$ source /etc/vim/vimrc
set nu "显示行号"
set tabstop = 4 "制表符宽度"
set cindent "C/C++ 语言的自动缩进方式"
set shiftwidth = 4 "C/C++ 语言的自动缩进宽度"
```

注: source 命令只能使配置在当前控制台有效,重启系统后都有效。

3. 修改工作目录 opt 的所有者

opt 目录是 Linux 自带的一个目录,通常这些目录下面没有任何其他文件或目录,因此可以作为开发目录使用,其所有者默认是 root,所以本书的普通用户 book 是不允许操控 opt 目录的。所以用 book 账户登录后,无法用 FTP 工具上传文件到 opt 目录下,需要修改 opt 的所有者(所属组 root 可以不修改),用命令 chown 修改。

```
book@jxes:~$ sudo chown -R book /opt
```

其中,“-R”是循环修改的意思,如果 opt 下面有其他目录存在,执行此命令后都会被修改。

修改后:

```
book@jxes:~$ ls / -l          //斜杠代表根目录,opt 是在根目录下面
drwxr-xr-x  3 book root  4096  4月  7 23:54 opt
```



视频讲解

3.2.4 制作交叉编译工具链

本书介绍的交叉工具链主要是针对 ARM 处理器的,相关的工具链可通过两种途径获得:一是直接用官方制作好的工具链,可以省去制作过程;二是自己手工制作。为了使读者全面了解工具链,本书对这两种途径分别进行详细介绍。

1. 使用制作好的工具链

制作好的工具链可以从所选芯片官方渠道获得,也可直接使用本书制作好的工具链。本书的工具链是基于 ARMv7 架构,适用于 Cortex-A8 处理器的交叉编译工具链。下面开始解压工具链:

```
book@jxes:~$ cd /opt/tools/crosstool
book@jxes:~$ tar xjf arm-cortex_a8-linux-gnueabi-gcc-4.9.3_glibc-2.22.tar.bz2
book@jxes:/opt/tools/crosstool$ ls
arm-cortex_a8-linux-gnueabi arm-cortex_a8-linux-gnueabi-gcc-4.9.3_glibc-2.22.tar.bz2
```

在环境变量 PATH 里添加工具链的 bin 目录,以便直接使用 bin 下面的工具:

```
$ export PATH = $ PATH:/opt/tools/arm-cortex_a8-linux-gnueabi/bin
```

以上设置只适用于当前终端(即当前控制台),重启系统后,设置就自动失效。要想当前用户不失效,可以在/etc/environment 文件里添加 bin 目录:

```
book@jxes:/opt/tools/crosstool$ sudo vim /etc/environment
PATH = "/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games: /opt/tools/arm-cortex_a8-linux-gnueabi/bin"
```

```
book@jxes:/opt/tools/crosstool $ source /etc/environment
```

由于现在很多芯片都是基于 ARM 架构的,因此,Ubuntu 系统也支持直接安装 ARM 交叉工具链,可以使用 `sudo apt-get install gcc-arm-linux-gnueabi` 安装,这样安装好后,就可以直接使用。但是需要注意的是,这样默认安装的工具链版本都是比较新的版本,有时 gcc 或 glib 的版本过高,可能与我们的程序不匹配,所以还是推荐去网站下载需要的版本,或自己制作。

2. 自己制作工具链

本书制作的工具链基于 crosstool-ng-1.22.0,源代码可以从官方网站下载,下载后的压缩包为 crosstool-ng-1.22.0.tar.xz,将下载后的文件通过 FTP 上传到 /opt/tools 目录下。

注: crosstool-ng 版本要根据实际需要编译的源码版本来选择,比如后面章节介绍的 U-Boot、Kernel 和 Qt 应用开发,这是定制交叉工具链的难点,也是很多人觉得定制很麻烦的原因。

(1) 解压。

```
$ cd /opt/tools
$ tar xJf crosstool-ng-1.22.0.tar.xz
book@jxes:/opt/tools $ ls
crosstool-ng crosstool-ng-1.22.0.tar.xz
```

(2) 安装 crosstool-ng 的软件依赖包。

安装 crosstool-ng 需要相关软件包支持,如果第一步不先安装所需的软件包,直接进入 crosstool-ng 安装会提示缺少软件。下面提供的软件包是作者制作时收集的一些软件包。

注: 如果选择其他版本的 crosstool-ng 制作工具链,可能会提示缺少其他软件包,可以先退出安装,把缺少的软件安装好后再安装。

所需软件包汇总安装如下,安装过程中会提示输入确认信息,直接输入“y”即可。

