

Operating Systems Assignment-1

Name: Vedurupaka Venkata Sai

Roll No: B210437CS

Batch: CS02

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.

Introduction

What is an Operating System?

An operating system is system software that manages computer hardware, software resources, and provides common services for computer programs. Your computer's operating system (OS) manages all of the software and hardware on the computer. Most of the time, there are several different computer programs running at the same time, and they all need to access your computer's central processing unit (CPU), memory, and storage. The operating system coordinates all of this to make sure each program gets what it needs. There are various types of Operating systems and Linux is one of them. Linux is an open-source operating system originally developed by Linus Torvalds.

What is a Linux Kernel?

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 2 managing resources as efficiently as possible. The kernel is so named because—like a seed inside a hard shell—it exists within the OS and controls all the major functions of the hardware, whether it's a phone, laptop, server, or any other kind of computer.

What does a kernel do?

The kernel has 4 jobs:

1. Memory management: Keep track of how much memory is used to store what, and where.
2. Process management: Determine which processes can use the central processing unit (CPU), when, and for how long.
3. Device drivers: Act as mediator/interpreter between the hardware and processes.
4. System calls and security: Receive requests for service from the processes.

Methodology

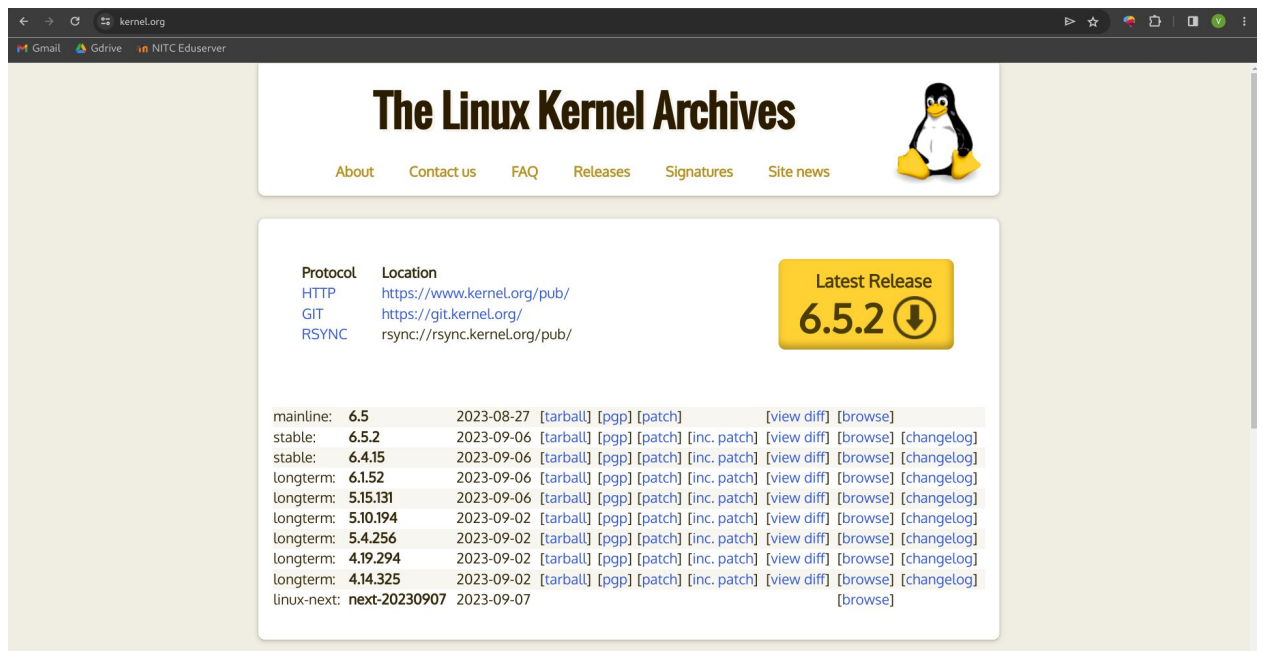
- Dual booting can be directly done with the host OS but if something goes wrong then OS could be corrupted. Hence, it is highly important to load the code accurately and load them.
- Obtain the kernel source code from kernel.org
- Install the development dependencies.
- Compile the kernel.
- Install the compiled kernel, add grub entry.
- Reboot the system.

