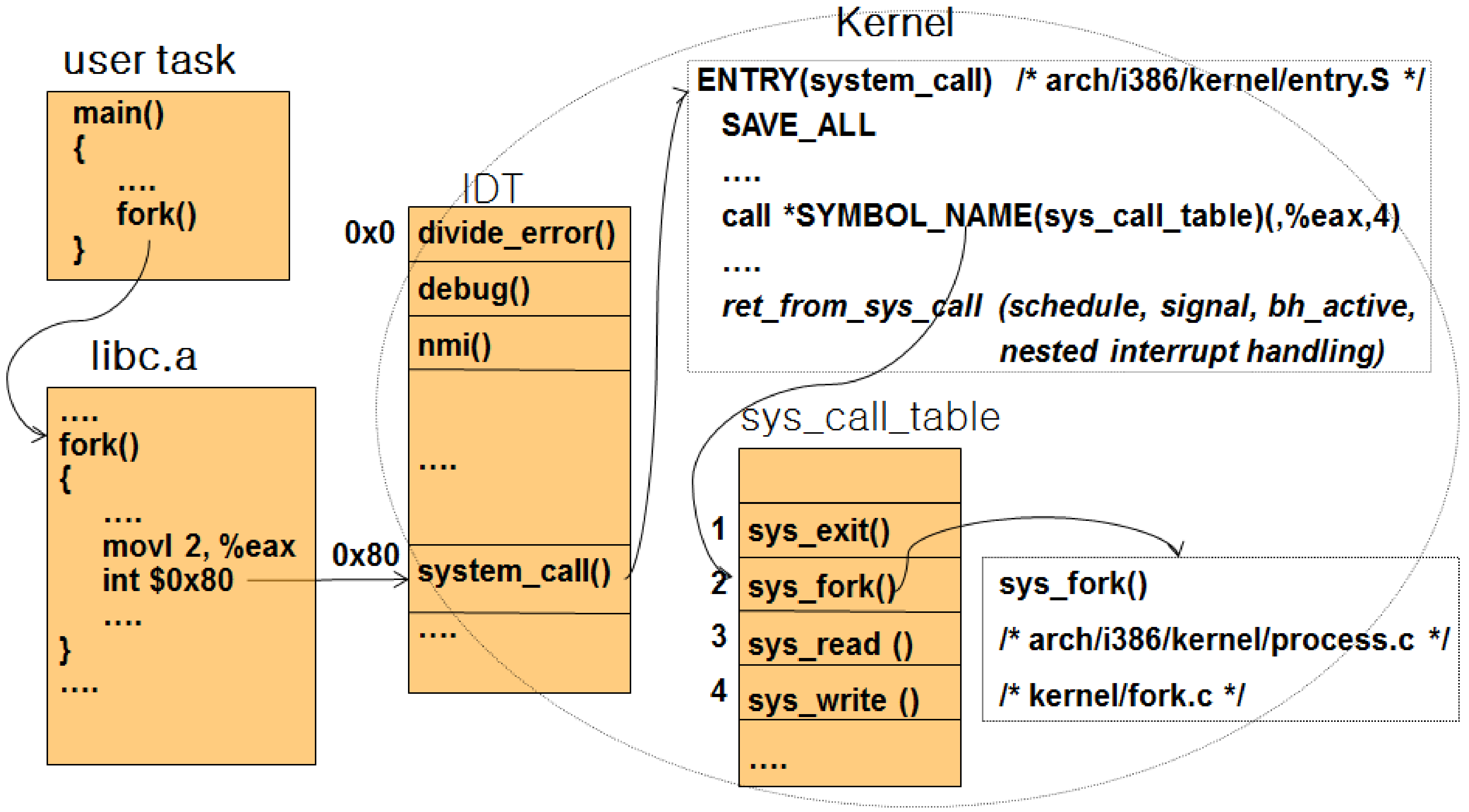
Linux System Call

What is System Call?

- User-level processes (clients) request services from the kernel (server) via special "**protected procedure calls**"

- System calls provide:
  - An **abstraction layer** between processes and hardware, allowing the kernel to provide access control, arbitration
  - A **virtualization** of the underlying system
  - A well-defined "**API**" for system services

System Call Procedure in Linux



Add New System Call (Tutorial 1)

- Download Kernel Source Code
- Kernel-level modification
  (1) Allocate an unused system call number
  (2) Register sys_call_table
  (3) Program new system call handler
  (4) Kernel compile and rebooting

- User-level modification
  (1) Make library interface
  (2) Make user program that call the library function

Add New System Call (Tutorial 1)

- Download Kernel Source Code
- Kernel-level modification
  - (1) Allocate an unused system call number
  - (2) Register sys_call_table
  - (3) Program new system call handler
  - (4) Kernel compile and rebooting
- User-level modification
  - (1) Make library interface
  - (2) Make user program that call the library function

