

學號：405410117

姓名：郭紘安

Email：andy8766kuo@gmail.com

實驗名稱：Exception

實驗目的：

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

實驗步驟：

➤ 例外處理

Step 1: 例外初始化

1. 下載Linux kernel source code

```
andy@ubuntu:~/disk$ git clone --depth=1 https://github.com/raspberrypi/linux  
-b rpi-4.9.y
```

2. 在<Linux kernel source code>/init/main.c的start_kernel函式中加入一個printk敘述。

(Linux Host的<Linux kernel source code>/init/main.c)

```
jump_label_init();

/*
 * These use large bootmem allocations and must precede
 * kmem_cache_init()
 */
setup_log_buf(0);
pidhash_init();
vfs_caches_init_early();
sort_main_extable();

/* Modify*/
printk("Initialize traps\n");
/*Modify*/
trap_init();
mm_init();

/*
 * Set up the scheduler prior starting any interrupts (such as the
 * timer interrupt). Full topology setup happens at smp_init()
 * time - but meanwhile we still have a functioning scheduler.
 */
sched_init();
/*
 * Disable preemption - early bootup scheduling is extremely
 * fragile until we cpu_idle() for the first time.
 */
preempt_disable();
if (WARN(!irqs_disabled(),
        "Interrupts were enabled *very* early, fixing it\n"))
    local_irq_disable();
idr_init_cache();
rcu_init();

/* trace_printk() and trace points may be used after this */
trace_init();
```

學號：405410117

姓名：郭紘安

Email：andy8766kuo@gmail.com

3.在<Linux kernel source code>/arch/arm/kernel/traps.c的 trap_init函式中加入一個printk敘述。(Linux Host的<Linux kernel source code>/arch/arm/kernel/traps.c)

```
void __init trap_init(void)
{
    /*Modify*/
    printk("arm: system call handler initialization -> see assembly code\n");
    /*Modify*/
    return;
}
```

Step 2: 編譯核心

andy@ubuntu:~/disk\$ cd linux/

andy@ubuntu:~/disk/linux\$ make ARCH=arm bcm2709_defconfig

```
andy@ubuntu:~/disk$ cd linux/
andy@ubuntu:~/disk/linux$ make ARCH=arm bcm2709_defconfig
HOSTCC scripts/basic/fixdep
HOSTCC scripts/kconfig/conf.o
SHIPPED scripts/kconfig/zconf.tab.c
SHIPPED scripts/kconfig/zconf.lex.c
SHIPPED scripts/kconfig/zconf.hash.c
HOSTCC scripts/kconfig/zconf.tab.o
HOSTLD scripts/kconfig/conf
#
# configuration written to .config
#
andy@ubuntu:~/disk/linux$
```

Add Path:

andy@ubuntu:~/disk/linux\$ PATH="/home/andy/WORK/crossgcc2/bin:\$PATH"

andy@ubuntu:~/disk/linux\$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-hf-bzImage

```
AS      arch/arm/boot/compressed/nyp-stub.o
SHIPPED arch/arm/boot/compressed/lib1funcs.o
AS      arch/arm/boot/compressed/lib1funcs.o
SHIPPED arch/arm/boot/compressed/ashldi3.S
AS      arch/arm/boot/compressed/ashldi3.o
SHIPPED arch/arm/boot/compressed/bswapsdi2.S
AS      arch/arm/boot/compressed/bswapsdi2.o
LD      arch/arm/boot/compressed/vmlinux
OBJCOPY arch/arm/boot/zImage
Kernel: arch/arm/boot/zImage is ready
```

