# 第14章 内核移植

研究内核的最终目的是为了应用内核，Linux内核支持多种处理器体系结构。内核的移植首先要获取内核源码，添加或修改平台（处理器）相关代码，主要是设备驱动程序，以及相应的配置、构建文件；然后，执行配置操作，选择需要编译、链接的源文件、函数等，执行构建操作，对内核源代码进行编译、链接，生成生成单一的内核目标文件（内核镜像）；最后，制作目标机根文件系统，将内核目标文件和根文件系统写入存储介质，修改目标机固件参数，指定加载内核目标文件的路径和挂载根文件系统的存储介质。系统上电启动后，固件从指定地点加载并运行内核，内核初始化完成后加载根文件系统，从根文件系统启动第一个用户进程，系统启动完成。

本章首先介绍内核移植需要搭建的主机环境，然后介绍内核配置、构建机制，以公版linux-4.2.4为例介绍配置、构建用于龙芯1B开发板内核镜像和模块的步骤及方法，最后介绍目标机根文件系统的制作和操作系统启动流程。

## 14.1 Deepin系统安装

主机是指用于浏览、修改内核源代码、配置、构建内核，生成内核目标文件所使用的机器，通常是台式机、笔记本电脑等。目标机是指运行内核镜像的机器，如各种开发板等。如果主机与目标机处理器体系结构是相同的，例如都是英特尔x86\_64处理器，则称主机对源代码的编译为本地编译，即本机编译生成的可执行目标文件可以直接在本机上运行。如果主机与目标机处理器体系结构不相同，例如主机是x86\_64处理器，目标机是MIPS处理器，则在主机对源程序的编译称为交叉编译，编译出的目标文件只能在目标机上运行，不能在主机上运行。

要编译、构建内核源代码首先要在主机上安装Linux发行版操作系统以及本地、交叉GCC编译器。GCC编译器是GNU组织开发的C语言编译器，是编译内核常用的编译器。

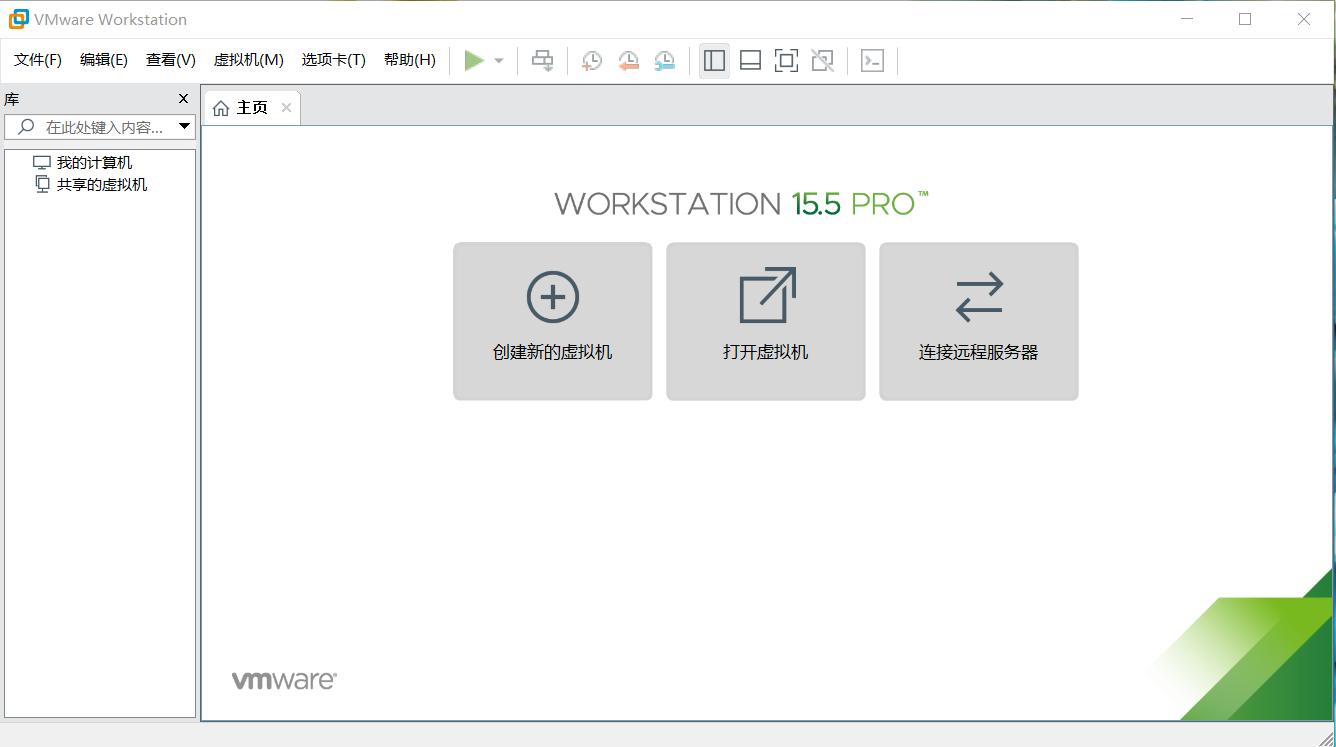
本节以国产Deepin操作系统（基于Linux的发行版Debian衍生的操作系统）为例介绍主机操作系统的安装、配置，下一节介绍本地、交叉GCC编译器的安装、配置。

Deepin（深度）操作系统是目前比较流行的国产Linux发行版操作系统，可从官网www.deepin.org下载安装镜像，以及启动U盘制作工具。启动U盘制作工具提供了一键操作功能，操作简单，可制作U盘安装盘，从U盘安装Deepin操作系统。这里假设读者已经下载好了Deepin安装镜像和启动U盘制作工具，下面分别介绍在虚拟机上安装Deepin操作系统，以及直接在硬盘上安装Deepin操作系统的方法。

### 14.1.1在虚拟机上安装Deepin

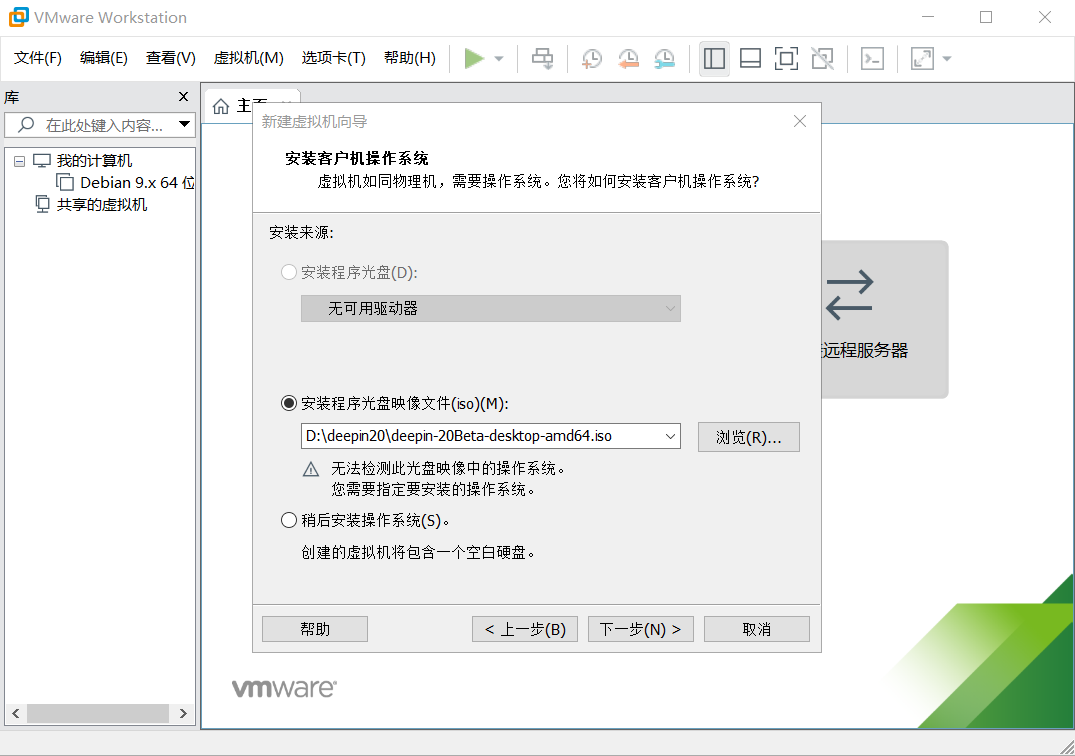
假设主机是Windows操作系统，可安装VMware虚拟机软件。VMware虚拟机软件的安装请读者自行完成，虚拟机软件可从官网（http://www.vmware.com/cn/）下载。下面介绍在VMware虚拟机软件上安装Deepin操作系统的方法。

**（1）**启动VMware虚拟机软件，点击“创建新的虚拟机”，运行新建虚拟机向导。

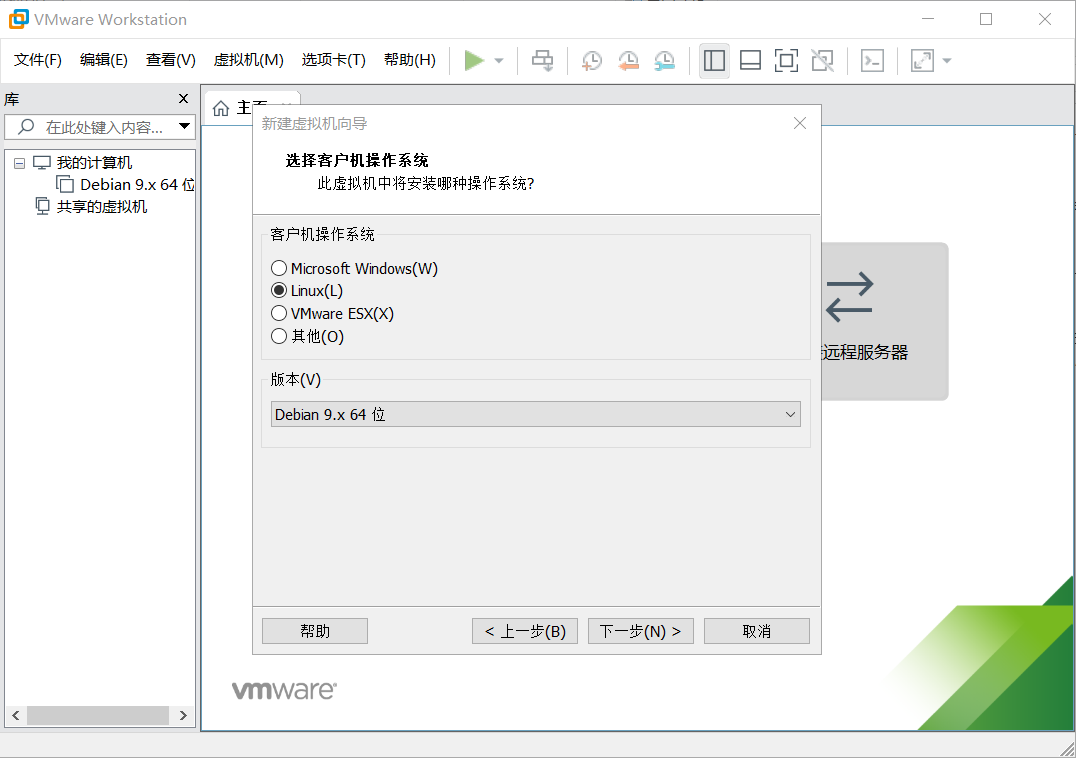


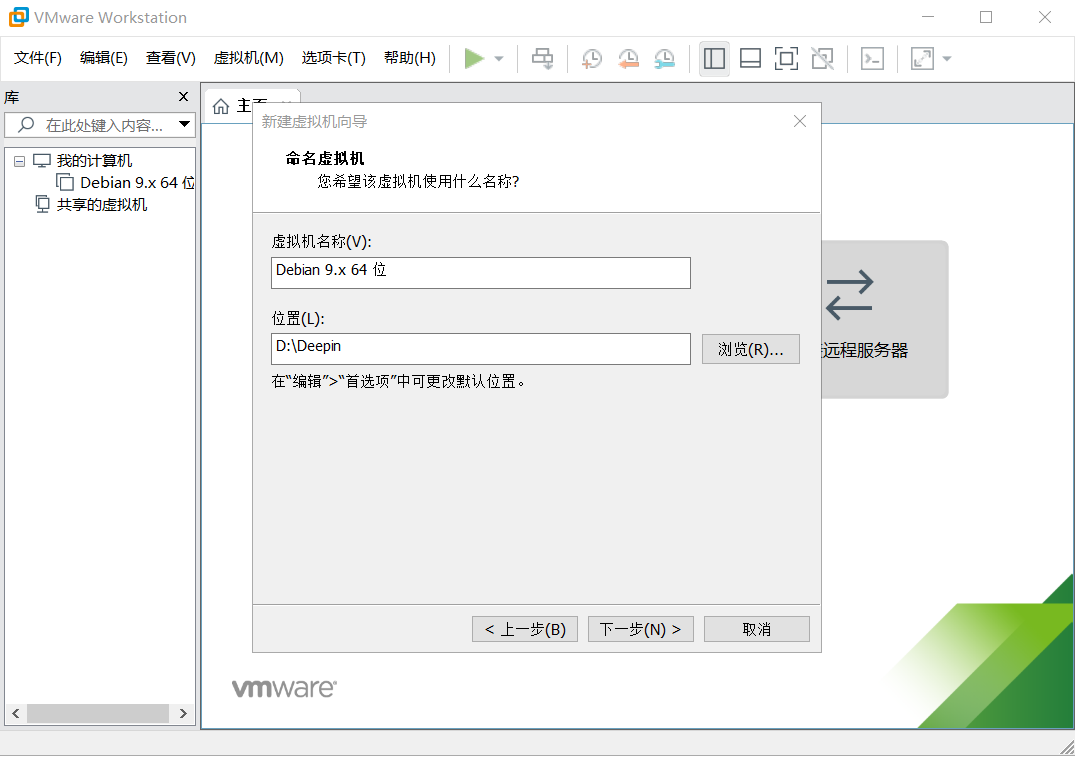


**（2）**选择安装程序光盘映像文件（下载的Deepin镜像文件）。

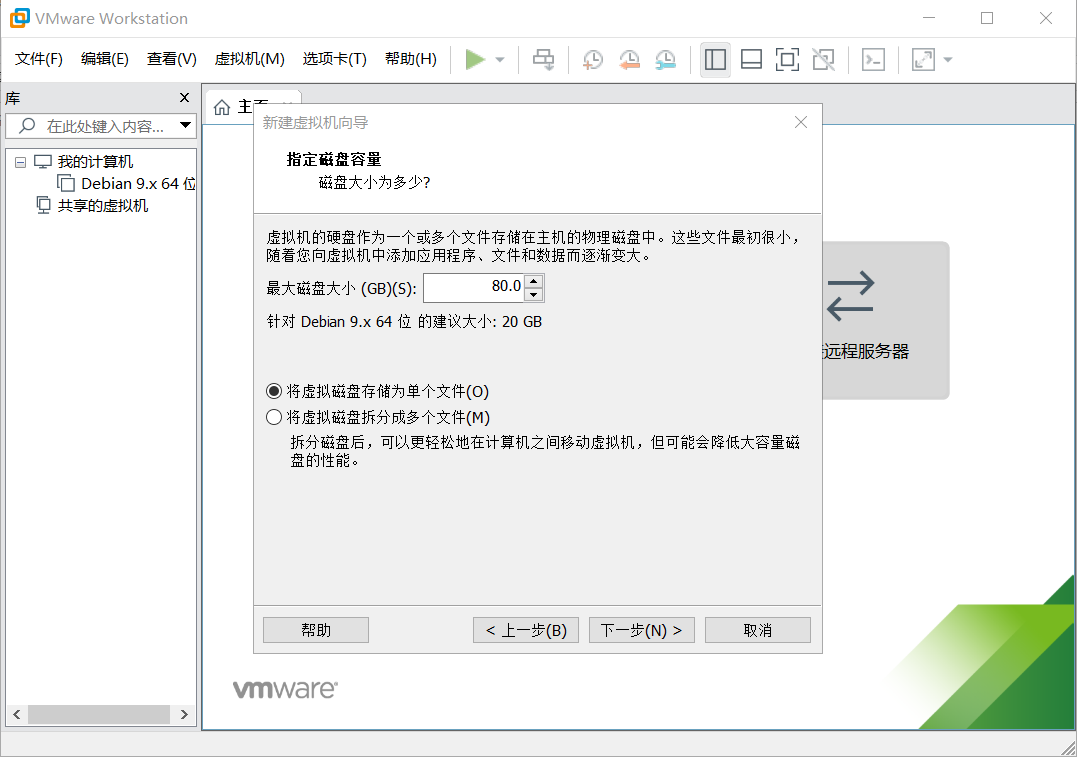


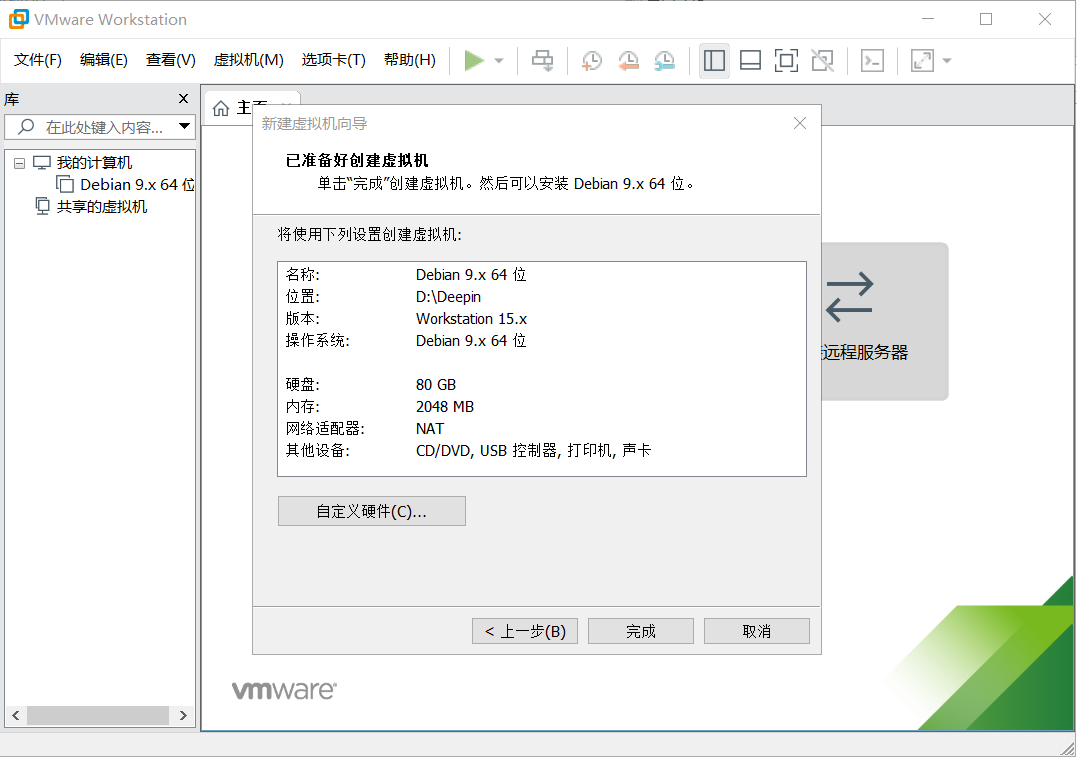
**（3）**选择客户机操作系统及安装位置（Debian版本号暂时选Debian 9.x 64位）。