Explanation

The process of building a Linux kernel takes seven easy steps to complete. However, the procedure requires a significant amount of time to complete, depending on the system speed.

1. Get the latest Linux kernel source code

Visit the official project website kernel.org and download the latest source code. Click on the big yellow button that read as “Latest Stable Kernel”.



The latest version, 6.5.2, was released on 06-09-2023. However, since I have already completed the installation of version 6.5.1, I will continue with this version.

The filename would be linux-x.y.z.tar, where x.y.z is actual Linux kernel version number. For example file linux-6.5.1.tar represents Linux kernel version 6.5.1

2. Extract .tar file

You can extract the source code in your home directory or any other directory using the following tar command

```
>> tar -xvf linux-6.5.1.tar
```

A terminal window titled 'Terminal' showing the output of the 'tar -xvf linux-6.5.1.tar' command. The output lists the files being extracted, including headers, source files, and Makefiles for various components like 'dummy', 'gen_init', 'include', 'virt', 'kvm', and 'lib'. The terminal shows the current directory as '~' and the user as 'venkatasai24'. The time is 15:55:39.

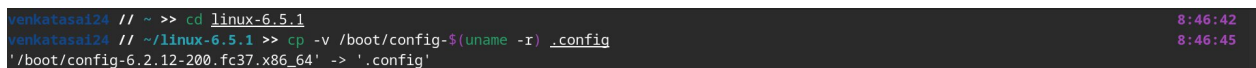
```
linux-6.5.1/usr/dummy-include/stdbool.h
linux-6.5.1/usr/dummy-include/stdlib.h
linux-6.5.1/usr/gen_init_cpio.c
linux-6.5.1/usr/gen_initramfs.sh
linux-6.5.1/usr/include/
linux-6.5.1/usr/include/.gitignore
linux-6.5.1/usr/include/Makefile
linux-6.5.1/usr/include/headers_check.pl
linux-6.5.1/usr/initramfs_data.S
linux-6.5.1/virt/
linux-6.5.1/virt/Makefile
linux-6.5.1/virt/kvm/
linux-6.5.1/virt/kvm/Kconfig
linux-6.5.1/virt/kvm/Makefile.kvm
linux-6.5.1/virt/kvm/async_pf.c
linux-6.5.1/virt/kvm/async_pf.h
linux-6.5.1/virt/kvm/binary_stats.c
linux-6.5.1/virt/kvm/coalesced_mmio.c
linux-6.5.1/virt/kvm/coalesced_mmio.h
linux-6.5.1/virt/kvm/dirty_ring.c
linux-6.5.1/virt/kvm/eventfd.c
linux-6.5.1/virt/kvm/irqchip.c
linux-6.5.1/virt/kvm/kvm_main.c
linux-6.5.1/virt/kvm/kvm_mm.h
linux-6.5.1/virt/kvm/pfncache.c
linux-6.5.1/virt/kvm/vfio.c
linux-6.5.1/virt/kvm/vfio.h
linux-6.5.1/virt/lib/
linux-6.5.1/virt/lib/Kconfig
linux-6.5.1/virt/lib/Makefile
linux-6.5.1/virt/lib/irqbypass.c
venkatasai24 ~/ ~ >>
```

3. Configure the Linux kernel features and modules

Before start building the kernel, one must configure Linux kernel features. You must also specify which kernel modules (drivers) needed for your system. The task can be overwhelming for a new user. I suggest that you copy existing config file using the cp command.

```
>> cd linux-6.5.1
```

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

A terminal window showing the execution of 'cd linux-6.5.1' and 'cp -v /boot/config-\$(uname -r) .config'. The output shows the source file path as '/boot/config-6.2.12-200.fc37.x86_64' and the destination as './.config'. The time is 8:46:45.

```
venkatasai24 ~/ ~ >> cd linux-6.5.1
venkatasai24 ~/linux-6.5.1 >> cp -v /boot/config-$(uname -r) .config
'/boot/config-6.2.12-200.fc37.x86_64' -> './.config'
```

4. Install the required compilers and other tools

You must have development tools such as GCC compilers and related tools installed to compile the Linux kernel.