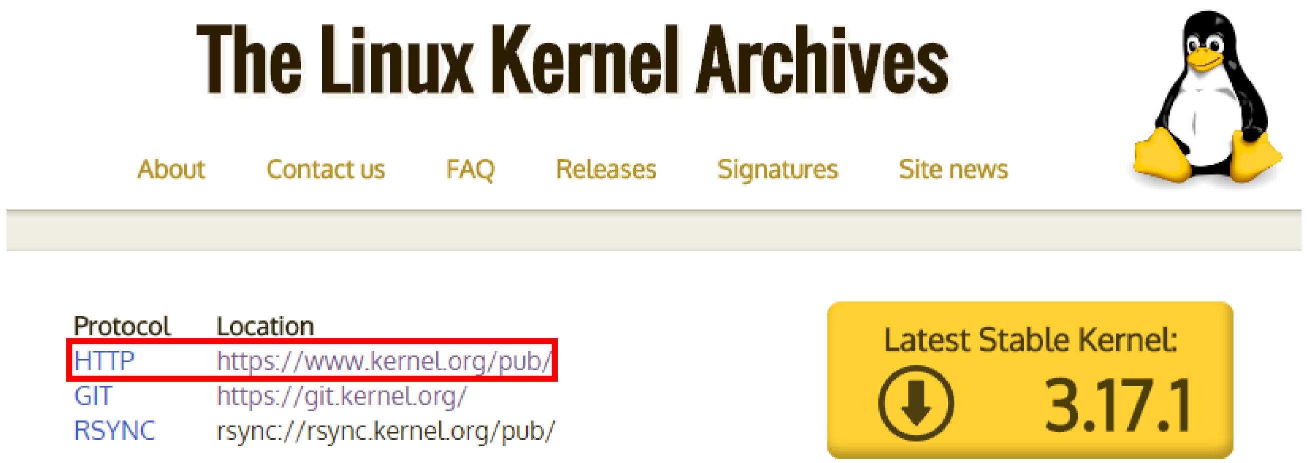
Download Kernel Source Code

- Tutorial environment
  - 32bit Ubuntu 12.04 (Kernel Version : 3.2.0) on VMware

- Download Kernel Source Code

  (1) $ apt-get source linux-image-$(uname -r)

  (2) or Download at http://www.kernel.org

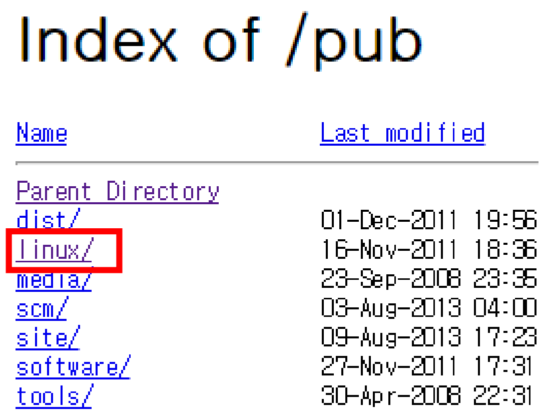


The Linux Kernel Archives

About Contact us FAQ Releases Signatures Site news

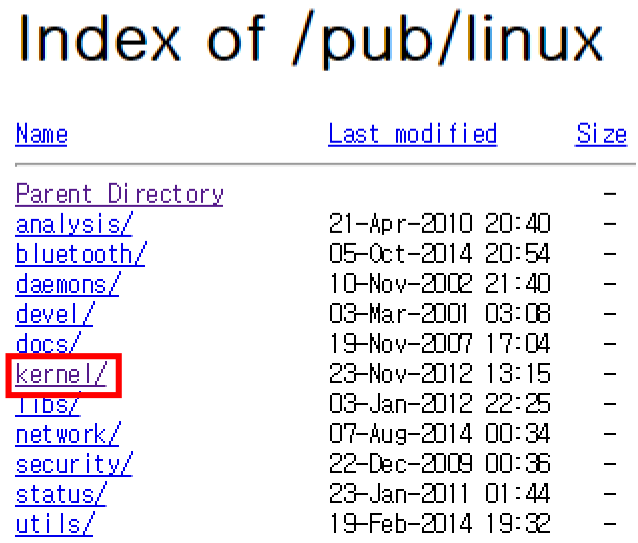
Protocol	Location
HTTP	https://www.kernel.org/pub/
GIT	https://git.kernel.org/
RSYNC	rsync://rsync.kernel.org/pub/

Latest Stable Kernel: 3.17.1



Index of /pub

Name	Last modified
Parent Directory	
dist/	01-Dec-2011 19:56
linux/	16-Nov-2011 18:36
media/	23-Sep-2008 23:35
scm/	03-Aug-2013 04:00
site/	09-Aug-2013 17:23
software/	27-Nov-2011 17:31
tools/	30-Apr-2008 22:31