**（4）**选择磁盘容量（64GB以上），点击“下一步”，完成虚拟机创建。

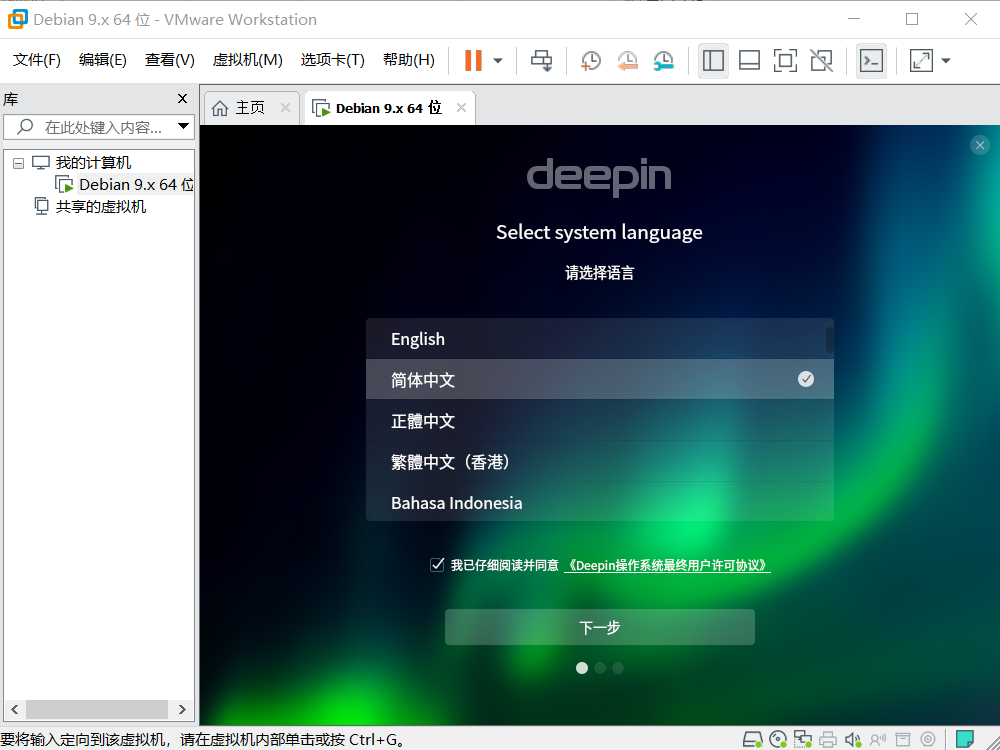




**（5）**点击“开启此虚拟机”，启动Deepin操作系统安装程序。



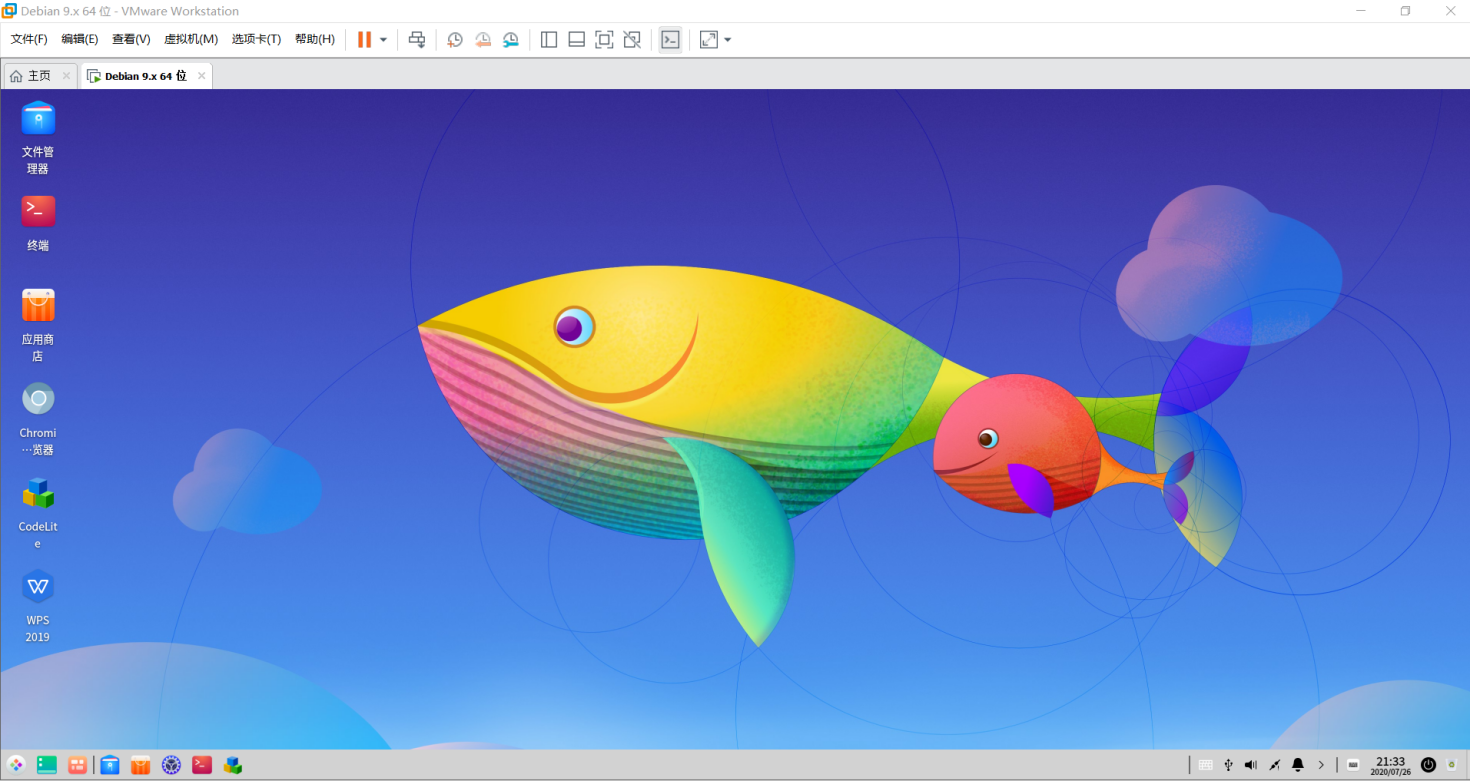
**（6）**选择“简体中文”，点击“下一步”。



**（7）**选择“全盘安装”，点击“开始安装”，稍等片刻即可开启Deepin之旅。

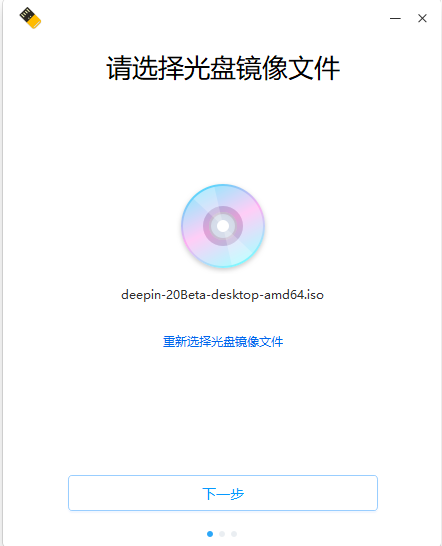


Deepin系统在第一次启动时需要设置密码，它将作为root用户的密码。开启虚拟机的步骤是：打开VMware虚拟机软件，在“我的计算机”里选择虚拟机，点击“开启此虚拟机”即可启动。在虚拟机中运动的Deepin操作系统界面如下：



### 14.1.2硬盘安装Deepin

在硬盘上安装Deepin操作系统需要制作U盘启动盘。启动盘制作方法如下：将U盘插入电脑，启动deepin-boot-maker工具，选择光盘镜像文件后（下载的Deepin镜像），点击“下一步”。



选择U盘，点击“开始制作”，稍等片刻，安装盘即可制作完成。



硬盘上最好准备两个分区用于安装Deepin操作系统，一个分区挂载到根目录“/”，一个分区挂载到家目录“/home”，因为用户数据都保存在/home目录下，如果重装系统可保留此目录下的用户数据。

从U盘安装操作系统方法如下：将U盘插入电脑，开机按F2键（通常是这样）进入BIOS，选择从U盘启动，即可运行Deepin操作系统安装程序。在安装程序中选择安装位置时选“手动安装”，分别将预留的两个分区挂载到根目录“/”和家目录“/home”，其它安装流程与在虚拟机中的安装步骤相同。

在虚拟机上运行的Deepin系统与在硬盘上安装的Deepin系统操作方法相同，但是安装在硬盘上的操作系统运行速度更快，数据也更安全，因此作者建议直接在硬盘上安装Deepin操作系统。如果主机原来装有Windows系统，在硬盘安装Deepin系统后，将会成为双系统，在启动时可选择引导哪个操作系统，默认为引导Deepin系统。

### 14.1.3系统命令

虽然Deepin系统提供了与Windows类似的图形操作界面，但是掌握常用的Linux系统命令也是非常重要的，如复制文件的cp、查看文件列表的ls、查看文件信息的file命令等，具体内容请读者参考Linux系统操作方面的书籍。

在Deepin操作系统中启动“终端”应用程序将开启一个虚拟终端，并运行一个shell程序（通常是bash），此shell程序将以此虚拟终端作为标准输入、标准输出和错误输出，在终端内可运行系统命令、脚本程序等。

Linux系统命令对应的可执行目标文件（程序二进制文件）保存在系统以下目录中：

●**/bin**：用于存放启动系统时用到的程序。

●**/usr/bin**：用于存放用户使用的标准程序。

●**/usr/local/bin**：用于存放软件安装程序。

系统管理员（root）还可执行存放系统管理程序目录下的程序，如/sbin和/usr/sbin目录。可选的操作系统组件或第三方应用程序可能被安装在/opt目录下。

shell程序中PATH环境变量保存了搜索命令（目标文件）的目录，目录间用“:”隔开，可用echo命令查询：



在终端执行命令时，shell将依次在PATH指定的目录中搜索同名的可执行目标文件，并创建子进程，加载目标文件运行，如果在以上路径中没有搜索到同名的可执行文件，命令将不能执行。

如果需要向PATH环境变量增加搜索目录，例如增加/home/lch/bin目录（先创建此目录），可在用户主目录（/home/lch）下的.bashrc文件（隐藏文件）最后增加以下两行并保存：

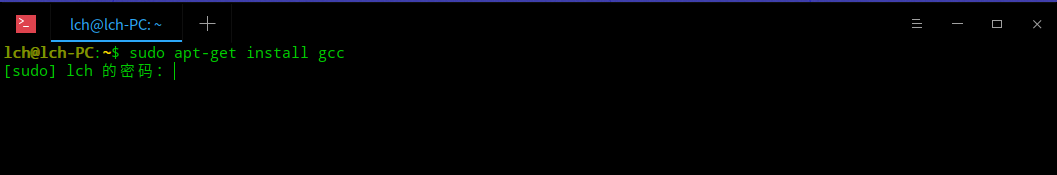
PATH=$PATH:/home/lch/bin

export PATH

export命令表示导出变量，PATH环境变量将会增加/home/lch/bin搜索目录，如下所示：



配置、构建内核所需的工具软件、库等可通过在终端运行apt-get命令下载安装（主机需联网），例如：安装主机GCC编译器的命令如下：



sudo表示以root权限运行apt-get命令，输入密码后（安装系统时设置的密码，不回显），将执行下载和安装操作。

下面是下载和安装库文件的命令：

sudo apt-get install *libxxx /\*libxxx*表示库名称*\*/*

## 14.2交叉编译工具

主机C编译器是用于将C源代码编译生成能在主机上运行的目标文件的编译器，交叉编译器是用于在主机上编译能在目标机（开发板）上运行的目标文件使用的编译器。通常编译内核和应用程式采用GNU GCC编译器。

主机在联网状态下可通过sudo apt-get install gcc命令为主机安装GCC编译器（操作系统通常已经安装），而交叉GCC编译工具需要用户安装。

### 14.2.1交叉编译工具的选择

通常我们用于开发的主机（电脑）都是x86-64架构的，而作者的目标机是MIPS32架构的，因此需要在主机上通过交叉编译工具,，将内核源代码（应用程序）编译生成能在MIPS32架构上运行的可执行目标文件，此目标文件不能在主机上运行。

交叉编译工具本身是一个在x86-64主机上运行的应用程序，需满足以下条件：

■**交叉编译工具能在x86\_64架构64位系统中运行**。由于Deepin系统是64位的，交叉编译工具作为一个应用程序需要能够在64位系统（Linux系统）上运行。龙芯1B开发板资料及龙芯开源社区提供的交叉编译工具（ls232）只能在32位系统上运行，不能直接在64位系统上运行。

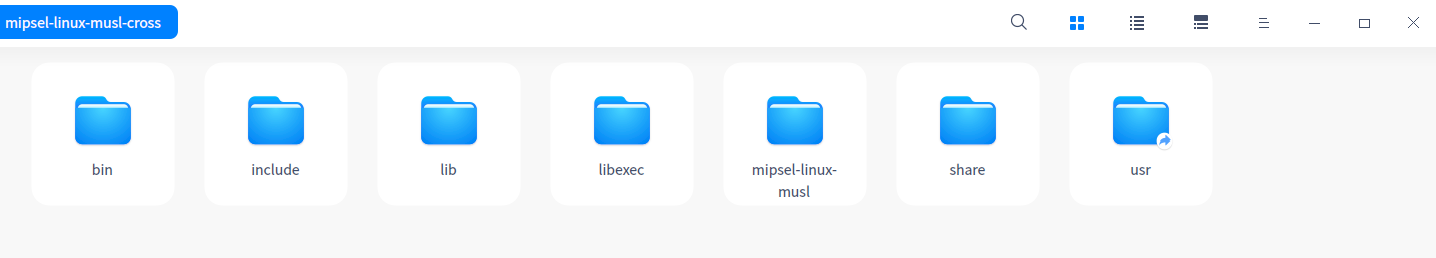
■**交叉编译工具应当能够编译出mips32架构、小端数据结构的目标文件**。MIPS架构有32位和64位之分，数据结构有小端、大端之分（整数字节在内存中保存的顺序，小端低字节保存在低地址，大端低字 节保存在高地址），龙芯1B兼容mips32r2、小端数据结构，编译出的目标文件必须是32位、小端的。

■**交叉编译工具提供的库文件必须是mips32架构、小端数据结构**。交叉编译工具自带有目标机使用的静态库和动态库，库文件必须也是32位小端的，有的交叉编译工具虽然可以编译出32位、小端结构目标文件，但是库文件却是64位的或大端数据结构的，因此也不能使用。

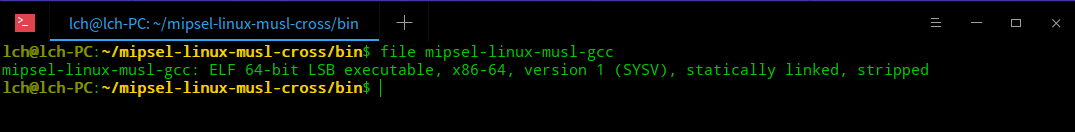
■**交叉编译工具版本不能太新，最好与内核版本处于同一个时期**。较老版本的内核可能不支持新版GCC编译器的某些特性，编译内核可能报错，不能通过。因此，最好选择与内核版本同期发布的GCC编译器版本。

经网友推荐，作者采用的是musl交叉编译工具，下载地址<http://musl.cc/>，作者下载的是5.5.0版本的适用于x86\_64架构的mipsel-linux-musl-cross交叉编译工具，前缀mipsel表示默认采用小端数据结构，还有一个版本前缀是mips，它也可以编译出小端结构目标文件，但其库文件是大端的，读者要特别注意。另外，还有适用于MIPS64架构的交叉编译工具，前缀标注了64标记，请读者下载时注意。

下载好交叉编译工具压缩包后，将其解压，作者主机解压的目录为：/home/lch/mipsel-linux-musl-cross，解压后子目录如下图所示：



下面通过file命令来检查下载的交叉编译工具是否符合要求。交叉编译工具命令位于/bin目录下，下面对GCC编译器进行检测：



由输出结果可看出GCC交叉编译器是支持x86-64架构、64位、小端数据结构的可执行程序，符合要求。

再检测一下/home/lch/mipsel-linux-musl-cross/mipsel-linux-musl/lib目录下库文件是否符合要求：



由上面可看出动态库libc.so是MIPS架构、32位、小端、位置无关可执行文件，其中LSB表示小端数据结构，显示MSB表示大端数据结构，请读者注意。

### 13.2.2配置交叉编译工具

交叉编译工具准备好后，需要将交叉编译工具所在目录和库文件所在目录通知主机操作系统，这是通过修改主目录下（/home/lch）的./bashrc文件来实现的。

交叉编译工具位于/home/lch/mipsel-linux-musl-cross/bin目录下，例如：mipsel-linux-musl-gcc为C编译器。注意mipsel-linux-musl-表示交叉编译工具的前缀，后面将会用到。目标机使用的静态库和动态库位于/home/lch/mipsel-linux-musl-cross/lib目录下。

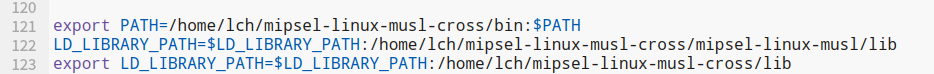
接下来的工作需要将交叉编译工具所在目录及库所在目录添加到环境变量中。打开主机用户家目录下的./bashrc文件（作者主机文件所在目录/home/lch，这是一个隐藏文件），在文件最后增加以下内容并保存：

export PATH=/home/lch/mipsel-linux-musl-cross/bin:$PATH

LD\_LIBRARY\_PATH=$LD\_LIBRARY\_PATH:/home/lch/mipsel-linux-musl-cross/mipsel-linux-musl/lib

export LD\_LIBRARY\_PATH=$LD\_LIBRARY\_PATH:/home/lch/mipsel-linux-musl-cross/lib

PATH表示命令搜索路径，用于查找交叉编译工具，LD\_LIBRARY\_PATH表示搜索库路径。



另外，在主机运行程序时经常会提示缺少某个库，前面介绍了可用sudo apt-get install *libname*命令下载安装需要的库，以下命令用于更新系统中的库文件：

sudo apt-get update

动态库具有不同的版本，有的应用程序指定需要某个版本的动态库，在该版本不存在的情况下，可以通过建立其到另一版本库文件的符号链接来解决。

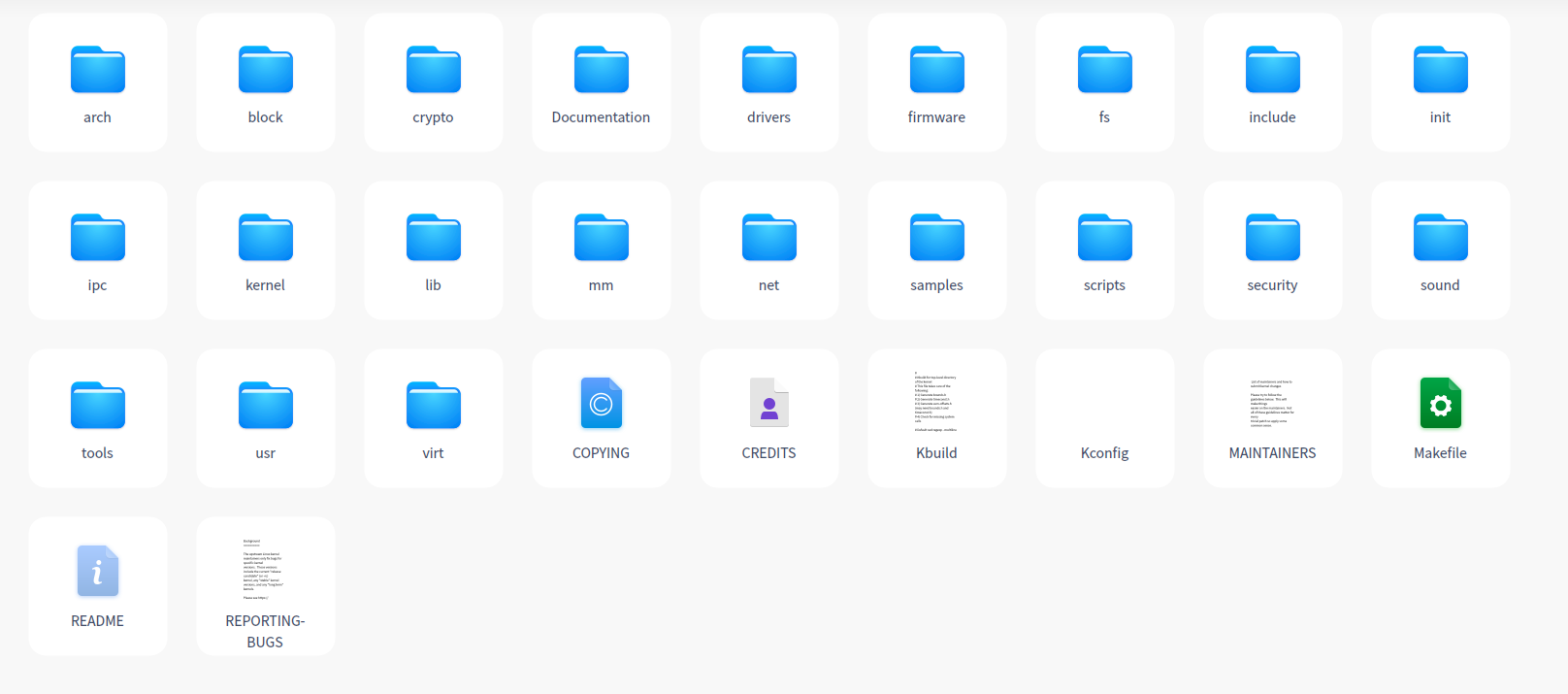
例如，在内核构建过程中会提示找不到libreadline.so.6库，但是系统内不存在此库，而libreadline.so.5.2库却存在，库文件位于/lib/x86\_64-linux-gnu目录下，可在此目录下运行终端，执行以下命令可创建名称为libreadline.so.6到libreadline.so.5.2库文件的符号链接：

