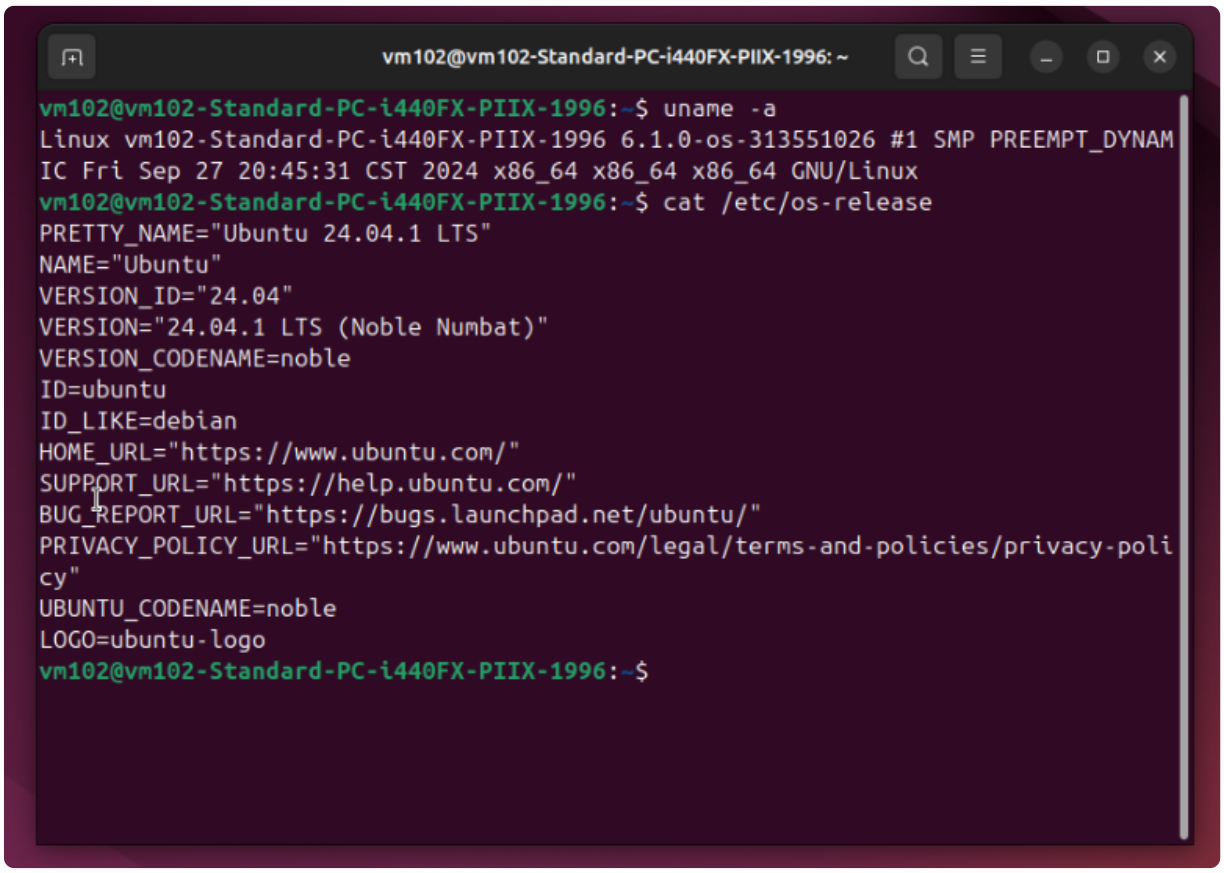
📖 2024-Fall Operating System Assignment I

> This page demonstrates my answer to the quiz in HW1, for further detail of this assignment, please check this link

- Student Name : 許登豪
- Student ID : 313551026

💻 Part 1: Compiling the Linux Kernel

📷 Successful Kernel Compilation Screenshot:



📝 Steps To Compile :

1. Run the command in home directory to clone the v6.1 branch :
   `git clone https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git --depth=1 --branch v6.1 --single-branch`

2. change directory to the cloned folder `~/linux` :
   `cd linux`
3. go to https://gitlab.archlinux.org/archlinux/packaging/packages/linux/-/blob/main/config and copy this `.config` file to `~/linux`
4. modify this `.config` file, and the following entries need to be set or check to succesfully compile my own version of kernel :

```
CONFIG_LOCALVERSION="-os-313551026"
CONFIG_LOCALVERSION_AUTO=y
CONFIG_SYSTEM_TRUSTED_KEYS=""
CONFIG_SYSTEM_REVOCATION_KEYS=""
```

