

CS3003D: Operating Systems

Assignment – 1

REPORT

Name: TOM SAJU | Roll No: B191290CS

- **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.

- **Report Overview**

The Linux® kernel is **the main component of a Linux operating system (OS)** and is the core interface between a computer's hardware and its processes. It communicates between the two, managing resources as efficiently as possible.

- **What exactly is a “Kernel”?**

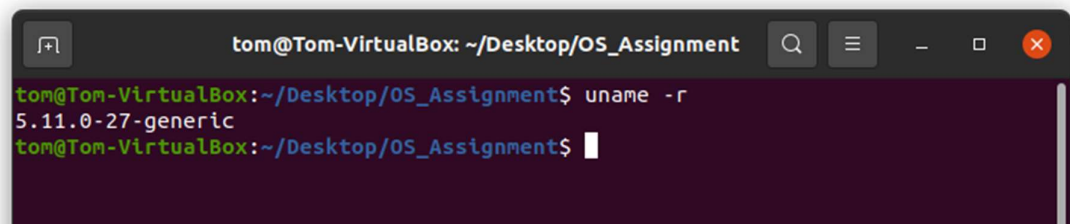
The Kernel is a computer program at the core of a computer's operating system with complete control over everything in the system. It is an integral part of any operating system. It is the "portion of the operating system code that is always resident in memory", and facilitates interactions between hardware and software components. The Linux kernel is open-source. Meaning anyone can view, modify and publish its source code.

- Methodology & Explanation

We need a Linux machine to compile a Linux kernel. For this assignment I used Ubuntu 20.04 as my machine.

Step 1: Boot machine and check current kernel version


Boot your linux machine and type ***uname -r*** on the terminal to check the current version of kernel installed in the system.



```
tom@Tom-VirtualBox: ~/Desktop/OS_Assignment
tom@Tom-VirtualBox:~/Desktop/OS_Assignment$ uname -r
5.11.0-27-generic
tom@Tom-VirtualBox:~/Desktop/OS_Assignment$
```

Step 2: Downloading the kernel

Open a browser and download the kernel source code from [kernel.org](https://www.kernel.org) . Download the latest stable kernel by clicking on the tarball link. Save the kernel source in an empty folder. (The latest stable kernel as of now is the version 5.14.1)



The Linux Kernel Archives

About Contact us FAQ Releases Signatures Site news

Protocol Location

HTTP	https://www.kernel.org/pub/
GIT	https://git.kernel.org/
RSYNC	rsync://rsync.kernel.org/pub/

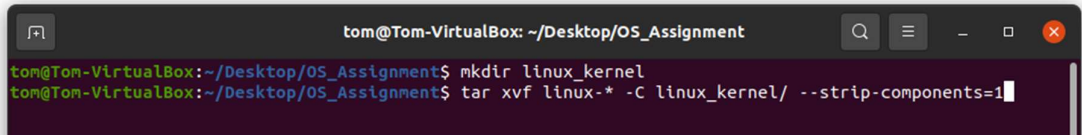
Latest Release

5.14.1

mainline:	5.14	2021-08-29	[tarball]	[pgp]	[patch]	[view diff]	[browse]
stable:	5.14.1	2021-09-03	[tarball]	[pgp]	[patch]	[view diff]	[browse] [changelog]
stable:	5.13.14	2021-09-03	[tarball]	[pgp]	[patch] [inc. patch]	[view diff] [browse]	[changelog]
longterm:	5.10.62	2021-09-03	[tarball]	[pgp]	[patch] [inc. patch]	[view diff] [browse]	[changelog]
longterm:	5.4.144	2021-09-03	[tarball]	[pgp]	[patch] [inc. patch]	[view diff] [browse]	[changelog]
longterm:	4.19.206	2021-09-03	[tarball]	[pgp]	[patch] [inc. patch]	[view diff] [browse]	[changelog]
longterm:	4.14.246	2021-09-03	[tarball]	[pgp]	[patch] [inc. patch]	[view diff] [browse]	[changelog]
longterm:	4.9.282	2021-09-03	[tarball]	[pgp]	[patch] [inc. patch]	[view diff] [browse]	[changelog]
longterm:	4.4.283	2021-09-03	[tarball]	[pgp]	[patch] [inc. patch]	[view diff] [browse]	[changelog]
linux-next:	next-20210903	2021-09-03					[browse]