To install GCC and development tools on Fedora linux use the following command.

```
>> sudo dnf install ncurses-devel bison flex openssl-devel dwarves
```

Here's a brief explanation of each package mentioned in the command for installing development tools on Fedora Linux:

ncurses-devel: The ncurses library is used to create text-based user interfaces in the terminal. The development package (ncurses-devel) provides header files and libraries necessary for building programs that utilize this library.

bison: Bison is a parser generator used to generate parsers for programming languages and other data formats. It is often used in the compilation process to generate code for parsing and analyzing source code.

flex: Flex, short for "Fast Lexical Analyzer Generator," is a tool for generating lexical analyzers (parsers for tokens) for text-based processing. It's commonly used in conjunction with Bison to create complete parsing solutions.

openssl-devel: OpenSSL is a widely-used cryptographic library. The development package (openssl-devel) contains header files and libraries that are needed for building applications that use OpenSSL for secure communication and encryption.

dwarves: Dwarves is a collection of debugging tools used to inspect and analyze debugging information in ELF (Executable and Linkable Format) files. These tools are valuable for developers debugging and profiling software.

These packages are essential for building and compiling software on Fedora Linux, especially when working with low-level system components like the Linux kernel or other development projects.

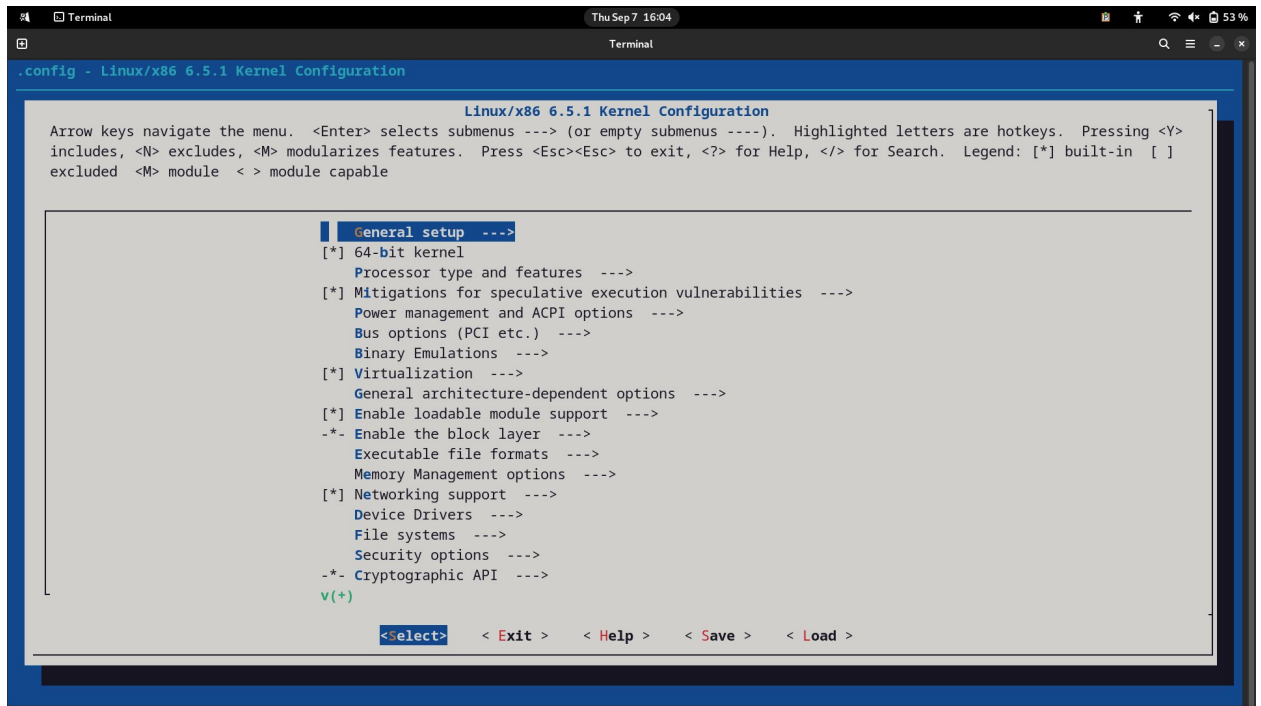
```
venkatasai24 // ~ >> sudo dnf install ncurses-devel bison flex openssl-devel dwarves 16:02:37
[sudo] password for venkatasai24:
Fedora 37 - x86_64 - Updates 23 kB/s | 5.2 kB 00:00
Fedora 37 - x86_64 - Updates 926 kB/s | 3.2 MB 00:03
Fedora Modular 37 - x86_64 - Updates 24 kB/s | 5.4 kB 00:00
created by dnf config-manager from https://repo.discord.com/rpm/stable/repodata/repomd.xml 0.0 B/s | 0 B 00:00
Errors during downloading metadata for repository 'repo.discord.com_rpm_stable_repodata_repomd.xml':
- Curl error (6): Couldn't resolve host name for https://repo.discord.com/rpm/stable/repodata/repomd.xml [Could not resolve host: repo.discord.com]
Error: Failed to download metadata for repo 'repo.discord.com_rpm_stable_repodata_repomd.xml': Cannot download repomd.xml: Cannot download repodata/repomd.xml: All mirrors were tried
Ignoring repositories: repo.discord.com_rpm_stable_repodata_repomd.xml
Package ncurses-devel-6.4-3.20230114.fc37.x86_64 is already installed.
Package bison-3.8.2-3.fc37.x86_64 is already installed.
Package flex-2.6.4-11.fc37.x86_64 is already installed.
Package openssl-devel-1:3.0.8-1.fc37.x86_64 is already installed.
Package dwarves-1.25-1.fc37.x86_64 is already installed.
Dependencies resolved.
Nothing to do.
Complete!
venkatasai24 // ~ >> 16:03:00
```

