313551137 官劉翔

1. **Compiling the Linux Kernel**

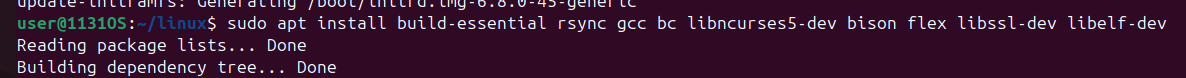
I will follow the following steps to implement:

* + 1. Install tool package
    2. Update Grub configuration.
    3. Copy existing Linux kernel config file.
    4. Custom kernel
    5. Compile and build Linux kernel
    6. Install Linux kernel and modules (drivers)
    7. Reboot

1. Install tool package

First, I ensured all necessary tools were installed on my Ubuntu system by using the command:

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



1. Update Grub configuration

To allow more time for selecting the custom kernel during boot, I modified the GRUB timeout:



I changed the GRUB\_TIMEOUT value to 15 seconds and ignored the GRUB\_TIMEOUT\_STYLE=hidden command

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

After saving the change, I updated GRUB

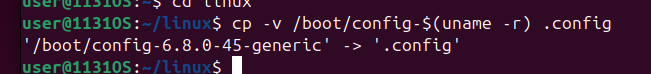
一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

When we reboot, we can have 15 second to select which kernel we want to use.

1. Copy existing Linux kernel config file.

To start with a known working configuration, I copied the existing kernel config to .config:



This step ensures that the new kernel will support the existing hardware and maintain system compatibility.

Starting from scratch could lead to missing critical drivers or features, potentially rendering the system unbootable.

1. Custom kernel

I customized the kernel configuration by:

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

In the configuration menu, I navigated to "General setup" -> "Local version - append to kernel release" and set it to "-os-313551137".

一張含有 文字, 螢幕擷取畫面, 電腦, 軟體 的圖片

自動產生的描述

一張含有 文字, 軟體, 電腦, 電腦圖示 的圖片

自動產生的描述

1. Compile and build Linux kernel

I experimented with three different approaches for this critical step:

Method1(error):

Initially, I attempted to use the make deb-pkg command, which is designed to create Debian packages for the kernel. However, this method encountered errors related to certificate files:

一張含有 文字, 字型, 螢幕擷取畫面, 圖形 的圖片

自動產生的描述

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

Even after disabling related configurations in menuconfig, the errors persisted. 一張含有 文字, 螢幕擷取畫面, 軟體, 電腦 的圖片

自動產生的描述

Method2(error?):

Next, I tried using make defconfig to generate a fresh configuration, followed by make -j

make defconfig

make -j$(nproc)

This method resets the kernel configuration to default values, which can be beneficial for starting with a clean slate. However, it resulted in an error in Makefile, and I think it is because default might be incompatible with our specific system or compilation requirements.

一張含有 文字, 字型, 螢幕擷取畫面, 圖形 的圖片

自動產生的描述

Although I change the Makefile and force it to ignore some error by CFLAGS\_KERNEL += -Wno-format -Wno-array-bounds.

一張含有 文字, 電子產品, 螢幕擷取畫面, 軟體 的圖片

自動產生的描述

It compile successfully. However, I don’t know whether this change effect the program in the future, so I changed to Method 3.

一張含有 文字, 字型, 螢幕擷取畫面, 圖形 的圖片

自動產生的描述

Method3(successful):

This method worked because it combined a known working configuration with targeted disabling of problematic features. The SYSTEM\_TRUSTED\_KEYS and SYSTEM\_REVOCATION\_KEYS options, which were causing issues in Method 1, were disabled without compromising the overall kernel functionality. Thus, I execute the following command.

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

Finally, it is working. I was crying and successfully compiled the kernel.

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

1. Install Linux kernel and modules:

After successful compilation,

* + - 1. Install the required modules:

sudo make modules\_install

* + - 1. Install the kernel by typing:

sudo make install

1. Reboot and Result:

Finally, I rebooted the system to load the new kernel.

After I reboot, the selection menu show

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

In the end, I check my kernel version.

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

1. **Implementing a new System Calls**

Step1~4 execute in ~/linux.