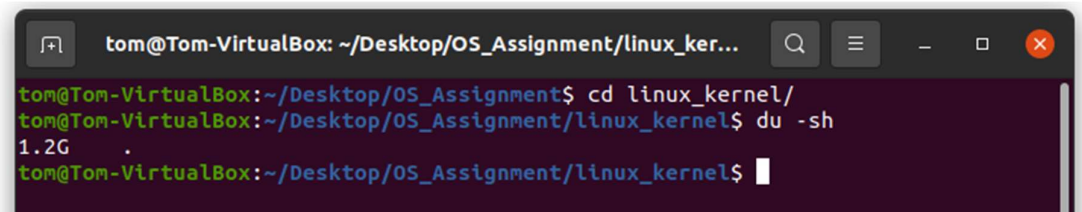
Step 3: Extracting the source

Open terminal in the folder where we downloaded the source code. Make a directory using command ***mkdir linux_kernel*** and then extract the source into the folder using the command ***tar xvf linux-* -C linux_kernel/ --strip-components=1*** .



```
tom@Tom-VirtualBox: ~/Desktop/OS_Assignment
tom@Tom-VirtualBox:~/Desktop/OS_Assignment$ mkdir linux_kernel
tom@Tom-VirtualBox:~/Desktop/OS_Assignment$ tar xvf linux-* -C linux_kernel/ --strip-components=1
```

Move to the `linux_kernel` directory by using the command ***cd linux_kernel/*** and the command ***du -sh*** returns the size of the extracted source code.

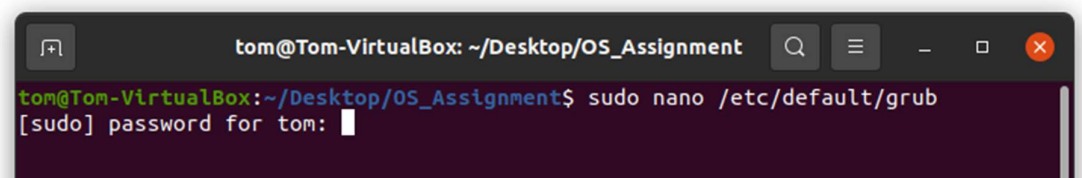


```
tom@Tom-VirtualBox: ~/Desktop/OS_Assignment/linux_ker...
tom@Tom-VirtualBox:~/Desktop/OS_Assignment$ cd linux_kernel/
tom@Tom-VirtualBox:~/Desktop/OS_Assignment/linux_kernel$ du -sh
1.2G  .
tom@Tom-VirtualBox:~/Desktop/OS_Assignment/linux_kernel$
```

Step 4: Increase timeout in our grub as a safety precaution

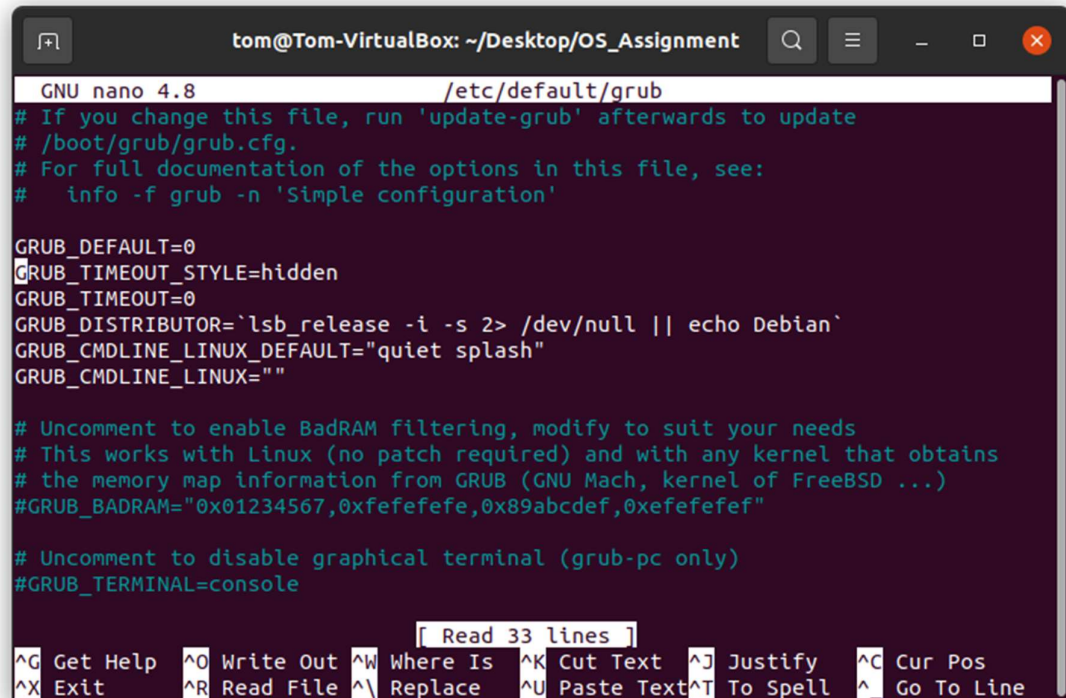
In case if our kernel installed fails to start then we need some time to select the stock kernel from the grub boot menu. For that we increase the time out in our grub.