5. Configuring the kernel

Now you can start the kernel configuration by typing any one of the following command in source code directory

>> make menuconfig

This command is a text based color menus, radiolists & dialogs. This option also useful on remote server if you wanna compile kernel remotely.



You have to select different options as per your need. Each configuration option has HELP button associated with it so select help button to get help.

Please note that 'make menuconfig' is optional. I used it here to demonstration purpose only. You can enable or disable certain features or kernel driver with this option. It is easy to remove support for a device driver or option and end up with a broken kernel. For example, if the ext4 driver is removed from the kernel configuration file, a system may not boot. When in doubt, just leave support in the kernel.

6. How to compile a Linux Kernel

Start compiling and to create a compressed kernel image, enter:

```
>> make
```

To speed up compile time, pass the -j as follows:

```
## use 4 core/thread ##
```

```
>> make -j 4
```

```
## get thread or cpu core count using nproc command ##
```

```
>> make -j $(nproc)
```

The \$ nproc on my system gave 8 using which I performed the make command.

```
>> make -j 8
```

The yes command generates an infinite stream of empty lines, and the pipe (|) sends this input to the make command.

Compiling and building the Linux kernel going take a significant amount of time. The build time depends upon your system's resources such as available CPU core and the current system load. So we must have some patience.

The terminal lists all Linux kernel components: memory management, hardware device drivers, filesystem drivers, network drivers, and process management.

