RV College of Engineering ®

Go, change the world

Experiential Learning

Operating System Presentation

MODIFYING KERNEL FOR FASTER BOOTING

Team Introduction

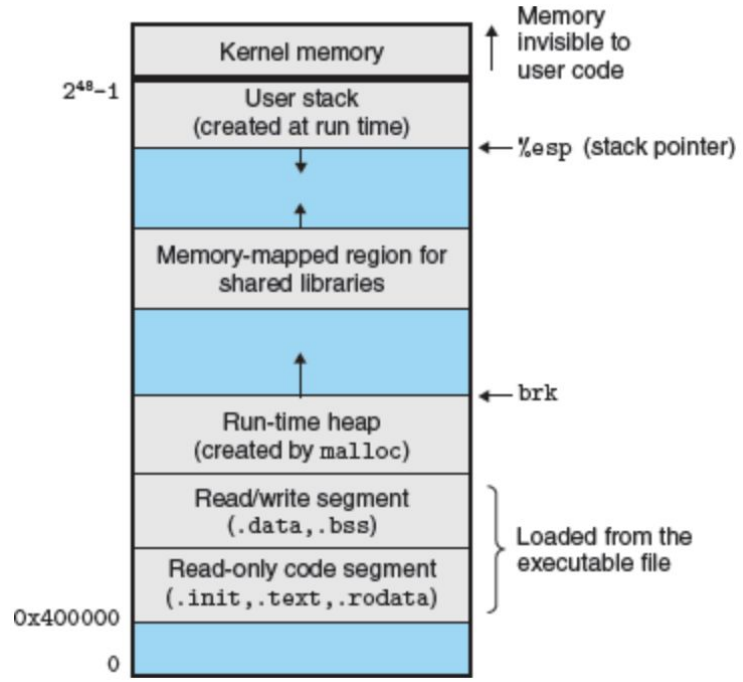
USN	Name	Email Id
1RV22CS243	Aaryan P	aaryanp.cs22@rvce.edu.in
RVCE22BCS059	Umang Mishra	umangmishra.cs22@rvce.edu.in

Problem Statement

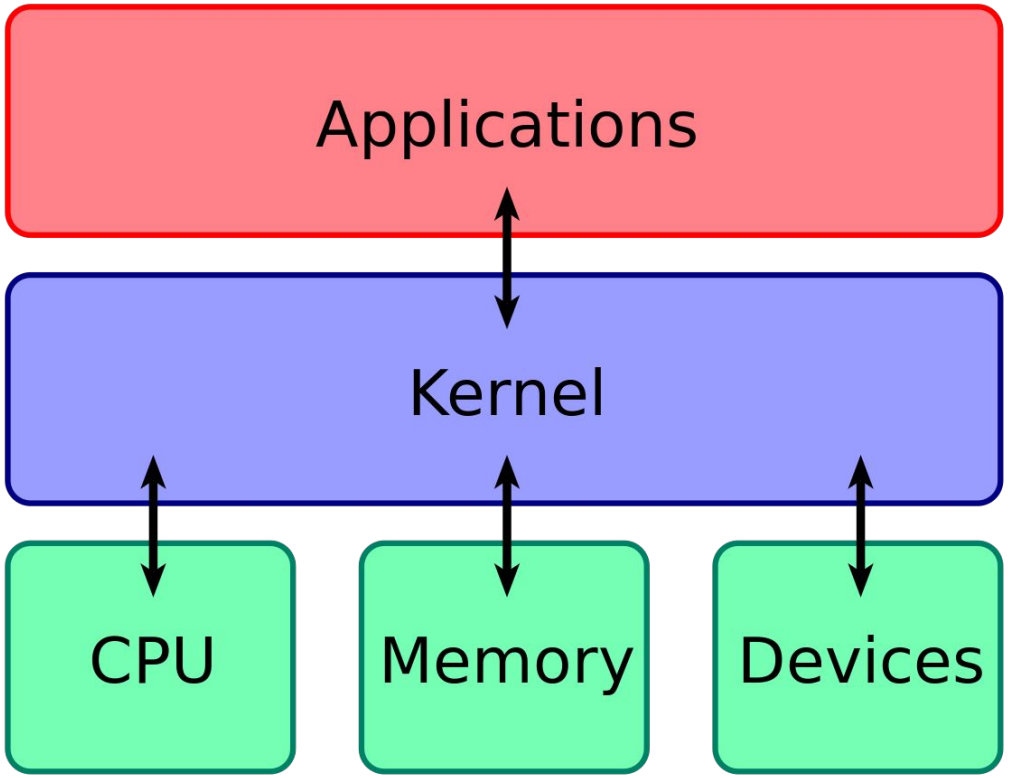
Custom kernel development essential for niche computing, demanding flexibility and optimization