To do that we should edit the grub config in nano by typing ***sudo nano /etc/default/grub*** .



```
tom@Tom-VirtualBox: ~/Desktop/OS_Assignment
tom@Tom-VirtualBox:~/Desktop/OS_Assignment$ sudo nano /etc/default/grub
[sudo] password for tom:
```

After entering the password, a text editor (nano) will pop up. Edit the **GRUB_TIMEOUT=0** to **GRUB_TIMEOUT=0** to give enough time to select the kernel. Save and exit from the editor by pressing **ctrl+o** and **ctrl+x** respectively.



```
tom@Tom-VirtualBox: ~/Desktop/OS_Assignment
GNU nano 4.8 /etc/default/grub
# If you change this file, run 'update-grub' afterwards to update
# /boot/grub/grub.cfg.
# For full documentation of the options in this file, see:
# info -f grub -n 'Simple configuration'

GRUB_DEFAULT=0
GRUB_TIMEOUT_STYLE=hidden
GRUB_TIMEOUT=0
GRUB_DISTRIBUTOR=`lsb_release -i -s 2> /dev/null || echo Debian`
GRUB_CMDLINE_LINUX_DEFAULT="quiet splash"
GRUB_CMDLINE_LINUX=""

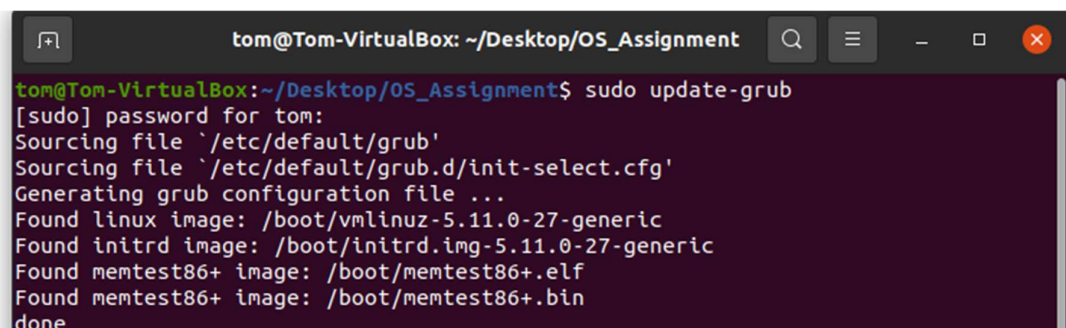
# Uncomment to enable BadRAM filtering, modify to suit your needs
# This works with Linux (no patch required) and with any kernel that obtains
# the memory map information from GRUB (GNU Mach, kernel of FreeBSD ...)
#GRUB_BADRAM="0x01234567,0xfefefefe,0x89abcdef,0xefefefef"

# Uncomment to disable graphical terminal (grub-pc only)
#GRUB_TERMINAL=console

[ Read 33 lines ]
^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify   ^C Cur Pos
^X Exit      ^R Read File ^\ Replace   ^U Paste Text ^T To Spell  ^_ Go To Line
```

Step 5: Re-build the initial RAM filesystem

Type command: **sudo update-grub** in the terminal to re-build the initial RAM filesystem so when we reboot after installing the compiled kernel the changes are applied to GRUB.