```
sudo apt-get install autoconf automake libtool libtool-bin libexpat1-dev libncurses5-dev
bison flex patch curl cvs texinfo build-essential subversion gawk python-dev gperf g++libexpat1-dev
help2man
```

缺少相关软件包(libexpat1-dev)时的提示信息:

```
[ INFO ] Installing cross-gdb
[ EXTRA ] Configuring cross-gdb
[ EXTRA ] Building cross-gdb
[ ERROR ] configure: error: expat is missing or unusable
[ ERROR ] make[2]: *** [configure-gdb] Error 1
[ ERROR ] make[1]: *** [all] Error 2
```

(3) 编译安装 crosstool-ng。

制作交叉工具前需要配置、安装 crosstool-ng,在安装前先创建两个目录分别用于安装和制作工具链。

```
$ mkdir crosstool_install crosstool_build
```

配置 crosstool-ng:

```
book@jxes:/opt/tools $ cd crosstool-ng
book@jxes:/opt/tools/crosstool-ng $ ./configure --prefix=/opt/tools/crosstool_install
checking build system type... x86_64-pc-linux-gnu
checking host system type... x86_64-pc-linux-gnu
checking for a BSD-compatible install... /usr/bin/install -c
```

```
checking for grep that handles long lines and - e... /bin/grep
...
checking ncurses.h presence... yes
checking for ncurses.h... yes
checking for library containing initscr... -lncurses
checking for library containing tgetent... none required
configure: creating ./config.status
config.status: creating Makefile
```

注:

- prefix 自定义安装的路径,如果路径的所有者不是当前用户或非 root 用户,配置不会成功,所以推荐使用当前用户,本书的/opt 目录已经是当前用户权限。
- crosstool-ng 默认是非 root 用户安装,root 用户安装需要修改/scripts/crosstool-ng.sh.in 这个文件,在“# Check running as root”这句上面添加“CT_ALLOW_BUILD_AS_ROOT_SURE=true”,这个方式安装一般很少用到。

编译、安装 crosstool-ng:

```
book@jxes:/opt/tools/crosstool-ng$ make
SED      'ct-ng'
SED      'scripts/crosstool-NG.sh'
SED      'scripts/saveSample.sh'
SED      'scripts/showTuple.sh'
...
CC      'nconf.o'
CC      'nconf.gui.o'
LD      'nconf'
SED      'docs/ct-ng.1'
GZIP     'docs/ct-ng.1.gz'
book@jxes:/opt/tools/crosstool-ng$ make install
GEN      'config/configure.in'
GEN      'paths.mk'
GEN      'paths.sh'
MKDIR    '/opt/tools/crosstool-install/bin/'
INST     'ct-ng'
...
INST     'docs/* .txt'
MKDIR    '/opt/tools/crosstool-install/share/man/man1/'
INST     'ct-ng.1.gz'
For auto-completion, do not forget to install 'ct-ng.comp' into
your bash completion directory (usually /etc/bash_completion.d)
```

注: 编译 crosstool-ng-1.22.0 时如果遇到如下错误提示:

```
DEP      'zconf.tab.dep'
CC      'zconf.tab.o'
In file included from zconf.tab.c:213:0:
zconf.hash.c:163:1: error: conflicting types for 'kconf_id_lookup'
kconf_id_lookup (register const char * str, register size_t len)
zconf.hash.c:34:25: note: previous declaration of 'kconf_id_lookup' was here
static struct kconf_id * kconf_id_lookup(register const char * str, register unsigned int len);
```

这是由于 kconf_id_lookup 这个函数的参数类型不一致,找到相关的源码后,将 size_t len 改为 unsigned int len,再编辑就不会有问题,修改如下:

```
book@jxes:/opt/tools/crosstool-ng$ vim kconfig/zconf.hash.c
```

```
163 kconf_id_lookup (register const char * str, register size_t len)
```

修改后:

```
163 kconf_id_lookup (register const char * str, register unsigned int len)
```

(4) 将 crosstool-ng 工具命令添加到环境变量 PATH。

```
$ cd /opt/tools
$ export PATH = $ PATH: /opt/tools/crosstool_install/bin
```

在当前终端就可以直接使用 ct-ng 命令。

```
$ ct-ng help
This is crosstool - NG version crosstool - ng - 1.22.0
...
Use action "menuconfig" to configure your toolchain
Use action "build" to build your toolchain
Use action "version" to see the version
See "man 1 ct - ng" for some help as well
```

上述信息说明 crosstool-ng 的配置与安装是成功的。另外,通过帮助提示可以看到, crosstool-ng 有点类似于 Linux,也有 menuconfig 用于配置工具链。

(5) 选择配置文件。

用命令 ct-ng list-samples 可以查看此版本的 crosstool-ng 默认支持哪些处理器。