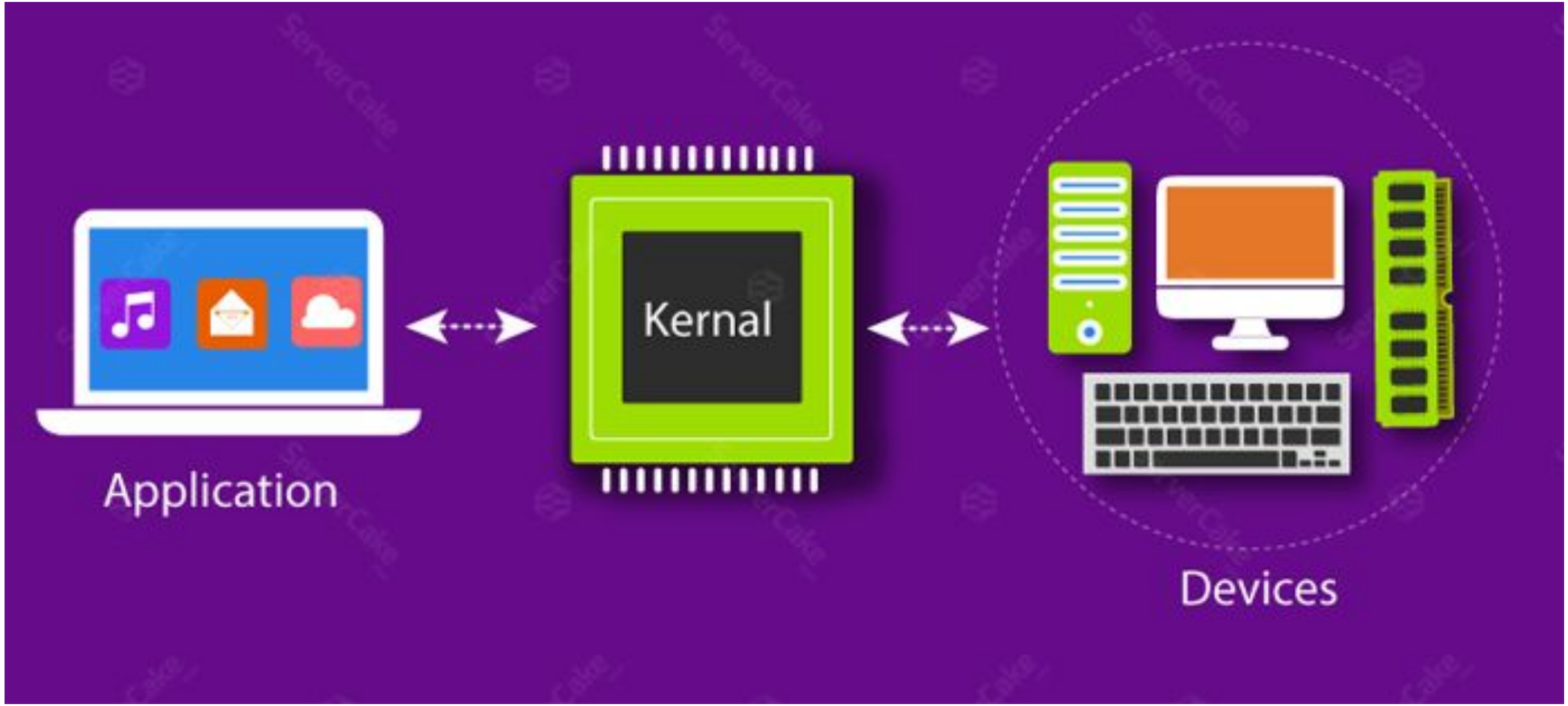
Relevance to the Course

Developing a custom kernel is paramount in an operating systems course, bridging theoretical knowledge with hands-on expertise. It provides students with a unique opportunity to delve into low-level system operations, memory management, and hardware interaction. This practical experience fosters a comprehensive understanding of operating system concepts, preparing students to address real-world challenges in optimizing, securing, and tailoring operating systems to diverse computing environments.