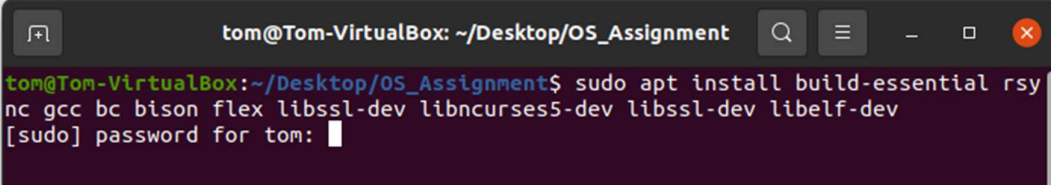


```
tom@Tom-VirtualBox: ~/Desktop/OS_Assignment
tom@Tom-VirtualBox:~/Desktop/OS_Assignment$ sudo update-grub
[sudo] password for tom:
Sourcing file `/etc/default/grub'
Sourcing file `/etc/default/grub.d/init-select.cfg'
Generating grub configuration file ...
Found linux image: /boot/vmlinuz-5.11.0-27-generic
Found initrd image: /boot/initrd.img-5.11.0-27-generic
Found memtest86+ image: /boot/memtest86+.elf
Found memtest86+ image: /boot/memtest86+.bin
done
```

Step 6: Installing the requirements

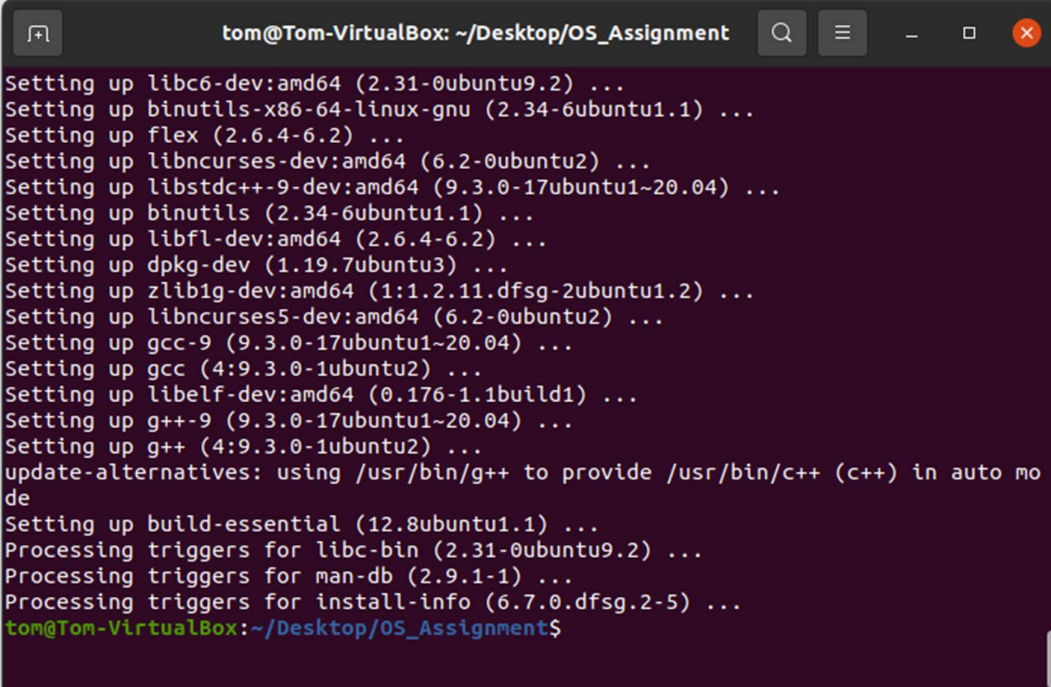
In order to compile the Linux kernel on ubuntu, we'll need to install dependency software packages . This can be done with a single command:

sudo apt install build-essential rsync gcc bc bison flex libssl-dev libncurses5-dev libelf-dev.



```
tom@Tom-VirtualBox: ~/Desktop/OS_Assignment
tom@Tom-VirtualBox:~/Desktop/OS_Assignment$ sudo apt install build-essential rsync gcc bc bison flex libssl-dev libncurses5-dev libelf-dev
[sudo] password for tom: 
```

