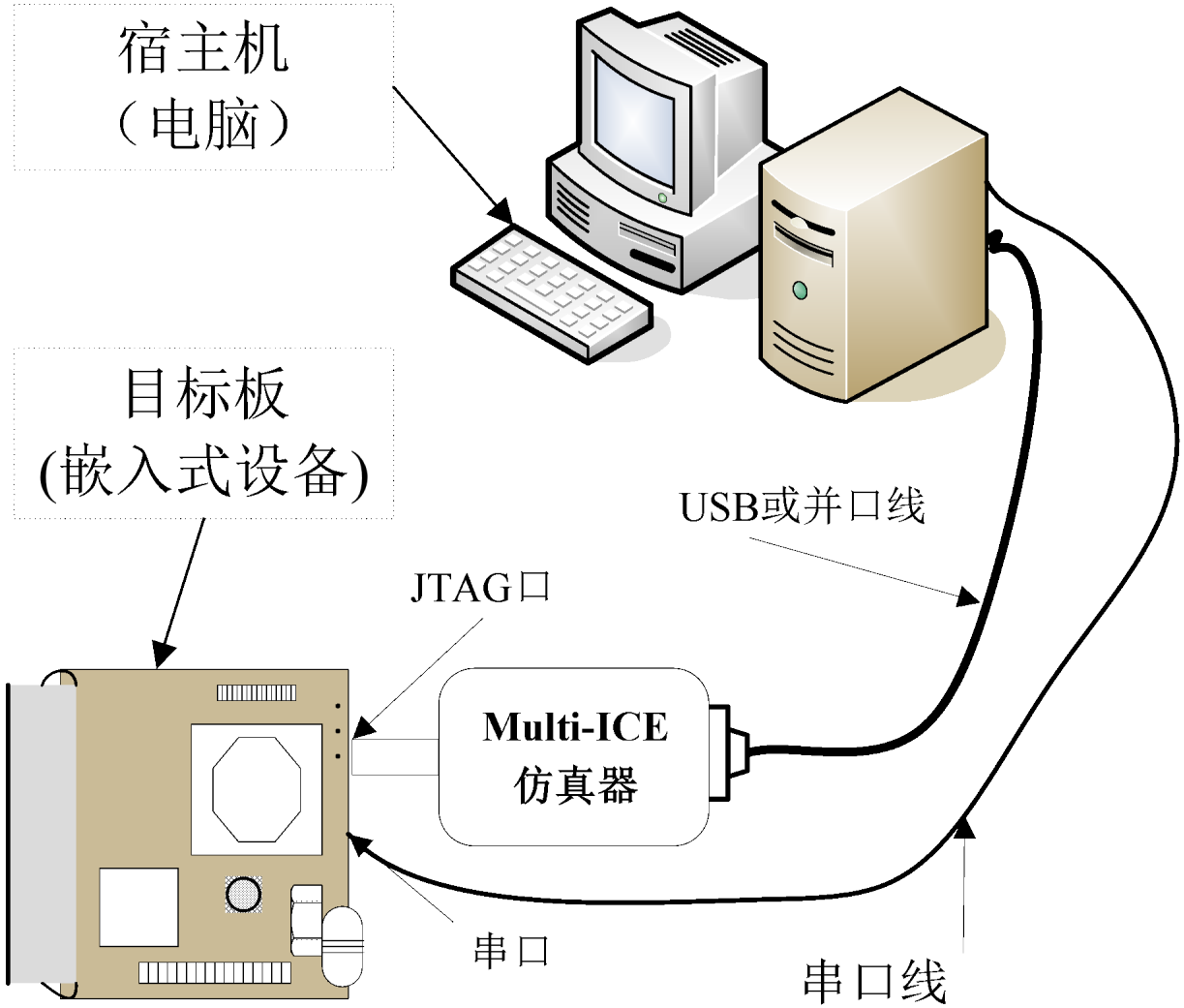
嵌入式Linux应用程序设计

1. Linux开发环境的建立
2. Linux开发环节
3. 嵌入式Linux交叉编译环境的建立
4. Linux开发环境的选择
5. 交叉编译过程
6. GNU工具的使用及开发流程
7. Linux常用命令

开发环境的建立

- **在进行项目开发前，首先要做的就是搭建一套基于Linux操作系统的应用开发环境，一般由目的板和宿主机所构成。目的板用于运营操作系统和系统应用软件，而目的板所用到的操作系统的内核编译、应用程序的开发和调试则需要经过宿主机来完毕。**
- **开发环境对硬件没有特殊的要求，但是为了双方之间建立连接关系，关键的几种接口电路如串口、以太网接口和USB接口是必不可少的。**用于开发和调试目的板上所用到的操作系统、应用程序等全部软件。
- 这种在宿主机上开发程序、在目的板上运营程序的方式，一般就叫做交叉开发。

开发环境的建立



开发环境的建立

- 目的板(Target)能够是嵌入式应用软件的实际运营环境，当然也能够是替代实际环境的仿真系统（如软件模拟器）。
- 宿主机(Host)经过串口、网络连接或调试接口(如JTAG仿真器)与目的机通信。宿主机的软硬件资源比较丰富，其操作系统主要有Windows和Linux两种，其上用于开发程序的那套软件工具，一般叫做开发工具链。

嵌入式Linux的构成

- 最基本的嵌入式Linux系统需要3个基本元素：
  - 系统引导程序：用于完毕机器加电后的系统定位引导；
  - Linux系统内核：为嵌入式应用提供一种软件环境，为应用程序完毕基本的底层资源管理工作；
  - 初始化过程：完毕基本的初始化。
- 为使这个最小嵌入式系统具有一定的实用性，还需加上硬件的驱动程序及一个或几种应用进程以提供必要的应用功能支持。假如应用比较复杂，可能还需要添加一种能够在ROM或RAM中使用的文件系统、TCP/IP网络协议栈等。在PDA领域，还需要加上一个GUI支持。