```
$ ct-ng list - samples
...
[L.X] arm - cortex_a15 - linux - gnueabi
[L..] arm - cortex_a8 - linux - gnueabi
[L.X] arm - cortexa9_neon - linux - gnueabihf.....
```

此版本已支持 Cortex-A8 处理器,接下来直接使用这个默认配置文件制作交叉工具链。

复制默认配置文件到 crosstool_build,并复制 crosstool.config 为 .config:

```
$ cd /opt/tools/crosstool - ng/samples/arm - cortex_a8 - linux - gnueabi
$ cp * /opt/tools/crosstool_build
$ cd /opt/tools/crosstool_build
$ cp crosstool.config .config
```

(6) 执行 menuconfig 配置工具链(见图 3-12)。

```
$ cd /opt/tools/crosstool_build
$ ct-ng menuconfig
```

针对打开的默认配置,做如下修改。

Paths and misc options → (/opt/tools/crosstool/src)Local tarballs directory

指定软件安装包目录,将下载好的安装包放在此目录可以避免安装时下载。

(/opt/tools/ crosstool/ \$ {CT_TARGET})Prefix directory

此项指定交叉编译器的安装路径。

(4) Number of parallel jobs

此项指定同时执行 4 个工作线程,提高编译速度。

注: 计算机配置的不同,这里的设置也不同。作者的处理器是 4 核,所里这里设为 4; 如



图 3-12 Crosstool 配置界面

果读者是 8 核,这里设为 8 就可以;如果还支持超线程技术,设为 16 也可以。

Target options →

Floating point:(Softfp(FPU)) →

此项表明使用软浮点技术,具体要视处理器情况而定。

Toolchain options →

(cortex_a8)Tuple's vendor string

由于本书选用默认配置,此处默认为 cortex_a8。这样制作后的交叉编译器名前缀即为“arm-cortex_a8-linux-gnueabi”,如果此处不设置,编译器名前缀即为:“arm-unknown-linux-gnueabi”。

(arm - linux)Tuple's alias

设置别名,这样会给每个工具创建一个软链接,比如: arm-linux-gcc 链接到 arm-cortex_a8-linux-gnueabi-gcc。

Operating System →

Linux kernel version(3.10.93) →

指定内核版本,这个选择对后续编译 Kernel 源码有帮助。

Linkers to enable (ld,gold,) →

支持 gold 连接,搭建 Qt 5 开发环境时用到。

C compiler →

gcc version(4.9.3) →

指定 gcc 软件包的版本(升级到 4.9.3 支持 Qt 5 开发,Qt 5 需要 C++11 的支持),本书提供的 gcc-4.9.3.tar.bz2 已经将下面错误修复,可直接使用。

注: 如果 Ubuntu 系统自带的 gcc 版本高于 4.9.3,即在高版本下编译低版本会报如下错误:

```
[ERROR]      cfns.gperf:101:1: error: 'const char * libc_name_p(const char *, unsigned int)'
redeclared inline with 'gnu_inline' attribute
```

修改如下两个源码文件:

```
/opt/tools/crosstool_build$ vim .build/src/gcc-4.9.3/gcc/cp/cfns.h
/opt/tools/crosstool_build$ vim .build/src/gcc-4.9.3/gcc/cp/cfns.gperf
cfns.h 第 56~57 行中间插入
# ifdef __GNUC_STDC_INLINE__
__attribute__((gnu_inline))
# endif
cfns.gperf 第 25~26 行中间插入
# ifdef __GNUC_STDC_INLINE__
__attribute__((gnu_inline))
# endif
[ * ]Compile libmudflap
```

这是一个软件调试工具,可以检查内存泄露,不是必选项。

```
C - library →
glibc version (2.22) →
```

指定 C 库版本。

```
[ * ] Force unwind support (READ HELP!)
```

C 库配置时需要此功能。

```
Paths and misc options →
[ * ] Try features marked as EXPERIMENTAL
```

选择此扩展项后即开启 Companion tools 功能选项。

```
Companion tools →
[ * ] Build some companion tools
[ * ] make
```

选择 make 后,编译时就会用选择的 make 来编译。

注: 建议直接使用本书配套软件包里 make-3.81.tar.bz2,如果是编译时自动下载的 make 软件包,需要做如下修改(与 C 库的兼容):

未修改的错误提示:

```
/opt/tools/crosstool_build/.build/src/make-3.81/glob/glob.c:1361: undefined reference to
'__alloca'
```

修改 glob.c 如下:

```
210 // #if !defined __alloca && !defined __GNU_LIBRARY__ //此处注释掉
211
212 # ifdef __GNUC__
213 # undef alloca
...
```

```

228
229 # define __alloca      alloca
230
231 // # endif           //此处注释掉

```

其他配置项,可以对照下面软件包中软件的版本选择即可,最后选择 Save an Alternate Configuration 将上面的修改保存后退出,所有配置信息会自动保存在.config 文件里。

注: 配置软件包版本时一定要注意软件包之间的匹配性,否则制作过程中会出错,一旦出错,可以退出制作,重新打开 menuconfig 修改相应软件包的版本。

(7) 下载工具链所依赖的软件包。

创建 crosstool/src 目录:

```

$ cd /opt/tools
$ mkdir -p crosstool/src

```

打开.config 查找所需要软件包及其版本信息,然后逐一下载,都下载完成后,通过 FTP 上传到/opt/tools/crosstool/src 目录。下面是本书工具链所需软件包:

```

cloop-ppl-0.18.4.tar.gz libelf-0.8.13.tar.gz glibc-2.16.tar.xz ltrace-0.7.3.orig.tar.bz2
glibc-ports-2.22.tar.xz dmalloc-5.5.2.tgz duma_2_5_15.tar.gz expat-2.1.0.tar.gz
binutils-2.24.tar.bz2 strace-4.10.tar.xz ncurses-6.0.tar.gz gdb-7.1.0.tar.xz libiconv-1.
14.tar.gz
gmp-6.0.0a.tar.xz mpfr-3.1.3.tar.xz ppl-0.10.2.tar.bz2 gcc-4.9.3.tar.bz2 isl-0.14.
tar.bz2
expat-2.1.0.tar.gz linux-3.10.93.tar.bz2 mpc-1.0.3.tar.gz make-3.81.tar.bz2 m4-1.4.13.
tar.xz
autoconf-2.65.tar.xz automake-1.11.1.tar.bz2 gettext-0.19.6.tar.xz libtool-2.4.6.tar.xz

```

注: ncurses-6.0.tar.gz 是修改后的软件包,如果从网络下载编译,可能会报下面错误:

```

[ALL ] from ../ncurses/lib_gen.c:19:
[ERROR] _29430.c:835:15: error: expected ')' before 'int'
[ALL ] ../include/curses.h:1594:56: note: in definition of macro 'mouse_trafo'
[ALL ] #define mouse_trafo(y,x,to_screen) wmouse_trafo(stdscr,y,x,to_screen)

```

修改如下:

```

/opt/tools/crosstool_build/.build/src/ncurses-5.9$ vim include/curses.tail
104 extern NCURSES_EXPORT(bool) mouse_trafo(int *, int *, bool); /*
generated */去掉后面的注释

```

(8) 编译。

进入 crosstool_build 目录执行 build 编译,整个编译过程主要是安装软件包。

```

$ cd /opt/tools/crosstool_build
$ ct-ng build

```

编译成功后会看到如下提示信息:

```

[INFO ] Performing some trivial sanity checks
[INFO ] Build started 20200401.083540
[INFO ] Building environment variables
[EXTRA] Preparing working directories
[EXTRA] Installing user-supplied crosstool-NG configuration
...
[EXTRA] Removing installed documentation

```

```
[INFO ] Cleaning - up the toolchain's directory: done in 2.47s (at 97:53)
[INFO ] Build completed at 20200401.101332
[INFO ] (elapsed: 97:52.18)
[INFO ] Finishing installation (may take a few seconds)...
[61:12] book@jxes:/opt/tools/crosstool_build$
```

整个编译过程大约需要 62 分钟,实际制作时间取决于读者计算机的性能。

(9) 修改环境变量 PATH 并测试工具链。

本书制作好的工具链 bin 目录为/opt/tools/crosstool/arm-cortex_a8-linux-gnueabi/bin
具体怎么修改环境变量,可以参考前面所讲,方法一致。

在环境变量 PATH 里添加好新制作的工具链后,可以用如下方法测试,如果看到工具的版本信息,说明制作成功。

```
$ arm-linux-gcc -v
gcc version 4.9.3 (crosstool-NG crosstool-ng-1.22.0)
```

关于嵌入式编程基础知识的介绍,见配套资源补充资料第 1 章,下载方式详见前言。