**sudo cp -s libreadline.so.5.2 libreadline.so.6**  /\*-s表示符号链接\*/

总之，在配置、构建过程中若提示找不到某个库可用apt-get install命令下载，若下载不成功，可在系统中查找同名称但不同版本的库，建立符号链接（系统中库文件通常位于/lib/目录下）。

## 14.3配置与构建

用户可从Linux内核官网（www.kernel.org）下载内核源码压缩包，经解压后获得内核源码树，如下图所示。



在添加/修改好平台及驱动代码（含配置文件）后，需要对内核进行配置，然后执行构建（编译链接）生成内核目标文件。本节简要介绍内核配置与构建的机制和方法，下一节将以龙芯1B平台为例介绍内核配置与构建的操作步骤。

配置就是内核提供一系列选项，用户可以通过设置选项的值，决定将哪些函数（源文件）包含在内核中、哪些函数（源文件）编译为模块、哪些函数（源文件）不进行编译，也可以对内核某项功能的实现方式进行选择等。内核采用了一种称之为Kconfig的配置语言，执行配置工作。配置文件名称一般为Kconfig，几乎每个源码目录下都有一个Kconfig文件。用户通过make menuconfig等命令执行配置工作，此命令将会读取各配置文件，向用户提供配置界面，由用户做出选择，最后生成配置结果保存在.config文件内。

配置完成后即可进行内核构建，内核提供了Kbuild机制用于执行构建操作。Kbuild将根据配置结果文件控制（选择）内核源文件的编译、链接。Kbuild还将根据配置结果文件生成配置头文件，用于内核源代码。

内核配置与构建机制可参考/Documentation/kbuild/目录下的说明文档，下面将简要介绍内核配置与构建机制。

### 14.2.1 Kconfig语言

Kconfig配置语言用于编写配置文件，内核配置脚本将读取配置文件，向用户提供配置界面，用户通过此界面可以设置配置选项的值，以控制内核的构建。

配置语言主要是定义配置选项，设置配置菜单，管理各配置选项之间的依赖关系等，配置语言必须解决下列问题：

●各组件可以持久编译到内核中、可以编译为模块或直接忽然掉。

●在配置选项之间，可能存在相互依赖关系。换言之，某些选项只能与一个或多个其他选项联合使用。

●必须能够给出一个可用选项列表，供用户从中选择。有些情形需要用户输入编号（或类似的值）。

●必须能够层次化地编排各种配置选项。

●配置选项可能依体系结构而不同。

●配置语言不应该过于复杂。

#### 1语言要素

Kconfig配置语言语法并不复杂，语言要素包括菜单、配置选项、属性、依赖关系等。

##### ■菜单

在配置文件中，菜单由以下命令表示：

**menu** “string” /\*配置界面上显示的菜单名称\*/

<attributes> /\*属性，见下文\*/

<configuration options> /\*配置选项\*/

**endmenu** /\*菜单结束\*/

menu表示菜单开始，endmenu表示菜单结束，string是菜单名称，在用户配置界面上显示，菜单内部是配置选项。例如：

**menu** "Machine selection" /\*平台选择菜单，/arch/mips/Kconfig\*/

**choice**  /\*单选菜单\*/

prompt "System type" /\*单选菜单名称\*/

default SGI\_IP22 /\*默认值\*/

...

**config** MACH\_LOONGSON32 /\*配置选项，平台类型\*/

bool "Loongson-1 family of machines"

select SYS\_SUPPORTS\_ZBOOT

help /\*可在配置界面中显示的帮助信息\*/

This enables support for the Loongson-1 family of machines.

Loongson-1 is a family of 32-bit MIPS-compatible SoCs developed by

the Institute of Computing Technology (ICT), Chinese Academy of

Sciences (CAS).

**config**  MACH\_LOONGSON64

bool "Loongson-2/3 family of machines"

select SYS\_SUPPORTS\_ZBOOT

help

This enables the support of Loongson-2/3 family of machines.

Loongson-2 is a family of single-core CPUs and Loongson-3 is a

family of multi-core CPUs. They are both 64-bit general-purpose

MIPS-compatible CPUs. Loongson-2/3 are developed by the Institute

of Computing Technology (ICT), Chinese Academy of Sciences (CAS)

in the People's Republic of China. The chief architect is Professor

Weiwu Hu.

...

**endchoice**  /\*单选菜单结束\*/

...

**endmenu** /\*菜单结束\*/

menu后面的字符串为菜单名称，会显示在用户配置界面上。menu与endmenu之间的配置选项为多选菜单项，可同时选择里面的多个选项。配置选项由config表示，见下文。

choice与endchoice之间是单选菜单，单选菜单里的配置选项只能选择其中的一项，上例中menu与endmenu表示的菜单中只包含一个单选菜单，单选菜单用于选择平台类型。

关键字menuconfig用于定义一个配置选项和一个菜单。例如：下面的写法可以进行调整：

menu “Bit bucket compression support”

config BIT\_BUCKET\_ZLIB

tristate “Bit bucket compression support”

采用menuconfig关键字调整后的格式：

menuconfig BIT\_BUCKET\_ZLIB

tristate “Bit bucket compression support”

另一个关键字mainmenu只能出现在配置层次结构的顶部（且只能出现一次），用于为整个层次结构指定一个标题，所有体系结构配置层次结构的起始点都在/Kconfig文件内：

**mainmenu** "Linux/$ARCH $KERNELVERSION Kernel Configuration"

config SRCARCH

string

option env="SRCARCH"

source "**arch/$SRCARCH/Kconfig**" /\*导入体系结构的配置文件\*/

##### ■配置选项

配置选项由关键字config开头，必须后接一个配置符号，如下所示：

config <**symbol**> /\*配置选项名称，配置符号\*/

<type-name> “Description” /\*类型，描述\*/

<attributes> /\*属性\*/

●**symbol：**配置选项名称，配置操作中若选中此选项，此配置选项会生成宏定义保存在配置结果文件中，宏名称为在配置选项名称前加上“CONFIG\_”。

例如：若选择了MACH\_LOONGSON32配置选项，生成的宏定义为：CONFIG\_MACH\_LOONGSON32，定义的宏可用于控制内核构建操作及内核源代码。

●**type-name**：配置选项类型，类似于C语言中的变量类型，类型值如下：

1. tristate：取值可为y、n或m，用于内核构建，分别表示将源文件持久编译入内核、将源文件排除在外（不编译），以及将源文件编译成模块。
2. bool：取值为y或n，表示是否选中该选项，y表示将某源文件、函数、变量等持久编译入内核，n表示排除在外（不编译）。
3. string：字符串。
4. hex、interger：表示十进制、十六进制数，类型后面的描述字符串表示其值。

在配置操作中对配置选项的操作就是对配置选项类型的赋值，在生成配置结果文件时，会将选项类型值赋予生成的宏。例如：若选择了CONFIG\_MACH\_LOONGSON32选项，则在配置结果文件中将会生成以下条目：

CONFIG\_MACH\_LOONGSON32=y

上面的例子还可用下面的语法表示：

config <**symbol**> /\*配置选项名称\*/

<type-name> /\*类型\*/

prompt “Description” /\*描述\*/

注意，没有指定描述字符串的配置选项将不会显示在用户配置界面上，只能在配置文件中修改选项类型值。

##### ■属性

属性用于更准确地指定配置选项的效果。下面的配置选项使用了属性：

config CPU\_LOONGSON1B /\*配置选项名称\*/

bool "Loongson 1B" /\*选项类型及描述\*/

**depends on** SYS\_HAS\_CPU\_LOONGSON1B /\*属性\*/

**select** CPU\_LOONGSON1

help

The Loongson 1B is a 32-bit SoC, which implements the MIPS32

release 2 instruction set.

●**depends on：**表示CPU\_LOONGSON1B选项依赖于SYS\_HAS\_CPU\_LOONGSON1B选项，只有后者被选中了（y），前者才有效，可供选则，否则前者无效。

●**select**：表示如果选择了CPU\_LOONGSON1B选项，将自动选择select之后的CPU\_LOONGSON1选项。

其它属性还有：

●**default**：配置选项类型的默认值。string、hex和interger类型配置选项必须指定默认值。

●**range**：限制配置选项类型值可能的范围。

●**help、--help--**：帮助文本，是对配置选项的说明，可在用户配置界面显示。

另外，配置语言中还有其它关键字，如用于逻辑控制的if、endif，导入其它配置文件的source等。source关键字相当于C语言中的include，如下所示：

source "drivers/pci/Kconfig" /\*导入配置文件\*/

Kconfig配置语言其它内容请读者参考内核说明文档。

#### 2配置文件结构

内核所有配置文件组成单一的树状层次结构，源代码根目录下的Kconfig文件是最顶层的配置文件。高层配置文件中通过source关键字导入下一层的配置文件，例如：Kconfig文件内容如下：

mainmenu "Linux/$ARCH $KERNELVERSION Kernel Configuration"

config SRCARCH

string

option env="SRCARCH"

source "arch/$SRCARCH/Kconfig" /\*导入体系结构相关的配置文件\*/

MIPS体系结构相关的配置文件为/arch/mips/Kconfig，内容简列如下：

config MIPS

bool

default y

select ARCH\_MIGHT\_HAVE\_PC\_PARPORT

...

menu "Machine selection" /\*第一个菜单\*/

choice

prompt "System type"

default SGI\_IP22

config MIPS\_ALCHEMY

bool "Alchemy processor based machines"

...

config AR7

bool "Texas Instruments AR7"

...

...

endmenu

source "net/Kconfig" /\*导入其它配置文件\*/

source "drivers/Kconfig"

source "drivers/firmware/Kconfig"

source "fs/Kconfig"

source "arch/mips/Kconfig.debug"

source "security/Kconfig"

source "crypto/Kconfig"

source "lib/Kconfig"

source "arch/mips/kvm/Kconfig"

内核配置文件树状层次结构简列如下图所示：



在执行配置操作时，配置脚本将导入所有的配置文件，然后向用户提供配置操作界面。如果其中任一配置文件不存在，或存在语法问题，将导致文件导入不成功，不能显示配置界面。

### 14.2.2配置操作

由配置语言编写的配置文件相当于C语言的源文件，要向用户提供配置界面还需要GNU Make工具和内核脚本的支持（/scripts）。GNU Make是一个主要用于构建目标文件的工具，它也有自己的语法，可将其视为一个命令解释器，类似于shell。调用GNU Make工具的命令为make，默认情况下make会查找并读取当前目录下的make文件（makefile），查找的顺序为GNUmakefile、makefile、Makefile（文件名称）。内核源码树中基本每个目录下都有一个名称为Makefile的文件，包含构建的规则。下一小节介绍内核构建时将会再次介绍make工具。

#### 1配置命令

在执行配置命令前，先要修改源代码树根目录下的Makefile文件中的部分内容，用于设置目标平台体系结构和交叉编译工具前缀（也可以不修改，在命令行参数中传递变量值）。

将Makefile文件内的ARCH、CROSS\_COMPILE变量值修改如下，并保存：

linux_makefile

其中ARCH表示体系结构名称，这里设为mips，CROSS\_COMPILE表示交叉编译工具前缀。

内核配置命令主要有（在源代码根目录下运行）：

**make menuconfig**

make oldconfig

make *xxx\_defconfig* /\*平台默认的配置结果文件写入.config\*/

make allyesconfig

make allnoconfig

●menuconfig：提供基于控制台的图形界面的配置前端，需要主机系统图形界面的支持。

●oldconfig：以/.config中配置选项为基础进行配置。

● *xxx\_defconfig*：将内核中平台定义的默认配置结果写入./config文件，默认配置结果文件保存在体系结构相关的目录下，例如：MIPS体系结构保存在/arch/mips/configs/目录下。

●allyesconfig：创建一个新配置，所有选择都设置成y。

●allnoconfig：创建一个新配置，所有选择都设置成n。

常用的配置命令为menuconfig，它需要主机ncurses库的支持，如果操作系统暂无此库需要执行以下命令安装：

**sudo apt-get install libncurses5-dev**

**sudo apt-get install flex**

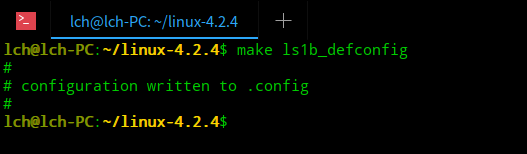
**sudo apt-get install bison**

执行完配置操作后，配置结果将保存在.config文件内，若此文件已经存在，则配置操作将会读取此文件中的配置结果，生成配置界面。也就是说，每次配置操作时将会导入上一次的配置结果。

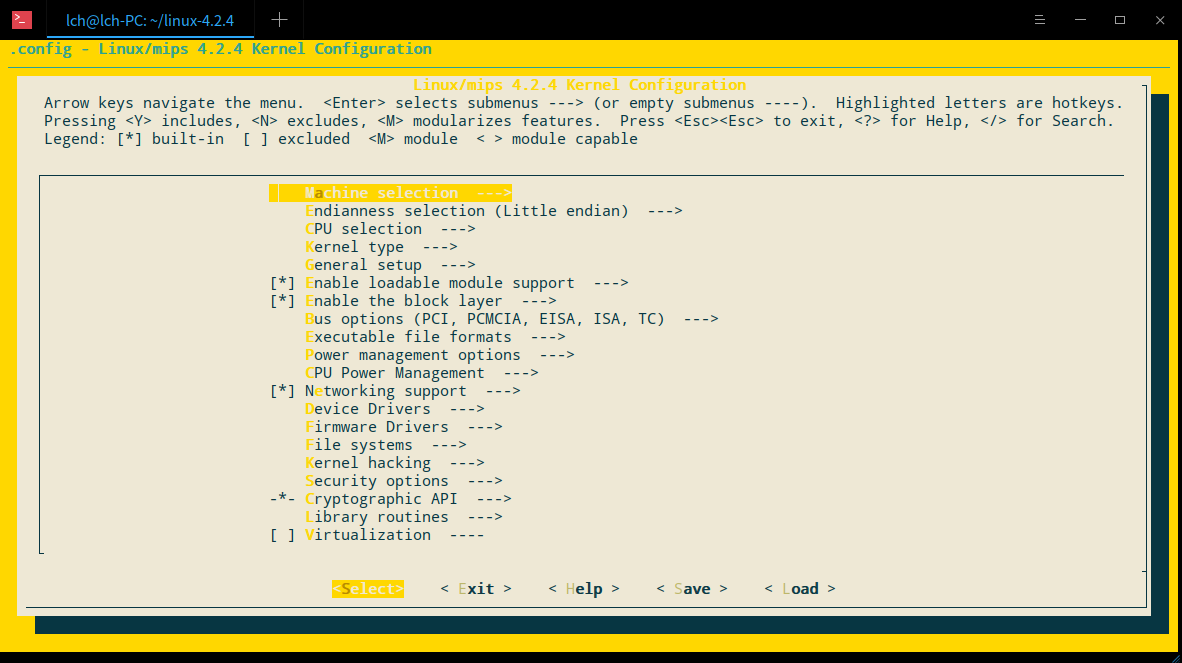
通常内核支持的平台（开发板）在内核中都已经定义了默认的配置结果文件，例如：龙芯1B平台默认配置结果文件为：/arch/mips/configs/ls1b\_defconfig。

用户可以先执行make *xxx\_defconfig*命令，将默认的配置文件写入.config文件，再执行make menuconfig命令，以排除一些不相关的配置选项，简化配置操作。

例如：执行make ls1b\_defconfig命令的结果如下图所示：



图中显示已将配置结果写入.config文件。接下来执行make menuconfig命令，显示的配置界面如下图所示：



make menuconfig命令将沿着配置文件树状层次结构导入所有的配置文件，在配置界面上显示配置菜单，配置界面中有二十一个一级子菜单。在配置介面中用上下键选择菜单项，也可以直接按每个菜单项黄色标识的字符键直接选中相应的菜单项，按回车键进入其子菜单项，Esc键（按两次）返回上一级菜单。

左右键选择最下面的Select、Exit、Help、Save、Load选项，各选项作用如下：

●Select：表示当前正在选择菜单项。

●Exit：选择此选项后按回车键返回上一级菜单，如果是顶层目录，则退出配置操作。

●Help：选择此选项后按回车键将显示光标当前选中菜单项的帮助文本。

●Save：选择此选项后按回车键将保存配置结果。

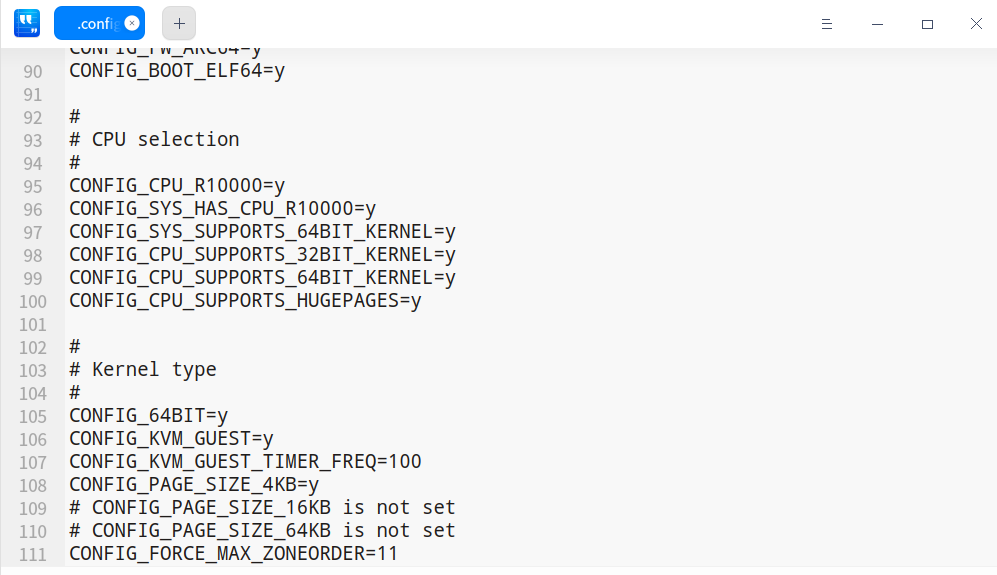
●Load：选择此选项后按回车键可从配置结果文件中导入配置结果。

在菜单项前面的< >标记表示配置选项类型为tristate，< >中可输入的字符为y、n、m，在< >中将显示的字符分别为\*、空格、m，分别表示持久编译入内核、排除在外、编译成模块。

前面是[ ]标记的配置选项类型为bool，[ ]中可输入的字符为y、n，在[ ]中将显示的字符分别为\*、空格，分别表示持久编译入内核、排除在外。

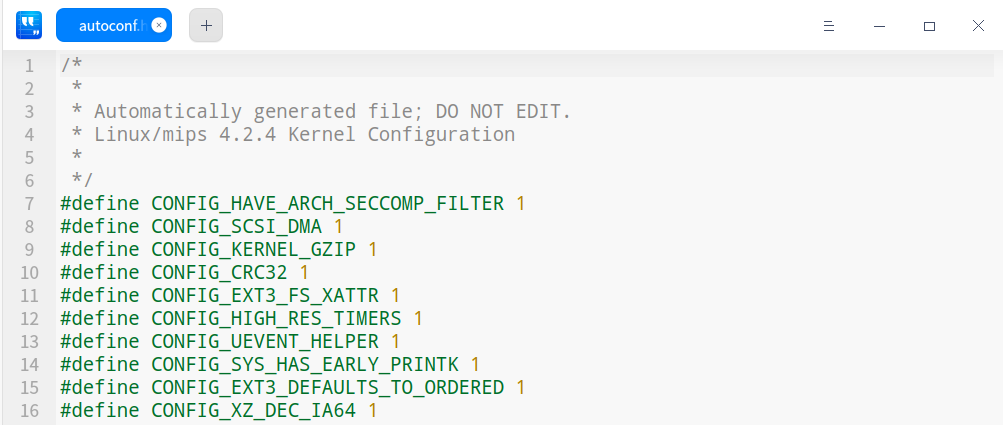
前面是( )标记的配置选项为单选菜单项，要选择哪一项则将光标移到此选项按回车键即可。