Press the y key on the keyboard, if asked to continue.



```
tom@Tom-VirtualBox: ~/Desktop/OS_Assignment
Setting up libc6-dev:amd64 (2.31-0ubuntu9.2) ...
Setting up binutils-x86-64-linux-gnu (2.34-6ubuntu1.1) ...
Setting up flex (2.6.4-6.2) ...
Setting up libncurses-dev:amd64 (6.2-0ubuntu2) ...
Setting up libstdc++-9-dev:amd64 (9.3.0-17ubuntu1~20.04) ...
Setting up binutils (2.34-6ubuntu1.1) ...
Setting up libfl-dev:amd64 (2.6.4-6.2) ...
Setting up dpkg-dev (1.19.7ubuntu3) ...
Setting up zlib1g-dev:amd64 (1:1.2.11.dfsg-2ubuntu1.2) ...
Setting up libncurses5-dev:amd64 (6.2-0ubuntu2) ...
Setting up gcc-9 (9.3.0-17ubuntu1~20.04) ...
Setting up gcc (4:9.3.0-1ubuntu2) ...
Setting up libelf-dev:amd64 (0.176-1.1build1) ...
Setting up g++-9 (9.3.0-17ubuntu1~20.04) ...
Setting up g++ (4:9.3.0-1ubuntu2) ...
update-alternatives: using /usr/bin/g++ to provide /usr/bin/c++ (c++) in auto mode
Setting up build-essential (12.8ubuntu1.1) ...
Processing triggers for libc-bin (2.31-0ubuntu9.2) ...
Processing triggers for man-db (2.9.1-1) ...
Processing triggers for install-info (6.7.0.dfsg.2-5) ...
tom@Tom-VirtualBox:~/Desktop/OS_Assignment$
```

Build essential is a group of software's that install tools required for the compilation.

- **Rsync** - a data backup and transfer utility
- **GCC** is the GNU C compiler which compiles C code.
- **Bc** - an extension to GCC.
- **Bison** - a parser.
- **Flex** - a lexical analyser.
- **Libssl-dev** - a development package for OpenSSL.
- **Libncurses5-dev** - a development package for ncurses
- **Libelf-dev** - a library for reading elf files.

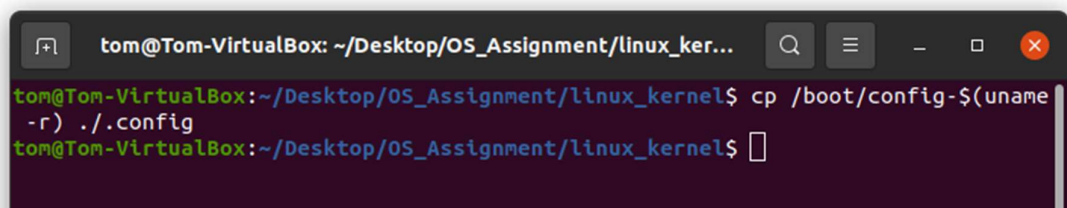
Step 7: Configuring the kernel

Before we actually compile the kernel, we must first configure which modules, components and drivers should be included in the compiled kernel. There are two ways to achieve this:

1. Configure the kernel compilation is to copy the config file that came with our distribution of Linux.
2. Generate a config file based on the currently connected hardware of your computer.

The method 1 is easy so we follow it. In this method we copy the existing kernel config file just in case you need to replace it in the event of a problem. To do that on an Ubuntu based system run the command:

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

A terminal window with a dark background. The title bar shows 'tom@Tom-VirtualBox: ~/Desktop/OS_Assignment/linux_ker...'. The prompt is 'tom@Tom-VirtualBox:~/Desktop/OS_Assignment/linux_kernel\$'. The command 'cp /boot/config-\$(uname -r) ./config' has been entered and executed. The prompt is now 'tom@Tom-VirtualBox:~/Desktop/OS_Assignment/linux_kernel\$' followed by a cursor.

```
tom@Tom-VirtualBox:~/Desktop/OS_Assignment/linux_kernel$ cp /boot/config-$(uname -r) ./config
tom@Tom-VirtualBox:~/Desktop/OS_Assignment/linux_kernel$
```


This will copy the config file of the currently running kernel that came from our distribution of Linux. The kernel from our Linux distribution comes a lot of drivers and modules. This is so that it can support wide range of hardware.

Step 8: Compiling and installing the kernel

To start the compilation process, run the command:

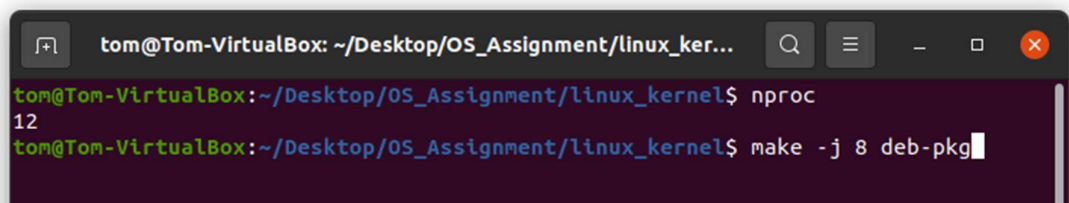
make deb-pkg

If we want to use multiple cores for the compilation, we use the command:

make -j x deb-pkg

where x is the number of cores you want to use for the compilation.

