DEVELOPING OUR OWN KERNEL

Phase-II Report

CS235AI

Submitted by

Umang Mishra	**1RV22CS220**
Aaryan P	**1RV22CS243**

Under the guidance of

1. Dr. Jyothi Shetty	**Operating Systems (CS235AI)**

Department of Computer Science Engineering 2023-2024

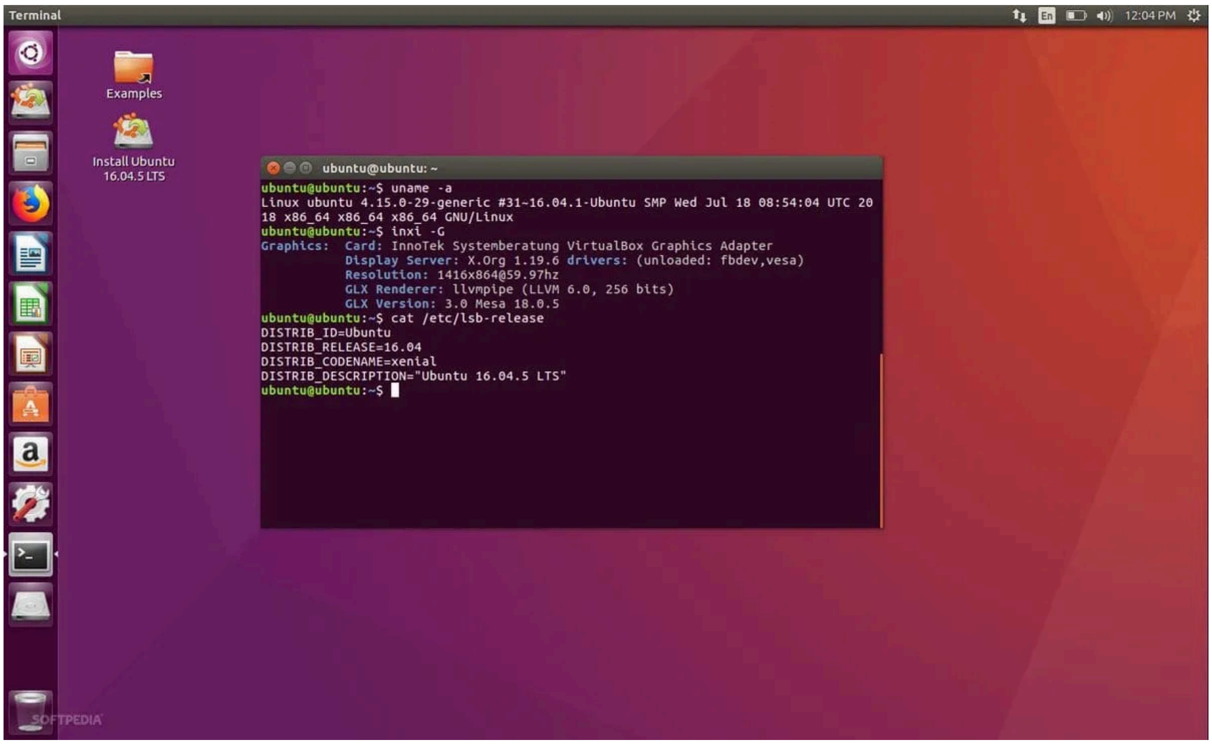
CONTENT

SYNOPSIS

In this project, the primary focus lies in the development of a custom Linux kernel, tailored to specific hardware requirements and performance optimizations. The objectives encompass optimizing system performance, tailoring feature sets, ensuring hardware compatibility, fostering educational understanding, and documenting the customization process for reproducibility. By meticulously configuring kernel options, users can delve deep into the inner workings of the Linux kernel, enhancing their understanding of operating system concepts. This project not only empowers users to create a personalized and high-performing Linux kernel but also encourages knowledge-sharing within the community or among team members.

The implementation involves obtaining the Linux kernel source code, installing necessary packages, and customizing the kernel through a meticulous exploration of configuration options. Key customization areas include general setup, processor types and features, device drivers, and error resolution. Through detailed documentation, the project provides step-by-step guidance on kernel customization, including resolving encountered errors and updating the GRUB bootloader. Furthermore, applications of the customized kernel extend to various domains, including embedded systems, security enhancements, real-time systems, education, research, and experimentation.