此处的y、n、m字符将赋予配置结果文件中的CONFIG\_XXX宏。配置选项选择完毕后，选择Save选项，保存配置结果，退出配置操作。配置结果默认保存在/.config文件内，这是一个隐藏文件，需要在操作系统文件管理器中设置显示隐藏文件才能显示此文件，文件内容如下。



文件内将对每个配置选项名称前加上“CONFIG\_”字符串，前面是“#”字符的选项表示没有设置。在下次执行配置操作时，将根据.config配置结果文件自动设置配置介面中配置选项的状态，也就是说可以保留上次配置结果。

在后面的构建操作中将根据配置结果文件（.config）自动创建**/include/generated/autoconf.h**头文件并被包含进内核源代码文件，使配置选项对内核源代码可见，文件内容如下：



#### 2配置选项简介

在make menuconfig命令启动的配置界面中一级子菜单共有二十一项，下面分别对主要的一级菜单项做简要介绍。

■Machine selection：选择平台。

■Endianness selection (Little endian) ：处理器大端、小端结构选择，选择平台后通常默认已设置。

■CPU selection：选择处理器。

■Kernel type：内核类型选项，主要是物理内存管理、内核抢占选项等。

■General setup：通用配置选项，此菜单下的选项比较多。如：进程间通信配置选项、审计子系统选项、控制组选项、设置初始化RAM文件系统源文件、slab分配器选项等。

■Enable loadable module support：内核模块加载方式支持选项。

■Enable the block layer：通用块设备层支持选项，分区类型选项等。

■Bus options (PCI, PCMCIA, EISA, ISA, TC)：支持的总线类型选项。

■Executable file formats：可执行目标文件（内核目标文件）格式。

■Power management options：功耗管理选项。

■CPU Power Management：CPU功耗管理选项，CPU Idle配置项等。

■ Networking support：网络层协议配置选项。

■Device Drivers：设备驱动配置选项，在内核移植过程中主要是对此选项下的子菜单项进行配置，包括通用驱动模型配置选项、字符设备驱动程序配置选项、块设备驱动程序配置选项以及网络设备驱动程序配置选项等。

■File systems：文件系统配置选项，内核支持几十种文件系统类型，可配置选择支持的文件系统类型。

■Kernel hacking：内核调试、跟踪、输出信息等配置选项。

■Security options：内核安全子系统配置选项。

■Library routines：内核代码使用的通用库配置选项。

■Virtualization：内核虚拟化配置选项。

### 14.3.3 Kbuild

内核构建就是根据配置选项对内核源文件进行选择性编译，对各源文件编译生成的目标文件进行链接，最后生成单一的可执行目标文件和内核模块。

内核构建需要主机GCC编译器和交叉GCC编译器的支持，通过执行make命令启动内核的Kbuild机制完成内核构建。Kbuild机制将根据源码各目录下的Makefile文件执行构建操作，Makefile文件将根据配置结果文件.config选择编译的源文件，生成的/include/generated/autoconf.h头文件将对内核源代码可见。

构建操作中将要使用到GNU Make工具，本节先简要GNU Make工具，然后介绍内核Kbuild机制和构建操作。

#### 1 GNU Make

Linux操作系统Make工具用于按照给定的规则构建目标，在此处用于控制内核源代码的编译、链接，在Linux系统中使用的Make工具通常是GNU Make，这是GNU组织开发的Make工具。Make类似于shell，可视为命令解释器，它有自己的语法、变量、规则等。Kbuild机制按Make工具规则在源代码每个目录下（基本每个目录下）编写了Makefile文件（相当于脚本文件），以控制内核的构建。

GNU Make默认情况下将在当前目录下依次搜索GNUmakefile、makefile、Makefile文件，如果有一个文件存在将不再搜索后面的文件，读取第一个搜索到的文件的内容并执行。

基本的makefile文件由规则组成，规则结构如下：

**target:prerequisites**

**recipe**

target表示构建目标的名称，prerequisites表示构建目标的依赖关系，即构建目标文件需要的源文件，recipe表示构建目标的命令，如gcc命令等。注意，默认情况下recipe前是tab字符，而不是空格。

例如：下面是一个简单的makefile文件，命名为makefile1：

myapp:main.o 2.o 3.o

gcc -o myapp main.o 2.o 3.o

main.o main.c a.h

gcc -c main.c

2.o:2.c a.h b.h

gcc -c 2.c

3.o:3.c b.h c.h

gcc -c 3.c

以上文件中包含4条规则，例如：第一条规则生成的目标名称为myapp，它依赖于目标文件main.o、2.o和3.o，构建myapp目标的命令是gcc。后面三条规则类似，分别用于生成main.o、2.o和3.o目标。

以下命令用于构建myapp目标：

make -f makefile1

-f命令行参数用于指示读取的makefile文件名称，如果没有指定-f参数，则默认依次搜索GNUmakefile、makefile、Makefile文件，命令行中如果没有指定生成目标的名称，则默认构建makefile文件中第一个目标，执行此命令将生成myapp目标。

除此之外，make与shell一样也有自己的语法，宏定义等，详细内容请参考GNU Make手册。

#### 2 Makefile

内核Kbuild机制使用的Makefile文件包含以下五个组件：

●**/.config**：配置文件。

●**/Makefile**：主（顶层）Makefile，它将读取配置文件.config，导向各层子目录下Makefile文件，主要负责生成内核镜像和模块。

●**/arch/$(ARCH)/Makefile**：体系结构相关的Makefile，负责在编译期间必须遵守的与处理器相关的微妙之处，如特别的编译优化选项等。

●/**scripts/Makefile.\***：包含了与一般编译、模块生成、各种实用程序的编译、从内核树删除目标文件和临时文件等任务相关的通用make规则。

●**各源代码目录下的Makefile文件**：负责根据配置选项选择编译目录下的源文件等。

内核源码树中的Makefile文件与配置文件Kconfig类似，也组成树状的层次结构。各源代码目录下的Makefile文件根据.config文件中的配置选项值来选择编译目录下的源文件，并将编译的流程导向到其下的子目录Makefile中。例如，/fs/Makefile文件内容如下图所示，用于构建内核支持的文件系统类型：



obj-y包含的目标文件将持久编译入内核，obj-m包含的目标文件将编译成模块，obj-n包含的目标文件将不编译，lib-y包含的目标文件将编译成库文件（lib.a）。

Kbuild机制会自动编译同名的源文件xxx.**c**或xxx.**S**生成xxx.o目标文件，例如，open.o目标文件由open.c源文件编译生成。

由前面介绍的.config文件内容可知，配置选项（如CONFIG\_BLOCK等），将赋值为y、m、n等，据此相关的目标文件将添加到obj-y、obj-m或obj-n列表，表示目标文件是持久编译入内核、编译成模块，还是不编译。

以上文件中ifeq...else...endif语句表示如果选择了BLOCK配置选项（选项值为y），则编译buffer.o、block\_dev.o等目标文件，否则编译no-block.o目标文件。

obj-y+=notify/表示将编译流程导向/fs/notify/子目录，调用执行其下的Makefile文件。

Kbuild机制会将同一目录下包含在obj-y列表下的所有目标文件都链接成一个目标文件built-in.o，随后将该目标文件再链接到整个内核中。

注意：obj-y目标依赖的目标文件排序是非常重要的，在链接时是按顺序链接目标文件，如果有同名的目标文件，将只链接第一个，后面的将忽略。

#### 3编译选项

在Makefile文件中以下宏分别表示编译选项、汇编选项和链接选项（交叉编译工具），且是全局的：

ccflags-y、asflags-y、ldflags-y

以前的表示方式为EXTRA\_CFLAGS,、EXTRA\_AFLAGS、 EXTRA\_LDFLAGS，虽然依然可用，但已不推荐使用。

subdir-ccflags-y、subdir-asflags-y表示只适用于当前目录及其子目录下Makefile文件的编译、汇编选项。

CFLAGS\_$@、AFLAGS\_$@表示只适用于特定目标文件的选项，例如：

CFLAGS\_aha152x.o = -DAHA152X\_STAT -DAUTOCONF

CFLAGS\_gdth.o = # -DDEBUG\_GDTH=2 -D\_\_SERIAL\_\_ -D\_\_COM2\_\_ \

-DGDTH\_STATISTICS

以上两者分别只适用于编译生成aha152x.o和gdth.o目标文件，AFLAGS\_$@选项与此类似，表示汇编选项。

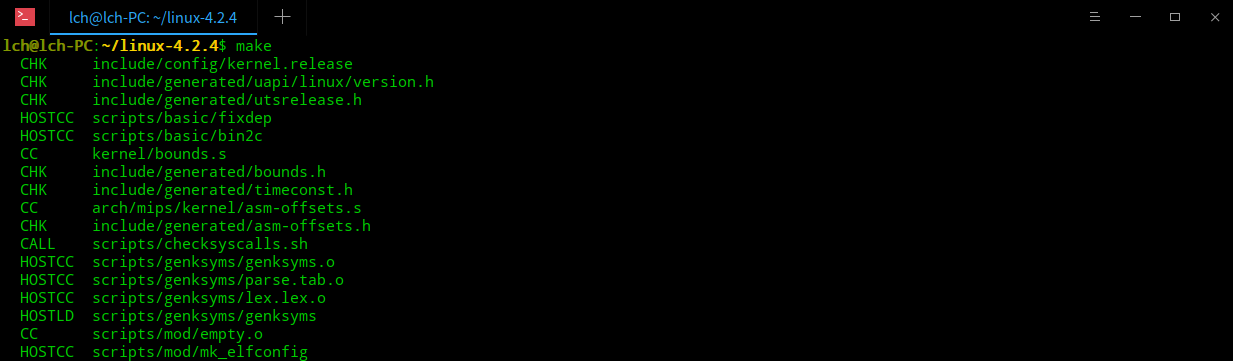
Makefile更详细的信息请参考内核说明文档/Documentation/kbuild/makefiles.txt。

### 14.3.4构建操作

本小节介绍构建内核和模块的方法，详情可参考/Documentation/kbuild/makefiles.txt说明文档，模块的编译可参考modules.txt文档。

#### 1构建内核

假设用户已按前面介绍的要求修改了主Makefile中的ARCH和CROSS\_COMPILE宏，且已完成内核的配置，则只需要在源代码根目录下运行终端执行make命令即可启动内核构建操作，如下图所示：



编译完成后（需要一段时间），在源代码根目录下将生成内核镜像文件vmlinux，以及经压缩过的vmlinuz文件和内核模块。构建过程中链接目标文件的链接文件为/arch/mips/kernel/vmlinux.lds。

make clean命令用于删除所有构建过程中产生在文件，恢复原有的源代码。

#### 2编译模块

前面介绍过，在Makefile文件中，obj-m列表下的目标将编译成模块，模块目标文件后缀为.ko。如果模块由单个文件组成，则Makefile文件内容如下（drivers/isdn/i4l/Makefile）：

obj-$(CONFIG\_ISDN\_PPP\_BSDCOMP) += isdn\_bsdcomp.o /\*配置选项值为m\*/

如果模块由多个文件组成，则Makefile文件格式如下（/drivers/isdn/i4l/Makefile）：

obj-$(CONFIG\_ISDN\_I4L) += isdn.o

isdn-y := isdn\_net\_lib.o isdn\_v110.o isdn\_common.o

isdn.o表示模块名称，在构建内核时Kbuild将编译、链接isdn-y内包含的目标文件，以生成模块。

在执行make命令构建内核时将编译内核模块，而后也可以使用make modules重新编译所有的模块。

模块也可以单独编译，但需要已构建好的内核源码树，单独编译模块的命令如下：

**make -C $KDIR M=$PWD**

$KDIR表示内核源代码所在目录（绝对路径），$PWD表示模块代码所在目录（绝对路径），执行此命令后，Kbuild机制将读取模块代码所在目录下的Makefile文件，编译模块。

例如：作者主机中内核源码所在目录为/home/lch/linux-4.2.4，在此目录下创建了hello目录，目录下包含两个文件，一个为源代码文件hello.c，一个为Makefile文件。源代码hello.c文件内容如下，内容很简单只是在加载、卸载模块时输出内核信息：

#include <linux/init.h>

#include <linux/module.h>

static int \_\_init hello\_init(void)

{

printk(KERN\_INFO "Hello World enter\n");

return 0;

}

module\_init(hello\_init);

static void \_\_exit hello\_exit(void)

{

printk(KERN\_INFO "Hello World exit\n");

return;

}

module\_exit(hello\_exit);

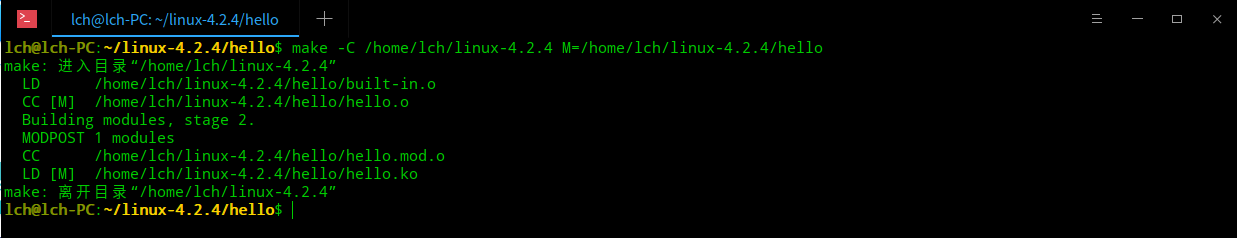
Makefile文件内容如下：

obj-m += hello.o

编译模块的命令如下：

**make -C /home/lch/linux-4.2.4 M=/home/lch/linux-4.2.4/hello**

执行结果如下图所示：



最终编译生成的模块目标文件为hello.ko。

以上命令还可以在末尾加上要构建的目标，例如：modules、modules\_install、clean等。

●modules：表示生成模块，与不加效果一样。

●clean：清除编译模块时产生的文件。

●modules\_install：安装模块，例如：

make INSTALL\_MOD\_PATH=/home/lch/rootfs modules\_install

INSTALL\_MOD\_PATH表示模块安装的路径，以上命令将内核所有模块安装到/home/lch/rootfs目录，在此目录下将创建./lib/modules/子目录用于存放模块目标文件。

以下命令用于编译并安装单独编译的模块：

make INSTALL\_MOD\_DIR=gandalf -C $KDIR M=$PWD modules\_install

模块安装路径为：/lib/modules/$(KERNELRELEASE)/gandalf/

/\*$(KERNELRELEASE)表示内核版本号，如4.2.4（目录名称）\*/

注意：以上路径都是在主机根文件系统中的路径。

## 14.4龙芯1B平台移植

本节介绍将公版linux-4.2.4内核移植到龙芯1B平台的内核配置与构建操作流程，公版内核中龙芯1B平台设备驱动程序较少，为降低难度，暂不介绍其它设备驱动程序的移植，先掌握内核配置与构建操作的流程，为后面驱动、应用的移植打下基础。下面先介绍内核源代码树中龙芯1B相关的平台代码、驱动代码以及相关的配置、构建文件，然后介绍配置、构建内核的操作。

### 14.4.1平台代码

龙芯1B处理器采用MIPS32体系结构，平台类型为loongson32（MACH\_LOONGSON32），处理器类型为LOONGSON1\_LS1B，平台及处理器相关代码目录及说明简列如下：

|  |  |  |
| --- | --- | --- |
| **目录** | **描述** | **备注** |
| /arch/mips/ |  |  |
| ./Kconfig | 含平台配置选项MACH\_LOONGSON32，  导入/arch/mips/loongson32/Kconfig配置文件。 |  |
| /arch/mips/include/asm/ |  |  |
| ./mach-loongson32/ | MACH\_LOONGSON32平台使用的头文件，中断编号、寄存器定义等。 |  |
| /arch/mips/loongson32/ |  |  |
| ./common/ | MACH\_LOONGSON32平台公共代码。 |  |
| ./ls1b | 龙芯1B专用代码。 |  |
| ./Kconfig | MACH\_LOONGSON32平台配置文件，选择处理器类型等。 |  |
| ./Makefile | MACH\_LOONGSON32平台代码构建文件。 |  |
| ./Platform | 平台特性参数设置，编译选项等。 |  |

#### 1配置与构建文件

在MIPS体系结构配置文件/arch/mips/Kconfig内选择平台类型菜单设置了MACH\_LOONGSON32配置选项：

menu "Machine selection"

choice

prompt "System type"

default SGI\_IP22

...

config MACH\_LOONGSON32

bool "Loongson-1 family of machines"

select SYS\_SUPPORTS\_ZBOOT

help

This enables support for the Loongson-1 family of machines.

Loongson-1 is a family of 32-bit MIPS-compatible SoCs developed by

the Institute of Computing Technology (ICT), Chinese Academy of

Sciences (CAS).

...

endchoice

...

source "arch/mips/loongson32/Kconfig" #导入平台配置文件

...

平台配置文件/arch/mips/loongson32/Kconfig用于选择处理器类型等，请读者自行阅读。

平台相关文件/arch/mips/loongson32/Platform是一个非常重要的文件，内容如下：

cflags-$(CONFIG\_CPU\_LOONGSON1) += \

$(call cc-option,-march=mips32r2,-mips32r2 -U\_MIPS\_ISA -D\_MIPS\_ISA=\_MIPS\_ISA\_MIPS32) \

-Wa,-mips32r2 -Wa,--trap

**platform-$(CONFIG\_MACH\_LOONGSON32) += loongson32/** /\*导向平台相关Makefile文件\*/

cflags-$(CONFIG\_MACH\_LOONGSON32) += -I$(srctree)/arch/mips/include/asm/mach-loongson32

load-$(CONFIG\_LOONGSON1\_LS1B) += **0xffffffff80100000**

第一个cflags-y用于设置编译选项，第二个cflags-y表示将/arch/mips/include/asm/mach-loongson32目录添加到头文件的搜索路径，load-y表示内核加载到内存的起始地址。

platform-y表示导向平台相关的Makefile文件，即/arch/mips/loongson32/目录下的Makefile文件。在MIPS体系结构的Makefile文件/arch/mips/Makefile中将导入/arch/mips/Kbuild.platforms文件，在此文件内又将导入/arch/mips/loongson32/Platform文件，在/arch/mips/Kbuild中也将导入/arch/mips/Kbuild.platforms文件（进而导入/arch/mips/loongson32/Platform文件），并将platform-y代表的平台Makefile中的目标文件添加到obj-y目标文件列表，如下所示：

include arch/mips/Kbuild.platforms

obj-y := $(**platform-y**)

从而将内核编译流程导向平台/arch/mips/loongson32/目录下的Makefile文件。

#### 2平台公共源代码文件

MACH\_LOONGSON32平台公共代码位于/arch/mips/loongson32/common/目录下，主要源文件及其内容简介如下：

**●irq.c：**主要实现plat\_irq\_dispatch()和arch\_init\_irq()函数，完成中断的初始化和处理函数。

**●prom.c**：主要实现prom\_init()函数，它在setup\_arch()函数内被调用，完成命令行参数的复制和内存大小的设置等。

**●setup.c：**主要实现plat\_mem\_setup()函数，完成内存段的注册，由setup\_arch()函数内arch\_mem\_init()函数调用。

**●reset：**实现ls1x\_reboot\_setup()函数，主要是实现关机、重启等函数，并将函数地址存入全局变量。

●**time.c：**注册PWM时钟源，时钟事件等。

**●platform.c：**定义设备信息结构，定义并注册设备xxx\_device实例，完成设备初始化等，这里是所有龙芯1x处理器公共的设备信息。

MACH\_LOONGSON32平台包含龙芯1B 、龙芯1C、龙芯1D等处理器，因此后面还有特定于处理器的板级源文件。

#### 3龙芯1B处理器代码

下面看一下/arch/mips/loongson32/common/platform.c文件内定义的外部设备信息和xxx\_device实例的定义，以及/arch/mips/loongson32/ls1b/board.c文件内注册设备xxx\_device实例的函数，先看公共的platform.c文件。

#include <linux/clk.h>

#include <linux/dma-mapping.h>

#include <linux/err.h>

#include <linux/phy.h>

#include <linux/serial\_8250.h>

#include <linux/stmmac.h>

#include <linux/usb/ehci\_pdriver.h>

#include <asm-generic/sizes.h>

#include <cpufreq.h>

#include <loongson1.h>

/\*8250/16550 兼容UART，串口设备信息\*/

#define LS1X\_UART(\_id) \

{ \

.mapbase = LS1X\_UART ## \_id ## \_BASE,\

.irq = LS1X\_UART ## \_id ## \_IRQ, \

.iotype = UPIO\_MEM, \

.flags = UPF\_IOREMAP | UPF\_FIXED\_TYPE, \

.type = PORT\_16550A, \

}

static struct plat\_serial8250\_port ls1x\_serial8250\_pdata[] = { /\*串口数据\*/

LS1X\_UART(0),

LS1X\_UART(1),

LS1X\_UART(2),

LS1X\_UART(3),

{},

};

struct platform\_device ls1x\_uart\_pdev = { /\*串口设备，平台总线设备\*/

.name = "**serial8250**", /\*设备名称，匹配驱动\*/

.id = PLAT8250\_DEV\_PLATFORM,

.dev = {

.platform\_data = ls1x\_serial8250\_pdata,

},

};

void \_\_init ls1x\_serial\_setup(struct platform\_device \*pdev) /\*串口初始化\*/

{

...