學號：405410117

姓名：郭紘安

Email：andy8766kuo@gmail.com

你可以在<linux kernel source目錄>/arch/arm/boot目錄下，找到zImage。

將 zImage 複製到 microSD 卡的第一個 partition。

andy@ubuntu:~/disk/linux/arch/arm/boot\$ sudo cp -rf zImage ~/mmc1

```
COM5 - PuTTY
[ 0.000000] Booting Linux on physical CPU 0x0
[ 0.000000] Linux version 4.9.80-v7+ (andy@ubuntu) (gcc version 4.9.3 (GCC) ) #1 SMP Tue May 19 01:53:07 CST
[ 0.000000] CPU: ARMv7 Processor [410fd034] revision 4 (ARMv7), cr=10c5383d
[ 0.000000] CPU: div instructions available: patching division code
[ 0.000000] CPU: PIPT / VIPT nonaliasing data cache, VIPT aliasing instruction cache
[ 0.000000] OF: fdt:Machine model: Raspberry Pi 3 Model B Rev 1.2
[ 0.000000] cma: Reserved 8 MiB at 0x39400000
[ 0.000000] Memory policy: Data cache writealloc
[ 0.000000] percpu: Embedded 14 pages/cpu @b8b8b000 s25600 r8192 d23552 u57344
[ 0.000000] Built 1 zonelists in Zone order, mobility grouping on. Total pages: 234465
[ 0.000000] Kernel command line: coherent_pool=1M 8250.nr_uaarts=1 bcm2708_fb.fbwidth=720 bcm2708_fb.fbheight
ole=ttyAMA0,115200
[ 0.000000] PID hash table entries: 4096 (order: 2, 16384 bytes)
[ 0.000000] Dentry cache hash table entries: 131072 (order: 7, 524288 bytes)
[ 0.000000] Inode-cache hash table entries: 65536 (order: 6, 262144 bytes)
[ 0.000000] Initialize traps
[ 0.000000] arm: system call handler initialization -> see assembly code
[ 0.000000] Memory: 91972K/946176K available (168K kernel code, 48K rwdata, 2032K rodata, 1024K init, 770
Virtual kernel memory layout:
[ 0.000000] vector : 0xffff0000 - 0xffff1000 ( 4 kB)
[ 0.000000] fixmap : 0xffc00000 - 0xffff0000 (3072 kB)
[ 0.000000] vmalloc : 0xba000000 - 0xff800000 (1112 MB)
[ 0.000000] lowmem : 0x80000000 - 0xb9c00000 ( 924 MB)
[ 0.000000] modules : 0x7f000000 - 0x80000000 ( 16 MB)
[ 0.000000] .text : 0x80008000 - 0x80800000 (8160 kB)
[ 0.000000] .init : 0x80b00000 - 0x80c00000 (1024 kB)
[ 0.000000] .data : 0x80c00000 - 0x80c79cbc ( 488 kB)
[ 0.000000] .bss : 0x80c7b000 - 0x80d3b9e4 ( 771 kB)
[ 0.000000] SLUB: HWalign=64, Order=0-3, MinObjects=0, CPUs=4, Nodes=1
[ 0.000000] Hierarchical RCU implementation.
[ 0.000000] Build-time adjustment of leaf fanout to 32.
```

➤ 系統呼叫運作

Step 1: 寫一個使用系統呼叫的程式

```
andy@ubuntu:~/disk/Lab9$ gcc hello.c -o hello
andy@ubuntu:~/disk/Lab9$ ./hello
hello world~
andy@ubuntu:~/disk/Lab9$
```

Step 2: Cross Compiler

編譯程式

反組譯程式碼

```
andy@ubuntu:~/disk/Lab9$ arm-linux-gnueabi-gcc -static hello.c -o hello.exe
andy@ubuntu:~/disk/Lab9$ arm-linux-gnueabi-objdump -d hello.exe > assembly
andy@ubuntu:~/disk/Lab9$
```