The end of this step looks like this

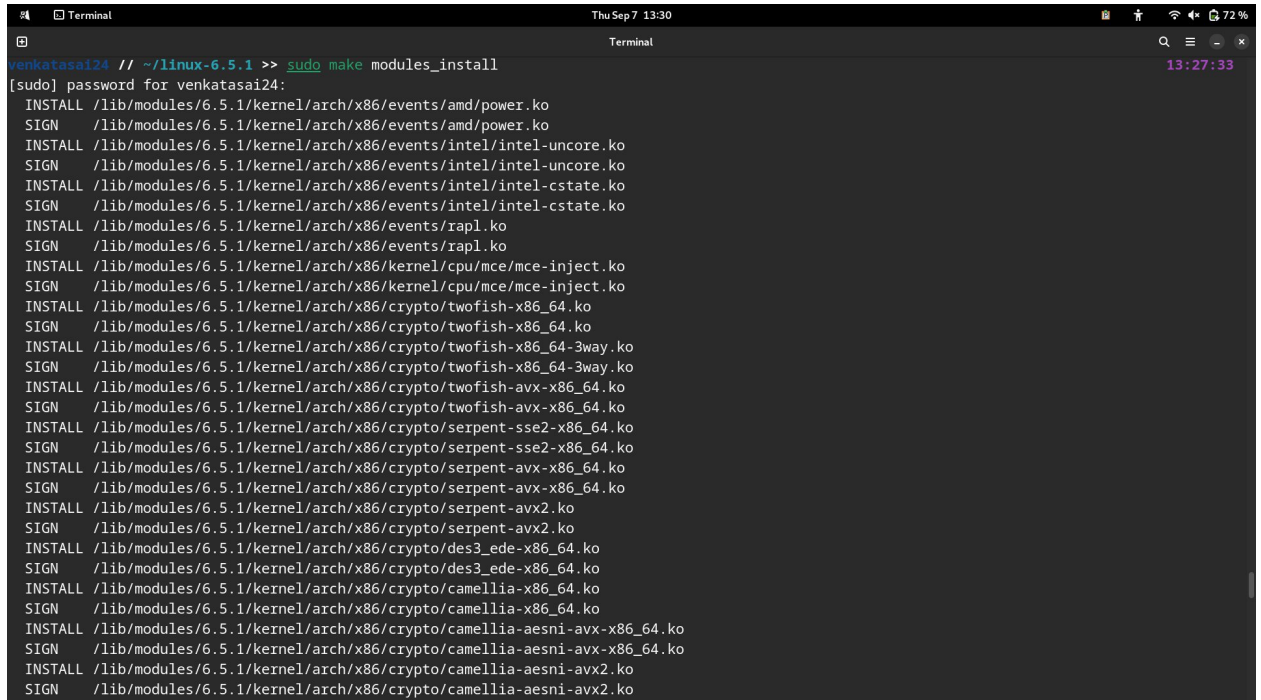
```
venkatasai24 // ~/linux-6.5.1 >> yes "" | make -j8
DESCEND objtool
DESCEND bpf/resolve_btfids
INSTALL libsubcmd_headers
INSTALL libsubcmd_headers
CALL scripts/checksyscalls.sh
CHK kernel/kheaders_data.tar.xz
UPD include/generated/utsversion.h
CC init/version-timestamp.o
LD .tmp_vmlinux.btf
BTF .btf.vmlinux.bin.o
LD .tmp_vmlinux.kallsyms1
NM .tmp_vmlinux.kallsyms1.syms
KSYM .tmp_vmlinux.kallsyms1.S
AS .tmp_vmlinux.kallsyms1.S
LD .tmp_vmlinux.kallsyms2
NM .tmp_vmlinux.kallsyms2.syms
KSYM .tmp_vmlinux.kallsyms2.S
AS .tmp_vmlinux.kallsyms2.S
LD vmlinux
BTFIDS vmlinux
NM System.map
SORTTAB vmlinux
RELOCS arch/x86/boot/compressed/vmlinux.relocs
RSTRIP vmlinux
HOSTCC arch/x86/tools/insn_decoder_test
HOSTCC arch/x86/tools/insn_sanitizer
TEST posttest
arch/x86/tools/insn_decoder_test: success: Decoded and checked 7155439 instructions
TEST posttest
arch/x86/tools/insn_sanitizer: Success: decoded and checked 1000000 random instructions with 0 errors (seed:0x7ed020a6)
CC arch/x86/boot/a20.o
AS arch/x86/boot/a20.o
LD arch/x86/boot/a20.o
CC arch/x86/boot/compressed/error.o
HOSTCC arch/x86/boot/compressed/mkpiggy
OBJCOPY arch/x86/boot/compressed/vmlinux.bin
CC arch/x86/boot/compressed/cpuflags.o
CC arch/x86/boot/compressed/early_serial_console.o
CC arch/x86/boot/compressed/kaslr.o
CC arch/x86/boot/compressed/ident_map_64.o
CC arch/x86/boot/compressed/idt_64.o
AS arch/x86/boot/compressed/idt_handlers_64.o
AS arch/x86/boot/compressed/mem_encrypt.o
CC arch/x86/boot/compressed/pgtable_64.o
CC arch/x86/boot/compressed/sev.o
CC arch/x86/boot/compressed/acpi.o
CC arch/x86/boot/compressed/tdx.o
AS arch/x86/boot/compressed/tdcall.o
CC arch/x86/boot/compressed/tdx-shared.o
CC arch/x86/boot/compressed/mem.o
CC arch/x86/boot/compressed/efi.o
AS arch/x86/boot/compressed/efi_mixed.o
CC arch/x86/boot/compressed/misc.o
ZSTD22 arch/x86/boot/compressed/vmlinux.bin.zst
MKPIGGY arch/x86/boot/compressed/piggy.S
AS arch/x86/boot/compressed/piggy.o
LD arch/x86/boot/compressed/vmlinux
ZOFFSET arch/x86/boot/zoffset.h
OBJCOPY arch/x86/boot/vmlinux.bin
AS arch/x86/boot/header.o
LD arch/x86/boot/setup.elf
OBJCOPY arch/x86/boot/setup.bin
BUILD arch/x86/boot/bzImage
Kernel: arch/x86/boot/bzImage is ready (#2)
