# KERNEL ARCHITECTURE

BLG413E – System Programming, Practice Session 2

# Contents

- System Calls
- Kernel Modules

# 1-Adding a system call

**Requirements (for Ubuntu OS):** linux-source, kernel_package, fakeroot, libncurses5-dev

**Steps:**

- define and write new system call
- modify system call table
- modify system header files
- compile kernel
- reboot to new kernel
- test new system call

# Writing a system call

- **mycall.c:** under /usr/src/linux-source-x.x.x/kernel/

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

asmlinkage int sys_mycall(int i, int j){
    return i + j;
}
```

- **Modify Makefile:** under /usr/src/linux-source-x.x.x/kernel/
  - add obj–y := mycall.o

# Modifying system call table and system header files

- In /usr/src/linux-source-x.x.x/arch/x86/kernel/syscall_table_32.S:
  - append `.long sys_mycall`
- In /usr/src/linux-source-x.x.x/include/linux/syscalls.h:
  - add prototype of the new system call as

```
asmlinkage int sys_mycall(int i, int j);
```

- In /usr/src/linux-source-x.x.x/arch/x86/include/asm/unistd_32.h:
  - append `#define __NR_mycall 349`
  - The index should be the number after the last system call in the list
  - You should also increment the system call count by 1 (**NR_syscalls**)

```
#define NR_syscalls 350
```

# Before compiling linux kernel

- **Configuration Options (1st one recommended):**
  1. copy config.txt file to linux-source-2.6.38 (under Ubuntu 11.04) and rename it as .config
  2. make oldconfig (only ask for new or different options)
  3. make menuconfig (menu for configuration)

# Compiling linux kernel

- make-kpkg clean → cleans up all from previous kernel compiles
- **Compilation**: fakeroot make-kpkg --initrd --append-to-version=-custom kernel_image kernel_headers
- **Output:** two files in parent directory:
  - linux-image-x.x.x-custom ....deb
  - linux-headers-x.x.x-custom ....deb

# Installing compiled kernel

- sudo dpkg -i linux-image-x.x.x-custom ....deb
- sudo dpkg -i linux-headers-x.x.x-custom ....deb

- **Uninstalling:**
  - sudo dpkg -r linux-image-x.x.x-custom
  - sudo dpkg -r linux-headers-x.x.x-custom

# Testing new system call

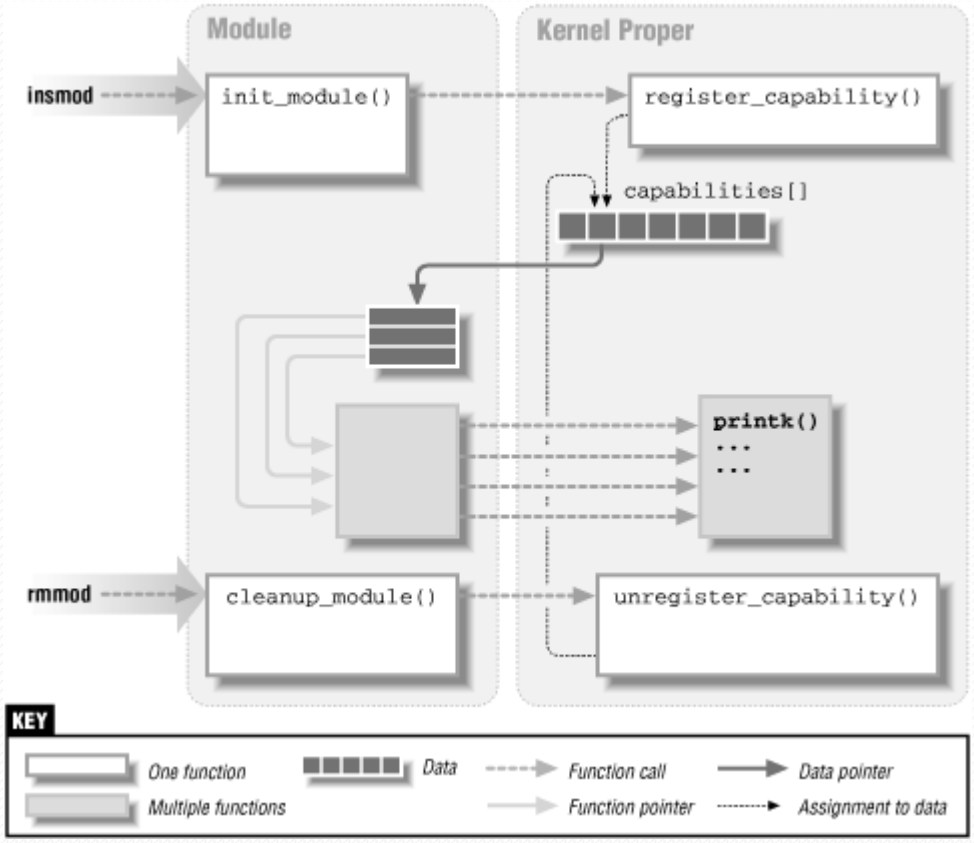
```c
#include <stdio.h>
#include <sys/syscall.h>

