## 2.2编译内核

如前所述，调试内核和内核模块、修改内核源码，是学习内核的重要技术手段之一。应用这些技术时，都有一本基本的要求，那就是编译内核。因此，在分析内核调试技术之前，本节首先讨论内核的编译方法。

### 2.2.1编译前的准备

在编译内核之前，有些准备工作需要进行。首先，就是从内核网站<https://www.kernel.org/>，下载好源码。这之前，需要确定下载哪个版本的内核源码。第1章提到过本书的环境是ubuntu 20.04，内核版本为5.4.0。而笔者选择的下载版本，就是5.4.0。至于原因，待会解释。

接着，使用apt-get安装一些必要的软件。在进行配置时，会执行命令“make menuconfig”，它需要libncurses5-dev、flex、bison等软件。ncurses库可用于管理字符终端界面，而bison和flex是生成词法分析器的工具。除此之外，还需要安装好kernel-package、libssl-dev。

最后一项准备工作，就是禁止内核地址随机化。出于安全考虑，内核支持ASLR (Address Space Layout Randomization)，即地址随机化。但是随机化在增加安全保障的同时，也给我们学习内核带来了一些麻烦，比如内核模块、函数等相关地址不方便直接确定。在本书的实验环境中，不管是开发机，还是目标机（后面分析kgdb时会讨论这两个概念），都需要禁止内核地址随机化。禁止的方法，即修改grub配置文件。具体步骤如下：

* 打开/etc/default/grub文件；
* 在该文件中，找到GRUB\_CMDLINE\_LINUX，修改其为：GRUB\_CMDLINE\_LINUX="nokaslr"；
* 执行命令update-grub，以更新grub配置文件等；
* 最后，重启系统即可。

上述步骤实质就是设置了内核命令行参数nokaslr，其意为禁止内核ASLR。一般来说，单纯编译内核是不需要这一步骤的。但是，本书的诸多实验，以及后面将讨论的kgdb克隆出的目标机，都需要禁止内核地址随机化。因此，笔者将其放在了编译前的准备工作中。

### 2.2.2编译步骤

准备工作做完，接下来就要编译内核了。这个编译其实也分了好几个大的步骤，下面首先介绍第一步配置。

原则上来讲，内核需要配置的项非常多，而且很多涉及到了一般读者并不太熟悉的硬件细节。这样一来，稍不留神就会导致配错项，而这个错误往往是在编译内核时才暴露出来，非常耽误时间。笔者曾碰到过一个学生，耗时两周才完成了编译工作，原因就在于诸多的配置项把他卡住了。鉴于这种情况，笔者一般建议直接使用系统已有的配置文件。系统在安装过程中，就已经根据当前的硬件环境做出了选择。我们直接拿来用就可以了，非常省事。下面就以本书实验环境为例，加以说明。

本书实验环境是ubuntu 20.04，其配置文件即/boot/config-5.4.0-28-generic（读者的配置文件名称可能会少有不同，比如config-5.4.0-29-generic等）。我们首先把该配置文件，复制到下载并解压好的内核源码目录中，比如本机环境的/root/kernelsrc/linux-5.4。前面提到，下载的内核源码版本是5.4.0，和当前系统所用的内核版本一致。其原因就在这里，可以非常方便地直接使用系统自带的内核配置文件。注意，在复制配置文件/boot/config-5.4.0-28-generic的同时，需要将其更名为.config。

前面提到过使用内核命令行参数，禁止内核地址随机化。其实也可以在.config文件中，注释掉CONFIG\_RANDOMIZE\_BASE配置项以实现禁用的目的。这之后，可以执行命令“make menuconfig”，在图形界面中对.config文件进行配置。就本书的实验目的而言，这一步其实没有什么需要配置的，该配置的默认就已经配好了，包括kgdb。

配置好了以后，先不着急编译。我们需要改一改内核的Makefile文件（即顶层Makefile文件），以去优化编译内核。我们编译内核的目的是为了调试，而优化很多时候会影响调试。比如代码的顺序被改变了，某个变量被优化掉了等等，这常常导致源码和实际运行情况不一致。怎么去优化呢？直接修改内核的Makefile文件，比如本书环境中为/root/kernelsrc/linux-5.4/ Makefile，将其中的-O2优化级别全部替换成-O1。可能有读者会问了，既然去优化，为什么不直接改成-O0呢？答案很简单，改成-O0或者-Og，后面编译过不了。能降低一点优化级别，也是好的。