venkatasai24 // ~/linux-6.5.1 >>
```

```
venkatasai24 // ~/linux-6.5.1 >> yes "" | make -j8
DESCEND objtool
DESCEND bpf/resolve_btfids
INSTALL libsubcmd_headers
INSTALL libsubcmd_headers
CALL scripts/checksyscalls.sh
CHK kernel/kheaders_data.tar.xz
UPD include/generated/utsversion.h
CC init/version-timestamp.o
LD .tmp_vmlinux.btf
BTF .btf.vmlinux.bin.o
LD .tmp_vmlinux.kallsyms1
NM .tmp_vmlinux.kallsyms1.syms
KSYM .tmp_vmlinux.kallsyms1.S
AS .tmp_vmlinux.kallsyms1.S
LD .tmp_vmlinux.kallsyms2
NM .tmp_vmlinux.kallsyms2.syms
KSYM .tmp_vmlinux.kallsyms2.S
AS .tmp_vmlinux.kallsyms2.S
LD vmlinux
BTFIDS vmlinux
NM System.map
SORTTAB vmlinux
RELOCS arch/x86/boot/compressed/vmlinux.relocs
RSTRIP vmlinux
HOSTCC arch/x86/tools/insn_decoder_test
HOSTCC arch/x86/tools/insn_sanitizer
TEST posttest
arch/x86/tools/insn_decoder_test: success: Decoded and checked 7155439 instructions
TEST posttest
arch/x86/tools/insn_sanitizer: Success: decoded and checked 1000000 random instructions with 0 errors (seed:0x7ed020a6)
CC arch/x86/boot/a20.o
AS arch/x86/boot/a20.o
LD arch/x86/boot/a20.o
CC arch/x86/boot/compressed/error.o
HOSTCC arch/x86/boot/compressed/mkpiggy
OBJCOPY arch/x86/boot/compressed/vmlinux.bin
CC arch/x86/boot/compressed/cpuflags.o
CC arch/x86/boot/compressed/early_serial_console.o
CC arch/x86/boot/compressed/kaslr.o
CC arch/x86/boot/compressed/ident_map_64.o
CC arch/x86/boot/compressed/idt_64.o
AS arch/x86/boot/compressed/idt_handlers_64.o
AS arch/x86/boot/compressed/mem_encrypt.o
CC arch/x86/boot/compressed/pgtable_64.o
CC arch/x86/boot/compressed/sev.o
CC arch/x86/boot/compressed/acpi.o
CC arch/x86/boot/compressed/tdx.o
AS arch/x86/boot/compressed/tdcall.o
CC arch/x86/boot/compressed/tdx-shared.o
CC arch/x86/boot/compressed/mem.o
CC arch/x86/boot/compressed/efi.o
AS arch/x86/boot/compressed/efi_mixed.o
CC arch/x86/boot/compressed/misc.o
ZSTD22 arch/x86/boot/compressed/vmlinux.bin.zst
MKPIGGY arch/x86/boot/compressed/piggy.S
AS arch/x86/boot/compressed/piggy.o
LD arch/x86/boot/compressed/vmlinux
ZOFFSET arch/x86/boot/zoffset.h
OBJCOPY arch/x86/boot/vmlinux.bin
AS arch/x86/boot/header.o
LD arch/x86/boot/setup.elf
OBJCOPY arch/x86/boot/setup.bin
BUILD arch/x86/boot/bzImage
Kernel: arch/x86/boot/bzImage is ready (#2)
venkatasai24 // ~/linux-6.5.1 >>
```