Looking forward, the project envisions further exploration into kernel development by leveraging C programming language and assembly language to develop a kernel from scratch. Additionally, virtual environments like Dockerfile and QEMU will be employed for testing and debugging purposes. By delving deeper into kernel development, the project aims to broaden its scope and contribute to advancing knowledge in operating system concepts, thereby addressing real-world challenges and fostering innovation in the field.



```
ubuntu@ubuntu:~$ uname -a
Linux ubuntu 4.15.0-29-generic #31~16.04.1-Ubuntu SMP Wed Jul 18 08:54:04 UTC 20
18 x86_64 x86_64 x86_64 GNU/Linux
ubuntu@ubuntu:~$ inxi -G
Graphics:  Card: InnoTek Systemberatung VirtualBox Graphics Adapter
           Display Server: X.Org 1.19.6 drivers: (unloaded: fbdev,vesa)
           Resolution: 1416x864@59.97hz
           GLX Renderer: llvmpipe (LLVM 6.0, 256 bits)
           GLX Version: 3.0 Mesa 18.0.5
ubuntu@ubuntu:~$ cat /etc/lsb-release
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=16.04
DISTRIB_CODENAME=xenial
DISTRIB_DESCRIPTION="Ubuntu 16.04.5 LTS"
ubuntu@ubuntu:~$ 
```

OBJECTIVES

These are the 5 objectives we plan to accomplish by developing our own Linux kernel:

1. Optimized System Performance:Fine-tune the Linux kernel to maximize performance on specific hardware, resulting in improved system responsiveness and efficiency.

2. Tailored Feature Set: Customize the kernel to include only necessary features, reducing unnecessary overhead and potentially enhancing security by minimizing attack surfaces.

3. Hardware Compatibility:Ensure seamless compatibility with specific hardware components by configuring the kernel to support and optimize for the targeted CPU architecture and peripherals.

4. Educational Understanding:Develop a deep understanding of kernel configuration options, enabling enhanced knowledge about the inner workings of the Linux kernel and empowering users to make informed choices.

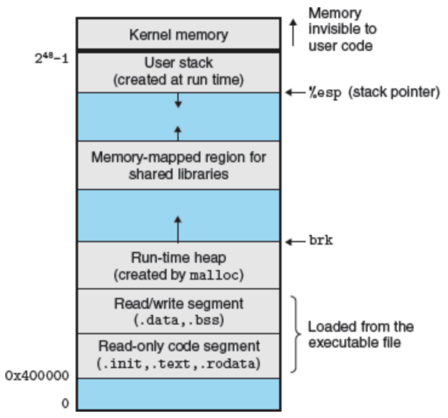
5. Documentation for Reproducibility:Create comprehensive documentation for the customization process, allowing for easy replication and sharing of the optimized kernel configurations with the community or team members.

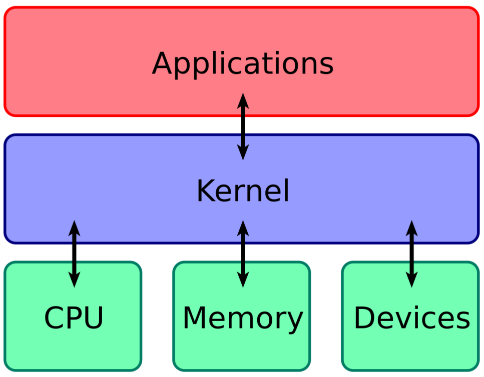
Problem Statement

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

Relevance to the Course

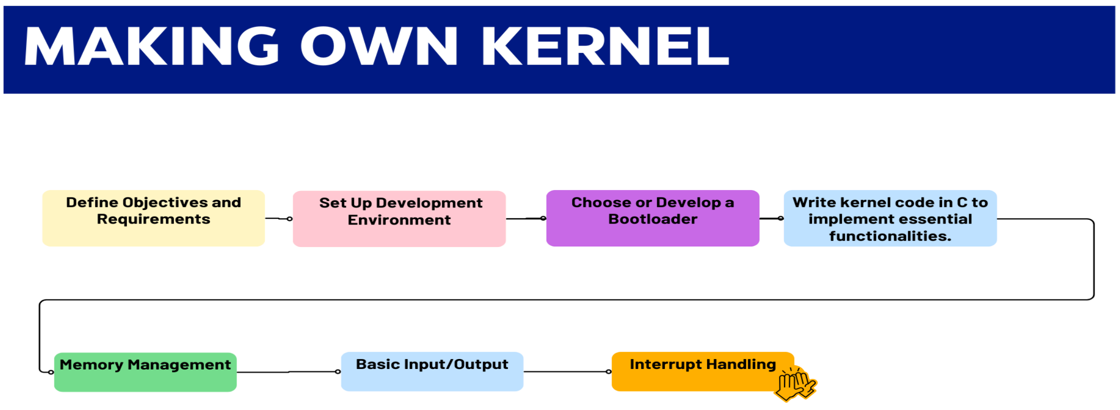
Developing a custom kernel is paramount in an operating systems course, bridging theoretical knowledge with hands-on expertise. It provides students with a unique opportunity to delve into low-level system operations, memory management, and hardware interaction. This practical experience fosters a comprehensive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


