Assignment - 1

Report

CS3003D: Operating Systems

Hanna Nechikkadan
B190420CS
CSE - B

NIT Calicut
05-09-2021

Contents

Problem Statement

Download the latest stable Linux kernel from kernel.org, compile it, and dual boot it with your current Linux version. Your current version as well as the new version should be present in the grub-menu.

Methodology

- Extract the latest kernel source from kernel.org.
- Install the development dependencies.
- Configure the kernel for building.
- Compile the kernel.
- Install the kernel modules and the compiled kernel.
- Update the grub configuration.
- Reboot the system and boot to the latest kernel.

Explanation

Introduction

A kernel is the most important software in an operating system. It controls everything in the system and it is that part of the operating system which always stays in the memory and mediates the interaction between the hardware and software elements of the system. It is responsible for the core functions of operating systems like disk management, memory management, etc.

Linux kernel, unlike many other operating systems, is open-source .i.e., anyone can access the kernel source code and use and configure it according to their requirements and distribute it adhering to the rules of GNU general public license, version 3. The linux kernel source code is mostly written using C programming language

in GNU extensions of GCC.This produces a highly optimized executable with respect to utilization of memory space and task execution times.

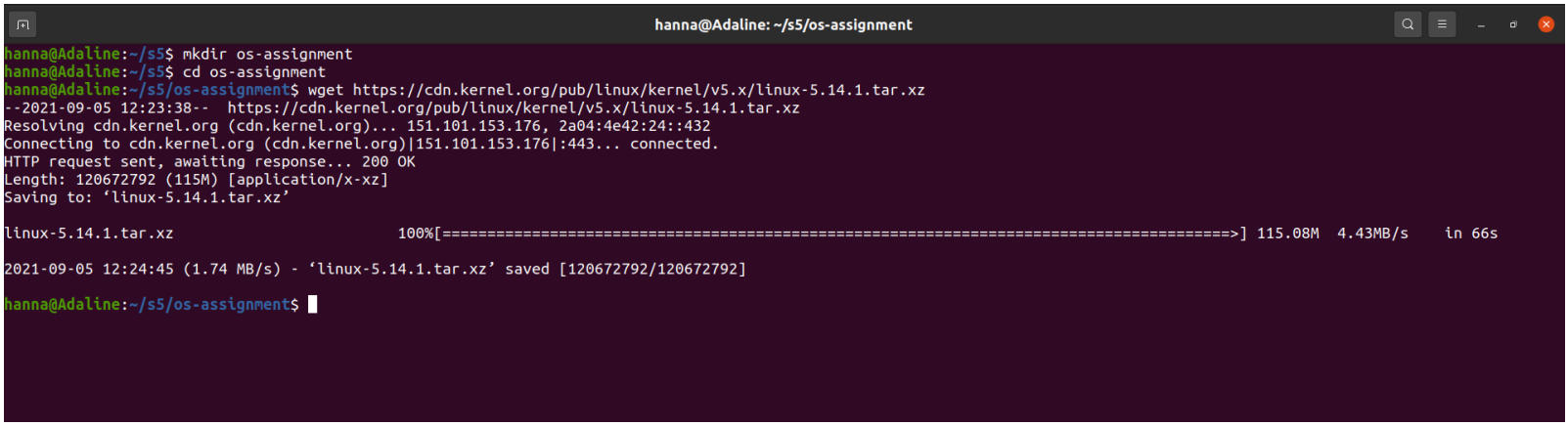
Commands to Build the kernel

1. Create a directory.
    a. `mkdir os-assignment`
    b. `cd os-assignment`

These commands create a directory named ‘os-assignment’ and the current directory is set as the newly created directory. So, we can download, compile and install the required source codes and modules at one place.

