OS Project 1
System Call Implementation

Advisor: Prof. Tei-Wei Kuo

TA: Tse-Yuan Wang

r03922064@csie.ntu.edu.tw

Outline

- Introduction
- Example: Hello System Call
- Project Requirements
- Submission Rules

What is System Call?

- System call is *how a program requests services from the kernel of an operating system*
- System call provides an essential interface between processes (user) and the operating system (kernel)
- System calls can be roughly grouped into five major categories:

1) Process control
2) File management
3) Device management
4) Information maintenance
5) Communication

System Call: Start

Once a system call occurs,

- The processor is switched to the system execution mode (or privileged execution mode)
- Key parts of the current thread context (e.g., *the program counter and the stack pointer*) are saved
- Then the thread context is then changed:
  - The *program counter* is set to a fixed (determined by the hardware) memory address, which is within the kernel's address space
  - The *stack pointer* is pointed at the top of a stack in the kernel's address space

System Call: Execute

Then,

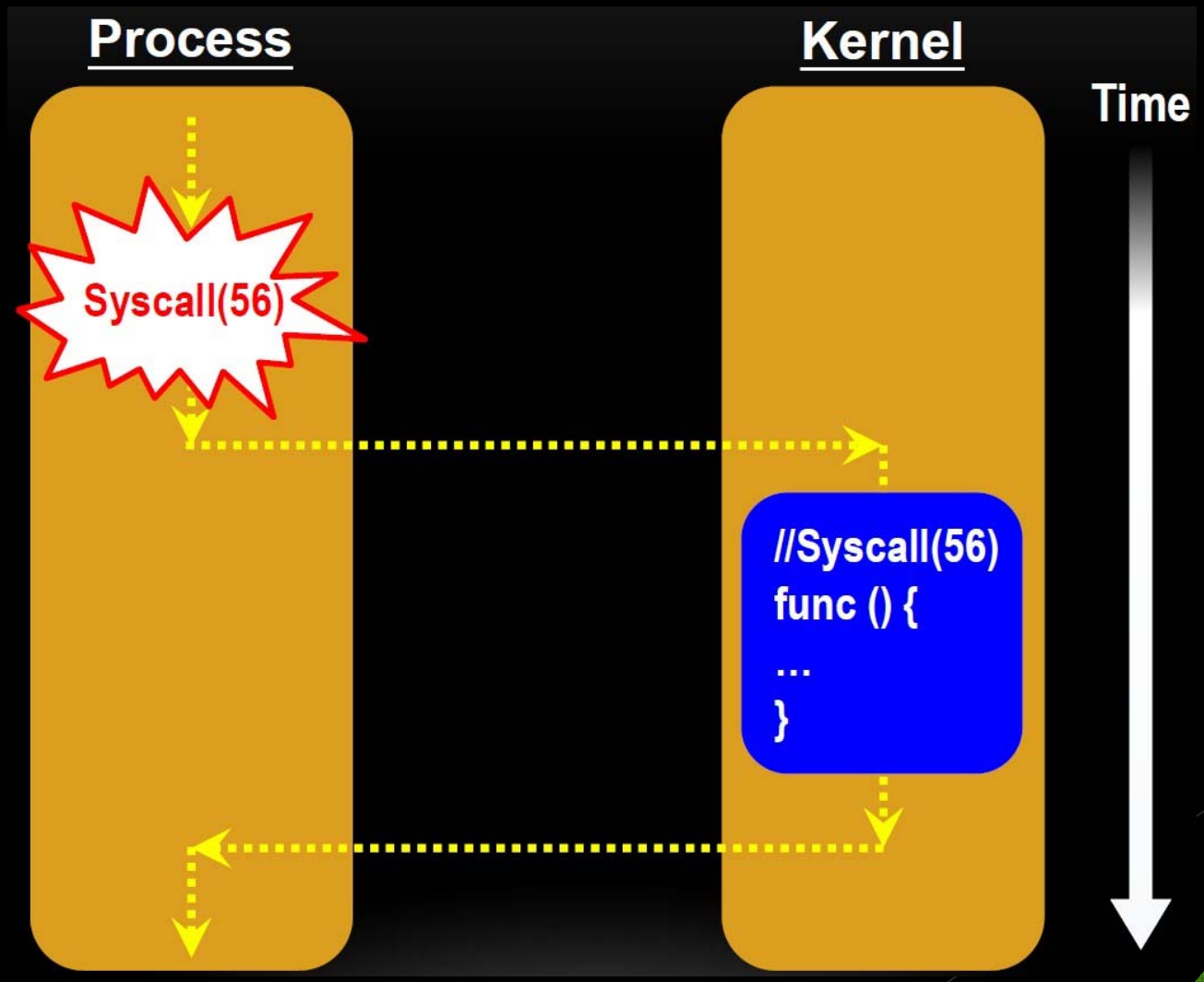
- The calling thread will be executing a system call handler, which is part of the kernel, in system mode
- This kernel's system call handler *determines which service the calling process wanted and then performs that service*

System Call: End

When the kernel finishes,

- It returns from the system call and this means to
  - Restore the key parts of the thread context that were saved when the system call was made
  - Switch the processor status back to the user execution mode (or unprivileged execution mode)
- Now the thread is executing the calling-process program again and picks up where it left when it made the system call

System Call: Diagram



Example: Hello System Call (1/5)

