Compile and boot a Linux Kernel

Anthony McGlone

October 20, 2022

Table of Contents

Chapter 1: Introduction **2**
1.1 Installing Oracle VirtualBox 2

Chapter 2: Ubuntu **3**
2.1 Prerequisites 3
2.2 Download the latest Linux kernel 3
2.3 Extract the source code 3
2.4 Configure the kernel 4
2.5 Compile the kernel 5
2.6 Install the kernel 5

Chapter 1 Introduction

This guide will demonstrate the basic process of compiling and booting a Linux kernel. More advanced kernel configuration (such as enabling device drivers or new kernel modules) will not be covered.

If you're already using a Linux operating system (such as Ubuntu, Red Hat or CentOS), you can skip ahead to the kernel installation instructions for your OS.

If you're on Windows or Mac, you can use Oracle's VirtualBox to test an installation.

1.1 Installing Oracle VirtualBox

1. Read the installation instructions for your OS.
2. Download the VirtualBox binary / package for your OS.
3. Download an ISO of a Linux OS (e.g. Ubuntu, CentOS, Fedora).
4. Create a virtual machine using the ISO / installation media.

Chapter 2 Ubuntu

2.1 Prerequisites

In your Linux OS or VM, open a terminal. Install the GCC compiler and the other tools required to compile the kernel:

```
sudo apt-get install build-essential libncurses-dev bison flex libssl-dev libelf-dev
```

2.2 Download the latest Linux kernel

Open the kernel home page by navigating to this website. Download the latest source code by clicking on the large yellow button.

A `tar.xz` file should be downloaded to the `Downloads` folder (e.g. `linux-6.0.2.tar.xz`).

2.3 Extract the source code

Navigate to your `Downloads` folder. In the `unxz` command, replace `<tar.xz file>` with the downloaded file. Run the command to extract the `tar` file:

```
unxz -v <tar.xz file>
```

The PGP signature for the `tar` file should be verified before its contents are extracted. On the kernel home page, find the table row corresponding to the latest kernel version and copy the link address from the `[pgp]` link. In the `wget` command, replace `<pgp link address>` with this link. Then run the command to download the `tar.sign` file:

```
wget <pgp link address>
```

In the `gpg` command, replace `<tar sign file>` with the downloaded `tar.sign` file. Then run it:

```
gpg --verify <tar sign file>
```

The output from the command contains the RSA key, which will be used in the verification of the `tar.sign` file's signature:

```
gpg:  assuming signed data in 'linux-6.0.2.tar'
gpg:  Signature made Sat 15 Oct 2022 07:04:02 IST
gpg:  using RSA key 647F28654894E3BD457199BE38DBBDC86092693E
gpg:  Can't check signature:  No public key
```

Replace `<RSA key>` in the following command with the `RSA key` from **your** terminal output. Then run the following command:

```
gpg --recv-keys --keyserver hkps://keyserver.ubuntu.com <RSA key>
```

This command should download a public key that will be used in the verification. The output will look like this:

```
gpg:  key 38DBBDC86092693E: 1 duplicate signature removed
gpg:  key 38DBBDC86092693E: public key "Greg Kroah-Hartman <gregkh@linuxfoundation.org>" imported
gpg:  Total number processed:  1
gpg:  imported:  1
```

Re-run the `--verify` command to complete the signature verification:

```
gpg --verify <tar sign file>
```

If successful, the output should look like this:

```
gpg:  assuming signed data in 'linux-6.0.2.tar'
gpg:  Signature made Sat 15 Oct 2022 07:04:02 IST
gpg:  using RSA key 647F28654894E3BD457199BE38DBBDC86092693E
gpg:  Good signature from "Greg Kroah-Hartman <gregkh@linuxfoundation.org>"
gpg:  aka "Greg Kroah-Hartman <gregkh@kernel.org>" [unknown]
gpg:  aka "Greg Kroah-Hartman (Linux kernel stable release signing key)
gpg:  WARNING: This key is not certified with a trusted signature!
gpg:  There is no indication that the signature belongs to the owner.
Primary key fingerprint:  647F 2865 4894 E3BD 4571 99BE 38DB BDC8 6092 693E
```

