目录

linux 内核编译

0x00 简介

一般在有特殊的需要的时候，我都会手动的编译内核，内核里面有其实有相当多的驱动是我不需要的，我的老设备有些声卡有需要的驱动，现成的内核里面又没有的情况，这时候我比较习惯的方式就是来编译内核了，当然也可以编译成模块来使用。

另外就是工作中有时候也会有要求os版本，例如ubuntu 16.04 又要使用非常新的cpu和主板的时候，使用新的内核，基础软件包不大变化也可以尝试做一些测试使用。

下面的例子我是以ubuntu的发型版本来做的各种说明，其他版本的，大体流程基本一致，特殊的部分还需要大家自己去修改验证。如果全部写出来就这个文档就实在太长了。

0x01 资料准备

1. ubuntu 官方kernel deb

如果你没有对内核编译非常的不熟悉，又想体验各个版本直接用下面的ubuntu 编译好的即可。下载后使用

ubuntu的编译好的各种二进制的版本链接

ubuntu kernel deb Index of /mainline

几个ubuntu的官方资料

Home - Ubuntu Wiki

Kernel - Ubuntu Wiki

https://wiki.ubuntu.com/Kernel/FAQ

Kernel/Handbook - Ubuntu Wiki

2. 手动编译的准备材料

2.1安装编译环境

```
sudo apt install  kmod cpio build-essential bc binutils bison dwarves flex gcc git gnupg2 gzip libelf-dev libncurses5-dev libssl-dev make openssl perl-base rsync tar xz-utils zstd
```

2.2 下载源码

kernel.org，在页面中寻找第一个稳定版本。你不会找不到它，因为它是最显眼的黄色方框哦 😉

也可以使用内核镜像地址去下载，在国内的速度会快一点
https://mirrors.edge.kernel.org/pub/linux/kernel/v5.x/

内核获取链接以后用下载工具motrix或者aria2c下载的比较快一些

2.3 配置源码

解压内核后到解压的目录

```
cd linux-*/

cp /boot/config-"$(uname -r)" .config
#复制现有的内核的配置文件到新的内核源码里面作为配置文件

make olddefconfig
#老的内核一般都有和新内核不同的地方，这里用这个配置方法来保留老的的配置，老内核不支持的内容
新内核采用默认配置
```

内核编译之前都需要配置一下。有很多的方式下面简单讲一下。具体可用的配置方法可以在内核的目录下，输入`make --help`来查看，里面有非常多的内容。

配置方面相关的有一些节选出来

```
"make config"      Plain text interface.
"make menuconfig"  Text based color menus, radiolists & dialogs.
"make nconfig"     Enhanced text based color menus.
"make xconfig"     Qt based configuration tool.
"make gconfig"     GTK+ based configuration tool.
"make oldconfig"   Default all questions based on the contents of
                   your existing ./.config file and asking about
                   new config symbols.
"make olddefconfig"
                   Like above, but sets new symbols to their default
                   values without prompting.
"make defconfig"   Create a ./.config file by using the default
                   symbol values from either arch/$ARCH/defconfig
                   or arch/$ARCH/configs/${PLATFORM}_defconfig,
                   depending on the architecture.
"make ${PLATFORM}_defconfig"
                   Create a ./.config file by using the default
                   symbol values from
                   arch/$ARCH/configs/${PLATFORM}_defconfig.
                   Use "make help" to get a list of all available
                   platforms of your architecture.
"make allyesconfig"
                   Create a ./.config file by setting symbol
                   values to 'y' as much as possible.
"make allmodconfig"
                   Create a ./.config file by setting symbol
                   values to 'm' as much as possible.
"make allnoconfig" Create a ./.config file by setting symbol
                   values to 'n' as much as possible.
"make randconfig"  Create a ./.config file by setting symbol
                   values to random values.
"make localmodconfig" Create a config based on current config and
                      loaded modules (lsmod). Disables any module
                      option that is not needed for the loaded modules.

                      To create a localmodconfig for another machine,
                      store the lsmod of that machine into a file
                      and pass it in as a LSMOD parameter.

                      Also, you can preserve modules in certain folders
                      or kconfig files by specifying their paths in
                      parameter LMC_KEEP.

                target$ lsmod > /tmp/mylsmod
                target$ scp /tmp/mylsmod host:/tmp

                host$ make LSMOD=/tmp/mylsmod \
                        LMC_KEEP="drivers/usb:drivers/gpu:fs" \
                        localmodconfig
```

```
                      The above also works when cross compiling.

"make localyesconfig" Similar to localmodconfig, except it will convert
                      all module options to built in (=y) options. You can
                      also preserve modules by LMC_KEEP.
"make kvmconfig"    Enable additional options for kvm guest kernel support.
"make xenconfig"    Enable additional options for xen dom0 guest kernel
                    support.
"make tinyconfig"   Configure the tiniest possible kernel.
```