}

/\* CPUFreq \*/

static struct plat\_ls1x\_cpufreq ls1x\_cpufreq\_pdata = {

.clk\_name = "cpu\_clk",

.osc\_clk\_name = "osc\_33m\_clk",

.max\_freq = 266 \* 1000,

.min\_freq = 33 \* 1000,

};

struct platform\_device ls1x\_cpufreq\_pdev = { /\*平台设备\*/

.name = "ls1x-cpufreq",

.dev = {

.platform\_data = &ls1x\_cpufreq\_pdata,

},

};

/\* Synopsys Ethernet GMAC，GMAC控制器\*/

static struct stmmac\_mdio\_bus\_data ls1x\_mdio\_bus\_data = {

.phy\_mask = 0,

};

static struct stmmac\_dma\_cfg ls1x\_eth\_dma\_cfg = {

.pbl = 1,

};

int ls1x\_eth\_mux\_init(struct platform\_device \*pdev, void \*priv) /\*GMAC初始化\*/

{

...

}

static struct plat\_stmmacenet\_data ls1x\_eth0\_pdata = {

.bus\_id = 0,

.phy\_addr = -1,

.interface = PHY\_INTERFACE\_MODE\_MII,

.mdio\_bus\_data = &ls1x\_mdio\_bus\_data,

.dma\_cfg = &ls1x\_eth\_dma\_cfg,

.has\_gmac = 1,

.tx\_coe = 1,

.init = **ls1x\_eth\_mux\_init**, /\*初始化函数\*/

};

static struct resource ls1x\_eth0\_resources[] = { /\*GMAC资源\*/

[0] = {

.start = LS1X\_GMAC0\_BASE,

.end = LS1X\_GMAC0\_BASE + SZ\_64K - 1,

.flags = IORESOURCE\_MEM,

},

[1] = {

.name = "macirq",

.start = LS1X\_GMAC0\_IRQ,

.flags = IORESOURCE\_IRQ,

},

};

struct platform\_device ls1x\_eth0\_pdev = {

.name = "**stmmaceth**",

.id = 0,

.num\_resources = ARRAY\_SIZE(ls1x\_eth0\_resources),

.resource = ls1x\_eth0\_resources,

.dev = {

.platform\_data = &ls1x\_eth0\_pdata,

},

};

static struct plat\_stmmacenet\_data ls1x\_eth1\_pdata = {

.bus\_id = 1,

.phy\_addr = -1,

.interface = PHY\_INTERFACE\_MODE\_MII,

.mdio\_bus\_data = &ls1x\_mdio\_bus\_data,

.dma\_cfg = &ls1x\_eth\_dma\_cfg,

.has\_gmac = 1,

.tx\_coe = 1,

.init = ls1x\_eth\_mux\_init,

};

static struct resource ls1x\_eth1\_resources[] = {

[0] = {

.start = LS1X\_GMAC1\_BASE,

.end = LS1X\_GMAC1\_BASE + SZ\_64K - 1,

.flags = IORESOURCE\_MEM,

},

[1] = {

.name = "macirq",

.start = LS1X\_GMAC1\_IRQ,

.flags = IORESOURCE\_IRQ,

},

};

struct platform\_device ls1x\_eth1\_pdev = {

.name = "**stmmaceth**",

.id = 1,

.num\_resources = ARRAY\_SIZE(ls1x\_eth1\_resources),

.resource = ls1x\_eth1\_resources,

.dev = {

.platform\_data = &ls1x\_eth1\_pdata,

},

};

/\*USB EHCI，USB控制器\*/

static u64 ls1x\_ehci\_dmamask = DMA\_BIT\_MASK(32);

static struct resource ls1x\_ehci\_resources[] = { /\*USB控制器资源\*/

[0] = {

.start = LS1X\_EHCI\_BASE,

.end = LS1X\_EHCI\_BASE + SZ\_32K - 1,

.flags = IORESOURCE\_MEM,

},

[1] = {

.start = LS1X\_EHCI\_IRQ,

.flags = IORESOURCE\_IRQ,

},

};

static struct usb\_ehci\_pdata ls1x\_ehci\_pdata = {

};

struct platform\_device ls1x\_ehci\_pdev = { /\*平台设备，USB控制器\*/

.name = "**ehci-platform**", /\*设备名称\*/

.id = -1,

.num\_resources = ARRAY\_SIZE(ls1x\_ehci\_resources),

.resource = ls1x\_ehci\_resources,

.dev = {

.dma\_mask = &ls1x\_ehci\_dmamask,

.platform\_data = &ls1x\_ehci\_pdata,

},

};

/\* Real Time Clock，RTC\*/

struct platform\_device ls1x\_rtc\_pdev = { /\*RTC设备\*/

.name = "ls1x-rtc",

.id = -1,

};

MACH\_LOONGSON32平台公共的platform.c文件内定义了串口设备、CPUFreq设备、GMAC控制器、USB总线控制器、RTC设备的平台设备platform\_device实例及其硬件资源信息。

龙芯1B处理器板级文件/arch/mips/loongson32/ls1b/board.c内容如下：

#include <platform.h> /\*/arch/mips/include/asm//mach-loongson32/目录下，声明platform\_device实例\*/

/\*红色代码为作者添加的代码\*/

static void ls1x\_start\_ehc(void) /\*复位USB控制器，参考龙芯1B处理器手册\*/

{

int i=100000;

\*(volatile int \*)0xbfd00424 &= ~0x800; /\* enable USB \*/

\*(volatile int \*)0xbfd00424 &= ~0x80000000; /\* ls1g usb reset \*/

while(i--) /\*延时一段时间\*/

{};

\*(volatile int \*)0xbfd00424 |= 0x80000000; /\* ls1g usb reset stop \*/

}

static struct platform\_device \*ls1b\_platform\_devices[] \_\_initdata = {

&ls1x\_uart\_pdev,

&ls1x\_cpufreq\_pdev,

&ls1x\_eth0\_pdev,

&ls1x\_eth1\_pdev,

&ls1x\_ehci\_pdev,

&ls1x\_rtc\_pdev,

};

static int \_\_init ls1b\_platform\_init(void)

{

int err;

ls1x\_serial\_setup(&ls1x\_uart\_pdev); /\*串口设备初始化\*/

ls1x\_start\_ehc(); /\*复位USB控制器\*/

err = platform\_add\_devices(ls1b\_platform\_devices,ARRAY\_SIZE(**ls1b\_platform\_devices**));

/\*注册平台设备\*/

return err;

}

arch\_initcall(ls1b\_platform\_init); /\*内核启动时调用此函数\*/

特别需要注意的是上面的红色代码为作者添加的代码（来自开发板内核源代码）用于复位（激活）USB控制器，否则USB控制器将不能正常工作。

### 14.4.2驱动源代码文件

下面简要介绍一下前面平台代码中注册的设备对应的驱动程序在内核源代码树中的位置。

●**串口设备驱动**：

/drivers/drivers/tty/serial/8250/

需配置选择TTY、SERIAL\_8250等配置选项。

●**CPUFreq驱动**：

/drivers/cpufreq/ls1x-cpufreq.c

需选择CPU\_FREQ、LOONGSON1\_CPUFREQ等配置选项。

●**GMAC控制器驱动**

/drivers/net/ethernet/stmicro/stmmac/

需选择NET、ETHERNET、NET\_VENDOR\_STMICRO等配置选项。

●**USB控制器驱动**

/drivers/usb/host/ehci-platform.c

需选择USB\_SUPPORT、USB\_EHCI\_HCD\_PLATFORM等配置选项。

●**RTC驱动**

/drivers/rtc/rtc-ls1x.c

需选择RTC\_CLASS、RTC\_DRV\_LOONGSON1配置选项。

### 14.4.3配置内核

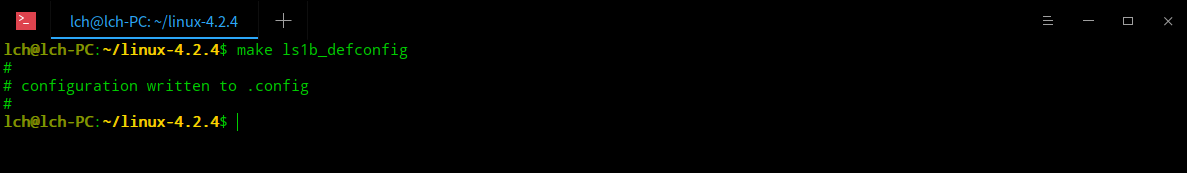
按照前面的介绍准备好交叉编译工具，按照下列要求修改内核源代码主Makefile文件内容：

linux_makefile

接下来就可以对内核源码进行配置了。

#### 1配置操作

内核在/arch/mips/config/目录下已经有龙芯1B开发板默认的配置文件ls1b\_defconfig，可先执行以下命令：

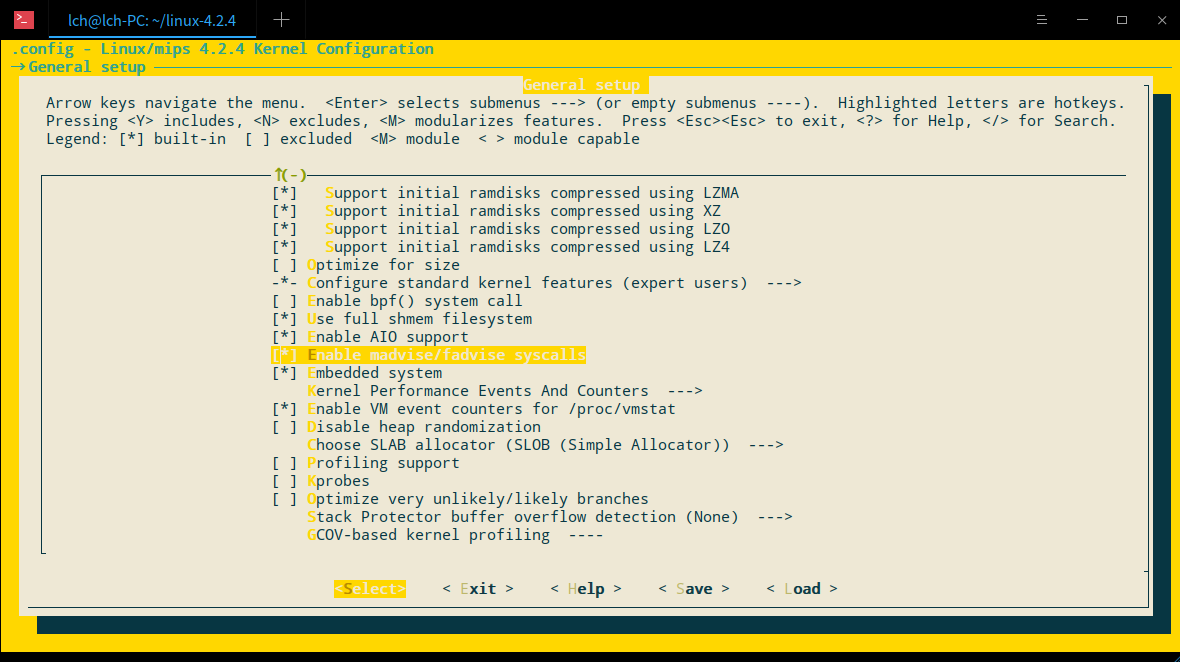


再执行make menuconfig命令启动配置操作，可查看或修改开发板的配置选项，配置完成后保存配置选项至.config文件，退出配置操作。

#### 2配置选项

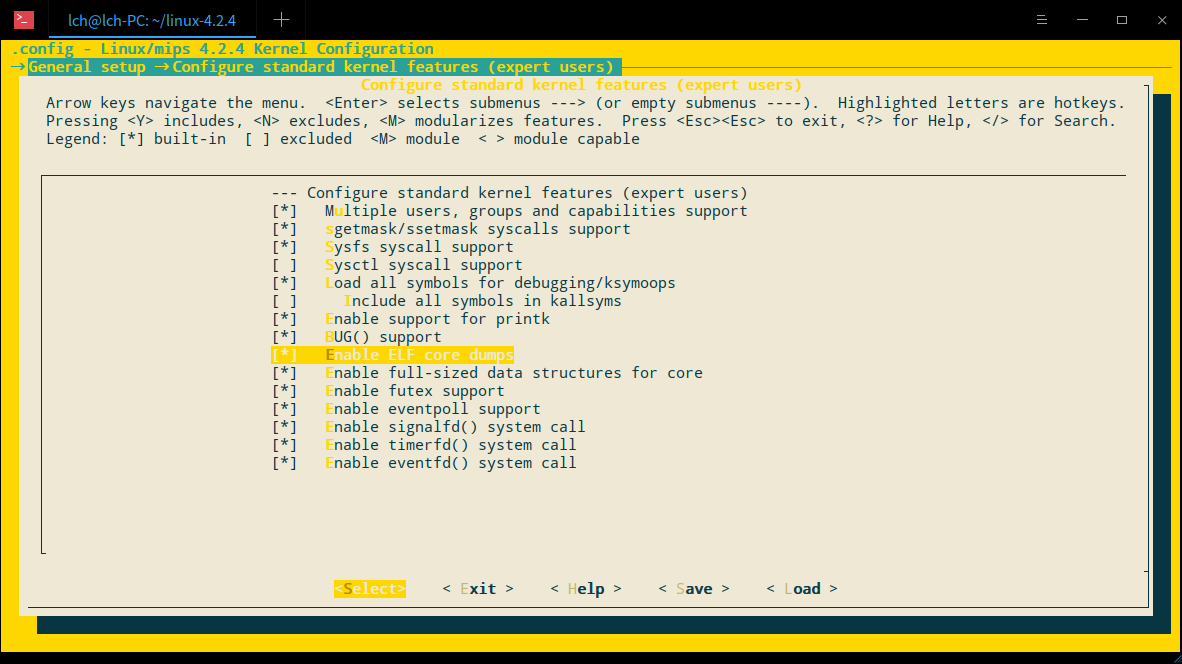
下面介绍一下在内核配置过程中作者检查或修改的几个配置选项：

■**General setup**



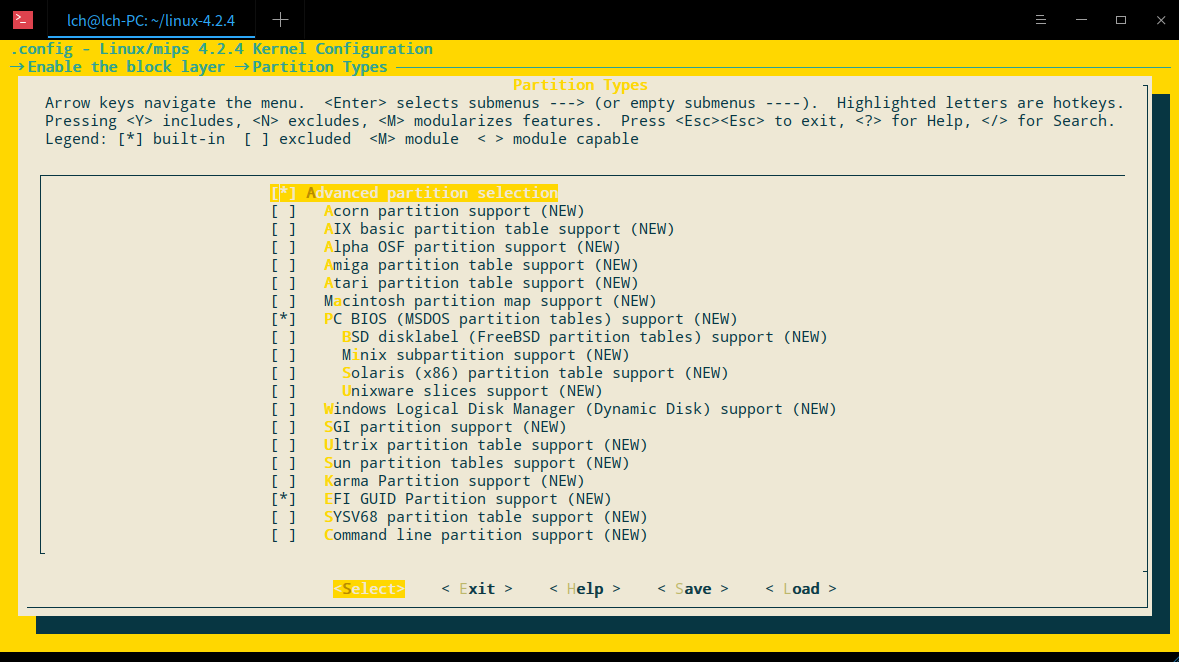
选择其中的**Embedded system**选项（适用于嵌入式系统），选择slob分配器（小型的分配器）。

■**General setup ->Configure standard kernel features (expert users)** （专家模式）



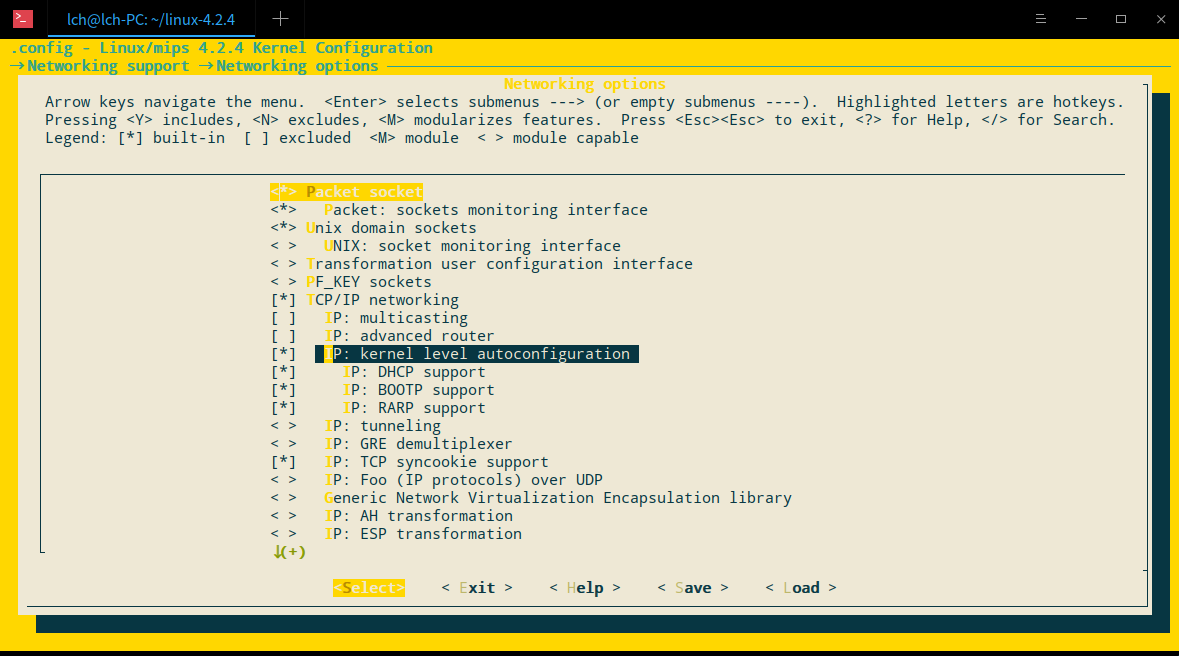
选择**Enable support for printk**选项，支持内核通过printk()函数输出信息。