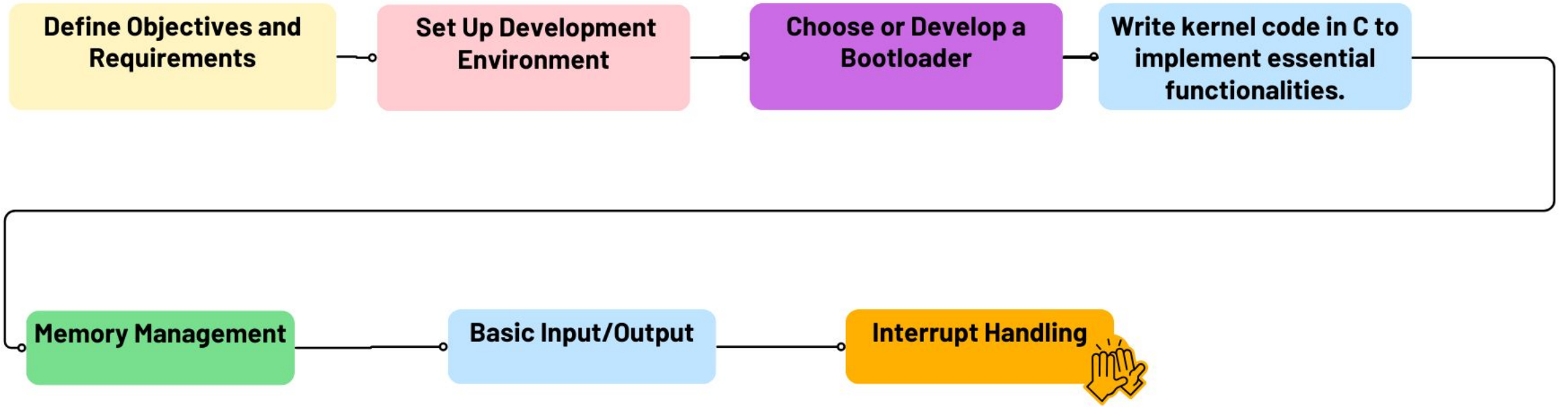


An operating system's kernel is its central component, handling system resources, delivering necessary functions, and serving as a bridge between hardware and software. We can obtain a thorough understanding of low-level programming, operating system internals, and computer architecture by creating own kernel.




Kernal
Application
Devices

MAKING OWN KERNEL



Tools/APIs Used

Assembly Language: Low-level programming language for tasks like bootloader development and interacting with hardware.
C Programming Language: Widely used for kernel development due to its efficiency and proximity to hardware.
GNU Compiler Collection (GCC): Compiles code for the target architecture, generating executable binaries.
Linker (LD): Links compiled code and libraries to create the final kernel image.
Make: Automates the build process, managing dependencies and compiling source code efficiently.
QEMU (Quick Emulator): Emulation tool for testing and debugging kernels in a virtual environment.
Bochs: Another emulator used for testing kernels, providing a virtual environment for development.
GRUB (GRand Unified Bootloader): Commonly used as a bootloader for x86-based systems, loading the kernel into memory during system boot.
Linux Kernel API: When building a Linux kernel, developers use the Linux Kernel API for interacting with kernel services and functions.
Hardware Abstraction Layer (HAL): Provides an abstraction layer for hardware interactions, enabling portability across different architectures.

Implementation

Download the Source Code from the official kernel website
Open the terminal and use the wget command to download the Linux kernel source code
Extract the Source Code
Install Required Packages: git fakeroot build-essential ncurses-dev xz-utils libssl-dev bc flex libelf-dev bison
Navigate to the linux-6.7.7 directory using the cd command
Copy the existing Linux config file using the cp command and then use the command make menuconfig
Build the kernel using make command
Use: sudo make modules_install and sudo make install
Update the initramfs to the installed kernel version 6.7.7
Update the GRUB bootloader
Verify the kernel version using the uname command

```
  DEPMOD  /lib/modules/6.7.7
aaryan@Ubuntu:~/linux-6.7.7$ sudo make install -j32
[sudo] password for aaryan:
  INSTALL /boot
run-parts: executing /etc/kernel/postinst.d/initramfs-tools 6.7.7 /boot/vmlinuz-
6.7.7
update-initramfs: Generating /boot/initrd.img-6.7.7
run-parts: executing /etc/kernel/postinst.d/unattended-upgrades 6.7.7 /boot/vmli
nuz-6.7.7
```