make config

是文本式，对话式，基于命令行的一种配置。 一个一个的询问，比较麻烦。

make menuconfig

这是有菜单选项以及有辅助图形界面的配置，基于ncurses库。（常常使用）

make oldconfig

使用旧的(之前)的配置，当前存在的 .config 文件询问新的特性该怎么处理（常用）

make olddefconfig

使用之前的配置，有新特性，默认不询问。（我常用）

make nconfig
也是一种字符界面的方式，我个人喜欢用这个。（我常用）

make xconfig

真正意义上的用到了图形界面的配置，使用的是QT的库。由于会加载额外的库，并不是用得很多。

make deconfig

直接使用某个默认的配置，和配置内容相关，与使用何种配置方式无关。

make gconfig

使用的图形界面进行内核配置，可能看起来好看一些？使用的是GTK的库。

make localmodconfig

自动检测当前linux内核加载的内核模块，根据当前的linux内核加载模块情况自动添加相关的参数到 .config 文件中，然后用 make menuconfig 等手动配置内核的方式进行内核配置，可以简化部分内核模块的选择。

使用 `make menuconfig` 修改的时候你会遇到下面的情况，做个说明，你可以根据各选项的类型来进行切换操作。还有几个方式都可以看看 `make help`

有两类可切换选项：

- 布尔状态选项：这类选项只能关闭（`[ ]`）或作为内建组件开启（`[*]`）。
- 三态选项：这类选项可以关闭（`< >`）、内建（`<*>`），或作为可加载模块（`<M>`）进行构建。

想要了解更多关于某个选项的信息，使用上/下箭头键导航至该选项，然后按 `<TAB>` 键，直至底部的 `< Help >` 选项被选中，然后按回车键进行选择。此时就会显示关于该配置选项的帮助信息。

在修改选项时请务必谨慎。

当你满意配置后，按 `<TAB>` 键直到底部的 `< Save >` 选项被选中。然后按回车键进行选择。然后再次按回车键（**记住，此时不要更改文件名**），就能将更新后的配置保存到 `.config` 文件中。

2.4 推荐做的选项

1. 关闭签名

```
./scripts/config --file .config --set-str SYSTEM_TRUSTED_KEYS ''

./scripts/config --file .config --set-str SYSTEM_REVOCATION_KEYS ''
```

2. 禁用debug

   CONFIG_DEBUG_INFO 改成 n在菜单中找到 `Kernel hacking`，然后找到 `Compile-time checks and compiler options`，在这里可以找到 `Compile the kernel with debug info` 选项，将其设置为 `n` 即可禁用调试信息。 这可以极大缩短编译时间，缩小\modules的体积，尤其是各种ko

   #或者 不用上面的方法，编译完成后采用

```
sudo make INSTALL_MOD_STRIP=1 modules_install

sudo make INSTALL_MOD_STRIP=1 install
```

0x02 手动编译内核

构建 Linux 内核实际上十分简单。然而，在开始构建之前，让我们为自定义内核构建添加一个标签。我将使用字符串 `-pratham` 作为标签，并利用 `LOCALVERSION` 变量来实施。你可以使用以下命令实现配置：

```
./scripts/config --file .config --set-str LOCALVERSION "-pratham"
```

这一命令将 `.config` 文件中的 `CONFIG_LOCALVERSION` 配置选项设为我在结尾指定的字符串，即 `-pratham`。当然，你也不必非得使用我所用的名字哦 😉

`LOCALVERSION` 选项可用于设置一个“本地”版本，它会被附加到通常的 `x.y.z` 版本方案之后，并在你运行 `uname -r` 命令时一并显示。

由于我正在构建的是 6.5.5 版本内核，而 `LOCALVERSION` 字符串被设为 `-pratham`，因此，对我来说，最后的版本名将会是 `6.5.5-pratham`。这么做的目的是确保我所构建的自定义内核不会与发行版所提供的内核产生冲突。