Index of /pub/linux

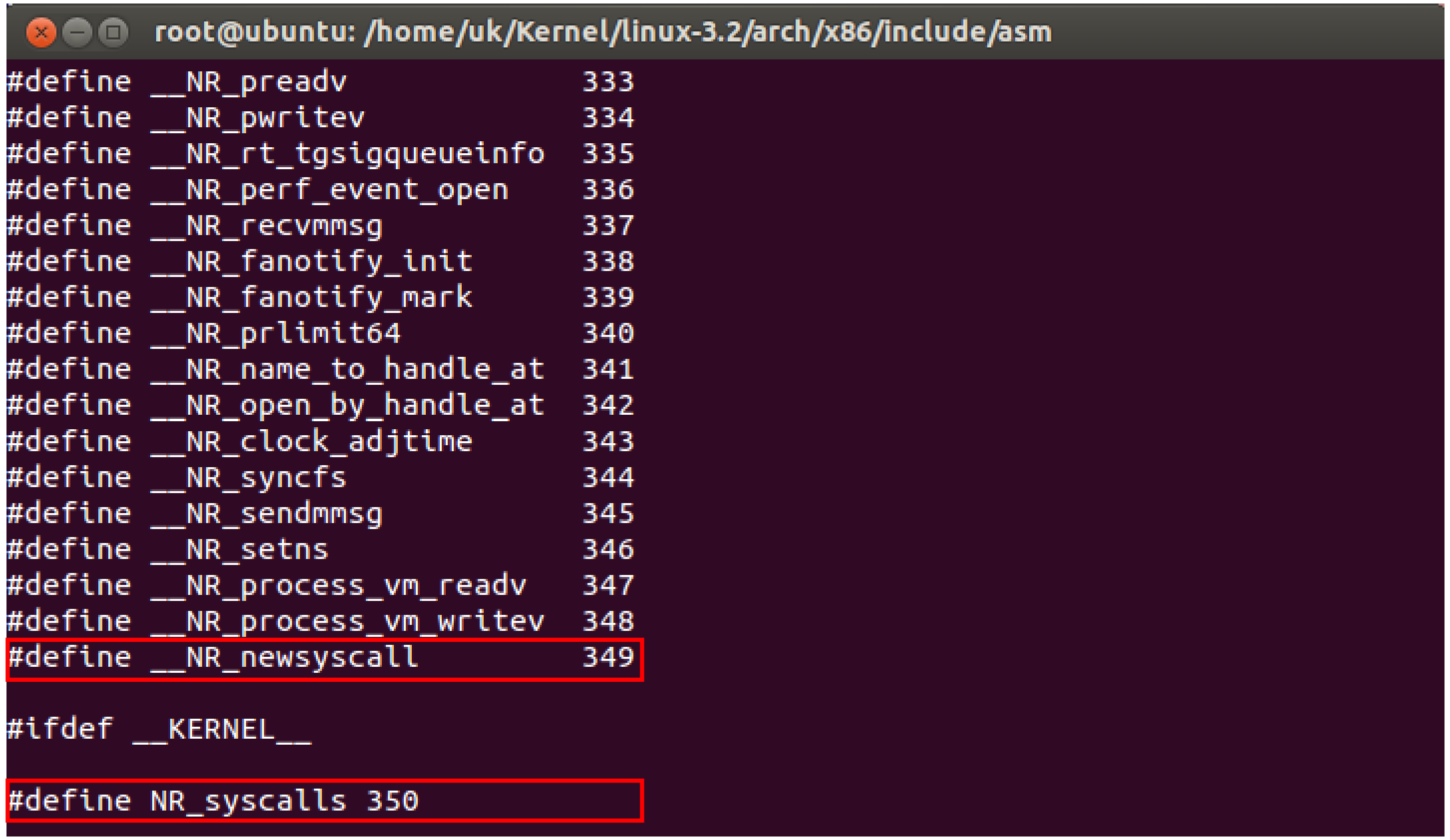
Name	Last modified	Size
Parent Directory		-
analysis/	21-Apr-2010 20:40	-
bluetooth/	05-Oct-2014 20:54	-
daemons/	10-Nov-2002 21:40	-
devel/	03-Mar-2001 03:08	-
docs/	19-Nov-2007 17:04	-
kernel/	23-Nov-2012 13:15	-
libs/	03-Jan-2012 22:25	-
network/	07-Aug-2014 00:34	-
security/	22-Dec-2009 00:36	-
status/	23-Jan-2011 01:44	-
utils/	19-Feb-2014 19:32	-

Add New System Call (Tutorial 1)

- Download Kernel Source Code
- Kernel-level modification
  - (1) Allocate an unused system call number
  - (2) Register sys_call_table
  - (3) Program new system call handler
  - (4) Kernel compile and rebooting
- User-level modification
  - (1) Make library interface
  - (2) Make user program that call the library function

Allocate Unused System Call Number

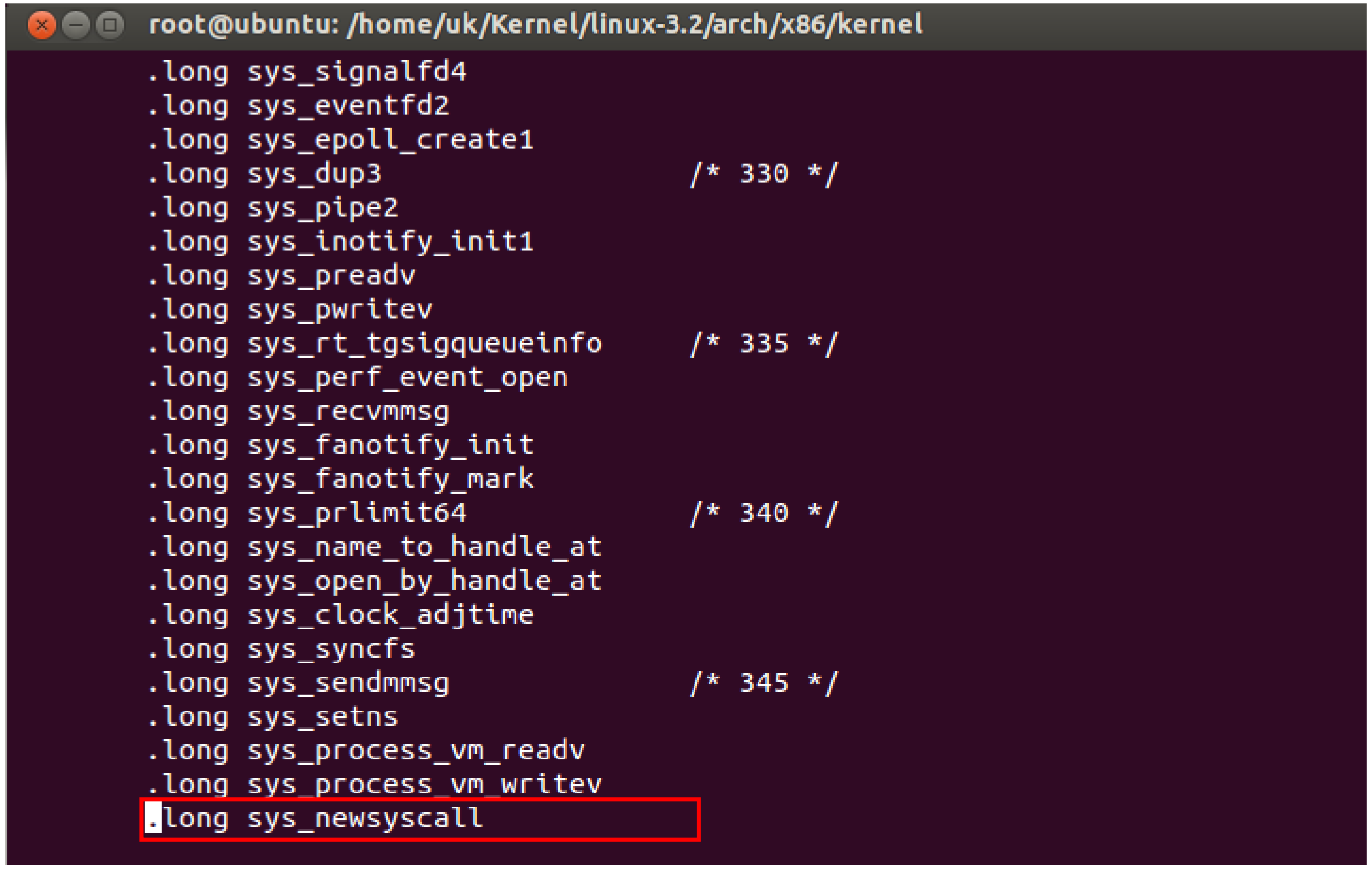
- /(source code path)/arch/x86/include/asm/unistd_32.h