Implementation

Used cd to set the current working to the linux kernel version we had manually installed.
Used ls-a to check the directories to see if the .config file was present or not.
ncurses library not present so it was installed using:
sudo apt install libncurses-dev
sudo apt-get install flex

```
aaryan@Ubuntu:~$ cd linux-6.7.7
aaryan@Ubuntu:~/linux-6.7.7$ cp -v /boot/config-$(uname -r) .config
'/boot/config-6.5.0-21-generic' -> '.config'
```

Kernel customization:
General Setup:
Kernel compression mode from GZip to ZTSD for faster boot time
Disable POSIX Messaging queue and auditing support
Periodic timer ticks enabled for better performance
Disable BSD process accounting
Change kernel log buffer size to 16
Keep initram support only for ZTSD
Processor Types and Features:
Enable Intel if on an Intel device and change processor family to core 2
Enable the block layer: Disable block layer debugging as it's only for kernel developers

Implementation

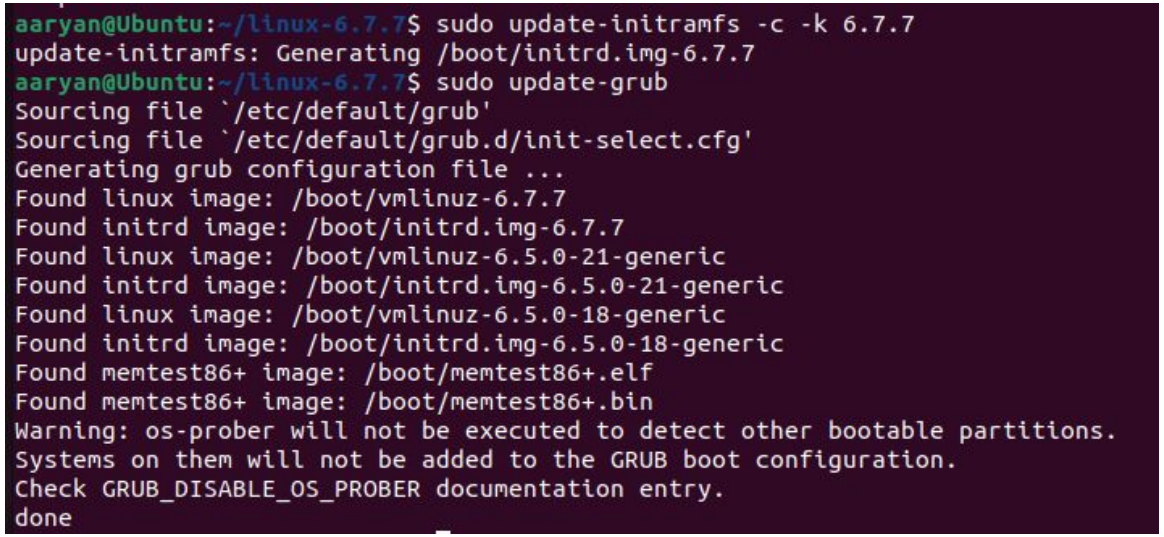
Device Drivers:
Disable PC Card support, enable NVME, disable most Misc Drives
Enable Asynchronous SCSI Scanning for faster booting
Disable Macintosh device drivers
Enable WireGuard secure network tunnel for VPN usage and multimedia support
Change maximum number of GPUs to 2
Enable laptop hybrid graphics and Intel sound card support
ASUS laptop extras enabled as an ASUS laptop was used for demonstration
Disable Miscellaneous File Systems