#define __NR_mycall 349

int main(void){
    int x1=10, x2=20, y;
    y = syscall(__NR_mycall, x1, x2);
    printf("%d\n", y);
    return 0 ;
}
```

# 2-Kernel modules

- A way to add new features to the kernel without rebuilding it.
- Unlike applications, modules register themselves for serving future requests.
- Applications can access the capabilities of a module through system calls.




http://www.xml.com/ldd/chapter/book/figs/ldr2_0201.gif

# An example module: hello

- **hello.c:**

```c
#include <linux/init.h> /* for module_init and module_exit */
#include <linux/module.h> /* needed by all modules */
MODULE_LICENSE("Dual BSD/GPL"); /* a macro to declare that this module is open source */

static int hello_init(void) /* static: unvisible outside the module */
{                           /* to avoid namespace pollution */
   printk(KERN_ALERT "Hello, world\n"); /* printk: kernel print function (macros for priority) */
   return 0;                            /* KERN_ALERT: a situation requiring immediate action */
}

static void hello_exit(void)
{
   printk(KERN_ALERT "Goodbye, cruel world\n");
}

module_init(hello_init);
module_exit(hello_exit);
```

- **Makefile:**

```
obj-m := hello.o
all:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
```

M=$(PWD) is to build external module in the working directory

# Using hello module

- **Compiling:**
  - *make*
- **Loading** (check with ***dmesg*** which is used to write the kernel messages):
  - *sudo insmod ./hello.ko*
- **Unloading** (check with ***dmesg***):
  - *sudo rmmod hello*
- check with ***lsmod*** (which prints the contents of the /proc/modules file) before and after loading and unloading

# An example module using load time parameters

- **hellop.c:**

```
/* $Id: hellop.c,v 1.4 2004/09/26 07:02:43 gregkh Exp $  */
#include <linux/init.h>
#include <linux/module.h>
#include <linux/moduleparam.h> /* to enable passing parameters at loadtime */
MODULE_LICENSE("Dual BSD/GPL");

/* A couple of parameters that can be passed in: how many times we say hello, and to whom */
static char *whom = "world";
static int howmany = 1;
module_param(howmany, int, S_IRUGO); /* S_IRUGO: read by the world but cannot be changed */
module_param(whom, charp, S_IRUGO);

static int hello_init(void){
   int i;
   for (i = 0; i < howmany; i++)
      printk(KERN_ALERT "(%d) Hello, %s\n", i, whom);
   return 0;
}

static void hello_exit(void){
   printk(KERN_ALERT "Goodbye, cruel world\n");
}

module_init(hello_init);
module_exit(hello_exit);
```

# Specifying module parameters

- sudo insmod ./hellop.ko whom='Mom' howmany=4
- dmesg

```
4555.764793] (0) Hello, Mom
4555.764796] (1) Hello, Mom
4555.764797] (2) Hello, Mom
4555.764798] (3) Hello, Mom
```

- sudo rmmod hellop
- dmesg

```
4555.764793] (0) Hello, Mom
4555.764796] (1) Hello, Mom
4555.764797] (2) Hello, Mom
4555.764798] (3) Hello, Mom
4611.350208] Goodbye, cruel world
```