root@ubuntu: /home/uk/Kernel/linux-3.2/arch/x86/include/asm

```
#define __NR_preadv             333
#define __NR_pwritev            334
#define __NR_rt_tgsigqueueinfo  335
#define __NR_perf_event_open    336
#define __NR_recvmmsg           337
#define __NR_fanotify_init      338
#define __NR_fanotify_mark      339
#define __NR_prlimit64          340
#define __NR_name_to_handle_at  341
#define __NR_open_by_handle_at  342
#define __NR_clock_adjtime      343
#define __NR_syncfs             344
#define __NR_sendmmsg           345
#define __NR_setns              346
#define __NR_process_vm_readv   347
#define __NR_process_vm_writev  348
#define __NR_newsyscall         349

#ifdef __KERNEL__

#define NR_syscalls 350
```

Register sys_call_table

- /(source code path)/arch/x86/kernel/syscall_table_32.S

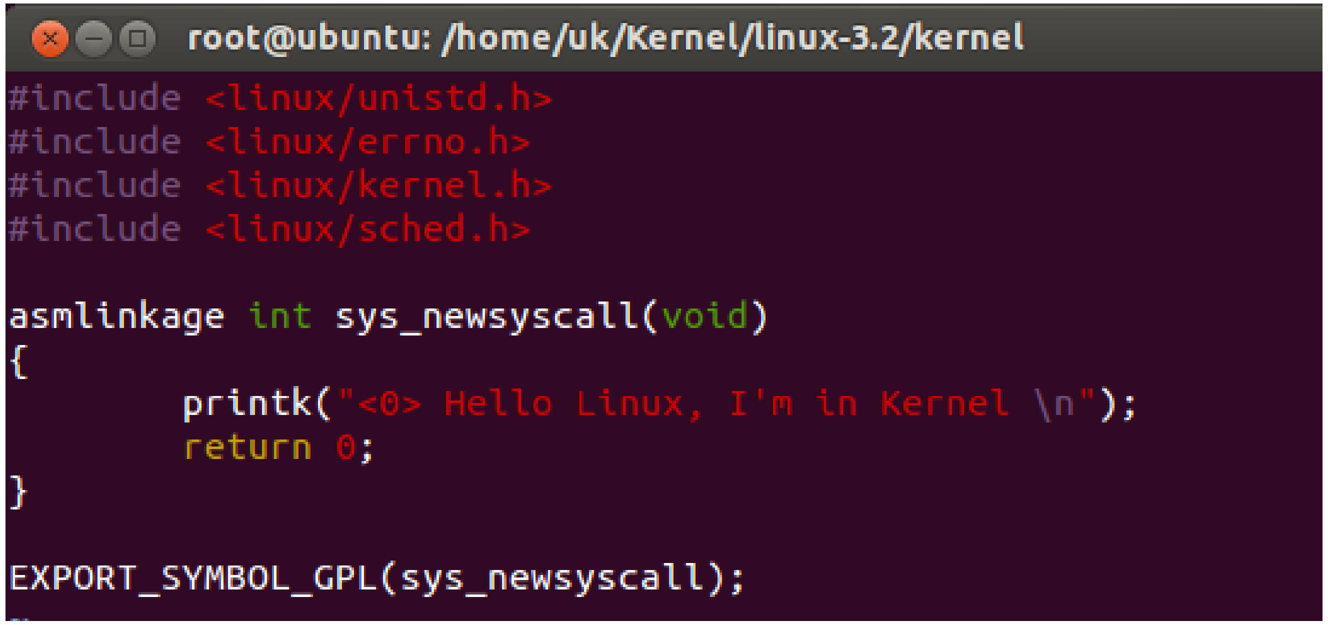


root@ubuntu: /home/uk/Kernel/linux-3.2/arch/x86/kernel

```
        .long sys_signalfd4
        .long sys_eventfd2
        .long sys_epoll_create1
        .long sys_dup3                  /* 330 */
        .long sys_pipe2
        .long sys_inotify_init1
        .long sys_preadv
        .long sys_pwritev
        .long sys_rt_tgsigqueueinfo     /* 335 */
        .long sys_perf_event_open
        .long sys_recvmmsg
        .long sys_fanotify_init
        .long sys_fanotify_mark
        .long sys_prlimit64             /* 340 */
        .long sys_name_to_handle_at
        .long sys_open_by_handle_at
        .long sys_clock_adjtime
        .long sys_syncfs
        .long sys_sendmmsg              /* 345 */
        .long sys_setns
        .long sys_process_vm_readv
        .long sys_process_vm_writev
        .long sys_newsyscall
```