Kernel memory
Memory invisible to user code
2^48−1
User stack (created at run time)
%esp (stack pointer)
Memory-mapped region for shared libraries
brk
Run-time heap (created by malloc)
Read/write segment (.data,.bss)
Read-only code segment (.init,.text,.rodata)
Loaded from the executable file
0x400000
0



Introduction

Customizing your own Linux kernel using the .config file opens up a realm of possibilities for optimizing system performance, tailoring feature sets, and enhancing overall hardware compatibility. In this project, the primary objectives revolve around empowering users to sculpt a Linux kernel that precisely aligns with their specific needs. The customization process involves a meticulous exploration of kernel configuration options, providing an educational opportunity to delve into the intricate workings of the Linux kernel. By optimizing the kernel for targeted hardware, users can unlock heightened system responsiveness and efficiency. Additionally, the deliberate inclusion or exclusion of features not only streamlines functionality but also contributes to potential security enhancements. Through thorough documentation of the customization journey, this project not only facilitates the creation of a personalized and high-performing Linux kernel but also encourages knowledge-sharing within the community or among team members.


MAKING OWN KERNEL
Define Objectives and Requirements
Set Up Development Environment
Choose or Develop a Bootloader
Write kernel code in C to implement essential functionalities.
Memory Management
Basic Input/Output
Interrupt Handling

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

1. Obtain the Linux kernel source code from the official website.
2. Use the terminal and employ the wget command to download the Linux kernel source code.
3. Extract the downloaded source code.
4. Install the necessary packages: git fakeroot build-essential ncurses-dev xz-utils libssl-dev bc flex libelf-dev bison.
5. Navigate to the linux-6.7.4 directory using the cd command.
6. Copy the existing Linux config file using the cp command, and then execute make menuconfig.
7. Build the kernel using the make command.
8. Execute sudo make modules_install and sudo make install.
9. Update the initramfs to the installed kernel version 6.7.4.
10. Update the GRUB bootloader.
11. Confirm the kernel version using the uname command.

For the manual installation of the Linux kernel version:
1. Use cd to set the current working directory to the Linux kernel version installed manually.
2. Employ ls -a to check directories for the presence of the .config file.
3. Install the ncurses library using sudo apt install libncurses-dev.
4. Install flex using sudo apt-get install flex.

Kernel customization:
1. In General Setup, modify the kernel compression mode from GZip to ZTSD for faster boot time.
2. Disable POSIX Messaging queue and auditing support.
3. Enable periodic timer ticks for improved performance.
4. Disable BSD process accounting.

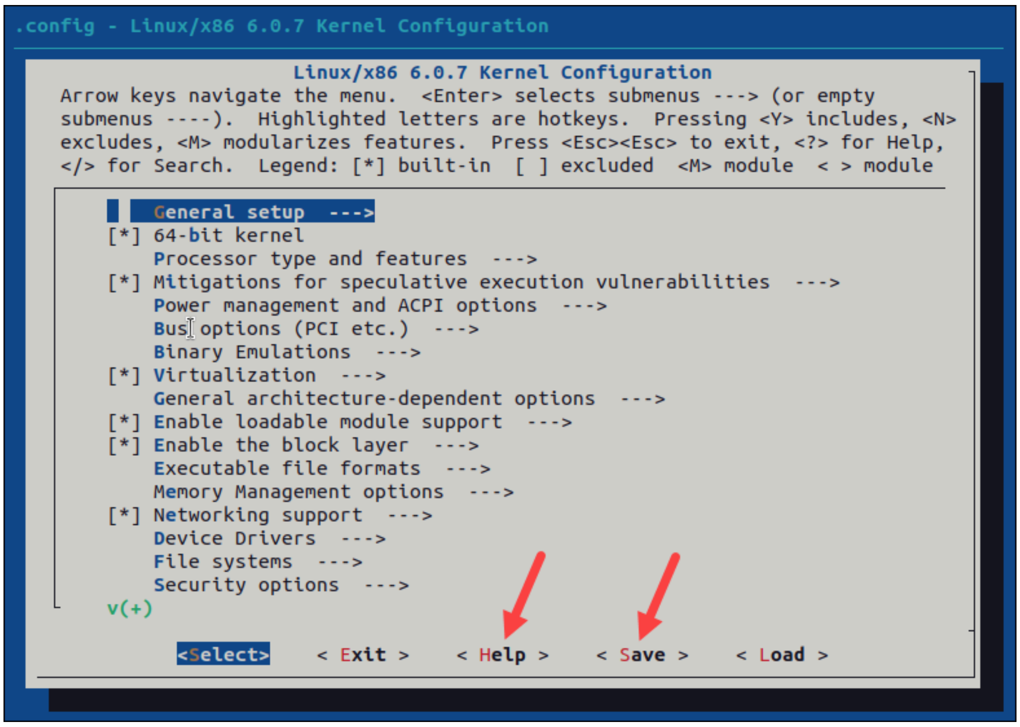
5. Adjust the kernel log buffer size to 16.
6. Retain initram support only for ZSTD.