虽然顶层Makefile文件中不能将优化级别变成-O0，但是有可能把某个模块的优化级别降成-O0。比如，当我们需要调试研究某一个模块时，可以在该模块自己的Makefile文件中加入-O0。但是，这个做法并不能保证适合于每一个模块。

去优化完成之后，我们就可以开始真正编译了。命令也十分简单“make-kpkg --initrd kernel-headers kernel\_image”。耐心等待较长时间后，会生成两个deb文件inux-headers-5.4.0\_5.4.0-10.00.Custom\_amd64.deb、linux-image-5.4.0\_5.4.0-10.00.Custom\_amd64.deb。在本书环境中，这两个文件位于/root/kernelsrc目录。之后，直接执行“dpkg -i \*.deb”进行安装就可以了。

到此为止，内核就编译完成了。不过，ubuntu 20.04默认情况下，开机是看不到grub界面的，也就无法选择进入新编译好的内核。所以，此时还得修改一个/etc/default/grub文件。将其中的“GRUB\_TIMEOUT\_STYLE=hidden”注释掉，以显示grub界面；并且将GRUB\_TIMEOUT修改成“GRUB\_TIMEOUT = 10”，以留出10秒时间让用户选择进入哪个内核。同样，最后还需要执行update-grub命令，并重启系统。

为了便于读者进行内核编译的实验，下面总结性地列出了主要步骤。

内核编译步骤

第一步：将下载的内核源码5.4.0，解压于目录/root/kernelsrc/linux-5.4中。并假设该目录，就是当前工作目录（即执行pwd命令，将得到路径/root/kernelsrc/linux-5.4）。

第二步：# cp /boot/config-5.4.0-28-generic .config

第三步：# vi .config，注释掉CONFIG\_RANDOMIZE\_BASE。

第四步：# make menuconfig

第五步：去优化编译，将顶层Makefile中的-O2，改成-O1。

第六步：# make-kpkg --initrd kernel-headers kernel\_image

第七步：# cd ../，回到上级目录，即/root/kernelsrc。

# dpkg -i \*.deb，安装生成的两个deb文件。

第八步：修改/etc/default/grub，注释掉“GRUB\_TIMEOUT\_STYLE=hidden”，并设置等待时间。

第九步：# update-grub

# reboot

### 2.2.3编译后的结果

接着上面第九步，重启系统后，就会等待十秒，以让用户选择进入哪个内核。由于我们编译的内核，和系统自带的内核都是5.4.0的版本，因此grub所显示的条目名称有点接近。比如，在笔者的环境中，自己编译出来的内核对应的grub条目为“Ubuntu, with Linux 5.4.0”，而系统自带的内核对应的条目则为“Ubuntu, with Linux 5.4.0-28-generic”。除了上述grub的变化外，编译内核源码后，还会有以下几个方面的输出。

首先，增加了目录/lib/modules/5.4.0。对于/lib/modules目录，读者应该不会陌生。因为我们在2.1.5节分析内核模块Makefile文件时，就提到过它。/lib/modules/5.4.0-28-generic/build中包含了内核的头文件、顶层Makefile文件，以及各个模块的Makefile文件等等。而新增加的/lib/modules/5.4.0中，也有build子目录，它也是一个符号链接，实际就指向了我们编译内核源码的目录，即/root/kernelsrc/linux-5.4。显然，/lib/modules/5.4.0/build里面要啥有啥，除了头文件、Makefile文件外，连源码都有。另外，/lib/modules/5.4.0/kernel目录中，存放了大量的内核模块文件。

其次，增加了目录/usr/src/linux-headers-5.4.0。前面分析内核模块Makefile文件时，提到/lib/modules/5.4.0-28-generic/build实际上是一个符号链接，指向了/usr/src/linux-headers-5.4.0-28-generic。我们新编译出来的内核，同样在/usr/src中增加了一个子目录。该目录包含了若干头文件、顶层Makefile文件，以及各个模块的Makefile文件等等。

第三，在/boot目录中增加了文件config-5.4.0、initrd.img-5.4.0、System.map-5.4.0、vmlinuz-5.4.0。config-5.4.0就是编译内核时的配置文件，initrd.img-5.4.0是一个用于mount根文件系统的内存盘，System.map-5.4.0中存放了若干符号信息，而vmlinuz-5.4.0则是内核映像。

最后，在/boot/grub/grub.cfg文件中，还增加了新编译内核对应的grub启动项，如“menuentry 'Ubuntu, with Linux 5.4.0'……”。