Program new system call handler

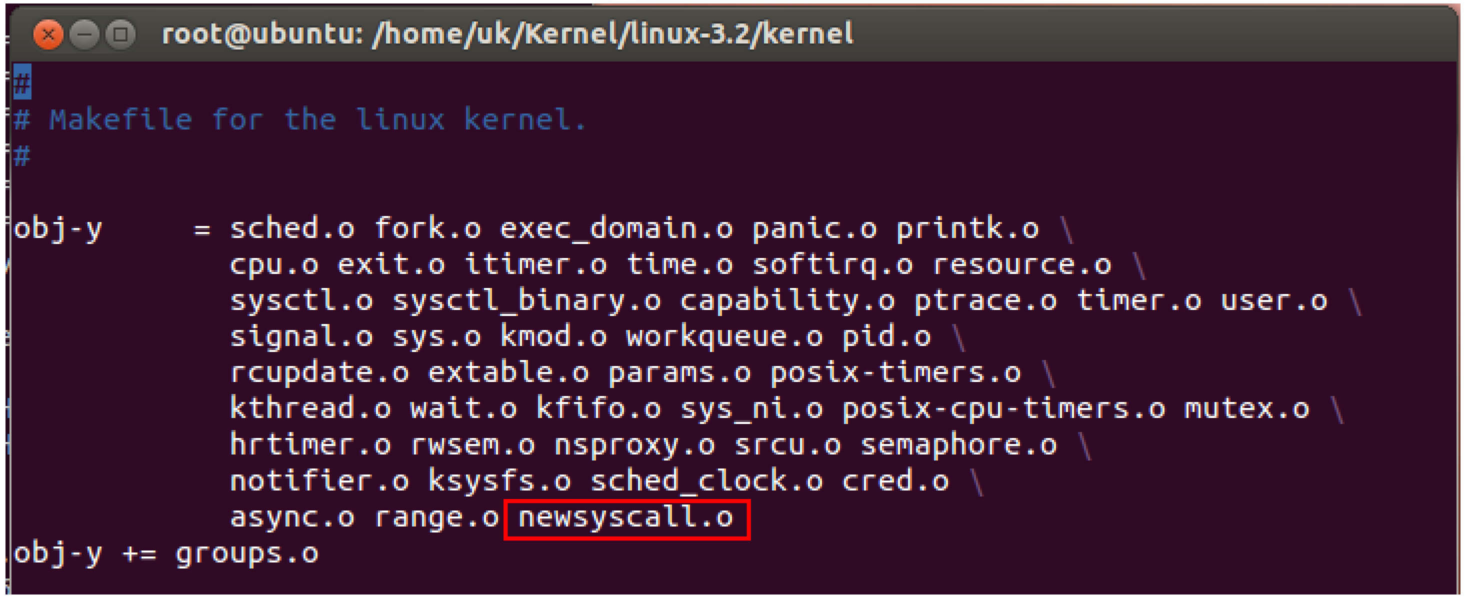
- /(source code path)/kernel/newsyscall.c



```
#include <linux/unistd.h>
#include <linux/errno.h>
#include <linux/kernel.h>
#include <linux/sched.h>

asmlinkage int sys_newsyscall(void)
{
        printk("<0> Hello Linux, I'm in Kernel \n");
        return 0;
}

EXPORT_SYMBOL_GPL(sys_newsyscall);
```

Modify Makefile

- /(source code path)/kernel/Makefile



root@ubuntu: /home/uk/Kernel/linux-3.2/kernel

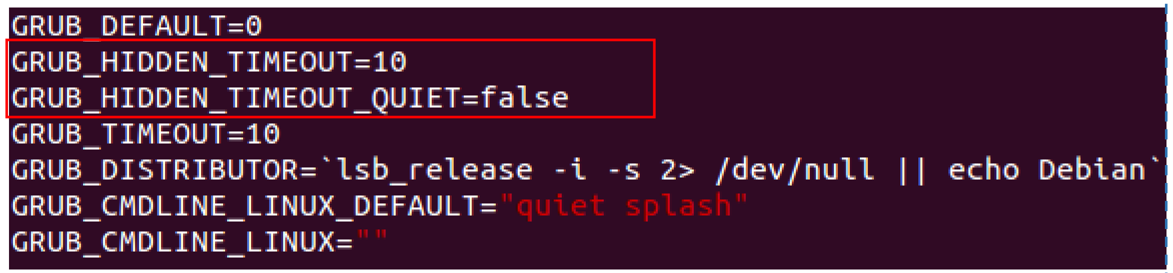
```
#
# Makefile for the linux kernel.
#

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 newsyscall.o
obj-y += groups.o
```

