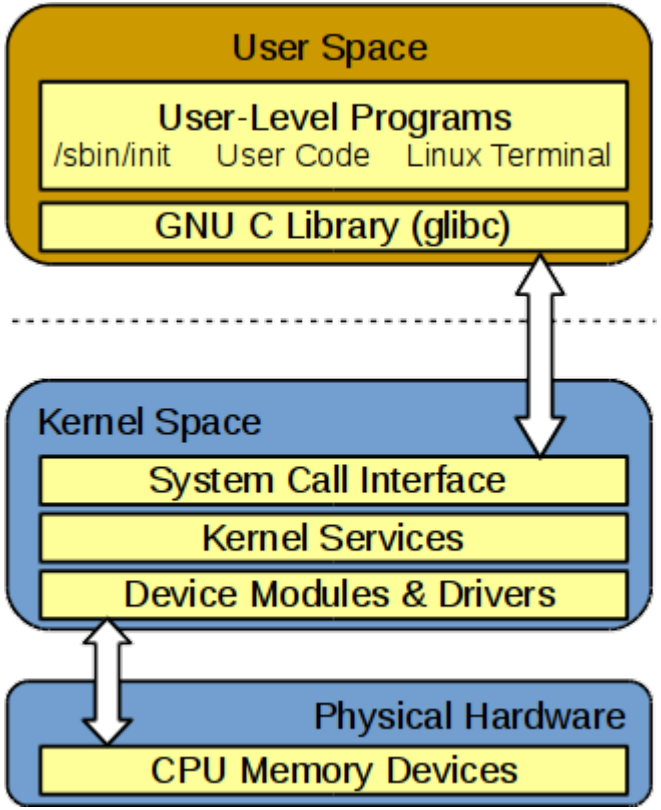
Linux Kernel Training

Kernel Module Interfaces

Agenda

1. System calls
2. Module interfaces
3. procfs
4. sysfs
5. devfs

System calls



API syscall-ов

All system function has implemented as wrapper over system call. The simplest way of call system function directly is call function:

```
#include <unistd.h>
#include <sys/syscall.h>

int syscall(int number, ...);
```

Для ARM64

```
#include <unistd32.h>
```

для x86

```
#include <unistd.h>
```

syscalls_32.h находится в папке собираемого ядра

/arch/x86/include/generated/asm/syscalls_32.h

содержит

```
__SYSCALL_I386(NR, name)
```

значение числа системных вызовов хранится в постоянной NR_syscalls

syscall table

arch/i386/kernel/syscall_table.S

Обработка системных вызовов

Low level implementation for system call on the some platform:

arm/OABI arm/EABI i386 x86_64	swi NR swi 0x0 int $0x80 syscall

См.: man syscall, man syscalls

File API

Communication between user and kernel space using files interface

- **`include/linux/fs.h`**
  - `struct file_operations`
  - `struct inode_operations`

The procfs filesystem

- **`include/linux/proc_fs.h`**
  - procfs API
- **`fs/proc/internal.h`**
  - struct proc_dir_entry

Sysfs file system

```
/sys
├── block
├── bus
├── class
├── dev
├── devices
├── firmware
├── fs
├── hypervisor
├── kernel
├── module
└── power
```

Sysfs file system

For udev services and devices automount

- **`include/linux/kobject.h`**
  - struct kobject
- **`include/linux/sysfs.h`**
  - struct sysfs_ops
    - `show()` - read data
    - `store()` - write data
  - struct attribute
- **`include/linux/device.h`** - high-level API

Device Filesystem

- `include/linux/device.h`
- `include/linux/cdev.h`

При "классическом" подходе в каталоге /dev - сотни специальных файлов

Device management

Major number (unsigned 8-bit integer) --> register_blkdev(), register_chrdev()

Проблемы Major number:

- ограниченное число
- процесс получения
- процесс выделения
- администрирование

Device Filesystem

Позволяет
автоматизировать процесс
получения Major number

Practice

1 Example proc_rw - development

Writing from user space to kernel and reading information from kernel space

1. Redefinition of only needed file_operations (usually ‘read’, ‘write’, ‘lseek’, ‘ioctl’)
2. Creation of dir/file in /proc:
    - proc_mkdir() - create directory
    - proc_create() - create file
1. Reading/writing data
    - Calculation and storing of offset position
    - copy_to_user() / copy_from_user()

open, check makefile

`CFLAGS :=`

`-m32` (для кросс компиляции и запуска в QEMU)

`-static` (shared libs from Ubuntu))

check environment, paths
check, install libc6-dev-i386

make kernel modules

```
$ make
```

check errors

```
$ run_new_quemu
```

copy keys
connect to virtual sys

copy compiled modules:

```
$ scp mp mplib mpsys *.ko myLinux:work/
```

connect to virtual host:

```
$ ssh myLinux
$ cd work/
$ ll
```

mplib.c (user-space) — пример обычной Си программы
mpsys.c — вместо write используется syscall с номером для write, аналогично с mknod и getpid
mp.c — аналогичен mpsys.c, делает то же что и syscall, только напрямую, через ассемблерную вставку

запускаем:

```
$ sudo ./mplib
$ ll
$ sudo rm -f ZZZ

$ sudo ./mpsys
$ sudo rm -f ZZZ
```

```
$ sudo ./mp
$ sudo rm -f ZZZ
```

mdu.c аналогичен mp.c для запуска syscall-ов из kernel space
mdc.c

check makefile

```
$ make
$ dmesg -c
$ sudo insmod mdu.ko
$ dmesg -c
```

(выводится ошибка, что нормально, смотри код return -1)

2 Example proc_rw — execution

Loading:

```
$ insmod ‘example’
```

```
$ echo ‘data’ > /proc/example/buffer
$ cat /proc/example/buffer
```

Unloading:

```
$ sudo rmmod ‘example’
```

Home reading

1. Практикум: модули ядра Linux Олега Цилюрика:
2. Вспоминаем: архитектура, ядро, модули...
3. Техника модулей ядра
4. Интерфейс /proc
5. Интерфейс /sys