Linux开发环节

- Linux操作系统作为一种多任务、稳定可靠、内核可裁剪的系统，是开发嵌入式软硬件产品的优异软件平台。
- 嵌入式Linux是一种开放源码、软实时、多任务的嵌入式操作系统。一般它是在原则Linux的基础上针对嵌入式系统进行裁剪和优化后形成的。裁剪和优化后的Linux体积更小，性能愈加稳定，而且源代码本身是免费的。这将大大降低开发商的成本，更具市场竞争力
- 详细进行Linux开发的环节如下：
  - BOOTLOADER开发；
  - Linux开发环境建立；
  - Linux内核移植；
  - 应用程序开发。

嵌入式Linux的开发环境

- 个人用Linux开发嵌入式应用程序，能够在自己的PC机上安装一套Linux操作系统，使用Linux中的X Windows打开若干个窗口进行编译、下载和调试等。
- 当多名工程师共同来开发一种系统时，能够用1台PC机运营Linux作为服务器，每个开发工程师都经过局域网用Telnet登录到这台服务器上，被开发的目的板也挂在网上。然后在服务器的Linux环境下用GNU gcc编译成生目的代码，再用FTP传回到自己的PC机上，最终经过串口或网络下载到目的机上即可完毕整个嵌入式系统的开发。

交叉编译环境的建立

- 所谓交叉编译，就是在一种平台上生成能够在另一种平台上执行的代码。
- 采用交叉编译的主要原因在于，多数嵌入式目的机不能提供足够的资源供编译过程使用，因而只好将编译工作转移到高性能的宿主机中进行。
- 交叉编译环境是一种由编译器、连接器和解释器构成的综合开发环境。交叉编译工具主要涉及针对目的系统的**编译器gcc**、目的系统的**二进制工具binutils**、目的系统的**原则c库glibc**和目的系统的**Linux内核头文件**。