Kernel Compile and Rebooting

- $ apt-get update
- $ apt-get install build-essential libncurses5 libncurses5-dev
- /(source code path)/
- Follow the below commands to compile the kernel
  (1) $ make menuconfig
  (2) $ make bzImage
  (3) $ make modules
  (4) $ make modules_install
  (5) $ make install

- Change grub configuration
- Reboot
  – Press 'esc' key to see the grub menu

modify '/etc/default/grub' like below



```
GRUB_DEFAULT=0
GRUB_HIDDEN_TIMEOUT=10
GRUB_HIDDEN_TIMEOUT_QUIET=false
GRUB_TIMEOUT=10
GRUB_DISTRIBUTOR=`lsb_release -i -s 2> /dev/null || echo Debian`
GRUB_CMDLINE_LINUX_DEFAULT="quiet splash"
GRUB_CMDLINE_LINUX=""
```

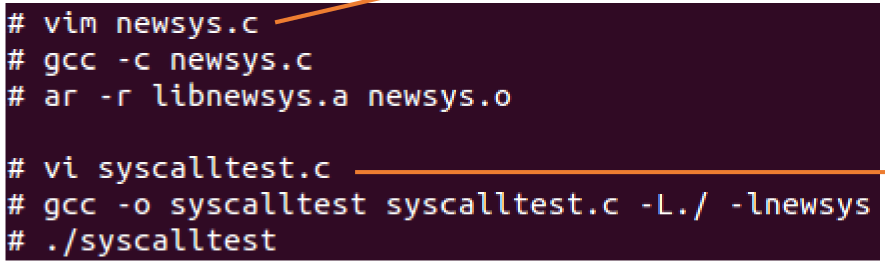
$ update-grub

Add New System Call (Tutorial 1)

- Download Kernel Source Code
- Kernel-level modification
  (1) Allocate an unused system call number
  (2) Register sys_call_table
  (3) Program new system call handler
  (4) Kernel compile and rebooting

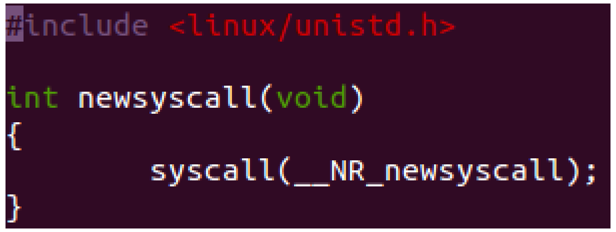
- User-level modification
  (1) Make library interface
  (2) Make user program that call the library function

Make Library Interface and User Program



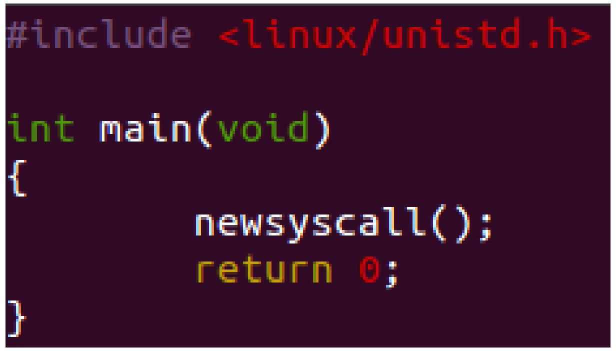
```
# vim newsys.c
# gcc -c newsys.c
# ar -r libnewsys.a newsys.o

# vi syscalltest.c
# gcc -o syscalltest syscalltest.c -L./ -lnewsys
# ./syscalltest
```

Library Interface



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

int newsyscall(void)
{
        syscall(__NR_newsyscall);
}
```

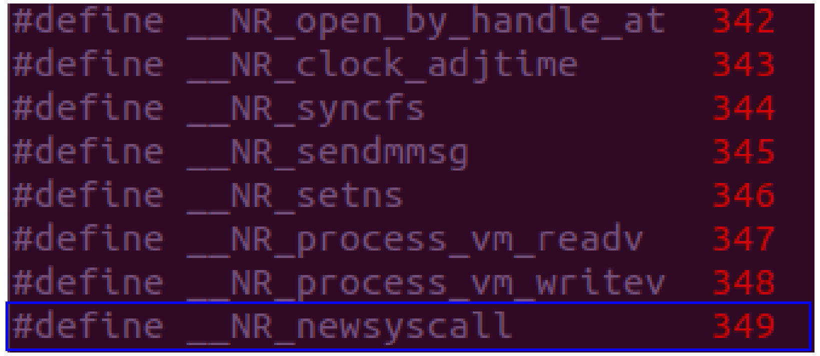
User Program



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

int main(void)
{
        newsyscall();
        return 0;
}
```

※ If you get the message such as "__NR_newsyscall is not defined", you can add a line in '/usr/include/i386-linux-gnu/asm/unistd_32.h' as follow