Compile using make -j32 && sudo make modules_install -j32

Error: openssl no such file or directory
Fix: sudo apt-get install libssl-dev

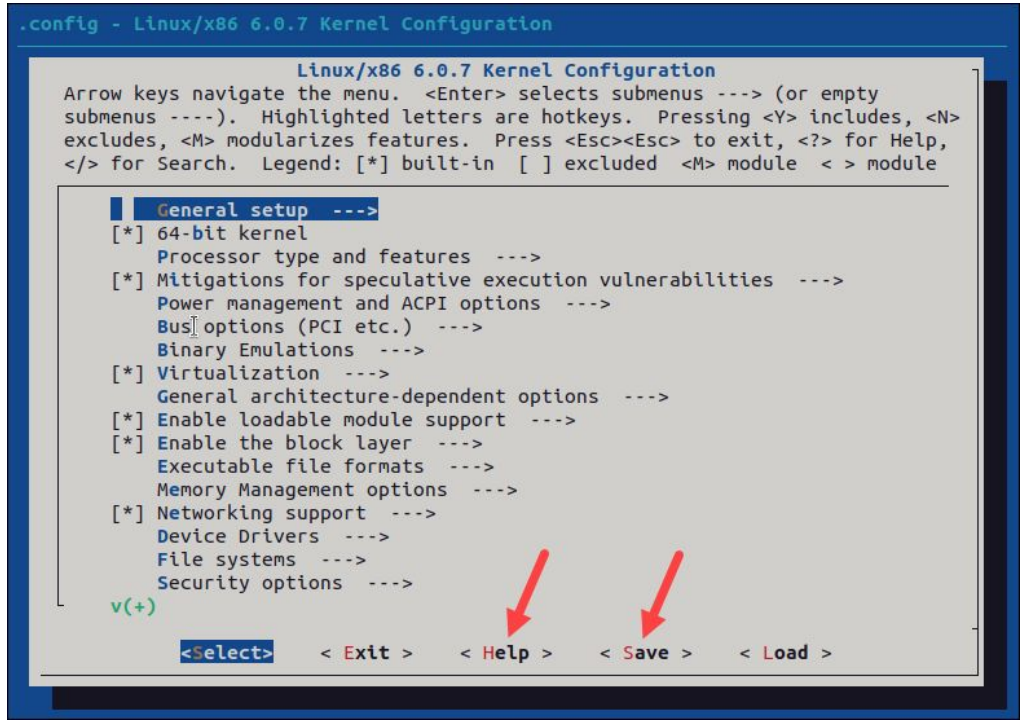
make bzImage
ls /boot/ to find vmlinuz

Updating the GRUB bootloader:
sudo update-initramfs -c -k 6.7.7
sudo update-grub



Implementation

Ubuntu Error debugging:
ERROR: No rule to make target 'debian/canonical-certs.pem
FIX: scripts/config --disable SYSTEM_TRUSTED_KEYS
scripts/config --disable SYSTEM_REVOCATION_KEYS
ERROR: user is not in the sudoers file
FIX: Change GRUB settings (root)
mount -o rw, remount/ using adduser username sudo command




```
aaryan@Ubuntu:~/linux-6.7.7$ uname -mrs
Linux 6.5.0-21-generic x86_64
```

```
aaryan@Ubuntu:~/linux-6.7.7$ free -h
               total        used        free      shared  buff/cache   available
Mem:            10Gi       798Mi       1.9Gi        34Mi       7.4Gi       9.0Gi
Swap:          2.0Gi       0.0Ki       2.0Gi
aaryan@Ubuntu:~/linux-6.7.7$
```

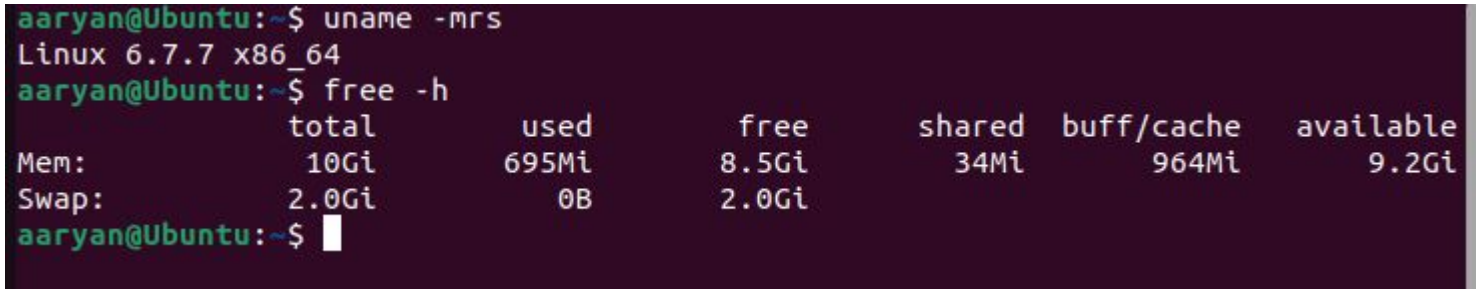
Results

Device Drivers:
Disable PC Card support, enable NVME, disable most Misc Drives
Enable Asynchronous SCSI Scanning for faster booting
Disable Macintosh device drivers
Enable WireGuard secure network tunnel for VPN usage and multimedia support
Change maximum number of GPUs to 2
Enable laptop hybrid graphics and Intel sound card support
ASUS laptop extras enabled as an ASUS laptop was used for demonstration
Disable Miscellaneous File Systems