5. Run the command in `~/linux` to accept the defaults of every configuration option that has changed between versions :

```
make olddefconfig
```

6. Run the command in `~/linux` to compile my own kernel :

```
sudo make -j4
```

7. Run the command in `~/linux` to build and install the kernel modules :

```
sudo make modules_install install
```

Note that `make modules_install` installs the kernel modules that have been built during the kernel compilation process. And `make install` installs the kernel itself along with the associated files (like the vmlinuz image, System.map, and initrd/initramfs) to the appropriate locations. The kernel image is copied to `/boot/`, and the bootloader configuration (e.g., GRUB) is updated to include the new kernel as a boot option.

8. Run the command to reboot :

```
sudo reboot
```

9. Hold down the shift key until the GRUB list shows up on the screen

10. Select the entries whose name ends with "-os-313551026" and press enter to launch the new kernel

💻 Part 2: Implementing a new System Calls

📷 Successful System Call Implementation Screenshot:

```
vm102@vm102-Standard-PC-i440FX-PIIX-1996: ~
vm102@vm102-Standard-PC-i440FX-PIIX-1996:~$ cat revstr_test.c
#include <unistd.h>

#include <string.h>
#include <stdio.h>
#include <assert.h>

#define __NR_revstr 451

int main(int argc, char *argv[]) {

    char str1[20] = "hello";
    printf("Ori: %s\n", str1);
    int ret1 = syscall(__NR_revstr, str1, strlen(str1));
    assert(ret1 == 0);
    printf("Rev: %s\n", str1);

    char str2[20] = "Operating System";
    printf("Ori: %s\n", str2);
    int ret2 = syscall(__NR_revstr, str2, strlen(str2));
    assert(ret2 == 0);
    printf("Rev: %s\n", str2);

    return 0;
}
vm102@vm102-Standard-PC-i440FX-PIIX-1996:~$ gcc revstr_test.c -o revstr_test
vm102@vm102-Standard-PC-i440FX-PIIX-1996:~$ ./revstr_test
Ori: hello
Rev: olleh
Ori: Operating System
Rev: metsyS gnitarepO
```

```
[ 1108.050164] The origin string: hello
[ 1108.050165] The reversed string: olleh
[ 1108.050168] The origin string: Operating System
[ 1108.050169] The reversed string: metsyS gnitarepO
```

📝 Steps To Implement :

1. create a new folder under `~/linux`, let's say it is `~/linux/revstr`

2. under `~/linux/revstr`, create a file `revstr.c` and with the following content :

```
vm102@vm102-Standard-PC-i440FX-PIIX-1996:~$ cat ~/linux/revstr/revstr.c
#include <linux/kernel.h>
#include <linux/syscalls.h>
#include <linux/uaccess.h>
#include <linux/slab.h>

SYSCALL_DEFINE2(revstr, char __user *, str, int, len){
    char *str_buf = kmalloc(len + 1, GFP_KERNEL); // +1 for '\0'
    if (!str_buf) {
        return -ENOMEM;
    }

    if (copy_from_user(str_buf, str, len)) {
        kfree(str_buf);
        return -EFAULT;
    }

    str_buf[len] = '\0';

    printk(KERN_INFO "The original string: %s\n", str_buf);

    char temp;
    for (int i = 0; i < len / 2; i++) {
        temp = str_buf[i];
        str_buf[i] = str_buf[len - i - 1];
        str_buf[len - i - 1] = temp;
    }

    printk(KERN_INFO "The reversed string: %s\n", str_buf);
```

```
    char temp;
    for (int i = 0; i < len / 2; i++) {
        temp = str_buf[i];
        str_buf[i] = str_buf[len - i - 1];
        str_buf[len - i - 1] = temp;
    }

    printk(KERN_INFO "The reversed string: %s\n", str_buf);

    if (copy_to_user(str, str_buf, len)) {
        kfree(str_buf);
        return -EFAULT;
    }

    kfree(str_buf);

    return 0;
}
```

That's break down the implementation :

`#include <linux/kernel.h>`

This header provides kernel-related macros, functions, and definitions, such as the `printk()` function used for logging messages to the kernel log.