1.内核编译

接下来，我们来真正地构建内核。可以用以下的命令完成此步骤：

```
make -j$(nproc) 2>&1 | tee log
```

这对大部分（99%）用户来说已经足够了。

其中的 `-j` 选项用于指定并行编译任务的数量。而 `nproc` 命令用于返回可用处理单位（包括线程）的数量。因此，`-j$(nproc)` 其实意味着“使用我拥有的 CPU 线程数相同数量的并行编译任务”。

`2>&1` 会将 STDOUT 和 STDIN 重定向到相同的文件描述符，并通过管道传输给 `tee` 命令，这会将输出存储在一个名为 `log` 的文件，并且在控制台打印出完全相同的文本。如果你在构建时遇到错误，并希望回顾日志来检查出了什么问题，这将会十分有用。遇到那种情况，你只需要简单执行 `grep Error log` 命令就能找到线索。

2.安装

```
sudo make modules_install -j$(nproc)

sudo make headers_install
#我建议还是装，以后的化要编译新东西还用的到

sudo make install
```

之后取改grub从新编译的内核启动即可，grub2的版本决定来能不能load起来initramfs

3.删除

先启用了别的内核以后，再删除自己编译的东西按下面的步骤来

```
sudo rm -rf /lib/modules/6.1.57-ivo-6157
sudo rm -rf /boot/config-6.1.57-ivo-6157
sudo rm -rf /boot/initrd.img-6.1.57-ivo-6157
sudo rm -rf /boot/System.map-6.1.57-ivo-6157
sudo rm -/boot/vmlinuz-6.1.57-ivo-6157
sudo update-grub2
#重新initramfs
```

4.另外的玩法做成deb

.config 设置以后

```
make -j$(nproc) bindeb-pkg 2>&1 | tee log
#生成deb来安装
sudo dpkg -i ../linux-*.deb

#卸载的话按deb包卸载
sudo apt purge linux-image-6.1.57-ivo-6157
sudo apt purge linux-headers-6.1.57-ivo-6157
sudo apt purge linux-libc-dev
```

0x03 精简命令版

```
sudo apt install vim kmod cpio build-essential bc binutils bison dwarves flex gcc git gnupg2 gzip libelf-dev libncurses5-dev libssl-dev make openssl perl-base rsync tar xz-utils zstd

cd linux-*/

cp /boot/config-"$(uname -r)" .config

make olddefconfig

make nconfig
# CONFIG_DEBUG_INFO 改成
#在菜单中找到 `Kernel hacking`，然后找到 `Compile-time checks and compiler options`，在这里可以找到 `Compile the kernel with debug info` 选项，将其设置为 `n` 即可禁用调试信息。这可以极大缩短编译时间，缩小modules的体积，尤其是各种ko

#或者 不用上面的方法，编译完成后采用
# sudo make INSTALL_MOD_STRIP=1 modules_install
# sudo make INSTALL_MOD_STRIP=1 install

./scripts/config --file .config --disable CONFIG_DEBUG_INFO

./scripts/config --file .config --enable DEBUG_INFO_NONE
#6.x内核没有上面的那个了，要换一下，最好还是进去看手动关最好
./scripts/config --file .config --set-str SYSTEM_TRUSTED_KEYS ''

./scripts/config --file .config --set-str SYSTEM_REVOCATION_KEYS ''

./scripts/config --file .config --set-str LOCALVERSION "-ivo"

make -j$(nproc) bindeb-pkg 2>&1 | tee log

sudo dpkg -i ../linux-*.deb



#安装之后一般都是第一项，重启查看
sudo reboot
uname -a
```

0x04疑难处理

16.04 编译6.x版本内核

需要升级gcc才可以，虽然是说可以用5.x版本gcc编译，实际语法会有错误

```
sudo apt-get install build-essential software-properties-common -y
sudo add-apt-repository ppa:ubuntu-toolchain-r/test -y
sudo apt-get update -y
sudo apt-get install gcc-9 g++-9 -y
sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-9 60 --slave
/usr/bin/g++ g++ /usr/bin/g++-9
sudo update-alternatives --config gcc
```

之后再去编译即可

本文有参考的链接

技术|Linux 内核动手编译实用指南

https://zhuanlan.zhihu.com/p/461516677