Compile using make && sudo make modules_install

Error: openssl no such file or directory
Fix: sudo apt-get install libssl-dev

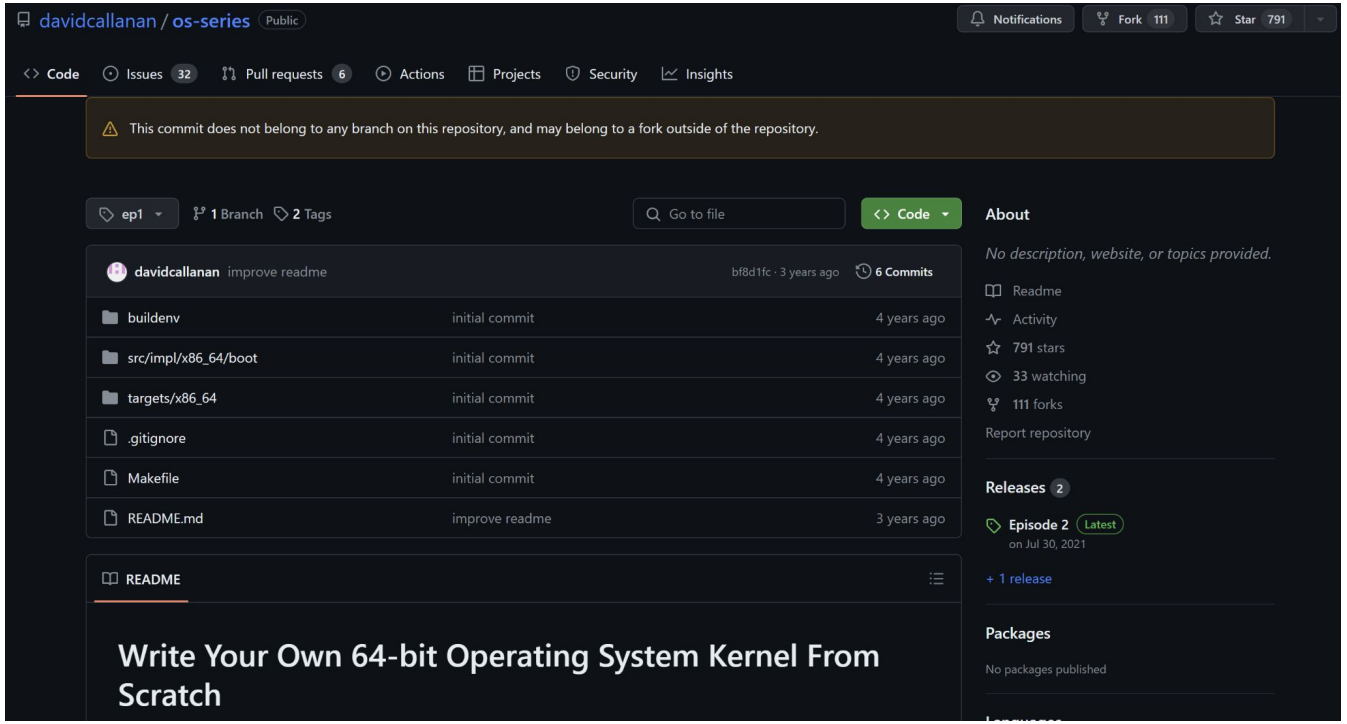
make bzImage
ls /boot/ to find vmlinuz



Future Scope

An existing Linux kernel source code has been used in this project
We have downloaded the code from kernel.org and built it using make command
We have customised it to our needs using the .config file such that lesser RAM is used

In the future, we plan to use C to develop our own kernel from scratch using assembly language and virtual environments like dockerfile and Qemu



Applications

1. Customized Operating Systems: tailored to specific hardware requirements or performance optimizations.

2. Embedded Systems: where resource constraints and specialized functionalities often demand a lightweight and efficient kernel.

3. Security Enhancements:Creating a custom kernel enables the implementation of security features tailored to specific needs, addressing vulnerabilities, and enhancing overall system resilience.

4. Real-time Systems:For applications requiring precise timing and responsiveness, such as in robotics or industrial control systems,

5. Educational Purposes: Building a kernel from scratch is an excellent educational exercise, providing insights into low-level system operations, memory management, fostering a deeper understanding of operating system concepts.

6. Research and Experimentation: allows researchers and enthusiasts to experiment with new ideas and test novel approaches.

REFERENCES

- https://www.linuxjournal.com/content/what-does-it-take-make-kernel-0
- https://phoenixnap.com/kb/build-linux-kernel
- kernel.org
- https://www.tecmint.com/fix-user-is-not-in-the-sudoers-file-the-incident-will-be-reported-ubuntu/