OS HW1

資科工所 312551105 羅羽軒

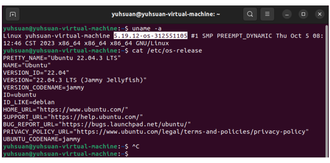
- 文章位址: https://hackmd.io/@lohsuan/rJlenpqg6
- 題目敘述: Compiling Linux Kernel and Adding Custom System Calls

Prob 1: Change kernel suffix

> Before compiling the kernel, you need to change your Linux kernel version suffix to `-os-<your-id>`, as an envidence that the kernel is built by yourself.

HW1-1 詳細過程: https://hackmd.io/@lohsuan/Hk7C4rUWa

Answer screenshot



Prob 2: Adding Custom System Calls

題目敘述: Adding Custom System Calls

> In this part, you are required to implement two system calls on top of the kernel you built. One `sys_hello` and the other one `sys_revstr`.

2-1 `sys_hello`

1. Define new system call: sys_hello()

- `mkdir workspace` under `linux-5.15.12`,
- add custom system call file `hello.c` under `workspace` directory
  - `SYSCALL_DEFINE0`: 0 means no parameter pass in

```
// hello.c
#include <linux/kernel.h>
#include <linux/syscalls.h>
SYSCALL_DEFINE0(hello)
{
 printk("Hello, world!\n");
 printk("312551105\n");
 return 0;
}
```

- Create new `Makefile` in `workspace`, It will
  - specifies the objects to be built
  - added to the source during the next kernel recompilation.

```
// Makefile
obj-y += hello.o
```

> This is to ensure that the hello.c file is compiled and included in the kernel source code.

2. Add `workspace/` to the kernel's `Makefile`

- switch back to `linux-5.19.12` directory
- add `workspace` directory to kernel's `Makefile`
  - find `core-y` line and append `workspace/`
  - this will tell the compiler to find our custom system call soure code in `workspace/` directory

```
ifeq ($(KBUILD_EXTMOD),)
core-y          += kernel/ certs/ mm/ fs/ ipc/ security/ crypto/
core-$(CONFIG_BLOCK)    += block/ workspace/
core-$(CONFIG_IO_URING) += io_uring/
```

> This is to tell the compiler that the source files of our new system call (sys_hello()) are in present in the workspace directory.

3. Add custom system call into the master's syscall table

- To wire up our new system call for x86 platforms, we need to update the master syscall tables.
  - Edit the system call table by `vim arch/x86/entry/syscalls/syscall_64.tbl`
  - add `sys_hello` with number 451

```
451    common   hello       sys_hello
```

4. Add function prototype in the system call header file

- The new entry point needs a corresponding function prototype
  - add it in `include/linux/syscalls.h`
  - marked as `asmlinkage` to match the way that system calls are invoked

```
asmlinkage long sys_hello(void);
```

5. Recompile the kernel

- Switch to the source directory `linux-5.19.12` and execute:

```
$ sudo make -j $(nproc)
$ sudo make modules_install install
```

6. Reboot

- For the system to now use the newly configured kernel, `reboot`.

Test with TA's code

```
// hello_test.c
#include <assert.h>
#include <unistd.h>
#include <sys/syscall.h>

#define __NR_hello 451
/* 451 is the system call number we declare in master's syscall table */
int main(int argc, char *argv[]) {
    int ret = syscall(__NR_hello);
    assert(ret == 0);

    return 0;
}
```

Compile and run the program:

```
$ gcc hello_test.c
$ ./a.out
```

show message with `sudo dmesg`

```
[  598.060929] Hello, world!
[  598.060933] 312551105
yuhsuan@yuhsuan-virtual-machine:~$
```

2-2 `sys_revstr`

The steps are the same as above

1. Define new system call: sys_revstr()

- add custom system call file `revstr.c` under `workspace` directory
  - `SYSCALL_DEFINE2`: 2 means 2 parameters pass in

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

SYSCALL_DEFINE2(revstr, int, len, char __user *, string)
{
    char str[200]; // Declare the size of character string
    unsigned long strlen = len;
    char temp;

    copy_from_user(str, string, strlen);
    printk("The origin string: %s\n", str);

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

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

    return 0;
}
```

- Add `revstr.o` in `Makefile`

```
// Makefile
obj-y += hello.o
obj-y += revstr.o
```

2. Add custom system call into the master's syscall table

- To wire up our new system call for x86 platforms, we need to update the master syscall tables.
  - Edit the system call table by `vim arch/x86/entry/syscalls/syscall_64.tbl`
  - add `sys_revstr` with number 452

```
452    common   revstr       sys_revstr
```

3. Add function prototype in the system call header file

- The new entry point needs a corresponding function prototype
  - add it in `include/linux/syscalls.h`
  - marked as `asmlinkage` to match the way that system calls are invoked

```
asmlinkage long sys_revstr(int len, char __user *string);
```

4. Recompile the kernel

Switch to the source directory `linux-5.19.12` and execute:

```
$ sudo make -j $(nproc)
$ sudo make modules_install install
```

5. Reboot

For the system to now use the newly configured kernel, `reboot`.

Test with TA's code

```
// revstr_test.c
#include <assert.h>
#include <unistd.h>
#include <sys/syscall.h>

#define __NR_revstr 452

int main(int argc, char *argv[]) {
    int ret1 = syscall(__NR_revstr, 5, "hello");
    assert(ret1 == 0);

    int ret2 = syscall(__NR_revstr, 11, "5Y573M C411");
    assert(ret2 == 0);

    return 0;
}
```

Compile and run the program:

```
$ gcc revstr_test.c
$ ./a.out
```

show message with `sudo dmesg`

```
[  517.654880] The origin string: hello
[  517.654880] The reversed string: olleh
[  517.654964] The origin string: 5Y573M C411
[  517.654965] The reversed string: 114C M375Y5
yuhsuan@yuhsuan-virtual-machine:~/ntnuclass/linux-5.19.12/workspace$
```