`#include <linux/syscalls.h>`

This header declares system call-related macros and functions, like `SYSCALL_DEFINE2`.

`#include <linux/uaccess.h>`

This header provides functions for safely copying data between user space and kernel space. It includes functions like `copy_from_user()` and `copy_to_user()`.

`#include <linux/slab.h>`

This header provides memory allocation functions like `kmalloc()` and `kfree()`.

`SYSCALL_DEFINE2(revstr, char __user *, str, int, len)`

Defines a system call named `revstr` that takes two arguments : A pointer to a user-space string `str`, the length of the string `len`.

`char* str_buf = kmalloc(len + 1, GFP_KERNEL)`

Allocates kernel memory buffer `str_buf` to store the string, with an extra byte for the null terminator.

`copy_from_user(str_buf, str, len)`

Copies the string from user space to kernel space. If it fails, the allocated memory is freed, and the function returns an -EFAULT.

`str_buf[len] = '\0'`

Appends a null terminator to the copied string for safe printing.

`printk(KERN_INFO "The original string: %s\n", str_buf)`

Logs the original string to the kernel log, which will appear in kernel ring buffer, maintained in memory and is shared between user space and the kernel.(reference : https://lwn.net/Articles/976836/)

`for (int i = 0; i < len/2; i++){...}`

A loop swaps characters at opposite ends of the string to reverse it.

`printk(KERN_INFO "The reversed string: %s\n", str_buf)`

Logs the reversed string, which will appear in kernel ring buffer.

`copy_to_user(str, str_buf, len)`

Copies the reversed string back to user space. If it fails, it frees the memory and returns an error.

`kfree(str_buf)`

The allocated memory is freed using kfree, and the system call returns 0 to indicate success.

3. under `~/linux/revstr`, create a Makefile and with the following content, to ensure that `revstr.c` is compiled and included in the kernel source code :

```
vm102@vm102-Standard-PC-i440FX-PIIX-1996:~$ cat ~/linux/revstr/Makefile
obj-y := revstr.o
vm102@vm102-Standard-PC-i440FX-PIIX-1996:~$
```

4. add `revstr/` in the Makefile under `~/linux`, to tell the compiler that the source files of my new system call is in the `~/linux/revstr` directory.

```
ifdef need-config
include include/config/auto.conf
endif

ifeq ($(KBUILD_EXTMOD),)
# Objects we will link into vmlinux / subdirs we need to visit
core-y          := revstr/
drivers-y       :=
libs-y          := lib/
endif # KBUILD_EXTMOD
```

5. register a system call number (in this example, 451) in `~/linux/arch/x86/entry/syscalls/syscall_64.tbl`, because my VM is 64 bits system :

```
438     common  pidfd_getfd             sys_pidfd_getfd
439     common  faccessat2              sys_faccessat2
440     common  process_madvise         sys_process_madvise
441     common  epoll_pwait2            sys_epoll_pwait2
442     common  mount_setattr           sys_mount_setattr
443     common  quotactl_fd             sys_quotactl_fd
444     common  landlock_create_ruleset sys_landlock_create_ruleset
445     common  landlock_add_rule       sys_landlock_add_rule
446     common  landlock_restrict_self  sys_landlock_restrict_self
447     common  memfd_secret            sys_memfd_secret
448     common  process_mrelease        sys_process_mrelease
449     common  futex_waitv             sys_futex_waitv
450     common  set_mempolicy_home_node sys_set_mempolicy_home_node
451     common  revstr                  sys_revstr
#
# Due to a historical design error, certain syscalls are numbered differently
```

6. add new system call to the system call header file, which locates at `~/linux/include/linux/syscalls.h` :

```
long __do_semtimedop(in
nsops,
                     const struct timespec64 *timeout,
                     struct ipc_namespace *ns);

int __sys_getsockopt(int fd, int level, int optname, char __user
*optval,
                int __user *optlen);
int __sys_setsockopt(int fd, int level, int optname, char __user
*optval,
                int optlen);
asmlinkage long sys_revstr(char __user *str, int len);
#endif
```

This defines the prototype of the function of my system call. `asmlinkage` is a key word used to indicate that all parameters of the function would be available on the stack. (reference : https://www.jollen.org/blog/2006/10/_asmlinkage.html)

7. recompile and install the new kernel using the same method mentioned above.