2. Download the kernel source code.
    a. `wget https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.6.9.tar.xz`

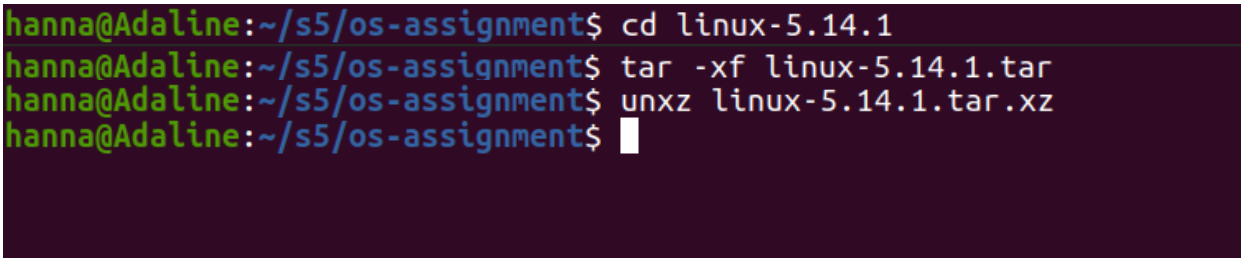
This command will download the kernel code of the specified kernel version (here 5.14.1) as a tarball file from kernel.org into the current directory. The file size is around 120mb.



3. Extract the source codes from the tarball file.

```
a. unxz linux-5.14.1.tar.xz
b. tar -xf linux-5.14.1.tar
c. cd linux-5.14.1
```

The first command will decompress the file and extract the kernel tarball file. The second command will extract the kernel source codes from the tarball file into the directory linux-5.14.1. The third command will set the current working directory to the directory containing extracted the kernel source codes.



4. Install the dependencies.

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

Before we move on to building the kernel, we have to ensure that all the required dependencies for compiling the kernel source codes are installed in our system. So, this command installs the GNU/GCC compiler and its related tools like make, etc. Here, the build-essential package contains a list of programs which are considered essential for building Debian packages. It consists of the following packages.

- libc6-dev – C standard library.
- gcc – C compiler.
- g++ – C++ compiler.
- make – GNU make utility to maintain groups of programs.
- dpkg-dev – Debian package development tools.

```
hanna@Adaline:~/s5/os-assignment/linux-5.14.1$ sudo apt-get install build-essential libncurses-dev bison flex libssl-dev libelf-dev
[sudo] password for hanna:
Reading package lists... Done
Building dependency tree
Reading state information... Done
bison is already the newest version (2:3.5.1+dfsg-1).
flex is already the newest version (2.6.4-6.2).
libncurses-dev is already the newest version (6.2-0ubuntu2).
libncurses-dev set to manually installed.
build-essential is already the newest version (12.8ubuntu1.1).
libssl-dev is already the newest version (1.1.1f-1ubuntu2.8).
The following packages were automatically installed and are no longer required:
  gconf-service gconf-service-backend gconf2-common libappindicator1 libc++1 libc++1-10 libc++abi1-10 libdbusmenu-gtk4 libfprint-2-tod1 libgconf-2-4 libgcrypt20-dev
  libgpg-error-dev libllvm10 libllvm11 libmbedtls-dev libssh2-1-dev shim
Use 'sudo apt autoremove' to remove them.
The following NEW packages will be installed
  libelf-dev
0 to upgrade, 1 to newly install, 0 to remove and 2 not to upgrade.
Need to get 57.0 kB of archives.
After this operation, 382 kB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 http://in.archive.ubuntu.com/ubuntu focal/main amd64 libelf-dev amd64 0.176-1.1build1 [57.0 kB]
Fetched 57.0 kB in 1s (44.1 kB/s)
Selecting previously unselected package libelf-dev:amd64.
(Reading database ... 199129 files and directories currently installed.)
Preparing to unpack .../libelf-dev_0.176-1.1build1_amd64.deb ...
Unpacking libelf-dev:amd64 (0.176-1.1build1) ...
Setting up libelf-dev:amd64 (0.176-1.1build1) ...
hanna@Adaline:~/s5/os-assignment/linux-5.14.1$
```

5. Configure the kernel.

    a. `cp -v /boot/config-$(uname -r) .config`
    b. `make olddefconfig`

Configuring the kernel is the most crucial step as it involves specifying the drivers and customizing the features of the kernel. I have used my current kernel's same configurations for the new kernel as well. The first command copies the existing config file to the .config file in the current directory. Here the command `` `uname -r` `` is a nested command. This command specifies the current kernel version, and thus the main command copies the config file of the current kernel.