學號：405410117

姓名：郭紘安

Email：andy8766kuo@gmail.com

Step 3: 找尋 SVC

```
00027280 <__libc_open>:
27280: e59fc060 ldr ip, [pc, #96] ; 272e8 <__libc_open+0x68>
27284: e79fc00c ldr ip, [pc, ip]
27288: e33c0000 teq ip, #0
2728c: e52d7004 push {r7} ; (str r7, [sp, #-4]!)
27290: 1a000005 bne 272ac <__libc_open+0x2c>
27294: e3a07005 mov r7, #5
27298: ef000000 svc 0x00000000
2729c: e49d7004 pop {r7} ; (ldr r7, [sp], #4)
272a0: e3700a01 cmn r0, #4096 ; 0x1000
272a4: 312fff1e bxcc lr
272a8: ea000e0c b 2aae0 <__syscall_error>
272ac: e92d400f push {r0, r1, r2, r3, lr}
272b0: eb000816 bl 29310 <__libc_enable_asynccancel>
272b4: e1a0c000 mov ip, r0
272b8: e8bd000f pop {r0, r1, r2, r3}
272bc: e3a07005 mov r7, #5
272c0: ef000000 svc 0x00000000
272c4: e1a07000 mov r7, r0
272c8: e1a0000c mov r0, ip
272cc: eb00083f bl 293d0 <__libc_disable_asynccancel>
272d0: e1a00007 mov r0, r7
272d4: e49de004 pop {lr} ; (ldr lr, [sp], #4)
272d8: e49d7004 pop {r7} ; (ldr r7, [sp], #4)
272dc: e3700a01 cmn r0, #4096 ; 0x1000
272e0: 312fff1e bxcc lr
272e4: ea000dfd b 2aae0 <__syscall_error>
272e8: 00073170 .word 0x00073170
272ec: e1a00000 .word 0xe1a00000
```

➤ 系統呼叫(System Call)

Step 1: 修改系統檔案

(Linux Host的<Linux kernel source code>/arch/arm/kernel/calls.S)

```
/* Modify */
CALL(sys_mysyscall)
/* Modify */
```

Step 2: Add define of system call in unistd.h

(Linux Host的<Linux kernel source

code>/arch/arm/include/uapi/asm/unistd.h)

```
/* Modify */
#define __NR_mysyscall (__NR_SYSCALL_BASE+397)
/* Modify */
```

Step 3: 撰寫欲新增的 system call 的內容

1. (Linux Host的<Linux kernel source

code>/arch/arm/kernel/mysyscall.c)(新增 mysyscall.c)

```
mysyscall.c (86 GB Volume ~/disk/linux/arch/arm/kernel) - gedit
Open [ ]
#include<linux/linkage.h>
#include <linux/kernel.h>
asmlinkage void sys_mysyscall(int a, char *b){
    printk("system call ...\n");
    printk("inti:%d, staring:%s in kernel\n", a, b);
}
```

學號：405410117

姓名：郭紘安

Email：andy8766kuo@gmail.com

2. (Linux Host的<Linux kernel source

code>/include/linux/syscalls.h) (修改)

```
asmlinkage long sys_pkey_alloc(unsigned long fl
asmlinkage long sys_pkey_free(int pkey);
asmlinkage void sys_mysyscall(int a, char* b);
#endif
```

3. 修改在<Linux kernel source code>/arch/arm/kernel下的Makefile，
其中的obj-y := compat.o entry-armv.o ...，在這行最後面加入
mysyscall.o。