■**Enable the block layer->Partition Types**



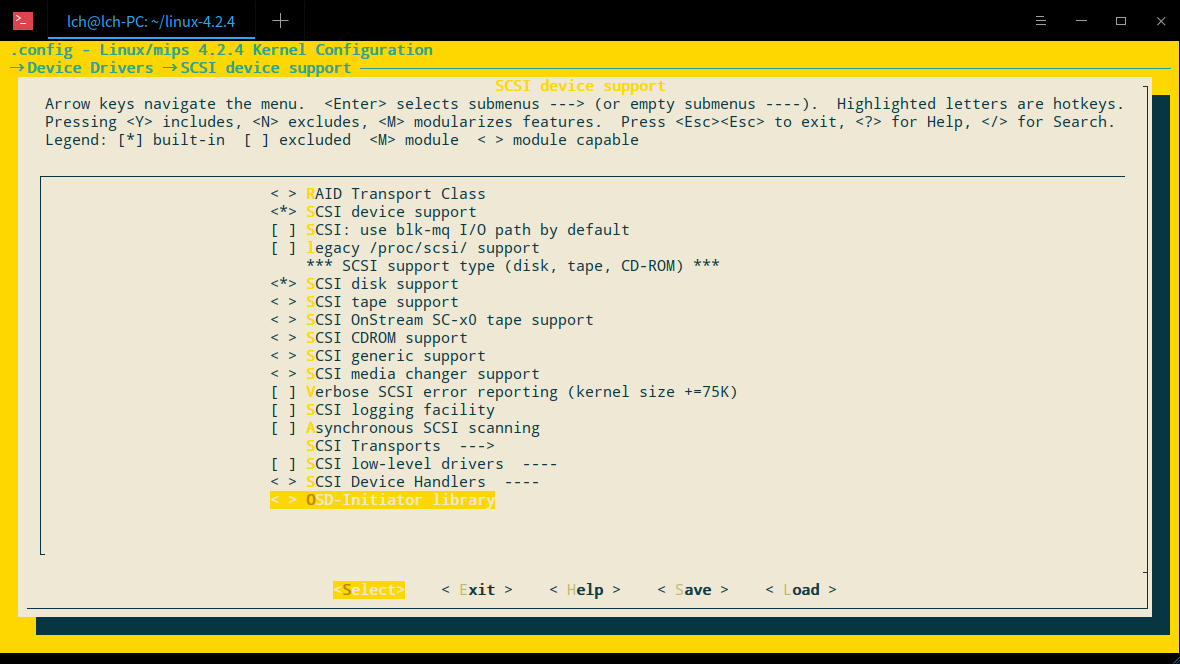
选择支持的块设备分区，如MSDOS分区等。

■**Networking support->Networking options**（支持的网络协议）



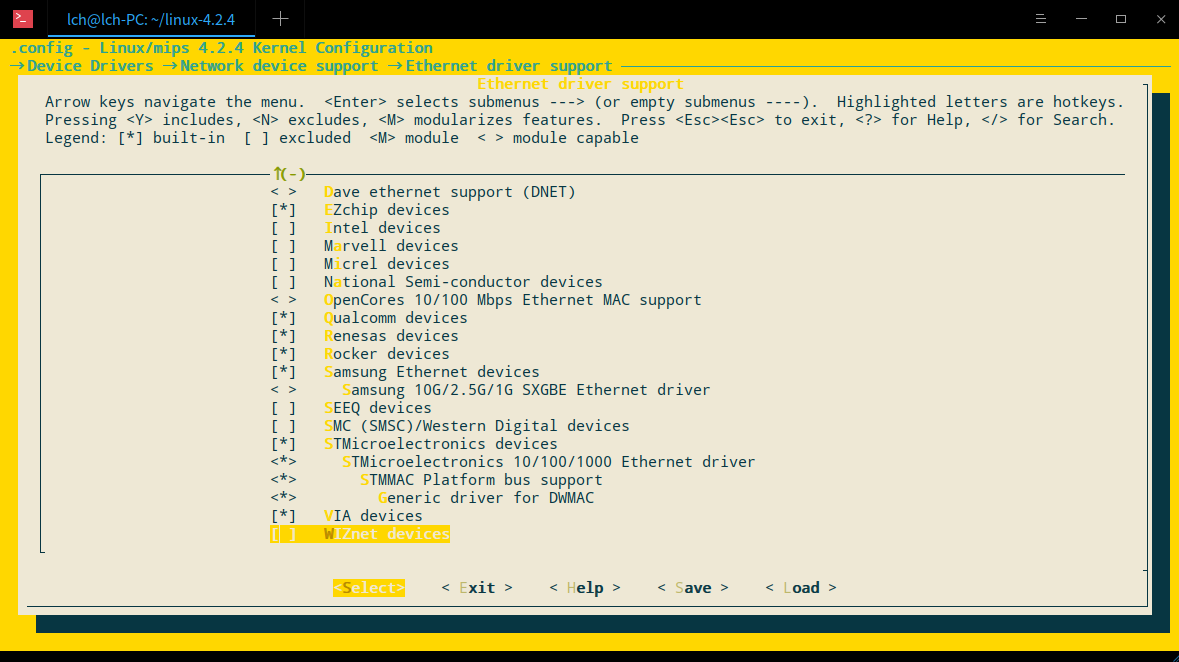
选择**IP: kernel level autoconfiguration**选项下的三个小项，用于支持NFS（网络文件系统）。

■**Device Drivers-> SCSI device support**



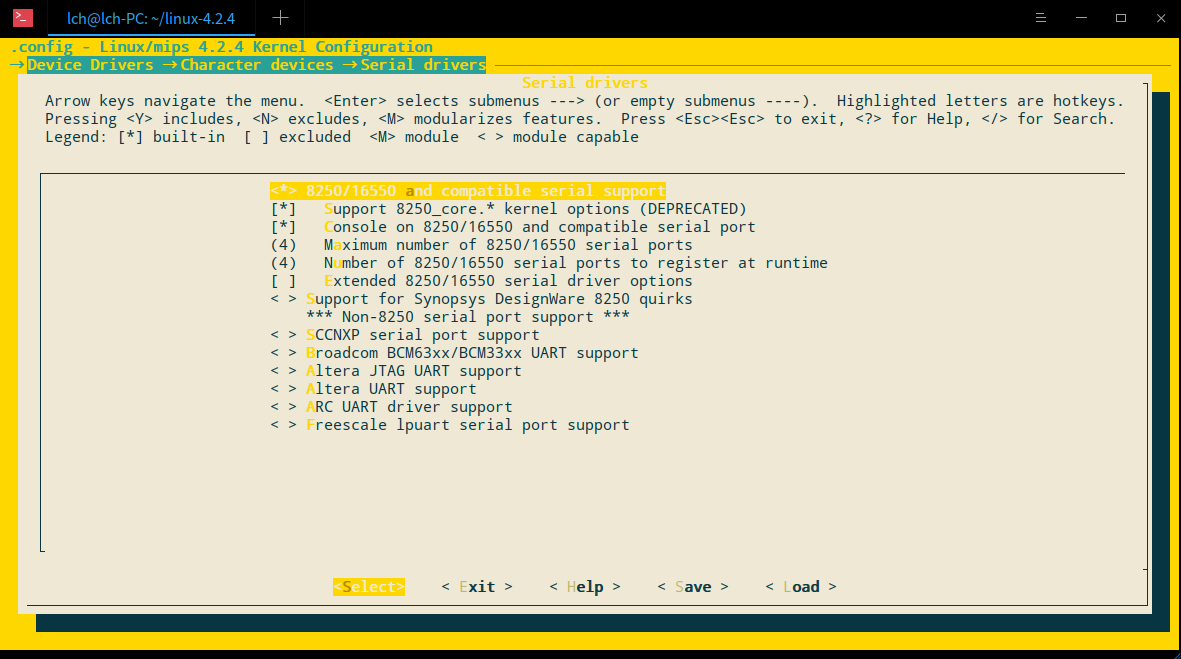
选择**SCSI device support**和**SCSI disk support**选项（持久编译入内核，不是模块），用于支持U盘驱动。

■**Device Drivers->Network device support->Ethernet driver support**（以太网驱动支持）



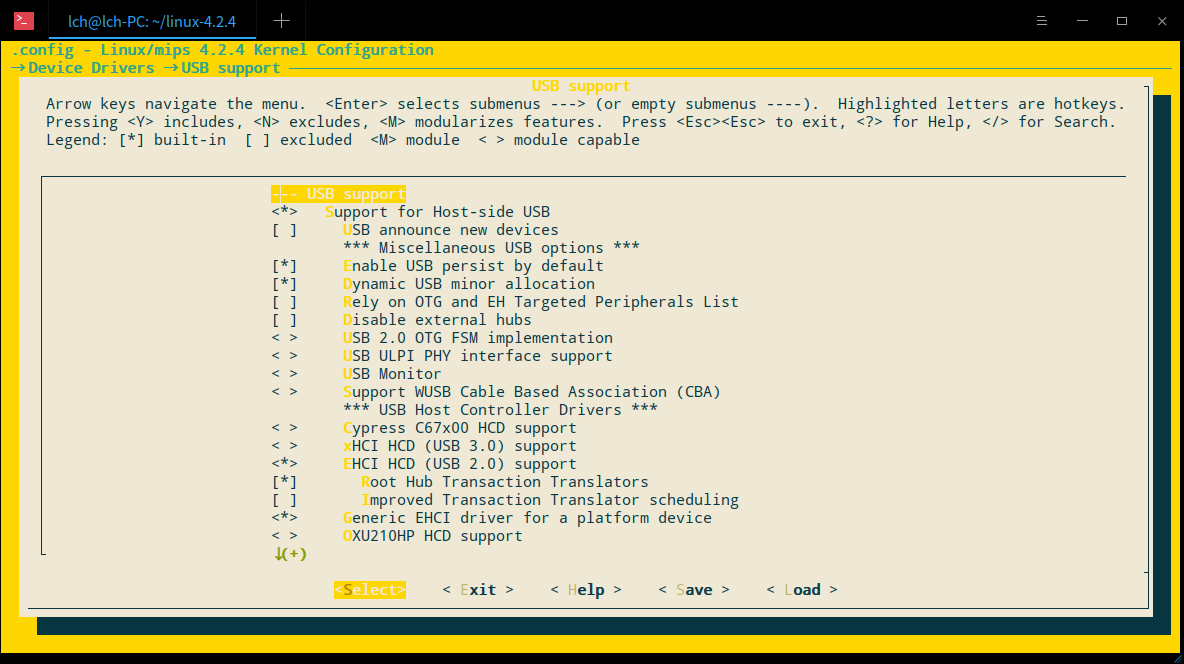
选中**STMicroelectronics devices**选项下的小项，用于支持GMAC网络控制器。

■**Device Drivers ->Character devices ->Serial drivers**



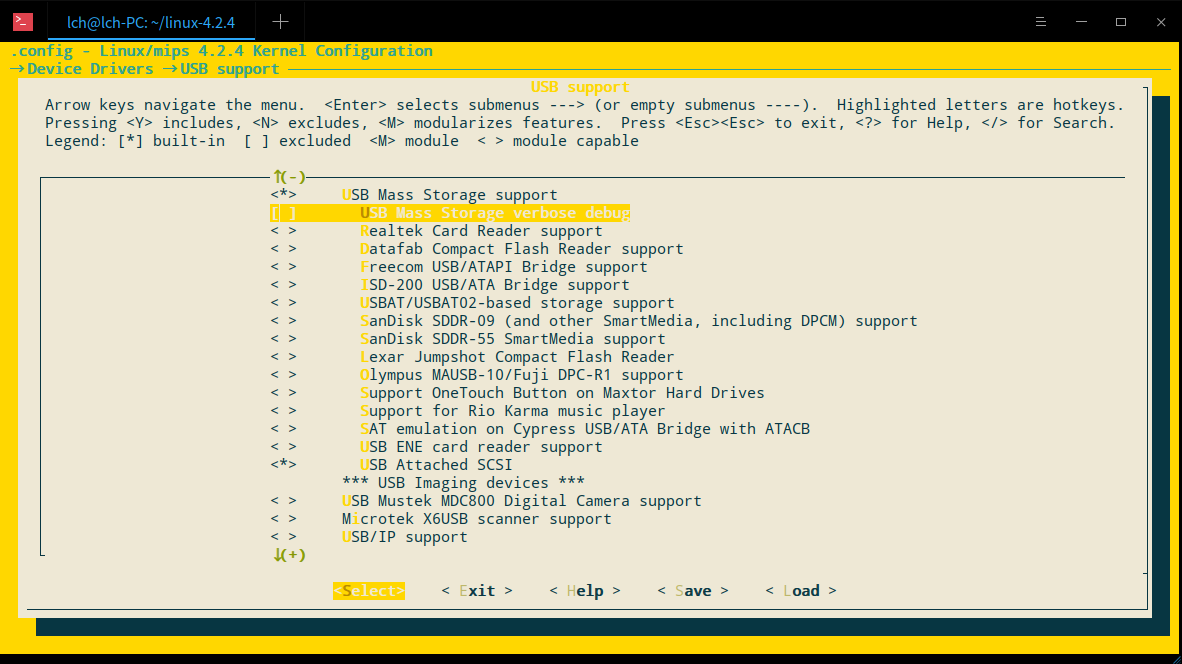
检查**8250/16550 and compatible serial support**选择是否选中，用于支持串口设备驱动。

■**Device Drivers ->USB support**



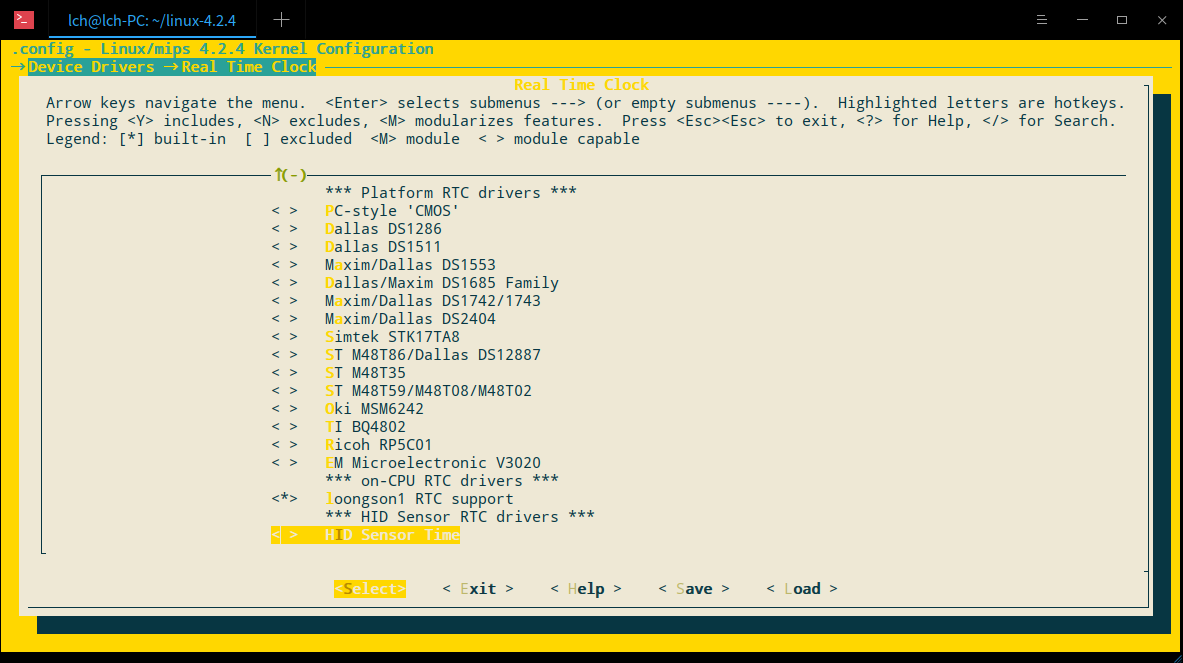
检查是否选中**EHCI HCD (USB 2.0) support**和**Generic EHCI driver for a platform device**选项，用于支持USB控制器驱动。

■**Device Drivers ->USB support**



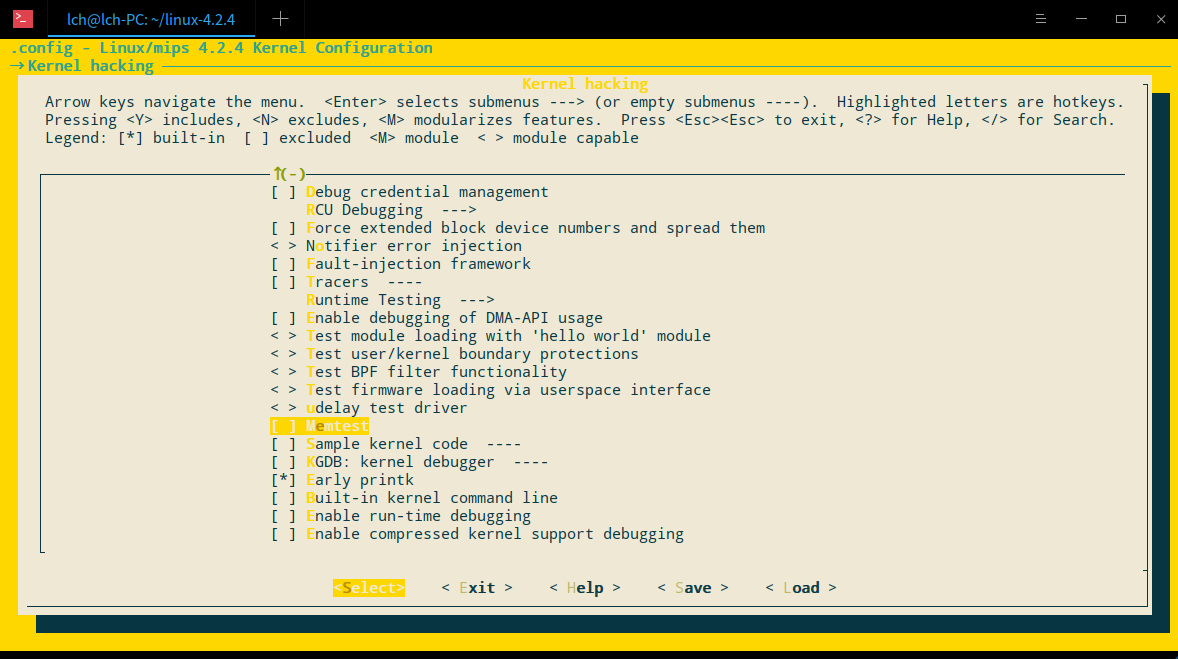
检查是否选择了**USB Mass Storage support**和**USB Attached SCSI**选项，用于支持U盘。