Now, to set the new config symbols in the new kernel version which may not be present in the existing kernel, we have to use another command like `` `make menuconfig` ``, which allows us to customize every configuration. Here, I have used `` `make olddefconfig` ``. This command sets everything as the existing configurations and sets the new symbols to default. The disadvantage of using this command is that it may include the modules and features we simply don't need. If one wants to configure everything afresh , we can use the command `` `make config` `` , but this could turn tiresome as we have to specify yes/no for enabling or disabling hundreds of modules. There are several other config commands that we could use according to our requirement.

```
hanna@Adaline:~/s5/os-assignment/linux-5.14.1$ cp -v /boot/config-$(uname -r) .config
'/boot/config-5.11.0-27-generic' -> '.config'
hanna@Adaline:~/s5/os-assignment/linux-5.14.1$
```

```
hanna@Adaline:~/s5/os-assignment/linux-5.14.1$ make olddefconfig
  HOSTCC  scripts/kconfig/conf.o
  HOSTCC  scripts/kconfig/confdata.o
  HOSTCC  scripts/kconfig/expr.o
  LEX     scripts/kconfig/lexer.lex.c
  YACC    scripts/kconfig/parser.tab.[ch]
  HOSTCC  scripts/kconfig/lexer.lex.o
  HOSTCC  scripts/kconfig/menu.o
  HOSTCC  scripts/kconfig/parser.tab.o
  HOSTCC  scripts/kconfig/preprocess.o
  HOSTCC  scripts/kconfig/symbol.o
  HOSTCC  scripts/kconfig/util.o
  HOSTLD  scripts/kconfig/conf
.config:2598:warning: symbol value 'm' invalid for PVPANIC
.config:8618:warning: symbol value 'm' invalid for ASHMEM
.config:9653:warning: symbol value 'm' invalid for ANDROID_BINDER_IPC
.config:9654:warning: symbol value 'm' invalid for ANDROID_BINDERFS
#
# configuration written to .config
#
hanna@Adaline:~/s5/os-assignment/linux-5.14.1$
```

6. Compile the kernel.
    a. `make -j8`

This command will compile the kernel source codes based on the configurations we specified. Here, the number 8 describes the no. of threads, that is parallel instances of GCC. The greater the number, the faster the compilation. The maximum number that we could specify depends on the number of processing units available in the system. We can see the no. of processing units available by using the command `` `nproc` ``. I have used 8 here because 8 processing units are available in my system. This process took around 1 hour for me. After compilation, the directory size has increased to around 12 GB.

```
#
hanna@Adaline:~/s5/os-assignment/linux-5.14.1$ make -j8
  LD [M]  sound/usb/line6/snd-usb-pod.ko
  LD [M]  sound/usb/line6/snd-usb-podhd.ko
  LD [M]  sound/usb/line6/snd-usb-toneport.ko
  LD [M]  sound/usb/line6/snd-usb-variax.ko
  LD [M]  sound/usb/misc/snd-ua101.ko
  LD [M]  sound/usb/snd-usb-audio.ko
  LD [M]  sound/usb/snd-usbmidi-lib.ko
  LD [M]  sound/usb/usx2y/snd-usb-us122l.ko
  LD [M]  sound/usb/usx2y/snd-usb-usx2y.ko
  LD [M]  sound/x86/snd-hdmi-lpe-audio.ko
  LD [M]  sound/xen/snd_xen_front.ko
```

This is a small part of the compilation process output which is significantly large.