GNU交叉编译

- **Linux采用GNU交叉编译器。**
- **GNU的交叉编译器，涉及下列组件：**

1. gcc交叉编译器，即在宿主机上开发编译目的上可运营的二进制文件；

2. binutils辅助工具，涉及objdump、objcopy等；

3. gdb调试器。

- **对于ARM能够采用如下两个版本的编译器：**

arm-elf-

arm-linux-

arm-elf- 交叉编译器

- **arm-elf-gcc**
- **arm-elf-ld**
- **arm-elf-as**
- **arm-elf-objdump**
- **arm-elf-objcopy**
- **arm-elf-gdb**

arm-linux- 交叉编译器

- **arm-linux-gcc**
- **arm-linux-ld**
- **arm-linux-as**
- **arm-linux-objdump**
- **arm-linux-objcopy**
- **arm-linux-gdb**

选择Linux开发环境

- 单机模式（一台计算机）
  - Linux环境，推荐RedHat 9.0；
  - Windows环境 + Cygwin；
  - Windows环境 + VMWare虚拟机（安装RedHat 9.0）。
- 双机模式（两台计算机）
  - Windows + Linux。

在linux环境下进行Linux开发

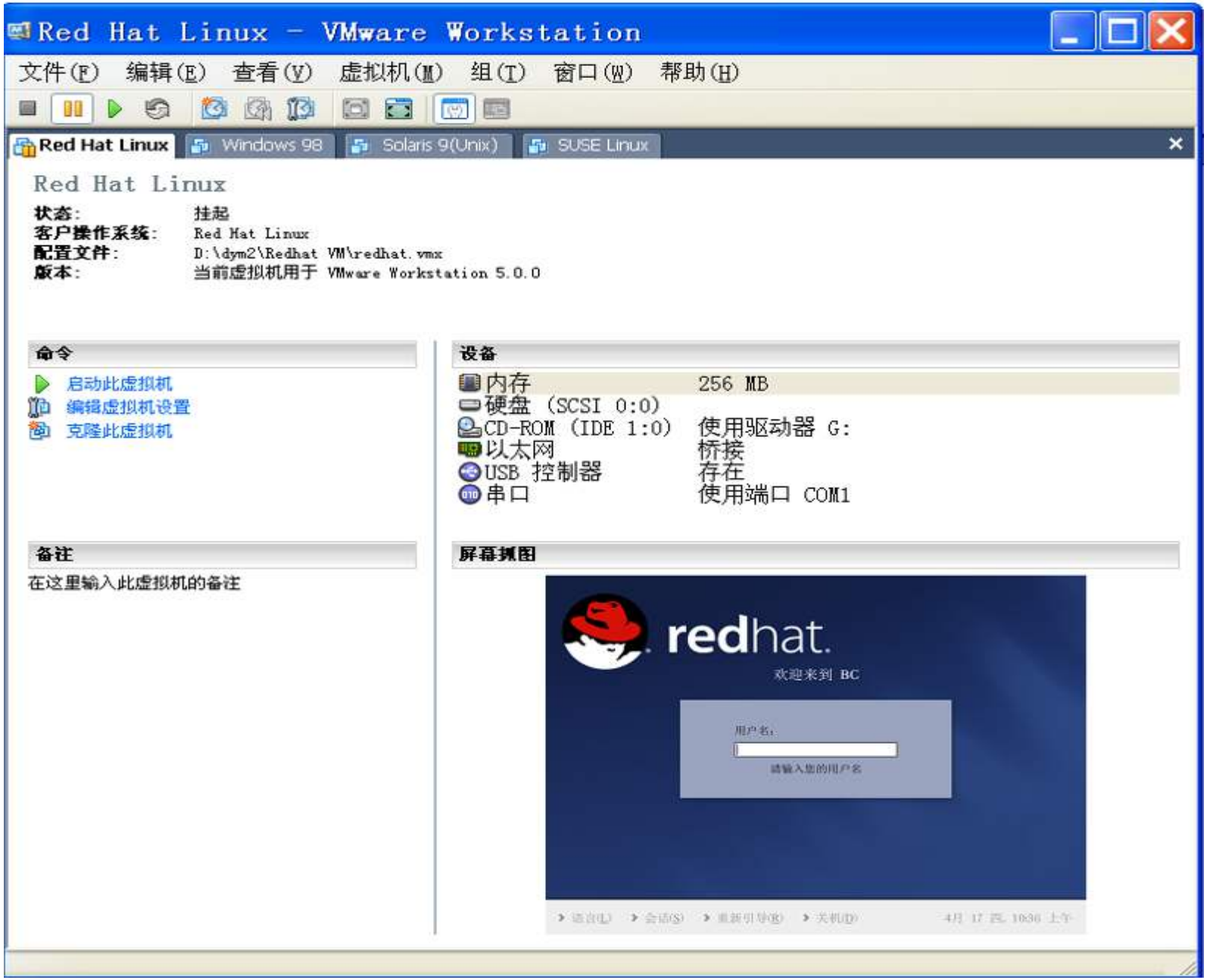
- Linux程序的编辑、编译和调试都是在Linux环境下进行的。
- 必须采用支持GDB的调试器（如ADT1000A）才干进行调试，不然只能经过BOOTLOADER进行简朴的程序烧写运营等。
- 调试信息的打印能够经过串口完毕，使用linux下的minicom超级终端程序（类似于Windows上的超级终端工具）接受并显示目的板传送的打印信息。
- Linux串口驱动完毕后，能够经过内核打印函数printk向串口打印信息，这是Linux内核调试过程中最有效的手段。