Processor Types and Features:
1. Enable Intel if using an Intel device and change the processor family to core 2.
2. In the block layer, disable block layer debugging, intended only for kernel developers.

Device Drivers:
1. Disable PC Card support, enable NVME, and disable most Miscellaneous Drives.
2. Enable Asynchronous SCSI Scanning for quicker booting.
3. Disable Macintosh device drivers.
4. Enable WireGuard secure network tunnel for VPN and multimedia support.
5. Change the maximum number of GPUs to 2.
6. Enable laptop hybrid graphics and Intel sound card support.
7. Enable ASUS laptop extras if an ASUS laptop is used for demonstration.
8. Disable Miscellaneous File Systems.
9. Compile using make -j32 && sudo make modules_install -j32.

Encountered error "openssl no such file or directory":
1. Resolve by executing sudo apt-get install libssl-dev.
2. Execute make bzImage.
3. Use ls /boot/ to locate vmlinuz.

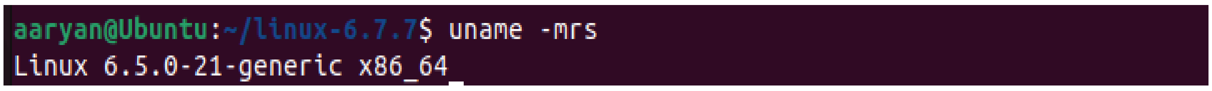
Update the GRUB bootloader:
1. Execute sudo update-initramfs -c -k 6.0.7.
2. Update GRUB using sudo update-grub.



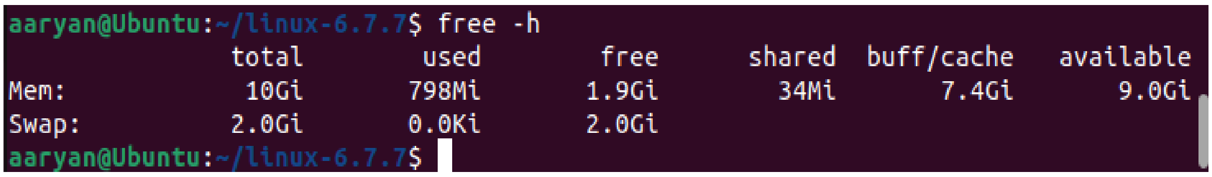
```
.config - Linux/x86 6.0.7 Kernel Configuration
                      Linux/x86 6.0.7 Kernel Configuration
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty
  submenus ----).  Highlighted letters are hotkeys.  Pressing <Y> includes, <N>
  excludes, <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help,
  </> for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < > module

        General setup  --->
    [*] 64-bit kernel
        Processor type and features  --->
    [*] Mitigations for speculative execution vulnerabilities  --->
        Power management and ACPI options  --->
        Bus options (PCI etc.)  --->
        Binary Emulations  --->
    [*] Virtualization  --->
        General architecture-dependent options  --->
    [*] Enable loadable module support  --->
    [*] Enable the block layer  --->
        Executable file formats  --->
        Memory Management options  --->
    [*] Networking support  --->
        Device Drivers  --->
        File systems  --->
        Security options  --->
    v(+)

          <Select>    < Exit >    < Help >    < Save >    < Load >
```



```
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$
```