7. Install the kernel modules.
    a. `sudo make modules_install`

In this step, we install the modules and drivers which were built in the previous step into the root filesystem for the new kernel. Here we have to run the command with root privileges as a root user and hence, the word `sudo` is added with the command.

```
  LD [M]  sound/xen/snd_xen_front.ko
hanna@Adaline:~/s5/os-assignment/linux-5.14.1$ sudo make modules_install
```

```
hanna@Adaline: ~/s5/os-assignment/linux-5.14.1
  SIGN    /lib/modules/5.14.1/kernel/sound/usb/line6/snd-usb-toneport.ko
  INSTALL /lib/modules/5.14.1/kernel/sound/usb/line6/snd-usb-variax.ko
  SIGN    /lib/modules/5.14.1/kernel/sound/usb/line6/snd-usb-variax.ko
  INSTALL /lib/modules/5.14.1/kernel/sound/usb/misc/snd-ua101.ko
  SIGN    /lib/modules/5.14.1/kernel/sound/usb/misc/snd-ua101.ko
  INSTALL /lib/modules/5.14.1/kernel/sound/usb/snd-usb-audio.ko
  SIGN    /lib/modules/5.14.1/kernel/sound/usb/snd-usb-audio.ko
  INSTALL /lib/modules/5.14.1/kernel/sound/usb/snd-usbmidi-lib.ko
  SIGN    /lib/modules/5.14.1/kernel/sound/usb/snd-usbmidi-lib.ko
  INSTALL /lib/modules/5.14.1/kernel/sound/usb/usx2y/snd-usb-us122l.ko
  SIGN    /lib/modules/5.14.1/kernel/sound/usb/usx2y/snd-usb-us122l.ko
  INSTALL /lib/modules/5.14.1/kernel/sound/usb/usx2y/snd-usb-usx2y.ko
  SIGN    /lib/modules/5.14.1/kernel/sound/usb/usx2y/snd-usb-usx2y.ko
  INSTALL /lib/modules/5.14.1/kernel/sound/x86/snd-hdmi-lpe-audio.ko
  SIGN    /lib/modules/5.14.1/kernel/sound/x86/snd-hdmi-lpe-audio.ko
  INSTALL /lib/modules/5.14.1/kernel/sound/xen/snd_xen_front.ko
  SIGN    /lib/modules/5.14.1/kernel/sound/xen/snd_xen_front.ko
  DEPMOD  /lib/modules/5.14.1
```

Screenshot of a part of the output

8. Install the kernel.
    a. `sudo make install`

As of now, we have compiled the kernel and installed kernel modules. Now, we can install the kernel itself. This command also had to be run with root privileges.

```
hanna@Adaline:~/s5/os-assignment/linux-5.14.1$ sudo make install
arch/x86/Makefile:148: CONFIG_X86_X32 enabled but no binutils support
sh ./arch/x86/boot/install.sh \
        5.14.1 arch/x86/boot/bzImage \
        System.map "/boot"
run-parts: executing /etc/kernel/postinst.d/apt-auto-removal 5.14.1 /boot/vmlinuz-5.14.1
run-parts: executing /etc/kernel/postinst.d/initramfs-tools 5.14.1 /boot/vmlinuz-5.14.1
update-initramfs: Generating /boot/initrd.img-5.14.1
run-parts: executing /etc/kernel/postinst.d/unattended-upgrades 5.14.1 /boot/vmlinuz-5.14.1
run-parts: executing /etc/kernel/postinst.d/update-notifier 5.14.1 /boot/vmlinuz-5.14.1
run-parts: executing /etc/kernel/postinst.d/xx-update-initrd-links 5.14.1 /boot/vmlinuz-5.14.1
I: /boot/vmlinuz.old is now a symlink to vmlinuz-5.11.0-27-generic
run-parts: executing /etc/kernel/postinst.d/zz-update-grub 5.14.1 /boot/vmlinuz-5.14.1
Sourcing file `/etc/default/grub'
Sourcing file `/etc/default/grub.d/init-select.cfg'
Generating grub configuration file ...
Found linux image: /boot/vmlinuz-5.14.1
Found initrd image: /boot/initrd.img-5.14.1
Found linux image: /boot/vmlinuz-5.11.0-27-generic
Found initrd image: /boot/initrd.img-5.11.0-27-generic
Found linux image: /boot/vmlinuz-5.11.0-25-generic
Found initrd image: /boot/initrd.img-5.11.0-25-generic
Found Windows Boot Manager on /dev/nvme0n1p1@/EFI/Microsoft/Boot/bootmgfw.efi
Adding boot menu entry for UEFI Firmware Settings
done
hanna@Adaline:~/s5/os-assignment/linux-5.14.1$
```