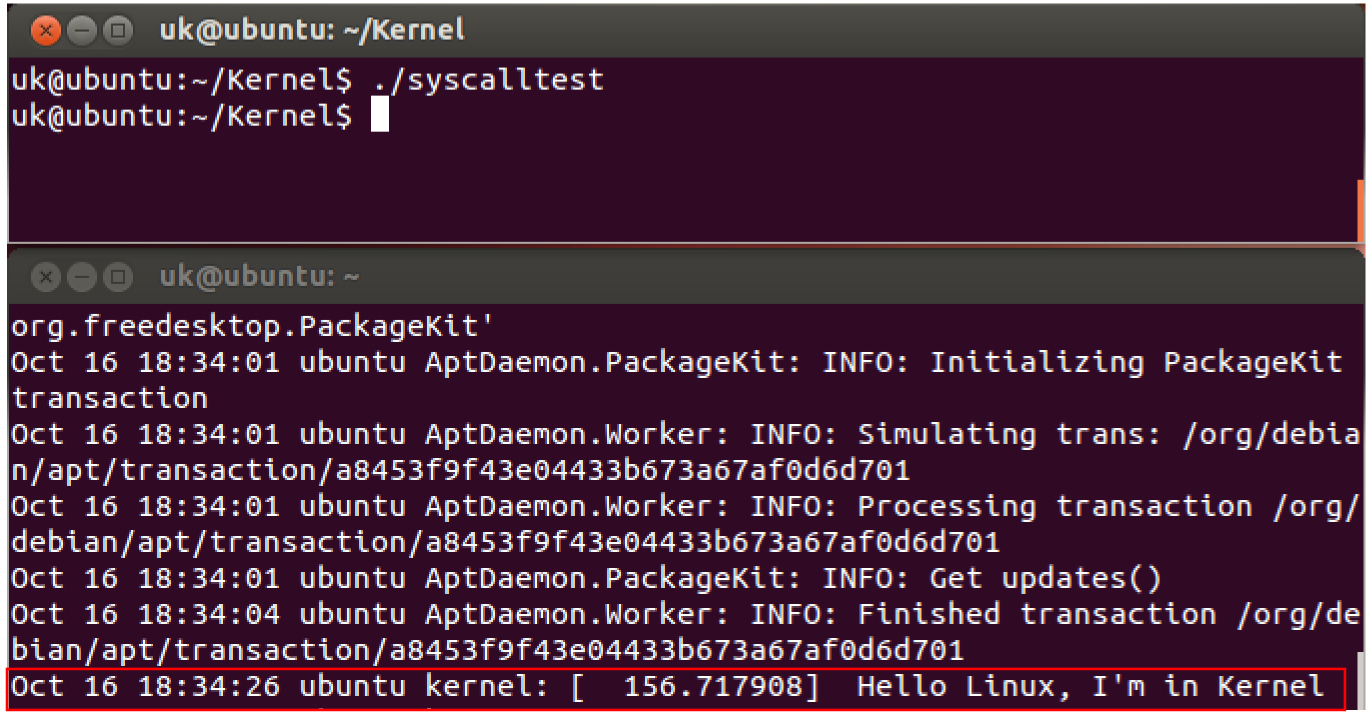
/usr/include/i386-linux-gnu/asm/unistd_32.h



```
#define __NR_open_by_handle_at  342
#define __NR_clock_adjtime      343
#define __NR_syncfs             344
#define __NR_sendmmsg           345
#define __NR_setns              346
#define __NR_process_vm_readv   347
#define __NR_process_vm_writev  348
#define __NR_newsyscall         349
```