* 1. Kernel Implementation:

In **kernel/sys.c,** implement the system call:

Besides, I also add some code to prevent common errors happening

一張含有 文字, 螢幕擷取畫面 的圖片

自動產生的描述

* + - 1. SYSCALL\_DEFINE2(NR\_revstr, char \_\_user \*, str, size\_t, n):
         1. This macro defines a system call named NR\_revstr with two arguments.
         2. char \_\_user \*str: A pointer to a user-space string that needs to be reversed.
         3. size\_t n: The length of the string.
      2. Variable declarations:
         1. char \*k\_str: A kernel-space buffer to store the string.
         2. int i, j: Loop counters for string reversal.
         3. char temp: Temporary variable for swapping characters.
      3. k\_str = kmalloc(n + 1, GFP\_KERNEL):
         1. Allocates n + 1 bytes of memory in kernel space.

(The extra byte is for the null terminator.)

* + - * 1. GFP\_KERNEL is the allocation flag for normal kernel allocations.
      1. if (!k\_str) return -ENOMEM:
         1. If memory allocation fails, return an "Out of memory" error.
      2. if (copy\_from\_user(k\_str, str, n)) { ... }:
         1. Copies n bytes from the user-space string to the kernel-space buffer.
         2. If copying fails, free the allocated memory and return an error.
      3. k\_str[n] = '\0':
         1. Null-terminates the kernel-space string.
      4. printk(KERN\_INFO "The origin string: %s\n", k\_str):
         1. Logs the original string to the kernel log.
      5. String reversal loop:
         1. for (i = 0, j = n - 1; i < j; i++, j--) {
         2. Reverses the string by swapping characters from both ends towards the middle.
      6. printk(KERN\_INFO "The reversed string: %s\n", k\_str):
         1. Logs the reversed string to the kernel log.
      7. if (copy\_to\_user(str, k\_str, n)) { ... }:
         1. Copies the reversed string back to user space.
         2. If copying fails, free the allocated memory and return an error.
      8. kfree(k\_str):
         1. Frees the allocated kernel memory.
      9. return 0:
         1. Returns 0 to indicate successful execution of the system call.
  1. System Call Prototype:

Add the system call prototype in **include/linux/syscalls.h**

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

* 1. Set system Call Table Entry:

Based on the context of the kernel document for x86

一張含有 文字, 字型, 螢幕擷取畫面, 白色 的圖片

自動產生的描述

Add the system call number in **arch/x86/entry/syscalls/syscall\_64.tbl**:

一張含有 文字, 螢幕擷取畫面, 功能表, 字型 的圖片

自動產生的描述

We set the new system call in 451 entry.

* 1. Re-compile kernel:

I use the method in Part I step3~step7. Recap:

Step 3: cp -v /boot/config-$(uname -r) .config

Step 4: make menuconfig

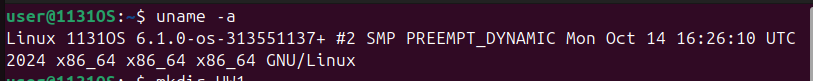
Step 5: make -j$(nproc)

Step 6: sudo make modules\_install

sudo make install

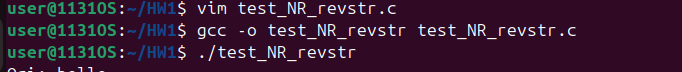
Step 7: reboot

Recompile result



* 1. Create && Compile && Run the Test Program:

I use vim to create a test program test\_NR\_revstr.c



Running Test result:

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

* 1. Check Kernel Logs

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

1. Patch
   1. Check initial status:

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

* 1. Stage all changes:

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

* + 1. Git add .:

This command stages all modified and new files in the current directory and its subdirectories. The dot (.) means "current directory and everything below it".

* + 1. git commit -m "Add revstr system call":

This makes a new commit with all your staged changes.

* + 1. git format-patch -1 HEAD:

creates a patch file for the most recent commit. The -1 flag means "create a patch for the last 1 commit", and HEAD refers to the most recent commit on the current branch.

* 1. Final check:

一張含有 文字, 字型, 螢幕擷取畫面 的圖片

自動產生的描述