Compiling a Kernel

LPIC-2: Linux Engineer (201-450)

Objectives:

At the end of this episode, I will be able to:

1. Describe the scenarios where compiling a custom kernel would be used.
2. Customize the kernel configuration and compile a custom kernel.

> Additional resources used during the episode can be obtained using the download link on the overview episode.

- Reasons for building a custom kernel
  - Loading required modules
  - Removing unnecessary modules
  - High security code review
- Pre-requisites
  - `sudo edit /etc/apt/sources.list`
    - `deb-src http://archive.ubuntu.com/ubuntu focal main restricted`
    - `deb-src http://archive.ubuntu.com/ubuntu focal-updates main restricted`
  - `sudo apt-get build-dep linux linux-image-$(uname -r)`
  - `sudo apt-get install build-essential libncurses5-dev gcc libssl-dev grub2 bc bison flex libelf-dev fakeroot`
- Install Source
  - From repo
    - `sudo apt-get source linux-image-unsigned-$(uname -r)`
  - From kernel.org
    - `cd /usr/src`
    - `wget http://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.9.16.tar.xz`
    - `tar -xvf linux-5.9.16.tar.xz`
- Create a configuration file
  - Manually edit the text file
    - `sudoedit /usr/src/linux/.config`
  - Import an existing configuration
    - `cp /boot/config-5.4.0-67-generic /usr/src/linux-5.9.16/.config`
- Modify the configuration
  - Use a default configuration file
    - `sudo make defconfig`
  - Re-use a config from a previous build
    - `sudo make oldconfig`
  - Use a text-based UI to build a config

        - `sudo make menuconfig`
    - Use a graphical UI to build a config
        - `sudo make xconfig`
- Compiling the kernel
    - `sudo make -j2 deb-pkg`
        - `-j` defines how many cores to use
    - Takes a long time
    - 2 1/2 hours in a VM
    - 1 hour on hardware
- Install the new kernel
    - Automatically as a package
        - `dpkg -i linux-*.deb`
    - Manually as a file
        - A `vmlinux` file is created
        - Copy it to `/boot` and point to it in GRUB
        - Or simlink it
- Kernel portability
    - Kernels can be copied between systems
    - Assuming similar hardware
- *initrd* and *initramfs*
    - Contain files used by the kernel during boot up
    - Required for things like RAID controller drivers
    - Updated automatically
    - Can be forced with *mkinitrd* and *mkinitramfs*
- Compiling Kernel Modules
    - A lot of times you can avoid building a custom kernel
    - Build a custom module instead
    - DKMS
        - Dynamic Kernel Module Support
        - Allows compiling dynamic modules
        - `dkms build -m <module_name> -v <#>`
        - `dkms install -m <module_name> -v <#>`