■**Device Drivers ->Real Time Clock**



检查是否选择了**loongson1 RTC support**选项，用于支持龙芯1B片上RTC。

■**Kernel hacking**



选择**Early printk**选项，用于内核输出早期启动信息。

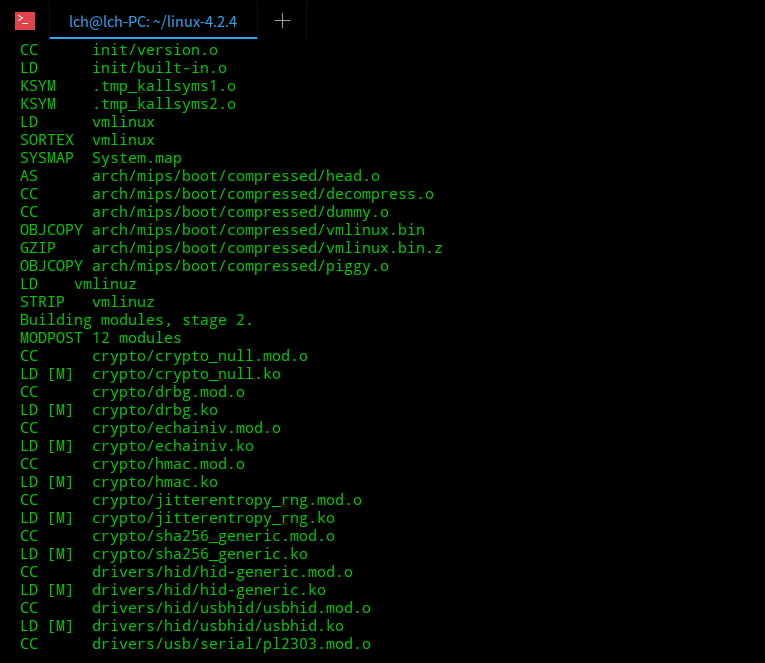
配置完成后，保存配置选项至.config配置结果文件，退出配置操作。

### 14.4.3构建内核

配置完成后执行make命令即可启动构建操作，将生成内核目标文件和模块。

#### 1构建操作

在内核源代码根目录下执行make命令即可启动构建操作，构建操作将需要一段时间。构建操作将编译各源代码文件，最终链接生成单一的内核目标文件和模块，生成目标文件列表如下图所示：



最终生成了vmlinux和vmlinuz内核可执行目标文件和12个模块。

#### 2导出头文件与模块

如果需要在目标机里编译C源文件，则需要将导出内核头文件供用户程序使用，导出头文件的命令如下：

make headers\_install INSTALL\_HDR\_PATH=*dir*

dir为用户指定的主机根文件系统下的绝对路径，导出头文件将在此目录下。

导出模块的命令前面已经介绍过了，即make INSTALL\_MOD\_PATH=*dir* modules\_install，例如以下为作者主机导出模块的命令（rootfs为根文件系统所在目录，见下文）：

make INSTALL\_MOD\_PATH=**/home/lch/rootfs** modules\_install

目录是指定的在主机根文件系统下的绝对路径，导出模块将保存在此目录下（创建/lib/modules/目录）。

## 14.5根文件系统

内核可视为一个程序，在系统启动时加载到内存中运行，只运行内核系统什么也做不了，不能做实际有用的工作。实际的工作由用户进程执行或发起，用户进程所执行的程序（目标文件）需要保存在外部介质中，在内核启动后期挂载到虚拟文件系统根目录下，内核从根文件系统中下搜索第一个用户程序运行。另外，内核还需要由根文件保存各种工具程序、配置文件、启动脚本程序和文件等。

在内核移植时需要事先将根文件系统的内容写入目标机外部存储介质中，在固件传递的命令行参数中通过root参数将介质设备文件名称传递给内核，内核在启动后期将此介质挂载到根目录。

嵌入式系统中通常采用busybox制作根文件系统，主要用于生成系统工具，用户还需要补充配置文件、库文件、头文件等数据。

### 14.5.1根文件系统目录

linux系统根文件系统中一般包含以下目录：

|  |  |  |
| --- | --- | --- |
| / | 根目录 |  |
| /bin | 存放二进制可执行文件，包含基本的命令。 | 通常由busybox生成 |
| /sbin | 存放二进制可执行文件，通常是系统命令。 | 通常由busybox生成 |
| /boot | 包含内核镜像及启动相关文件。 |  |
| /dev | 设备文件，用于访问外部设备。 |  |
| /home | 用户主目录，一个用户对应一个子目录。 |  |
| /lib | 存放系统库（动态库、静态库）、内核模块等。 | 模块由内核导出 |
| /media | 媒体目录，可移动介质挂载点。 |  |
| /mnt | 另一个外部移动介质挂载点。 |  |
| /opt | 可选目录，存放用户软件包等。 |  |
| /root | root用户主目录。 |  |
| /tmp | 临时目录 |  |
| /usr | 存放用户程序、库及头文件等。 |  |
| /var | 可变目录，存放日志文件等。 |  |
| /sys | 挂载sysfs文件系统。 |  |
| /proc | 挂载procfs文件系统。 |  |
| /etc | 存放系统配置文件，启动脚本。 | 此目录下文件由用户创建 |
| /inittab | init进程的配置文件，是init进程第一个访问的脚本文件。 |  |
| /init.d/rcS | 存放init进程启动脚本等。 |  |
| /fstab | mount -a命令挂载的文件系统列表，在rcS中写入mount -a命令。 |  |
| /profile | 运行/bin/sh命令时首先执行的脚本。 |  |

### 14.5.2 busybox

busybox用于生成各种系统工具（命令的执行程序），它与内核一样，需要下载源代码压缩包、解压，然后进行配置、构建，最后执行安装操作，将生成的系统工具存入指定目录下（/bin、/sbin、/usr/bin和/usr/sbin）。

busybox是在嵌入式系统中广泛使用的工具集，它集成了许多常用的工具软件（Linux命令执行的程序）。busybox可从官网https://busybox.net/下载，作者下载的是busybox-1.32.0，解压后对源码进行配置、构建操作的命令如下：

（1）**make menuconfig**：配置busybox，主要用于设置构建选项、选择生成的系统工具。

（2）**make**：构建操作，编译生成系统工具目标文件。

（3）**make install**：生成根文件系统目录，包含生成的工具，用于移植到目标机根文件系统内。

另外，执行以上步骤前需要修改根目录下的Makefile文件，设置交叉编译工具前缀，如下：

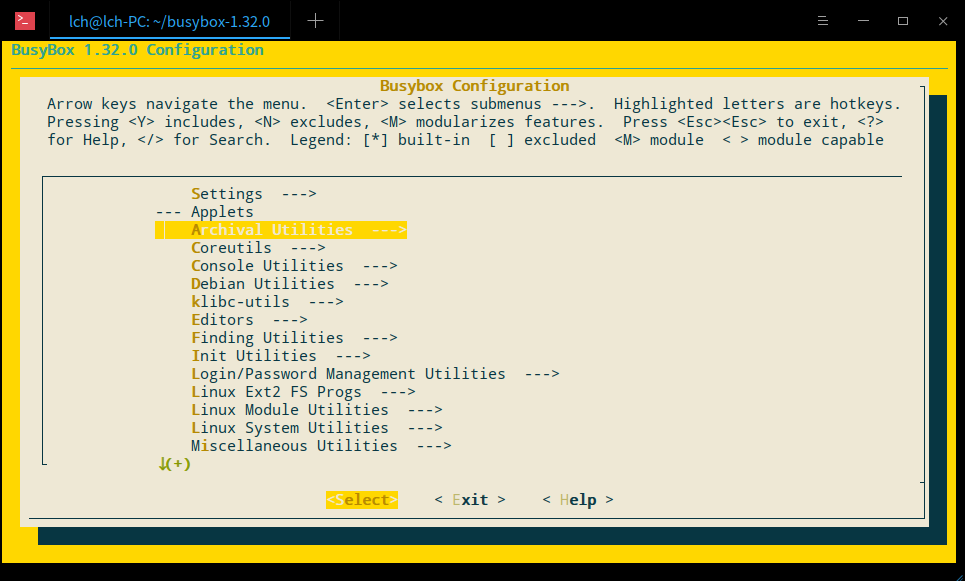
busybox

busybox0

交叉编译工具前缀可以在下面介绍的配置选项中指定，也可以在构建时通过CROSS\_COMPILE命令行参数指定，三者任选其一即可。但是，如果有两个以上同时设置了，则Makefile文件内的设置具有最高优先权。

#### 1配置

配置busybox需执行make menuconfig命令，配置界面如下图所示：



主界面下主要包含以下几个菜单：

●**Settings**：通用设置选项，包含构建选项等。

●**Applets**：选项要编译构建的系统工具。

●**Load an Alternate Configuration File**：加载配置结果文件。

●**Save Configuration to an Alternate File**：保存配置选项至配置结果文件，默认为.config。

Applets菜单下主要包含一些工具集的配置选项，用户可根据需要选择需要编译构建的系统工具，这里就不详细介绍了。

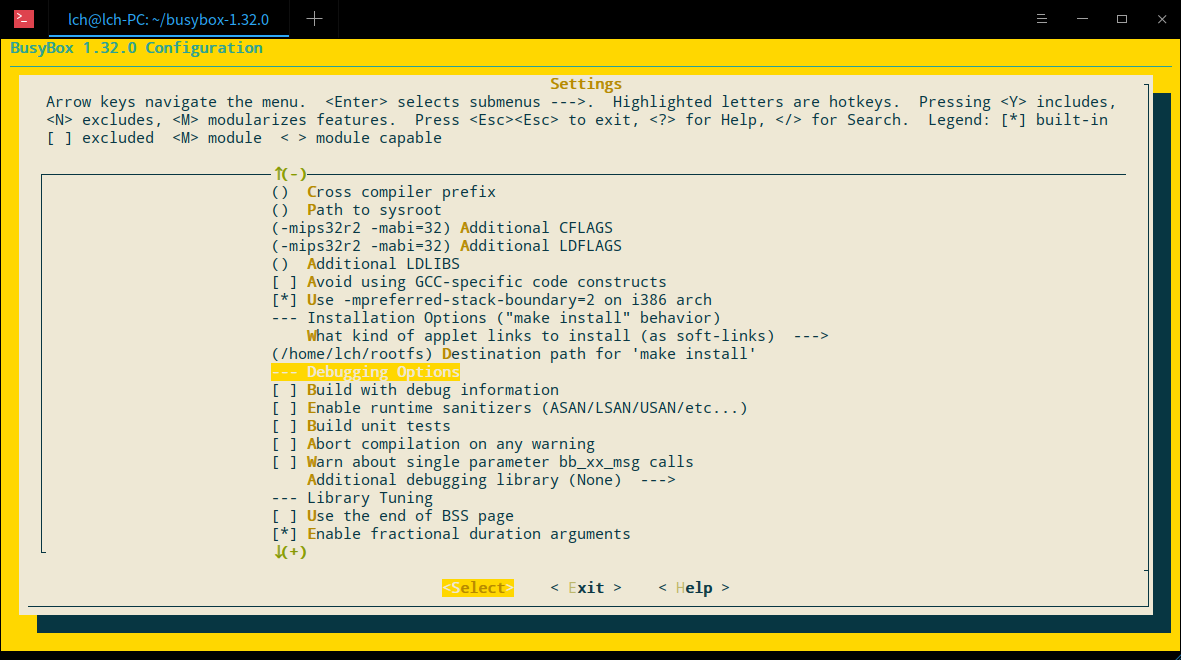
下面简要介绍一下Settings菜单下的几个构建选项（Build Options），如下图所示：

●**Build static binary (no shared libs)** ：构建时使用静态库，否则使用动态库。如果使用动态库，还要在目标机根文件系统中移植动态库。使用静态库则将库函数静态链接到目标文件中，运行时不需要加载动态库，这里选择使用静态库。

●**Cross compiler prefix**：交叉编译工具前缀（可不写）。

●**Additional CFLAGS、Additional LDFLAGS**：增加编译、链接选项，这里添加**-mips32r2 -mabi=32**选项，表示目标文件体系结构及ABI类型（o32），编译器默认编译的是小端数据结构，因此没有添加-EL编译选项。

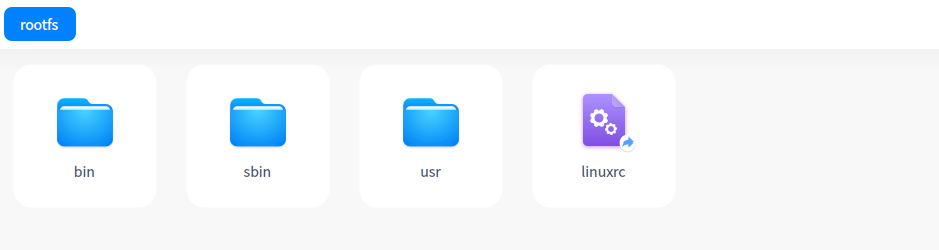
●**Destination path for 'make install'**：执行make install命令后，生成的目录在主机上保存的路径。默认为./\_install，用户可以自行修改，这里作者创建了/home/lch/rootfs目录保存根文件系统内容。



配置完成后，保存配置文件，下一步即可执行构建操作。

#### 2构建与安装

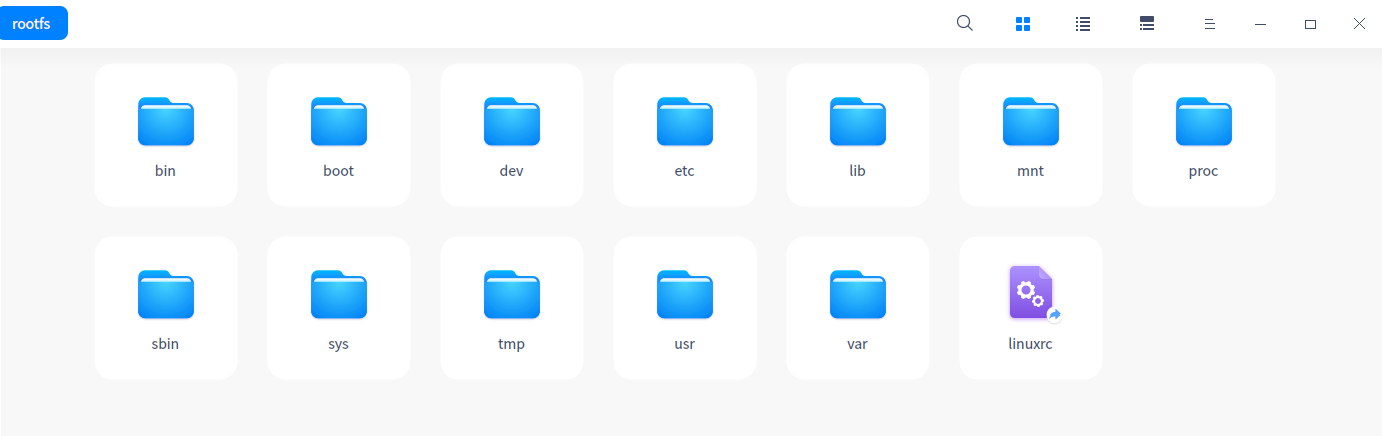
busybox构建命令为make，执行完make命令后，执行make install命令将生成用于根文件系统的目录（含系统工具），默认情况下生成的目录保存在./\_install目录下，如果配置时修改了路径将保存在指定的路径下。作者主机上生成的目录如下图所示：



其中/bin、/sbin、/usr/bin、/usr/sbin目录是工具集，linuxrc用于内核启动后加载第一个用户进程，常用的init进程可执行目标文件为/sbin/init。所有的工具，含linuxrc和init都是到/bin/busybox可执行目标文件的符号链接，busybox可执行程序相当于一个分配器，它会根据不同的命令名称执行不同的操作。

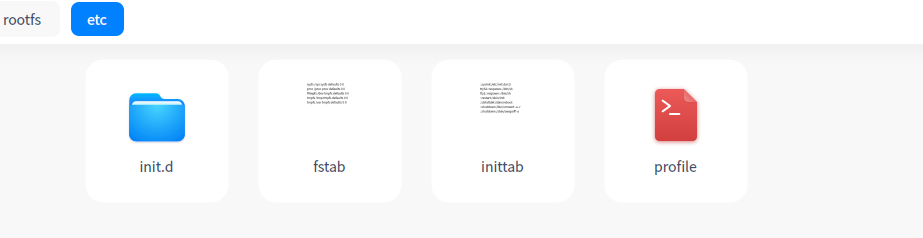
### 14.5.3配置文件与库

目标机根文件系统中除了busybox创建的目录外，还需要用户手工创建其它的目录，例如：/sys、/proc、/lib、/etc等，其中最重要的是/etc目录下的配置文件和/lib目录下的库及模块文件等。作者在前面busybox生成的根文件系统目录下创建其它的目录如下图所示：



#### 1 /etc

/etc目录用来存放初始化脚本和其它配置文件，在此目录下需要创建以下文件及目录：



其中创建文件用touch命令，例如创建fstab文件的命令为：touch fstab。init.d目录下还需要创建rcS文件，下面分别介绍各文件内容及功用。

##### ■inittab

inittab文件是init进程的配置文件，是init进程访问的第一个脚本文件，后续启动的程序都由它指定。

inittab文件内容由数个条目组成，每个条目表示init进程执行流程的一个步骤或状态，条目格式如下：

**<id>:<runlevels>:<action>:<process>**

**<id>：**在busybox init中表示进程关联的控制台设备文件名，如ttyS2，若为空则默认为标准输入输出设备。

**<runlevels>：**运行级别，在busybox init中忽略，不支持运行级别，此处为空。

**<process>：**表示要执行的命令或脚本。

**<action>：**表示动作或者说步骤，包括sysinit、wait、once、respawn、askfirst、shutdown、restart和ctrlaltdel，每个条目可在各选项中选择其一。以上各动作由init进程按顺序执行，或在收到某条命令或操作时执行。

●sysinit：表示init进程启动时执行的动作，init进程将等待动作结束，<process>表示动作执行的命令或脚本，下同。

●wait：sysinit动作之后的动作，init进程等待其结束。

●once：下一个执行的动作，init进程不等待其结束。

●**respawn**、askfirst：接下来执行的动作，askfirst执行前会输出提示字符串，要求用户按回车键，这两个是重复执行的动作。

●shutdown：在执行halt/reboot/poweroff命令，或处理SIGQUIT信号前执行的操作。

●restart：init重启时的动作（重启由信号SIGQUIT触发），init进程将由此条目中指示的进程（命令）覆盖。

●ctrlaltdel：init进程收到SIGINT信号时执行的动作，信号由CTRL-ALT-DEL组合键触发，此动作执行完后将继续执行respawn、askfirst动作。

注：inittab相关说明详见busybox源代码下/examples/inittab和/init/init.c文件。

下例中是根文件系统中inittab文件的内容：

::sysinit:/etc/init.d/rcS #指定init进程启动后首先执行的脚本文件。

**ttyS2::respawn:-/bin/sh** #在串口终端运行shell，串口号ttyS2要与启动参数一致。

**tty1::respawn:-/bin/sh** #用于开发板屏幕终端显示，在开发板显示屏运行shell。

::restart:/sbin/init #重启后还是运行init进程。

::ctrlaltdel:/sbin/reboot #捕捉 ctrl+alt+del 键，重启系统。

::shutdown:/bin/umount -a -r #当关机时卸载所有文件系统。

::shutdown:/sbin/swapoff -a

注意：ttyS2::respawn:-/bin/sh：在/bin/sh前的‘-’表示需要登陆，一般由login程序处理，但此处忽略。

##### ■rcS

上面的inittab文件指示init进程启动后第一个运行的脚本文件为/etc/init.d/rcS，主要用于设置环境变量、挂载文件系统等，下面是rcS文件内容：

#!/bin/sh

export PATH=/bin:/sbin:/usr/bin:/usr/sbin #设置PATH环境变量

/bin/hostname "lch" #设置主机名称

mkdir -p /dev/pts

mount -t devpts devpts /dev/pts

**/bin/mount -a** #挂载/etc/fstab文件中文件系统列表

echo /sbin/mdev > /proc/sys/kernel/hotplug

/sbin/mdev -s #运行mdev工具

ifconfig eth0 192.168.1.107 up #设置网络IP地址

ifconfig lo 127.0.0.1

##### **■fstab**

前面介绍的rcS文件内的mount -a命令将挂载/etc/fstab文件指定的文件系统，此文件内容如下：

sysfs /sys sysfs defaults 0 0

proc /proc proc defaults 0 0

tmpfs /tmp tmpfs defaults 0 0

tmpfs /var tmpfs defaults 0 0

##### ■profile

init进程在respawn动作中启动的/bin/sh命令，在启动后将首先运行/etc/profile脚本文件，文件内容设置如下：

#!/bin/sh

echo "Processing /etc/profile....." #输出信息。

export LD\_LIBRARY\_PATH=/lib:/usr/lib #搜索库路径（环境变量）。

export PATH=/bin:/sbin:/usr/bin:/usr/sbin

USER="root" #用户名称。

HOSTNAME='/bin/hostname'

PS1='[$USER@\h:\w]\$'

echo "Done!"