To check the number of core in your system give command ***nproc***. I am using 8/12 cores for the compilation so that it would compile faster than using single core.

A terminal window with a dark background and light green text. The window title is "tom@Tom-VirtualBox: ~/Desktop/OS_Assignment/linux_ker...". The prompt is "tom@Tom-VirtualBox:~/Desktop/OS_Assignment/linux_kernel\$". The first command entered is "nproc", which returns "12". The second command entered is "make -j 8 deb-pkg", which is partially visible with a cursor at the end.

```
tom@Tom-VirtualBox: ~/Desktop/OS_Assignment/linux_ker...
tom@Tom-VirtualBox:~/Desktop/OS_Assignment/linux_kernel$ nproc
12
tom@Tom-VirtualBox:~/Desktop/OS_Assignment/linux_kernel$ make -j 8 deb-pkg
```

The compilation process will take some time according to the number of processors allotted and efficiency of the system. The compilation should be successful if we don't get any errors.

```
tom@Tom-VirtualBox: ~/Desktop/OS_Assignment/linux_ker...
HDRINST usr/include/asm/unistd_64.h
HDRINST usr/include/asm/ioctl.h
HDRINST usr/include/asm/poll.h
HDRINST usr/include/asm/resource.h
HDRINST usr/include/asm/param.h
HDRINST usr/include/asm/socket.h
HDRINST usr/include/asm/ipcbuf.h
HDRINST usr/include/asm/errno.h
HDRINST usr/include/asm/unistd_32.h
INSTALL debian/linux-libc-dev/usr/include
dpkg-deb: building package 'linux-libc-dev' in '../linux-libc-dev_5.14.0-1_amd64.deb'.
dpkg-deb: building package 'linux-image-5.14.0' in '../linux-image-5.14.0_5.14.0-1_amd64.deb'.
dpkg-deb: building package 'linux-image-5.14.0-dbg' in '../linux-image-5.14.0-dbg_5.14.0-1_amd64.deb'.
dpkg-genbuildinfo
dpkg-genchanges >../linux-upstream_5.14.0-1_amd64.changes
dpkg-genchanges: info: including full source code in upload
dpkg-source -i.git --after-build .
dpkg-buildpackage: info: full upload (original source is included)
tom@Tom-VirtualBox:~/Desktop/OS_Assignment/linux_kernel$
```

Now we must get the deb packages in our parent directory. To do that move to the parent directory by command

`cd ..`

Now to install those packages run command:

`sudo dpkg -i linux-*.deb`

Now the installation for the newly compiled kernel will begin.

Once the installation is done, dpkg will automatically re-build the initial RAM filesystem.

```
tom@Tom-VirtualBox:~/Desktop/OS_Assignment$ sudo dpkg -i linux-*.deb
[sudo] password for tom:
```