7. Install the Linux kernel modules

```
>> sudo make modules_install
```

This step is essential to ensure that the kernel functions correctly by incorporating the required modules.

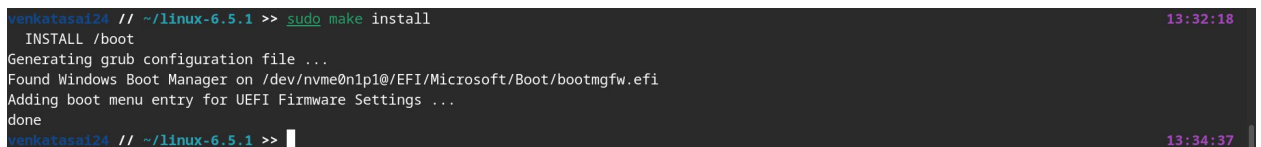
A terminal window titled 'Terminal' showing the output of the command 'sudo make modules_install'. The output lists various kernel modules being installed and signed, including events/amd/power.ko, events/intel/intel-uncore.ko, events/intel/intel-cstate.ko, events/rapl.ko, kernel/cpu/mce/mce-inject.ko, crypto/twofish-x86_64.ko, crypto/twofish-x86_64-3way.ko, crypto/twofish-avx-x86_64.ko, crypto/serpent-sse2-x86_64.ko, crypto/serpent-avx-x86_64.ko, crypto/serpent-avx2.ko, crypto/des3_ede-x86_64.ko, crypto/camellia-x86_64.ko, crypto/camellia-aesni-avx-x86_64.ko, and crypto/camellia-aesni-avx2.ko. The terminal shows the user 'venkatasai24' and the time '13:27:33'.

```
venkatasai24 // ~/linux-6.5.1 >> sudo make modules_install
[sudo] password for venkatasai24:
INSTALL /lib/modules/6.5.1/kernel/arch/x86/events/amd/power.ko
SIGN /lib/modules/6.5.1/kernel/arch/x86/events/amd/power.ko
INSTALL /lib/modules/6.5.1/kernel/arch/x86/events/intel/intel-uncore.ko
SIGN /lib/modules/6.5.1/kernel/arch/x86/events/intel/intel-uncore.ko
INSTALL /lib/modules/6.5.1/kernel/arch/x86/events/intel/intel-cstate.ko
SIGN /lib/modules/6.5.1/kernel/arch/x86/events/intel/intel-cstate.ko
INSTALL /lib/modules/6.5.1/kernel/arch/x86/events/rapl.ko
SIGN /lib/modules/6.5.1/kernel/arch/x86/events/rapl.ko
INSTALL /lib/modules/6.5.1/kernel/arch/x86/kernel/cpu/mce/mce-inject.ko
SIGN /lib/modules/6.5.1/kernel/arch/x86/kernel/cpu/mce/mce-inject.ko
INSTALL /lib/modules/6.5.1/kernel/arch/x86/crypto/twofish-x86_64.ko
SIGN /lib/modules/6.5.1/kernel/arch/x86/crypto/twofish-x86_64.ko
INSTALL /lib/modules/6.5.1/kernel/arch/x86/crypto/twofish-x86_64-3way.ko
SIGN /lib/modules/6.5.1/kernel/arch/x86/crypto/twofish-x86_64-3way.ko
INSTALL /lib/modules/6.5.1/kernel/arch/x86/crypto/twofish-avx-x86_64.ko
SIGN /lib/modules/6.5.1/kernel/arch/x86/crypto/twofish-avx-x86_64.ko
INSTALL /lib/modules/6.5.1/kernel/arch/x86/crypto/serpent-sse2-x86_64.ko
SIGN /lib/modules/6.5.1/kernel/arch/x86/crypto/serpent-sse2-x86_64.ko
INSTALL /lib/modules/6.5.1/kernel/arch/x86/crypto/serpent-avx-x86_64.ko
SIGN /lib/modules/6.5.1/kernel/arch/x86/crypto/serpent-avx-x86_64.ko
INSTALL /lib/modules/6.5.1/kernel/arch/x86/crypto/serpent-avx2.ko
SIGN /lib/modules/6.5.1/kernel/arch/x86/crypto/serpent-avx2.ko
INSTALL /lib/modules/6.5.1/kernel/arch/x86/crypto/des3_ede-x86_64.ko
SIGN /lib/modules/6.5.1/kernel/arch/x86/crypto/des3_ede-x86_64.ko
INSTALL /lib/modules/6.5.1/kernel/arch/x86/crypto/camellia-x86_64.ko
SIGN /lib/modules/6.5.1/kernel/arch/x86/crypto/camellia-x86_64.ko
INSTALL /lib/modules/6.5.1/kernel/arch/x86/crypto/camellia-aesni-avx-x86_64.ko
SIGN /lib/modules/6.5.1/kernel/arch/x86/crypto/camellia-aesni-avx-x86_64.ko
INSTALL /lib/modules/6.5.1/kernel/arch/x86/crypto/camellia-aesni-avx2.ko
SIGN /lib/modules/6.5.1/kernel/arch/x86/crypto/camellia-aesni-avx2.ko
```