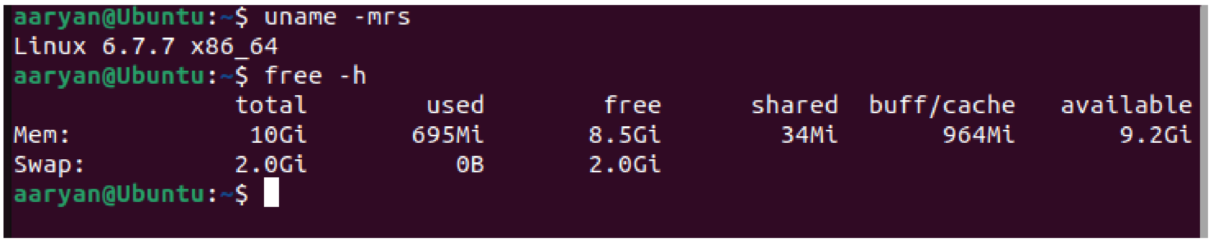
Results

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



```
aaryan@Ubuntu:~$ uname -mrs
Linux 6.7.7 x86_64
aaryan@Ubuntu:~$ free -h
               total        used        free      shared  buff/cache   available
Mem:            10Gi       695Mi       8.5Gi        34Mi       964Mi       9.2Gi
Swap:          2.0Gi          0B       2.0Gi
aaryan@Ubuntu:~$
```

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,

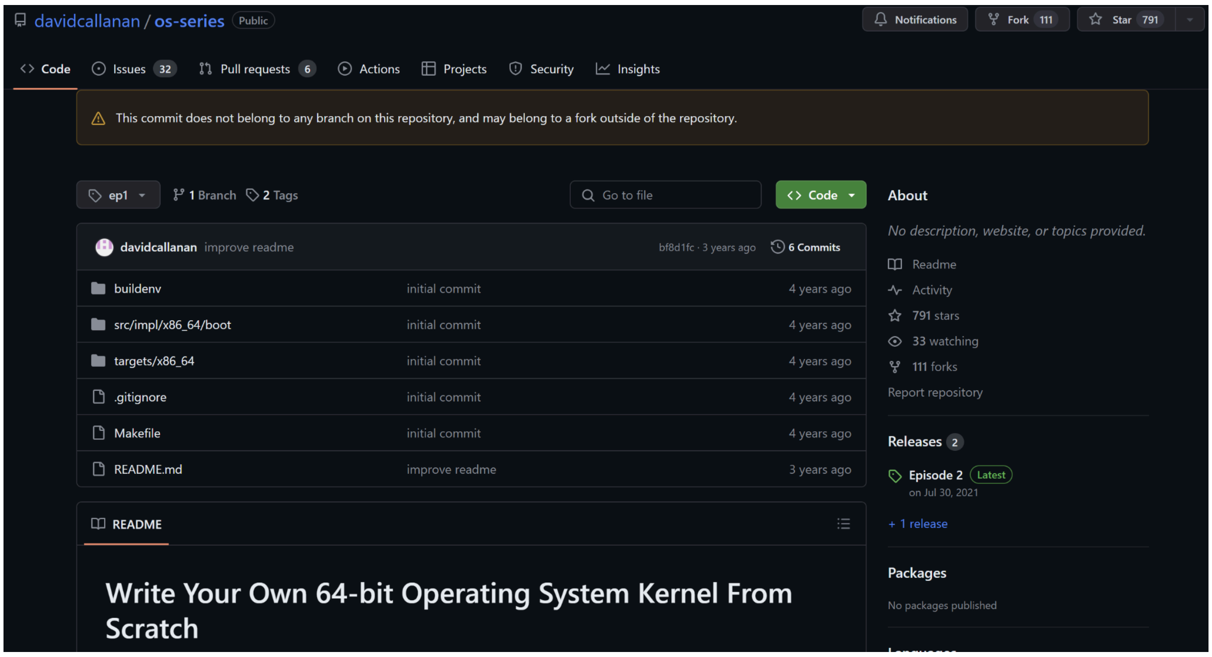
5. Educational Purposes: Building a kernel from scratch is an excellent educational exercise, providing insights into low-level system operations, memory management, fostering a deeper understanding of operating system concepts.

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

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


davidcallanan / os-series Public

Notifications | Fork 111 | Star 791

Code | Issues 32 | Pull requests 6 | Actions | Projects | Security | Insights

This commit does not belong to any branch on this repository, and may belong to a fork outside of the repository.

ep1 | 1 Branch | 2 Tags | Go to file | Code

davidcallanan improve readme — bf8d1fc · 3 years ago — 6 Commits

buildenv	initial commit	4 years ago
src/impl/x86_64/boot	initial commit	4 years ago
targets/x86_64	initial commit	4 years ago
.gitignore	initial commit	4 years ago
Makefile	initial commit	4 years ago
README.md	improve readme	3 years ago

README

Write Your Own 64-bit Operating System Kernel From Scratch

About

No description, website, or topics provided.

Readme
Activity
791 stars
33 watching
111 forks
Report repository

Releases 2

Episode 2 Latest
on Jul 30, 2021

+ 1 release

Packages

No packages published