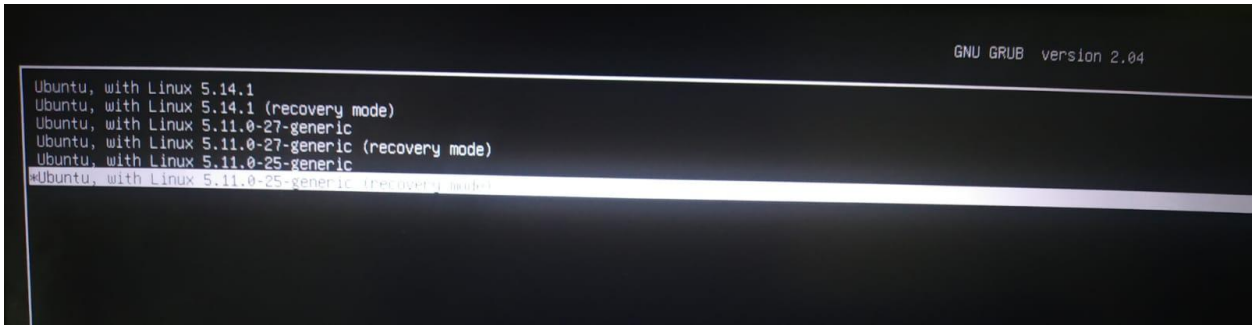
The command also creates new initramfs for our kernel. The initramfs is a gzipped cpio archive which will be unpacked by the kernel into RAM disk at the boot time and uses it as the initial root filesystem. So, the next time we boot up, the new kernel will be used.

It also updates the grub configuration with the new kernel details, so that we can see the new kernel along with previously installed kernels in the grub menu.

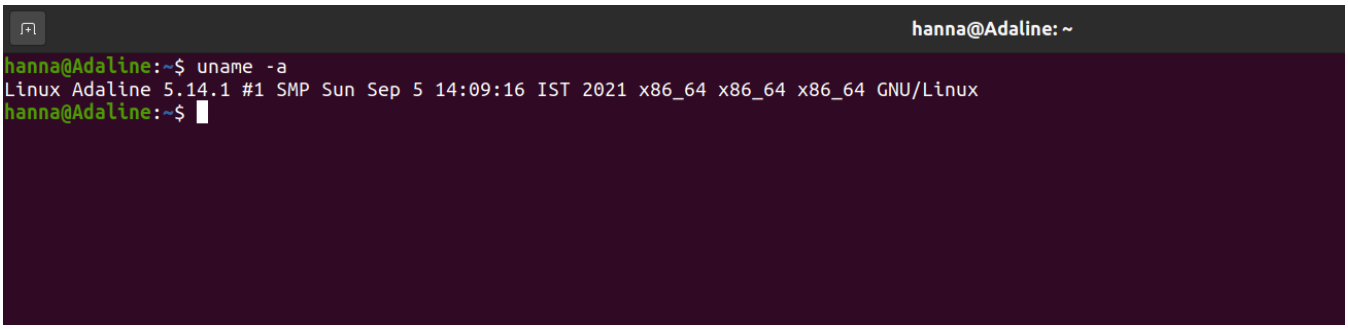
By now, we have successfully installed the kernel and we can boot to the new kernel by using the command `sudo reboot`.

Conclusion

The compilation and installation of the latest linux kernel was successfully completed. And now, we can easily boot to the newer kernel. As of today, the latest Linux kernel version published in kernel.org is the version 5.14.1. which is what I have used to install. My existing kernel is of version 5.11.0-27-generic and my operating system is Ubuntu.



In the grub menu, we can see the first entry is the new kernel entry.



After booting to the new kernel, we can see the current kernel and operating system details by running the command `uname -a`.

References

- https://www.cyberciti.biz/tips/compiling-linux-kernel-26.html
- https://wiki.archlinux.org/title/Kernel/Traditional_compilation#Kernel_configuration
- https://www.kernel.org/doc/html/latest/admin-guide/README.html
- https://www.freecodecamp.org/news/building-and-installing-the-latest-linux-kernel-from-source-6d8df5345980/
- https://wiki.linuxquestions.org/wiki/How_to_build_and_install_your_own_Linux_kernel#The_build_step_by_step
- https://www.itdev.co.uk/blog/building-linux-kernel-introduction
- https://en.wikipedia.org/wiki/Linux_kernel
- https://www.systutorials.com/docs/linux/man/