在Windows + Cygwin环境下进行Linux开发

- Cygwin是运营于Windows中的一种应用程序，它能够使得Linux环境下的应用程序能够在Cygwin环境下进行编译，即能够在Windows进行编译。
- Linux内核配置和编译等在Cygwin环境下运营，程序编辑和调试都在Windows环境下进行，必须采用支持Windows下进行linux调试的调试器（如ADT1000A）才干进行调试，不然只能经过BOOTLOADER进行简朴的程序烧写运营等
- 调试信息的打印能够经过串口完毕，使用Windows上的超级终端工具接受并显示目的板传送的打印信息。

在Windows + VMWare虚拟机环境下进行Linux开发

- VMWare是运营于Windows中的一种应用程序，是一种虚拟机，能够在其上安装多种操作系统，相当于在Windows上安装一种虚拟的操作系统。
- 假如需要使用只支持Windows环境下调试的调试器，能够经过在VMWare中安装linux虚拟机，在该虚拟机中进行编辑、编译Linux，然后经过网络（ftp、nfs、ssh）等手段传送到Windows中，进行调试，这么编译和调试能够分别在linux和windows环境下但是是在一台电脑中完毕，它的缺陷是系统要求较高，运营速度慢。
- 也能够在两台电脑中分别安装linux和windows，分别实现编译和调试。


Red Hat Linux - VMware Workstation
文件(F) 编辑(E) 查看(V) 虚拟机(M) 组(T) 窗口(W) 帮助(H)
Red Hat Linux
Windows 98
Solaris 9(Unix)
SUSE Linux
Red Hat Linux
状态: 挂起
客户操作系统: Red Hat Linux
配置文件: D:\dym2\Redhat VM\redhat.vmx
版本: 当前虚拟机用于 VMware Workstation 5.0.0
命令
启动此虚拟机
编辑虚拟机设置
克隆此虚拟机
设备
内存 256 MB
硬盘 (SCSI 0:0)
CD-ROM (IDE 1:0) 使用驱动器 G:
以太网 桥接
USB 控制器 存在
串口 使用端口 COM1
备注
在这里输入此虚拟机的备注
屏幕抓图
redhat.
欢迎来到 BC
用户名:

交叉编译过程

在Linux交叉编译环境下，整个编译过程大致上能够分为下列几种环节：

- **编译binutils**

  运营configure文件，并使用--prefix=$PREFIX参数指定安装途径，使用--target=arm-linux参数指定目的机类型，然后执行make install。

- **配置Linux内核头文件**

  执行make mrproper进行清理工作，然后执行make config ARCH=arm（或make menuconfig/xconfig ARCH=arm）进行配置。一定要在命令行中使用ARCH=arm指定CPU架构，因为默认架构为主机的CPU架构。

交叉编译过程

- **第一次编译gcc**

  这一步执行make install，生成一种最简朴的gcc。因为编译整个gcc需要目的机的glibc库，而它在刚开始时还不存在，所以需要首先生成一种最简朴的gcc，它只需要具有编译目的机glibc库的能力即可。

- **交叉编译glibc**

  这一步经过执行configure和make install来对glibc库进行交叉编译。

  configure的运营参数如下：

  CC＝arm-linux-gcc ./configure

  --prefix=$PREFIX/arm-linux

  --host=arm-linux

  --enable-add-ons

交叉编译过程

- **第二编译gcc**

  首先运营configure，参数设置为

  --prefix=$PREFIX

  --target=arm-linux

  --enable-languages=c,c++

  接着运营make install。这么整个交叉编译环境就生成了。

  建立一种交叉编译工具链是一种相当复杂的过程，为了节省时间，能够直接从网上下载某些已编译好的交叉编译工具链，如arm-linux-toolchains.tgz。

Linux开发工具的使用

- 运营于Linux操作系统下的自由软件GNU gcc编译器，不但能够编译Linux操作系统下运营的应用程序，还能够编译Linux内核本身，甚至能够交叉编译运营于其他CPU上的程序。所以，在进行嵌入式系统应用程序开发时GNU gcc得到了广泛的应用。
- GNU开发工具的主要缺陷是采用命令行方式，顾客掌握和使用起来比较困难，不如基于Windows系统的开发工具易用，可一旦掌握了其使用方法就能够非常以便地进行项目开发。
- GNU的操作系统和开发工具都是免费的，只要遵照GPL协议，任何人都能够随意获取并使用。

Linux开发工具GNU的使用

- GNU提供的编译工具涉及汇编器as、C编译器gcc、C++编译器g++ 、链接器ld和二进制转换工具objcopy等。其中基于ARM平台的工具分别为：arm-linux-as 、arm-linux-gcc、 arm-linux-g++、 arm-linux-ld 和arm-linux-objcopy。
- Linux下的GNU调试工具主要有gdb、gdbserver和kgdb。其中gdb和gdbserver可完毕对目的板上Linux应用程序的远程调试。

GNU工具的开发流程

在Linux操作系统下使用GNU工具开发程序的流程主要有下列几种环节：

- 编写C、C++语言或汇编语言源程序；
- 用gcc或g++生成目的文件；
- 编写链接脚本文件；
- 用链接器生成最终目的文件（elf格式）；
- 用二进制转换工具生成可下载的二进制代码。

gcc编译器的使用

- 最初gcc的意思是GNU C Compiler，即只作为C程序语言的编译器；目前，经过数年的发展， gcc已经变成了GNU Compiler Collection，即GNU编译器家族的意思，除支持C语言以外，还支持Ada语言、C++语言、Java语言等多种语言。
- gcc是GNU组织提供的免费C编译器，大多数Linux版本都默认安装了这种编译器。

gcc编译器的使用方法

- Gcc的使用格式为： gcc [option] file…

  其中option是以“-”开始的多种选项，file是要编译的文件名。
- 在使用gcc进行编译时，必须要给出必要的选项和文件名。
- Gcc的整个编译过程分为预处理、编译、汇编和链接四个过程。

