Exception

開發學生:黃彥筑、許展富、王冠人
開發教師: 曹孝櫟、陳鵬升
國立中正大學 資訊工程學系

Exceptions

- Exceptions are usually used to handle unexpected events which arise during the execution of a program

Exception in Linux

- When an exception is generated, a function: vector_swi() is called.
- vector_swi() is defined in <kernel source code/arch/arm/kernel/entry-common.S>.
- vector_swi() gets the system call number in the R7 register and finds the system call address in the sys_call_table and invokes it.
- Registers R0-R6 are used to send arguments to the system calls.

Outline

- Environment
- Goal
- Steps
  - Initialization
  - Operation a system call
  - Write a system call
- Reference

Environment

- PC x 1
  - Linux Ubuntu
    - Editor, arm cross compiler
- Raspberry Pi

Goal

- Observe the initialization of exception.
- Observe the implementation of system calls.
- Add a new system call.

Initialization

- System boot
  - Start kernel
- Lab. steps
  - Download kernel
  - Compile kernel

Observation of a system call (1)

- In general, a user level process is not supposed to access kernel.
- Linux uses software interrupt to generate exception to implement system calls.
- The user-level calling process fills the registers with the appropriate values and then calls a special instruction which jumps to a previously defined location in the kernel.

Observation of a system call (2)

- I/O is conducted by the system
  - Use a printf program to investigate its assembly
- Cross-compile with static link

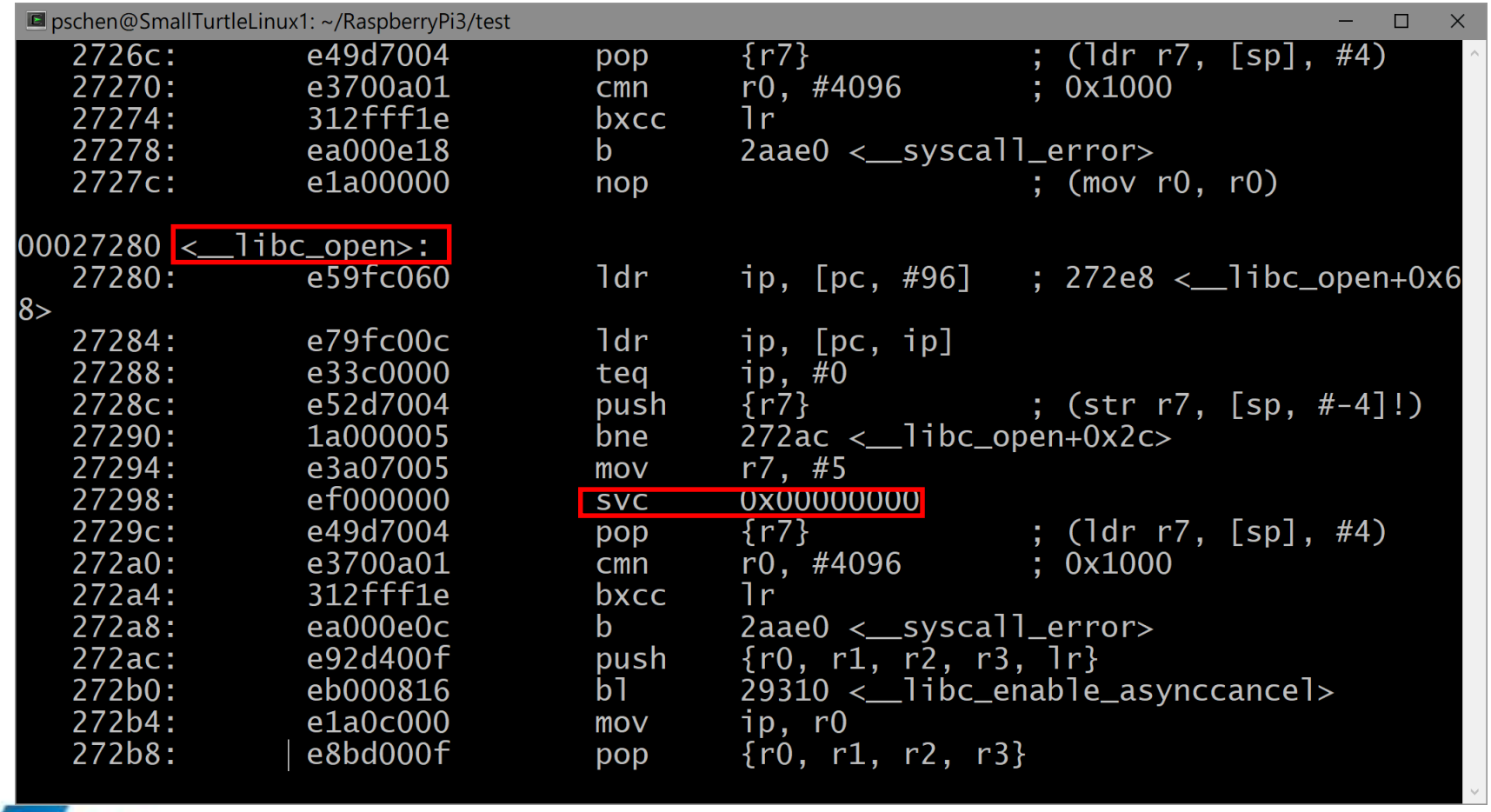
```
$arm-linux-gnueabihf-gcc -g -static -o
test.exe test.c
```