(Linux Host 的<Linux kernel source c
ode>/arch/arm/kernel/Makefile)

```
# Object file lists.

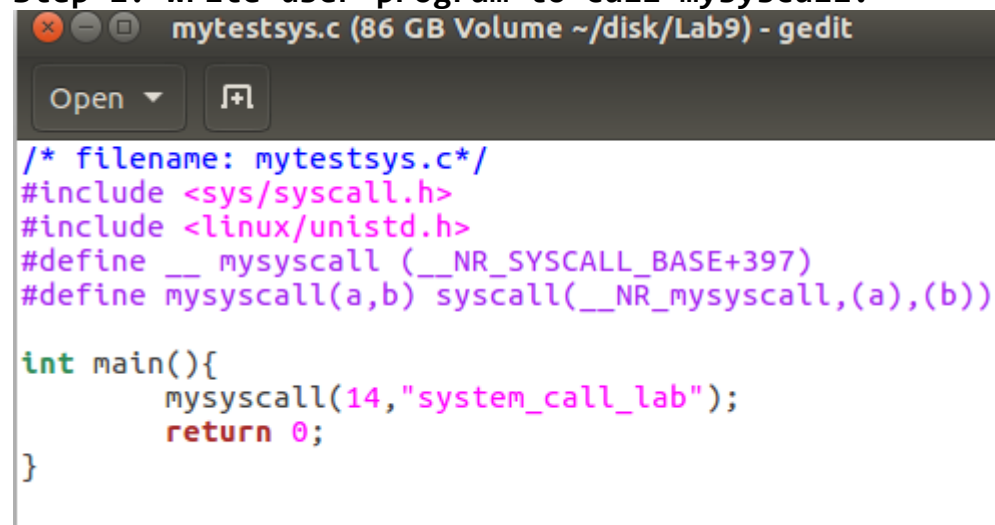
obj-y      := elf.o entry-common.o irq.o opcodes.o \
               process.o ptrace.o reboot.o return_address.o \
               setup.o signal.o sigreturn_codes.o \
               stacktrace.o sys_arm.o time.o traps.o mysyscall.o
```

4. 重編 kernel

```
andy@ubuntu:~/disk/linux$ make ARCH=arm bcm2709_defconfig
#
# configuration written to .config
#
andy@ubuntu:~/disk/linux$
```

➤ 寫一個使用者應用程式，來測試新增的系統呼叫

Step 1: write user program to call mysyscall.



```
/* filename: mytestsys.c */
#include <sys/syscall.h>
#include <linux/unistd.h>
#define __mysyscall (__NR_SYSCALL_BASE+397)
#define mysyscall(a,b) syscall(__NR_mysyscall,(a),(b))

int main(){
    mysyscall(14,"system_call_lab");
    return 0;
}
```

學號：405410117

姓名：郭紘安

Email：andy8766kuo@gmail.com

Step 2: 使用 Cross Compiler 編譯使用者程式 mytestsys.c

```
andy@ubuntu:~/disk/Lab9$ arm-linux-gnueabi-gcc -I/home/andy/disk/linux/include/ -static -g mytestsys.c -o mytestsys.exe
mytestsys.c: In function 'main':
mytestsys.c:5:24: warning: implicit declaration of function 'syscall' [-Wimplicit-function-declaration]
#define mysyscall(a,b) syscall(__NR_mysyscall,(a),(b))
                        ^
mytestsys.c:8:2: note: in expansion of macro 'mysyscall'
mysyscall(14,"system_call_lab");
^
```

Step 3: Pi 上執行 mytestsys.exe

```
# ls
bin          lib32        mnt          root         tmp
dev          linuxrc      mytestsys.exe run          usr
etc          lost+found  opt          sbin         var
lib          media       proc         sys

# ./mytestsys.exe
[ 239.496759] system call ...
[ 239.502699] intl:14, staring:system_call_lab in kernel
#
```

問題與討論：

- `asm`linkage 的描述是什麼意義？

For example:

```
asm linkage int sys_myservice(...)
{
...
}
```

"asm linkage" 是在 i386 system call 實作中相當重要的一個 gcc 標籤 (tag)。

當 system call handler 要呼叫相對應的 system call routine 時，便將一般用途暫存器的值 push 到 stack 裡，因此 system call routine 就要由 stack 來讀取 system call handler 傳遞的參數。這就是 asm linkage 標籤的用意。

system call handler 是 assembly code，system call routine (例如：sys_nice) 是 C code，當 assembly code 呼叫 C function，並且是以 stack 方式傳參數 (parameter) 時，在 C function 的 prototype 前面就要加上 "asm linkage"。

加上 "asm linkage" 後，C function 就會由 stack 取參數，而不是從 register 取參數 (可能發生在程式碼最佳化後)。

學號：405410117

姓名：郭紘安

Email：andy8766kuo@gmail.com

心得：

這次實驗讓我實作了如何修改並且編譯 linux kernel,以及創建自己的 system call.