Assignment 1: Compiling Linux Kernel and Adding Custom System Calls

Student: `109550132吳念蓉`

Part 1: kernel compilation

Part 2: system call

- 把linux-5.19.12資料夾放在/root 下，用root 權限操作

1. 建立一個資料夾，放自定義的`syscall`
    - `$ cd linux-5.19.12`
    - `$ mkdir hello`

- `$ mkdir revstr`

2. 進去資料夾，按照需求寫自己定義的 `syscall`

- `$ vim hello/hello.c`

```
# include <linux/kernel.h>

asmlinkage long __x64_sys_hello(void)
{
    printk(KERN_INFO "Hello, World!\n");
    printk(KERN_INFO "109550132\n");
    return 0;
}
```

- `$ vim revstr/revstr.c`

  `SYSCALL_DEFINEn` 後面的n，為這個function所需要的參數數量，先從userspace拿到string，revserse後再傳回去

```
#include <linux/kernel.h>
#include <linux/syscalls.h>
#include <linux/uaccess.h>

SYSCALL_DEFINE2(revstr, size_t, length, char __user *, str)
{
    char *buffer;
    buffer = kmalloc(length+1, GFP_KERNEL);

    if (copy_from_user(buffer, str, length) < 0)  {
        kfree(buffer);
        return -EFAULT;
    }

    pr_info("The origin string: %s\n", buffer);

    // reverse the string
    for (int i = 0; i < length / 2; i++) {
        char temp = buffer[i];
        buffer[i] = buffer[length - i - 1];
        buffer[length - i - 1] = temp;
    }
    buffer[length] = '\0';

    pr_info("The reversed string: %s\n", buffer);

    if (copy_to_user(str, buffer, length) < 0) {
        kfree(buffer);
        return -EFAULT;
    }
```

```
    kfree(buffer);
    return 0;
```

3. 再建立一個 `Makefile`
   - `$ vim hello/Makefile`

     ```
     obj-y := hello.o
     ```

   - `$ vim revstr/Makefile`

     ```
     obj-y := revstr.o
     ```

4. 修改 kernel 的 `Makefile` 來告訴 compiler 新的 system call 可以在哪個位置找到
   - `$ vim Makefile`
   - 找到 `core-y` 並在最後面添加之前所建立的資料夾，這樣 kernel 編譯時才能找到我們自己定義的資料夾

     ```
     ...
     core-y += kernel/ certs/ mm/ fs/ ipc/ security/ crypto/ block/ hello/ revstr/
     ...
     ```

5. 修改 system call entry table
   - `$ vim arch/x86/entry/syscalls/syscall_64.tbl`
   - 在最後一行上沒有被分配到的entry添加自定義的 `syscall`，這裡的話需要以 `sys_` 作為對應的system call 開頭，按照上方定義的格式填入，在最上面的comment有說明

     ‘*The __64_sys_*() stubs are created on-the-fly for sys_*() system calls*’

     ```
     <number> <abi> <name> <entry_point>
     ...
     451 common  hello        sys_hello
     452 common  revstr       sys_revstr
     ```

6. 在 system call header file 上添加我們自己定義的 `syscall` prototype

- `$ vim include/linux/syscalls.h` 添加在asmlinkage後方，須以`__x64_sys_`作為開頭

```
...
asmlinkage long __x64_sys_hello(void);
asmlinkage long __x64_sys_revstr(void, size_t length, char __user *str);
```

7. 重新編譯kernel，並重新開機。

   (可以用`nproc`看一下有幾個處理器可用)

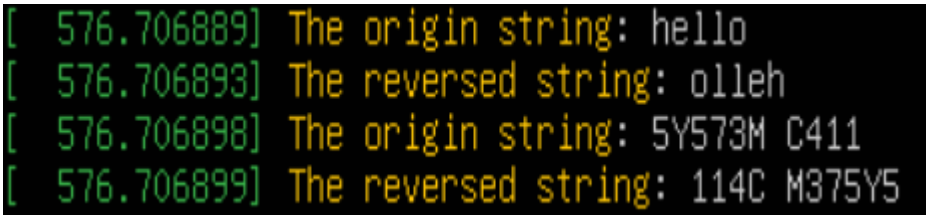
   - `$ make -j 4`
   - `$ make modules_install -j 4`
   - `$ make install -j 4`
   - `$ reboot`

8. 登入並測試程式

   - 因為我們在entry table 設定值為451、452，所以我們可以看到我們__NR_hello、__NR_revstr也對應到相同數字，所以只要將助教給的code改成自己的system call number ，即可編譯與執行成功

9. 結果 `screen shot`

   - (這裡hello world格式上，spec的圖片說明與文字有點小差異，我是按照文字說明)
   - `$ sudo dmesg`



- 一些過程中遇到的狀況

1. No space left on device 已經按照spec去擴充空間，但因為舊的kernel太佔空間，需到/boot 刪除舊的kernel
2. undefine reference to ‘sys_hello’: From linux kernel v4.17 onwards, x86_64 system calls may begin with "__x64_sys".