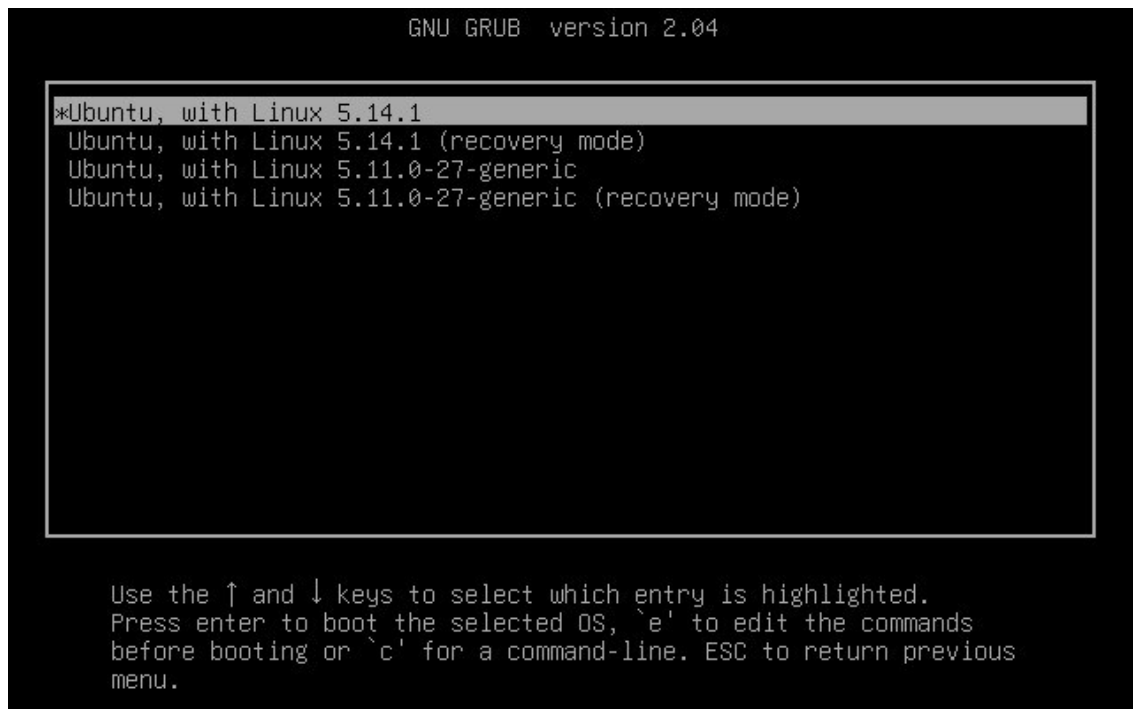
```
tom@Tom-VirtualBox: ~/Desktop/OS_Assignment
Selecting previously unselected package linux-image-5.14.0.
Preparing to unpack linux-image-5.14.0_5.14.0-1_amd64.deb ...
Unpacking linux-image-5.14.0 (5.14.0-1) ...
Selecting previously unselected package linux-image-5.14.0-dbg.
Preparing to unpack linux-image-5.14.0-dbg_5.14.0-1_amd64.deb ...
Unpacking linux-image-5.14.0-dbg (5.14.0-1) ...
Preparing to unpack linux-libc-dev_5.14.0-1_amd64.deb ...
Unpacking linux-libc-dev:amd64 (5.14.0-1) over (5.4.0-81.91) ...
Setting up linux-headers-5.14.0 (5.14.0-1) ...
Setting up linux-image-5.14.0 (5.14.0-1) ...
update-initramfs: Generating /boot/initrd.img-5.14.0
Sourcing file `/etc/default/grub'
Sourcing file `/etc/default/grub.d/init-select.cfg'
Generating grub configuration file ...
Found linux image: /boot/vmlinuz-5.14.0
Found initrd image: /boot/initrd.img-5.14.0
Found linux image: /boot/vmlinuz-5.11.0-27-generic
Found initrd image: /boot/initrd.img-5.11.0-27-generic
Found memtest86+ image: /boot/memtest86+.elf
Found memtest86+ image: /boot/memtest86+.bin
done
Setting up linux-image-5.14.0-dbg (5.14.0-1) ...
Setting up linux-libc-dev:amd64 (5.14.0-1) ...
tom@Tom-VirtualBox:~/Desktop/OS_Assignment$
```

Step 9: GRUB Menu

GRUB is a multi-boot bootloader for most of the linux distros. After successfully completing till step 7, you have to reboot the machine to check our results.

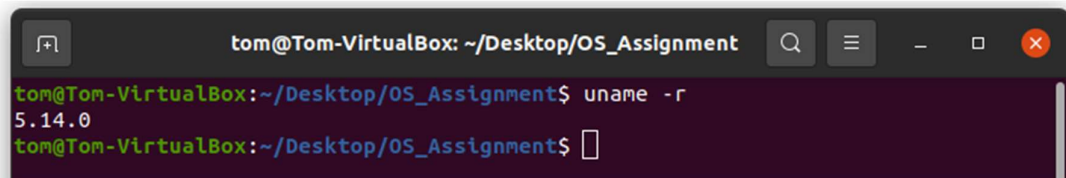
Reboot your machine and hold shift key while booting to access the GRUB bootloader menu.

Choose Advanced options for Ubuntu which will direct to a menu with the list of kernels available. Since we installed latest version, this will be the first option.



To show that the latest kernel is installed, run the command from the terminal:

uname -r



- Conclusion

We have successfully compiled our Linux kernel from source and successfully dual booted it with the current Linux version.