gcc编译器的使用方法

gcc编译器的常用选项有：

- [-o]：表达要求编译器生成指定文件名的可执行文件；
- [-c]：表达只要求编译器进行编译，而不要进行链接，生成以源文件的文件名命名但把其后缀由.c或.cc变成.o的目的文件。
- [-g]：要求编译器在编译的时候提供后来对程序进行调试的信息。
- [-E]：表达编译器对源文件只进行预处理就停止，而不做编译、汇编和链接。
- [-S]：表达编译器只进行编译，而不做汇编和链接。
- [-O]：是编译器对程序提供的编译优化选项，以提升执行效率。
- [-Wall]：指定产生全部的警告信息。

gcc编译器的使用方法

例：给出下列Hello源程序

```
void main( )
  {
    printf(“Hello the world\n”)
  }
```

要编译这个程序，只需输入下列命令即可完毕

```
$ gcc –o hello hello.c
```

其中gcc表达用gcc来编译源程序，-o选项表达要求编译器生成可执行文件的文件名， hello.c是要编译的源程序文件。

Makefile文件和Make命令

Linux内核的配置系统由下列三个部分构成：

- Makefile：用于定义Linux内核的编译规则；
- 配置文件（config.in）：给顾客提供配置选择的功能；
- 配置工具：涉及配置命令解释器和配置顾客界面等。

其中，Makefile文件描述了目的文件之间的依赖关系，以及指定编译过程中使用的工具，即根据配置的情况，构造出需要编译的源文件列表，然后分别编译，并把目的代码链接到一起，最终形成Linux内核的二进制文件。

- Makefile带来的好处就是自动化编译（批处理），在需要编译时只需要一种Make命令即可自动完毕，极大的提升了软件开发的效率。

Makefile文件和Make命令

在Linux内核中，与Makefile直接有关的文件有：

- Makefile：顶层Makefile，是整个内核配置、编译的总体控制文件。
- Config：内核配置文件，包括由顾客选择的配置选项，用来存储内核配置后的成果。
- Arch/*/Makefile：位于各CPU体系目录下的Makefile。
- 各子目录下的Makefile：负责所在子目录下源代码的管理。
- Rules.make：规则文件，被全部的Makefile使用。

顶层Makefile有两个主要任务：一是产生vmlinux文件；二是产生内核模块（module）。

Makefile文件和Make命令

- **Makefile中的变量**

  顶层Makefile定义并向环境中输出了诸多变量，为各个子目录下的Makefile传递某些信息。详细常用变量如下：

  - 版本信息：用来定义目前内核的版本；
  - CPU体系构造：用于定义目的CPU的体系构造；
  - 途径信息：用于定义内核等源代码所在的目录；
  - 内核构成信息：涉及头文件、内核文件等；
  - 编译信息：用于定义交叉编译的环境；
  - 配置变量config_*：用来阐明顾客配置的成果。

Makefile文件和Make命令

- **Rules.make变量**

  Rules.make定义了全部Makefile共用的编译规则。Linux把全部共用的编译规则统一放置到Rules.make中，并在各自的Makefile中经过语句“include Rules.make”包括Rules.make，这么可防止多种Makefile中反复这些规则。

- **子目录Makefile**

  用来控制本级目录下源代码的编译规则。

- **Make**

  Make是一种解释Makefile中指令的命令工具。Make命令执行时，需要一种Makefile文件，以告诉make命令怎么去编译和链接程序。

配置文件

- 除了Makefile的编写，另外一种主要的工作就是把新功能加入到Linux的配置选项中，并提供此项功能的阐明，让顾客有机会选择此项功能。这些工作需要在config.in文件中用配置语言编写配置脚原来实现。
- 在Linux内核中，配置命令有多种方式：
  - Make config(配置命令) scripts/configure(解释脚本)
  - Make oldconfig scripts/configure
  - Make menuconfig scripts/menuconfig
  - Make xconfig scripts/tkparse
- 以字符界面配置Make config为例，其配置过程为顶层Makefile调用scripts/configure按照arch/arm/config.in来进行配置，命令执行完后产生配置文件.config，其中保存着有关配置信息。