1. Download the linux-3.2.54 kernel source code
   - sudo wget https://www.kernel.org/pub/linux/kernel/v3.x/linux-3.2.54.tar.xz
2. Decompress the kernel source code
   - sudo tar xvf linux-3.2.54.tar.xz
3. Add system call to the system call table
   - Open the file linux-3.2.54/arch/x86/kernel/syscall_table_32.S and add the following line.
   - Add ".long sys_hello"

```
.long sys_syncfs
.long sys_sendmmsg              /* 345 */
.long sys_setns
.long sys_process_vm_readv
.long sys_process_vm_writev
.long sys_hello                 /* 349 this is my os demo */
                                                351,2-9       Bot
```

Example: Hello System Call (2/5)

4. Define macros associated with system call
   - Open the file linux-3.2.54/arch/x86/include/asm/unistd_32.h
   - You will notice that a macro is defined for each system call. At the end of the huge macro definition, add a definition for our new system call and accordingly incremented the value of the macro NR_SYSCALLS
   - Add “#define __NR_hello349”
   - Update “#define NR_syscalls350”

```
#define __NR_process_vm_readv    347
#define __NR_process_vm_writev   348
#define __NR_hello               349     //this is my os demo

#ifdef __KERNEL__

#define NR_syscalls 350          //This number is 349 originally.

#define __ARCH_WANT_IPC_PARSE_VERSION
#define __ARCH_WANT_OLD_READDIR
#define __ARCH_WANT_OLD_STAT
#define __ARCH_WANT_STAT64
#define __ARCH_WANT_SYS_ALARM
```

Example: Hello System Call (3/5)

5. Define macros associated with system call

- Now to the file linux-3.2.54/arch/x86/include/asm/syscalls.h, add the prototype of the system call.

- Add the prototype of the system call “asmlinkage long sys_hello(void);”

```
/* kernel/sys_x86_64.c */
asmlinkage long sys_mmap(unsigned long, unsigned long, unsigned long,
                         unsigned long, unsigned long, unsigned long);

/* This is my os demo */
asmlinkage int sys_hello(void);
```

- Now, in the directory of the kernel sources inux-3.2.54/kernel/, create a file “hello.c” with the following contents:

```
#include<linux/kernel.h>
#include<linux/linkage.h>

asmlinkage int sys_hello(void)
{
        printk("hello system call ! \n");
        return 0;
}
```

Example: Hello System Call (4/5)

6. After you create the function definition, we have to modify the Makefile(inux-3.2.54/kernel/Makefile) so as to compile the new system call to merge it into the kernel

- Add “hello.o” to “obj-y”

```
#
# Makefile for the linux kernel.
#

obj-y     = sched.o fork.o exec_domain.o panic.o printk.o \
            cpu.o exit.o itimer.o time.o softirq.o resource.o \
            sysctl.o sysctl_binary.o capability.o ptrace.o timer.o user.o \
            signal.o sys.o kmod.o workqueue.o pid.o \
            rcupdate.o extable.o params.o posix-timers.o \
            kthread.o wait.o kfifo.o sys_ni.o posix-cpu-timers.o mutex.o \
            hrtimer.o rwsem.o nsproxy.o srcu.o semaphore.o \
            notifier.o ksysfs.o sched_clock.o cred.o \
            async.o range.o hello.o
obj-y += groups.o
```

7. **REBUILD** the whole kernel and reboot the Ubuntu

- Please reference to the OS2016 Project 0 slides.

Example: Hello System Call (5/5)

8. After you REBUILD and REBOOT into the kernel that you just compiled, try to run the following program:

- Test program

```
#include <sys/syscall.h>
#include <unistd.h>
#include <stdio.h>
int main()
{
        syscall(349);
        return 0;
}
```

9. The output of printk() is written to the kernel log. To view it, type the command “dmesg”

```
[13960.623254] hello system call !
[13960.623258] hello system call !
[13960.623259] hello system call !
kib65060@ubuntu:~/test$ dmesg
```

Requirements of Project 1 (1/2)

- Implement 3 new system calls into your Linux kernel (60%, 20%for each):

1. Show (void)
   - Show Student_ID(s) and Name(s)of your team member(s)
2. Multiply (long, long)
   - Return the calculation result
3. Min (long, long)
   - Return the calculation result

Requirements of Project 1 (2/2)

- Write a test program to test your implemented system calls (20%)
  - Notably, the test program should call the 3 system calls to demonstrate the results
- Report (20%)
  - Implementation details or faced difficulties
  - Your results (please print-screen)
  - At most 2 pages
- Bonus (at most 20%)
  - Any other implementation regarding system calls in Linux
    - E.g., the process status, some kernel's information

Submission Rules

- Project deadline: 2016/04/6 (Wednesday) 23:59
- Upload to FTP Server
  - IP: 140.112.28.132
  - Port: 5566
  - Account name: os2016
  - Password: ktw2016os
- The team project should
  - Contain your test program, modified source files (NOT the whole kernel), and your report (PDF format, within 2 pages)
  - Be packed as one file named "OSPJ1_Team##.**ZIP**"
- **DO NOT COPY THE HOMEWORK**

Contact TAs

- If you have any problem about the projects, you can contact TAs by the following ways:
- Facebook: NTU OS2016 Spring Group
  - https://www.facebook.com/groups/1683988081869980/
  - **HIGHLY RECOMMENDED**
- E-mail:
  - Tse-Yuan Wang: r03922064@csie.ntu.edu.tw