- Disassemble the executable code: test.exe

```
$arm-linux-gnueabihf-objdump -d test.exe
> assembly
```

Observation of a system call (3)

- Find an svc(swi) instruction for the system call



Write a system call (1)

- Define a system call
  - <Linux kernel source code>/arch/arm/kernel/calls.S
- You also need to generate a system call "stub" so that an ordinary user program can invoke your system call.
  - <Linux kernel source code>/arch/arm/include/uapi/asm/unistd.h
- Write your system call
  - Find the place suitable for your system call
  - Add asmlinkage in front of the function and sys_ in fornt of the call
  - include <linux/linkage.h> and <linux/kernel.h>

Write a system call (2)

- **Define a system call**

```
                CALL(sys_pkey_mprotect)
/* 395 */       CALL(sys_pkey_alloc)
                CALL(sys_pkey_free)
                CALL(sys_mysyscall)
```

- **Establish a "stub"**

```
#define __NR_pkey_mprotect    (__NR_SYSCALL_BASE+394)
#define __NR_pkey_alloc       (__NR_SYSCALL_BASE+395)
#define __NR_pkey_free        (__NR_SYSCALL_BASE+396)
#define __NR_mysyscall        (__NR_SYSCALL_BASE+397)
```

Write a system call (3)

- Add mysyscall.c for the system call

```
#include <linux/linkage.h>
#include <linux/kernel.h>
asmlinkage void sys_mysyscall(int a, char *b)
{
        printk("system call ...\n");
        printk("int1:%d, staring:%s in kernel\n",a,b);
}
```

Write a system call (4)

- User applications use the system call
  - mytestsys.c

```
#include <sys/syscall.h>
#include <linux/unistd.h>
#define __NR_mysyscall (__NR_SYSCALL_BASE+397)
#define mysyscall(a,b) syscall(__NR_mysyscall,(a),(b))
int main()
{
     mysyscall(14,"system_call_lab");
     return 0;
}
```

Write a system call (5)

- Compilation and verification
  - Modify makefile under <linux source>/arch/arm/kernel for the system call
    - Add obj-y += …… mysyscall.o
  - Re-compile the kernel
  - Compile mytestsys.c for the target, ARM-Linux
    - Cross compile
    - arm-linux-gnueabihf-gcc -I<Linux Source Code>/include/ -g mytestsys.c -o mytestsys.exe
  - Execute the user program(ex: ./mytestsys.exe) in Raspberry Pi
    - type dmesg in terminal and the system call message will be displayed.

❖Q1: asmlinkage的描述是什麼意義？

```
For example:
asmlinkage int sys_myservice (…)
{
   ...
}
```

請在實驗報告裡回答。