配置文件

- 配置语言
  - 顶层菜单
  - mainmenu_name /prompt/ /prompt/

  用于指定本CONFIG语言文件顶层的名字，其中/prompt/是一串提醒符。

  - 问询语句
  - Bool /prompt/ /symbol/
  - hex /prompt/ /symbol/ /word/
  - int /prompt/ /symbol/ /word/
  - string /prompt/ /symbol/ /word/
  - tristate /prompt/ /symbol/

  问询语句首先显示一串提醒符/prompt/,等待顾客输入，并把输入的成果赋给/symbol/ 所代表的配置变量。

配置文件

- 定义语句
  - define_bool /symbol/ /word/

  定义语句显式的给配置变量/symbol/赋值/word/ 。
- 依赖语句
  - dep_bool /prompt/ /symbol/ /dep/…
  - 依赖语句与问询语句一样，也是定义新的变量。不同的是，/symbol/的取值将依赖于配置变量列表/dep/…。
- 选择语句
  - choice /prompt/ /work/ /word/
  - 选择语句主要用于给出一串选择列表，以供顾客选择。
- If语句
- 菜单块
- Source语句

开启加载程序bootloader

- SRAM、SDRAM等存储设备属于易失性的存储器，掉电后来其中的内容会全部丢失，所以必须把操作系统的内核镜像存储在Flash等不易失性的存储介质上。而操作系统在运营时，需要动态的创建某些如数据段、堆栈、页表（针对使用虚拟地址的操作系统）等内容，所以需要在RAM中运营操作系统。
- 所以，这就需要一种引导程序把操作系统的内核镜像从Flash存储器拷贝到RAM中，然后再从RAM中执行操作系统的内核。
- Bootloader就是能够完毕这么一种功能的程序。

开启加载程序bootloader

- 从本质上来讲，bootloader不属于操作系统内核。它采用*汇编语言*编写，所以针对不同的CPU体系构造，这一部分代码不具有可移植性。
- 在移植操作系统时，这部分代码必须加以改写。详细来讲，bootloader在系统开启时主要完毕下列几项工作：
  - 将操作系统内核从Flash拷贝到SDRAM中，假如是压缩格式的内核，还要将之解压缩。
  - 改写系统的memory map，原先flash起始地址映射为0地址，这时需要将RAM的起始地址映射为0。
  - 设置堆栈指针并将*bss*段清零。将来执行C语言程序和调用子函数时要用到。
  - 变化*pc*值，使得CPU开始执行真正的操作系统内核。

运营操作系统内核

- **bootloader程序执行完上述的各项工作后，经过一条跳转指令，转而执行init目录下C语言源文件main.c中的函数start_kernel( )。因为在此之前bootloader已经创建好一种初始化环境，C函数能够开始执行了。**

Linux操作系统移植

- 在交叉编译环境和BootLoader建立后，下面的工作就是对操作系统的移植。
- 对于系统移植，Linux系统实际上是由两个比较独立的部分构成，即内核部分和系统部分，详细过程如下。
  - 系统开启时，加载程序（BootLoader）首先将Linux的部分内核调入内存，并将控制权交给内存中Linux内核的第一行代码，加载程序的工作就算完毕了，然后Linux内核再将剩余的部分全部加载到内存，初始化全部的设备，在内存中建立好所需的数据构造。这部分工作属于内核部分。
  - 内核加载设备并开启init守护进程， init守护进程再根据配置文件加载文件系统、配置网络、服务进程等。这部分工作属于系统部分。
  - 即内核部分的工作是初始化并控制大部分硬件设备，为内存管理、进程管理等工作作好准备；而系统部分的工作是加载必需的设备，配置多种环境以便顾客能够使用整个系统。