Extract the Linux kernel code using the `tar` command:

```
tar xvf <the tar file>
```

There should now be a folder with the extracted source code (e.g. `linux-6.0.2/`)

2.4 Configure the kernel

`cd` into the extracted folder. Before compiling the kernel, we must choose the modules that we wish to include. This can be done in two steps. First, copy the modules from the existing kernel configuration to the new configuration file:

```
cp -v /boot/config-$(uname -r) .config
```

The next step is to run `make menuconfig`. This will load a graphical dialog (see Figure 2.1). The dialog can be used to add or remove modules from the config (we will not edit the config at this time). A quick note on the dialog controls. Use the up/down arrow keys to navigate vertically. Use the Enter/return key to select a submenu in the dialog. Typing `Y` will include a feature. Typing `N` will exclude it. Use the left/right arrow keys to navigate to `Exit` when you want exit a submenu or the dialog.

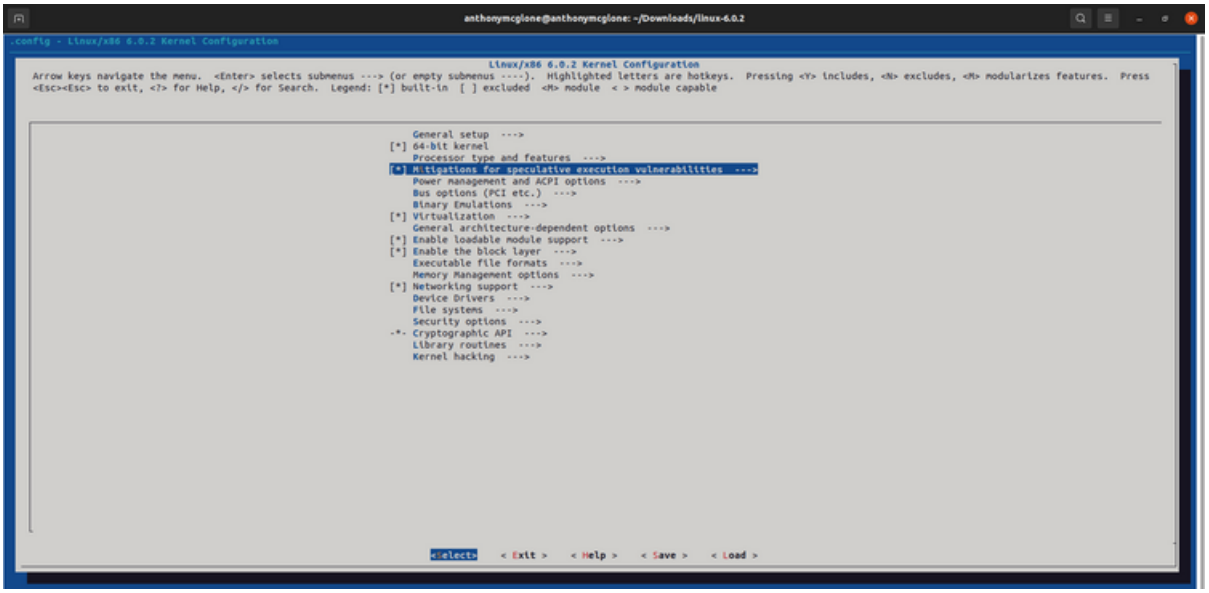

Figure 2.1: Linux kernel configuration dialog

Navigate around the dialog to examine the submenus. Then exit the dialog.

2.5 Compile the kernel

The kernel compilation process can take some time to run. To compile the kernel, use the `make` command. Specify the number of cores to use in the compilation with the `-j` option. The following command will run the compilation with two cores:

```
make -j 2
```

When the compilation is finished, the console will be returned to the user.

2.6 Install the kernel

To prepare for installation, install the kernel modules:

```
sudo make modules_install
```

Once the command prompt is returned, run the kernel installation command:

```
sudo make install
```

When the install command is finished, run `ls -l /boot`. Three files should be updated (with `X.X.X` corresponding to the version number of the kernel):

```
initramfs-X.X.X.img    System.map-X.X.X    vmlinuz-X.X.X
```