8. Install the Linux kernel

So far we have compiled the Linux kernel and installed kernel modules. It is time to install the kernel itself:

```
>> sudo make install
```

A terminal window titled 'Terminal' showing the output of the command 'sudo make install'. The output shows the installation of the kernel to the boot directory, generating the GRUB configuration file, and adding a boot menu entry for UEFI Firmware Settings. The terminal shows the user 'venkatasai24' and the time '13:32:18' and '13:34:37'.

```
venkatasai24 // ~/linux-6.5.1 >> sudo make install
INSTALL /boot
Generating grub configuration file ...
Found Windows Boot Manager on /dev/nvme0n1p1@EFI/Microsoft/Boot/bootmgfw.efi
Adding boot menu entry for UEFI Firmware Settings ...
done
venkatasai24 // ~/linux-6.5.1 >>
```

9. Update GRUB Configuration

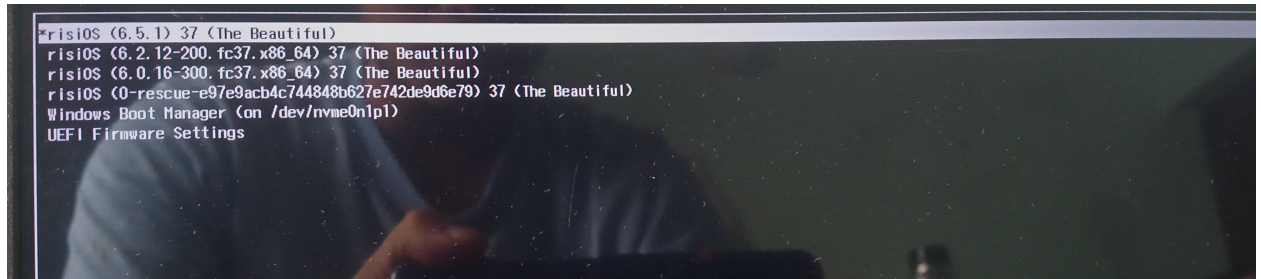
Update the GRUB bootloader configuration to include the new kernel.

```
>> sudo grub2-mkconfig -o /boot/grub2/grub.cfg
```



```
venkatasai24 // ~/linux-6.5.1 >> sudo grub2-mkconfig -o /boot/grub2/grub.cfg
[sudo] password for venkatasai24:
Generating grub configuration file ...
Found Windows Boot Manager on /dev/nvme0n1p1@/EFI/Microsoft/Boot/bootmgfw.efi
Adding boot menu entry for UEFI Firmware Settings ...
done
venkatasai24 // ~/linux-6.5.1 >>
```

You should ideally reboot the system after the completion of this stage.



To show that the latest kernel is installed, use the following command.

```
>> uname -r
```

This will print a more specific string with actual release.

```
venkatasai24 // ~ >> uname -r
6.5.1
venkatasai24 // ~ >>
```

Congratulations! You completed various steps to build the Linux kernel from source code and compiled kernel should be running on your system.