综上所述，系统启动的第一个用户进程，即init进程的执行流程及各配置文件的调用关系如下图所示：



#### 2 /lib

根文件系统/lib目录用于存放库文件、模块等，库文件来自交叉编译工具，模块在内核构建时生成。在前面介绍的安装模块的操作中将指定目录下生成/lib目录，模块文件存放在/lib/modules/目录下。如果此/lib目录不在制作的根文件系系统目录下，则需要执行复制操作。

目标机用户程序使用的动态库来源于交叉编译工具，作者所用的交叉编译工具动态库所在目录为：

/home/lch/mipsel-linux-musl-cross/mipsel-linux-musl/lib

以下命令用于将动态库文件（.so）复制到目标机根文件系统/lib目录下：

sudo cp -d ./\*.so\* /home/lch/rootfs/lib

其中终端在/home/lch/mipsel-linux-musl-cross/mipsel-linux-musl/lib目录下运行，-d参数用于保留符号链接文件。

#### 3复制内核镜像

内核镜像（可执行目标文件）通常也保存在根文件系统所在的介质中（/boot），系统从中加载内核镜像。当然内核镜像也可以保存在其它介质中，例如：系统从U盘中加载内核镜像，挂载NAND Flash作为根文件系统。由于作者后面是从U盘加载内核镜像，同时也挂载U盘作为根文件系统，因此需要复制内核镜像到根文件系统/boot目录下，内核镜像为源码所在目录下的vmlinux文件（未压缩）。

至此，目标机根文件系统在主机上已经制作完成，接下来就可以将其写入到目标机挂载的介质中。这里需要注意的一点是，需要保证根文件系统下的工具、脚本文件具有可读、可执行权限，可通过ls -l命令查看，如果不符合要求可通过chmod命令修改权限。

### 14.5.4制作启动盘

嵌入式系统中通常挂载外部存储介质作为根文件系统，如NAND Flash、SD卡、U盘等。这里以制作U盘启动盘例，说明制作的流程，因为制作U盘启动盘比较简单，系统也从U盘内核镜像。

#### 1格式化U盘

首先准备一个U盘对其进行分区（作者分了2个区，应该也可以不分区）并将其格式化为ext2文件系统，步骤如下：

■将U盘插入主机（连接到虚拟机），执行以下命令获取U盘设备文件名称：

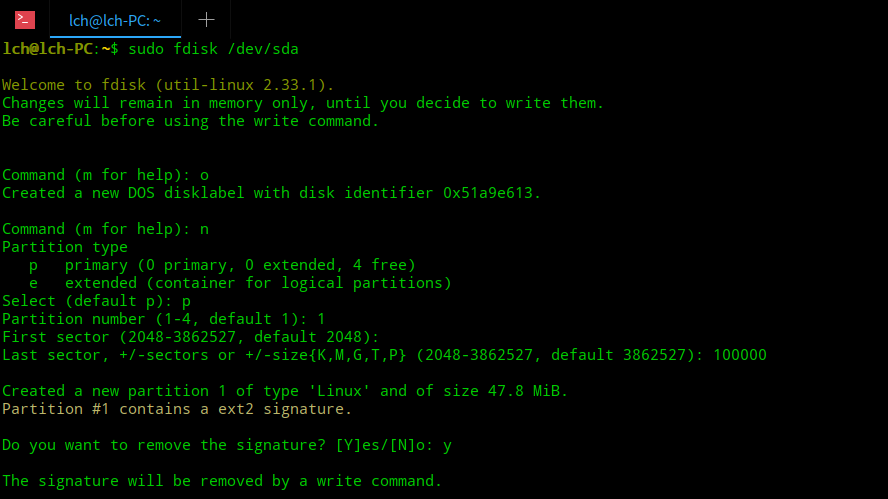
sudo fdisk -l

作者主机上U盘设备文件名为/dev/sda。（要特别注意，小心别把主机硬盘格式化了！）

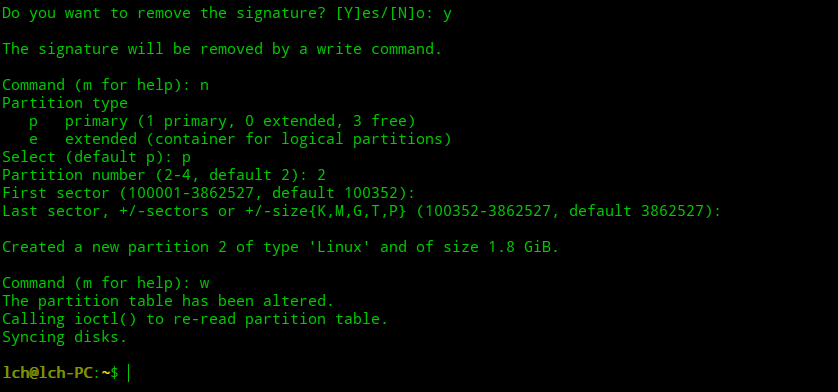
■卸载U盘：

sudo umount /dev/sda

■执行fdisk /dev/sda命令进行分区



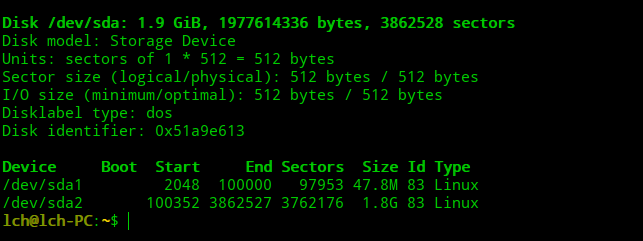
在Command命令行输入o，创建DOS分区，随后输入n创建一个新的分区，再输入p表示创建主分构，输入分区号1，分区起始扇区号采用默认值，结束扇区号为100000，由于作者U盘中原先存在ext2分区，因此确定删除它。按照同样的方法创建第二个分区，如下图所示：



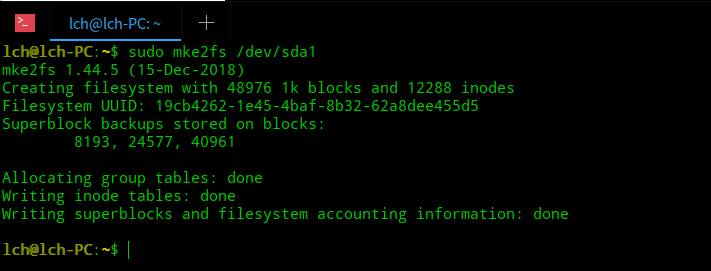
第二个分区的结束扇区采用默认值，即到U盘的结束，整个U盘分两个区。

■格式化U盘分区

先用fdisk -l命令来查看一下分区的结果，如下图所示：



如上图所示，U盘sda被分为2个分区，设备名称分别为/dev/sda1和/dev/sda2。下面通过mke2fs命令将/dev/sda1分区格式化为ext2文件系统：



用同样的方法可格式化/dev/sda2分区。

#### 2复制根文件系统

格式化U盘后，需要挂载U盘到主机文件系统中，以下命令用于将U盘/dev/sda1分区挂载到主机文件系统：

sudo mount -t ext2 /dev/sda1 /home/lch/mnt

以上命令将U盘第一个分区挂载到主机/home/lch/mnt目录下，此目录要先创建。

以下命令用于将主机/home/lch/rootfs/目录下的根文件系统内容复制到U盘第一个分区：

**sudo cp -rd /home/lch/rootfs/\* /home/lch/mnt**

至此，U盘启动盘制作完成，现在万事俱备，只欠东风了。将U盘插入目标机，开机进入PMON修改环境变量和命令行参数后即可启动系统了。

## 14.6运行操作系统

现在终于可以进行最后一步了，从U盘启动系统。不过，在启动系统前还需要完成以下工作：

■开发板接上电源、插上U盘，将串口线串口一端与开发板com2连接，USB与主机相连。

■在主机上安装配置串口工具minocom，用作目标机控制台，输入输出信息。

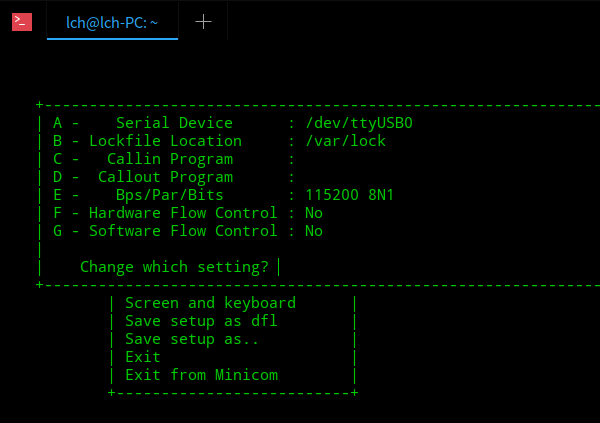
■配置固件，设置加载内核镜像的路径和命令行参数。

### 14.6.1串口工具

在主机上执行以下命令安装串口工具minicom：

sudo apt-get install minicom

执行sudo minicom -s命令设置串口，设置结果如下（Serial port setup 选项，/dev/ttyUSB0为USB转串口设备文件名称），保存并退出，此时处于mincom界面。 后面启动串口工具不再需要进行设置，直接执行sudo minicom命令即可。



主机终端中的minicom将作为开发板内核输出信息的控制台，以及用户进程的控制台。

### 14.6.2从U盘启动系统

引导加载程序（固件）是系统启动运行的第一段代码，保存在系统启动ROM内，它负责加载位于外部介质中的内核镜像到系统内存，启动内核并向内核传递命令行参数。引导加载程序也可以看成一个小系统，它能初始化处理器、内存等，检测外部设备，包括存储介质，串口等，可识别介质文件系统等。可修改引导加载程序中的参数从不同的介质加载内核镜像，也可以修改传递给内核的命令行参数。

嵌入式系统中比较常用引导加载程序是U-boot，但龙芯处理器采用的是PMON，龙芯1B开发板已经烧写了PMON。

#### 1从U盘加载内核镜像

要系统从U盘加载内核镜像需要修改PMON环境变量。将USB转串口线两端分别与主机和开发板相连，将启动U盘插入开发板，主机启动minicom。开发板上电，在主机键盘迅速按下任意键将进入PMON（类似于电脑的BIOS）。PMON常用命令如下：

　　●**load**：加载内核。

　　●**set**：列出所有设置好的环境变量（PMON中的环境变量）。

●**set *envname* value**：将环境变量envname设置成value值。

●**reboot**：重启。

　　●**devls**：查看系统可用的设备，如硬盘，U盘等。

环境变量是PMON中保存的系统信息以及传递给内核的加载信息等。PMON中set命令可设置环境变量，env命令可查看PMON中的全部环境变量。

系统启动过程中加载内核镜像的路径通过环境变量al来设置，而传递给内核的命令行参数通过环境变量append传递。

PMON中设备采用文件形式管理位于/dev/fs路径下，设置al环境变量命令如下：

set al /dev/fs/fat@usb0/boot/vmlinux

其中fat为介质文件系统类型，可为fat，ext2等，usb0为通过devls查阅到的介质名称，/boot/vmlinux为介质下文件的路径，fat@usb0表示第一个分区。以上环境变量值表示加载U盘第一个分区中/boot/vmlinux内核镜像。

若U盘格式化为ext2文件系统则设置环境变量命令为：

**set al /dev/fs/ext2@usb0/boot/vmlinux**

#### 2挂载U盘根文件系统

下面通过设置append环境变量来向内核传递命令行参数。挂载U盘作为根文件系统的命令行参数设置如下：

set append "root=/dev/sda1 console=ttyS2,115200 noinitrd rootfstype=ext2 rw rootdelay=3"

命令行参数简介如下：

■**root=\*\*\***：加载为根文件系统的外部设备文件名称，这里为/dev/sda1表示U盘第一个分区。

■**rootfstype=\*\*\***：加载根文件系统的类型，如ext2。

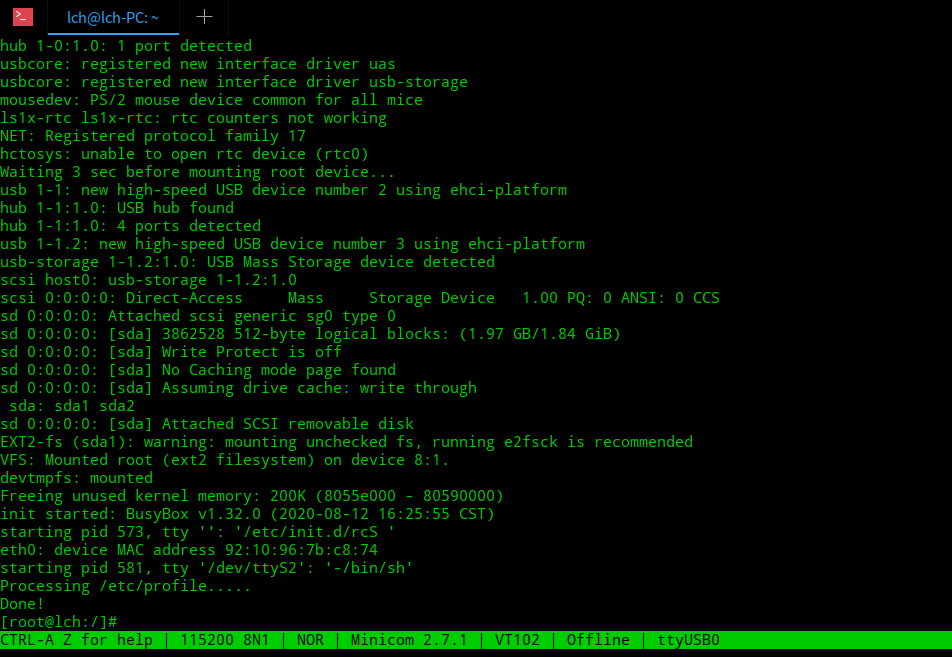
■**rw/ro**：以读写/只读方式挂载外部介质。

■**noinitrd**：设置采用ramfs不采用ramdisk。

■**rootdelay=\*\*\***：设置加载根文件系统前等待的时间（单位秒），以便外部介质初始化完成，这里设置为3秒，如果不设置U盘将挂载不成功。

■**console=\*\*\***：表示用户进程标准输入、输出、错误输出的控制台信息，这里为串口2。

修改完al和append环境变量后，重启系统或运行PMON的reboot命令，将从U盘加载内核镜像并运行，内核初始化完成后加载U盘第一个分区作为根文件系统，运行根文件系统中/sbin/init程序作为第一个用户进程，运行结果如下图所示：



至此，内核启动完成，开发板运行着Linux操作系统！！！

#### 3挂载NFS服务目录

NFS不是传统意义上的文件系统，而是访问远程文件系统的协议。其最主要的功能就是让网络上的 linux机器之间共享目录及文件。用户可以将远端所共享出来的文件目录，挂载（mount）在本地端系统上，从而可以很方便的使用远端的文件（目录）。

开发板通过NFS共享主机的数据，目标机与主机之间通过网线相连，挂载主机NFS到根文件系统某一目录下，从使目标机可通过NFS获取主机的数据，不需要通过U盘在主机与目标机之间来回复制数据（目标机根文件系统中的数据），以方便应用程序的调试。

作者主机配备了USB以太网卡，其中一端通过USB接口与主机相连，另一端通过网线接口与开发板LAN1接口相连。

##### ■安装配置NFS服务器

（1）在主机终端上运行以下命令在线安装NFS软件：

**sudo apt-get install nfs-kernel-server**

（2）创建NFS目录，并更改其权限：

可通过mkdir命令创建NFS目录，这里作者创建了/home/lch/mrootfs目录。通过chmod命令来改变目录及子目录（文件）的权限：

**sudo chmod -R 777 /home/lch/mrootfs**

（3）配置 NFS 服务器

修改配置文件/etc/exports

sudo vim /etc/exports

在文件的最后新建一行，添加如下内容：

**/home/lch/mrootfs \*(rw,sync)**

内容含义：

/home/lch/mrootfs：要共享的目录，需要先创建后更改权限。

\* ：表示所有主机，也可指定特定的主机，如：192.168.1.13 指定 IP 地址的主机，nfsclient.test.com指定域名的主机，192.168.1.0/24 指定网段中的所有主机，\*.test.com 指定域下的所有主机。

rw：读写访问。

sync：同步。

（4）重启NFS服务

使用以下命令重启NFS服务，或重启系统（以后的使用中无需再重启，除非修改了配置文件）：

**sudo /etc/init.d/nfs-kernel-server restart**

（5）在本机上测试NFS服务

将USB以太网卡与主机相连，另一端与开发板LAN1网络接口相连，开发板上电开机。在Deepin系统设置里设置主机IP地址并保存，IP地址值设为：192.168.1.117，如下图所示：



执行以下命令可将NFS服务挂载到本机目录下：

sudo mount -t nfs -o nolock 192.168.1.117:/home/lch/mrootfs /home/lch/mnt

挂载成功后，/home/lch/mnt目录下内容将与/home/lch/mrootfs目录下内容相同。

##### **■挂载到目标机**

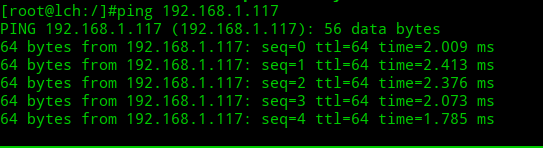
前面讲解内核配置时要求在**Networking support->Networking options**选项卡下选择**IP: kernel level autoconfiguration**选项下的三个小项，用于目标机支持NFS（网络文件系统），这里不再重复了。

通过USB转串口线将主机与开发板相连，插入启动U盘，主机启动minicom，开发板上电，启动开发板系统。开发板从U盘启动，并挂载U盘分区作为根文件系统后，可将主机NFS服务目录挂载到其根文件系统某个目录下。

在minicom终端输入以下命令，确认开发板与主机间的网络是否连通：

ping 192.168.1.117

输出结果如下图所示，表示已连通：

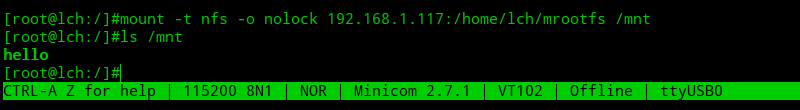


按Ctrl+C键结束ping进程。

在minicom终端执行以下命令，将主机NFS服务目录挂载到目标机根文件系统/mnt目录下：

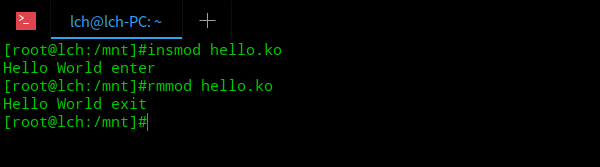
**mount -t nfs -o nolock 192.168.1.117:/home/lch/mrootfs /mnt**

下面通过ls命令查看是否挂载成功：



作者在主机/home/lch/mrootfs目录下创建了hello目录，/mnt目录下内容与主机/home/lch/mrootfs目录下内容相同，说明挂载成功。在主机上对此目录下内容的修改在目标机上（/mn）也将立即可见。

下面试验一下前面编译的hello.ko模块，将此模块文件复制到主机/home/lch/mrootfs目录下，并确保具有可执行权限，下面是在目标机上加载和卸载模块的操作结果（模块加卸载操作成功）：



读者也可以将mount -t nfs -o nolock 192.168.1.117:/home/lch/mrootfs /mnt命令写入目标机/etc/init.d/rcS脚本文件内，以使目标机启动后自动挂载NFS文件系统。同时可以将/mnt目录写入目标机/etc/profile文件中的PATH环境变量，从而使目标机可以直接运行/mnt目录下的程序。

## 14.7小结

本章介绍了将Linux内核移植到目标机（龙芯1B开发板）的流程。首先主机要安装Linux发行版操作系统（Deepin系统）和交叉GCC编译工具，然后对内核源代码进行配置和构建，生成内核镜像和模块，随后要制作根文件系统并复制到外部存储介质中（内核镜像一并写入），最后启动目标系统进入固件程序设置加载内核镜像的路径和挂载到根目录的块设备文件名称。系统启动后将从指定路径加载内核镜像并运行，内核启动后期将挂载外部介质到根目录，并从中查找第一个用户进程的目标文件，加载运行，系统启动完成。

为了简化操作，使读者快速地掌握内核配置、构建、运行操作系统的操作流程，作者直接选取了公版的linux-4.2.4内核进行移植（目标机驱动程序较少），制作了U盘启动盘，从U盘加载内核镜像，挂载U盘分区作为根文件系统，运行操作系统。本章最后介绍了挂载主机NFS文件系统到目标机根文件系统目录下的方法，以方便应用程序的调试。