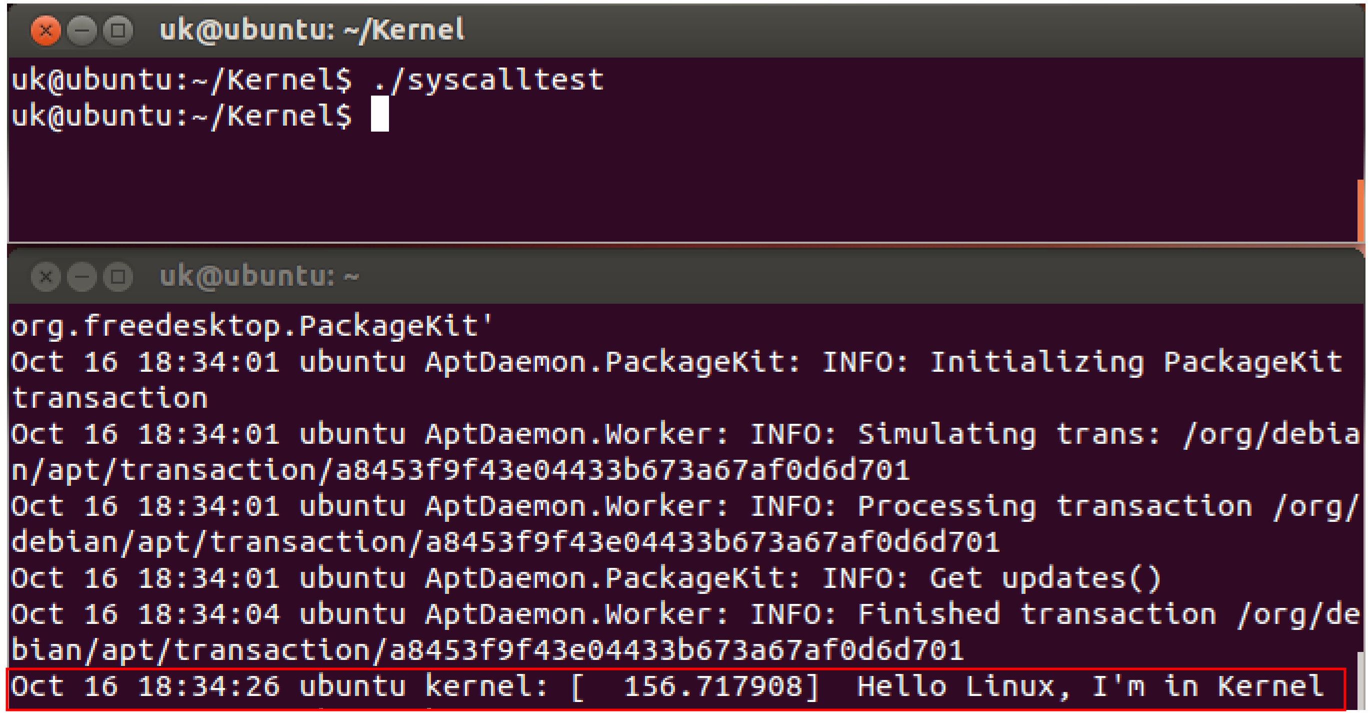
Check the Results

- Check the kernel print message
  - $ dmesg
  - $ tail –f /var/log/syslog



```
uk@ubuntu:~/Kernel$ ./syscalltest
uk@ubuntu:~/Kernel$
```

```
org.freedesktop.PackageKit'
Oct 16 18:34:01 ubuntu AptDaemon.PackageKit: INFO: Initializing PackageKit
transaction
Oct 16 18:34:01 ubuntu AptDaemon.Worker: INFO: Simulating trans: /org/debia
n/apt/transaction/a8453f9f43e04433b673a67af0d6d701
Oct 16 18:34:01 ubuntu AptDaemon.Worker: INFO: Processing transaction /org/
debian/apt/transaction/a8453f9f43e04433b673a67af0d6d701
Oct 16 18:34:01 ubuntu AptDaemon.PackageKit: INFO: Get updates()
Oct 16 18:34:04 ubuntu AptDaemon.Worker: INFO: Finished transaction /org/de
bian/apt/transaction/a8453f9f43e04433b673a67af0d6d701
Oct 16 18:34:26 ubuntu kernel: [  156.717908]  Hello Linux, I'm in Kernel
```

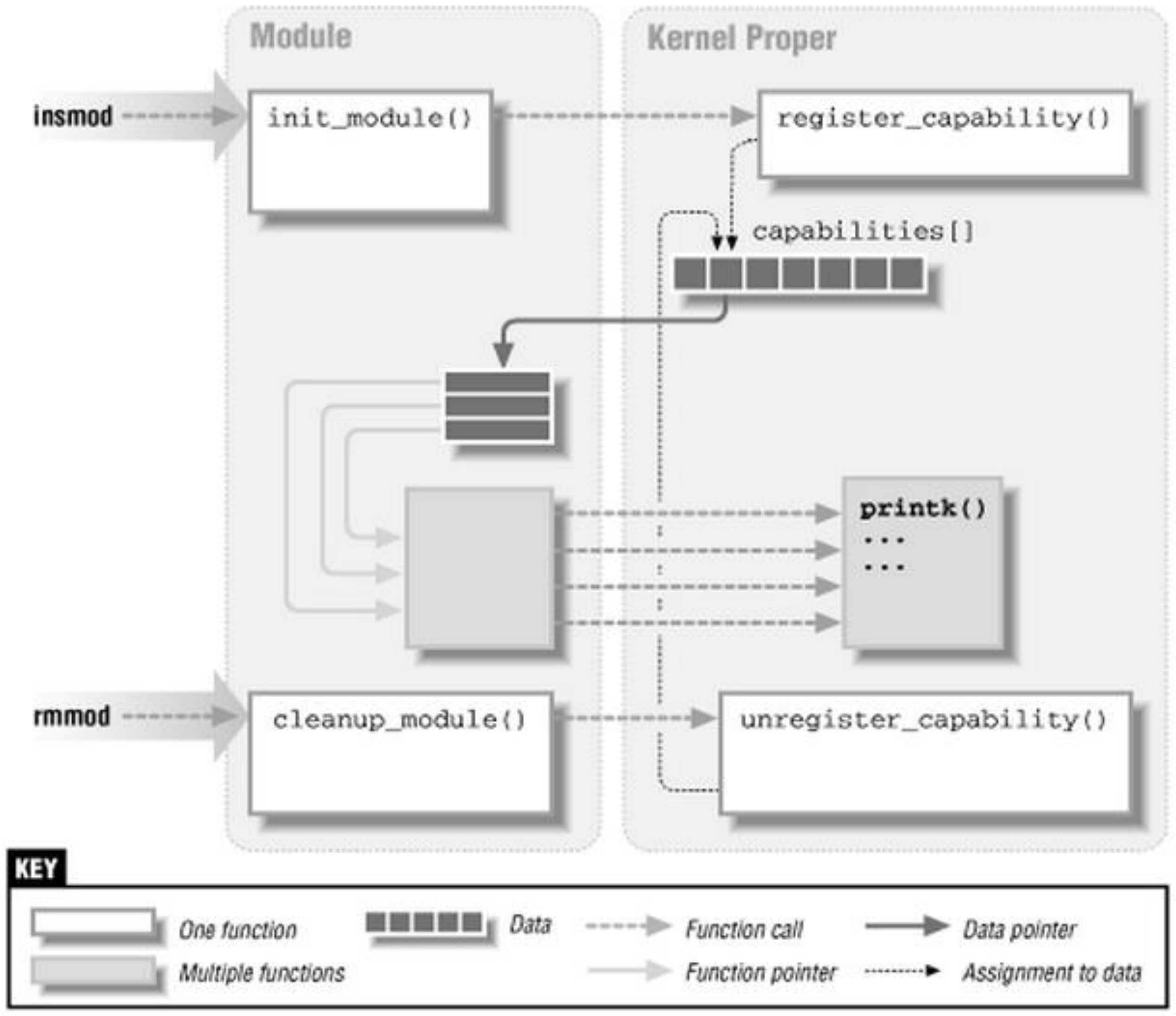


Linux Kernel Module

What is Kernel Module?

- A Module is a object file that contains code to extend the functionality of the base kernel
- Modules are used to add support for new hardware and file systems
- Also used to add new system calls and executable interpreters

Linking a Module to the Kernel

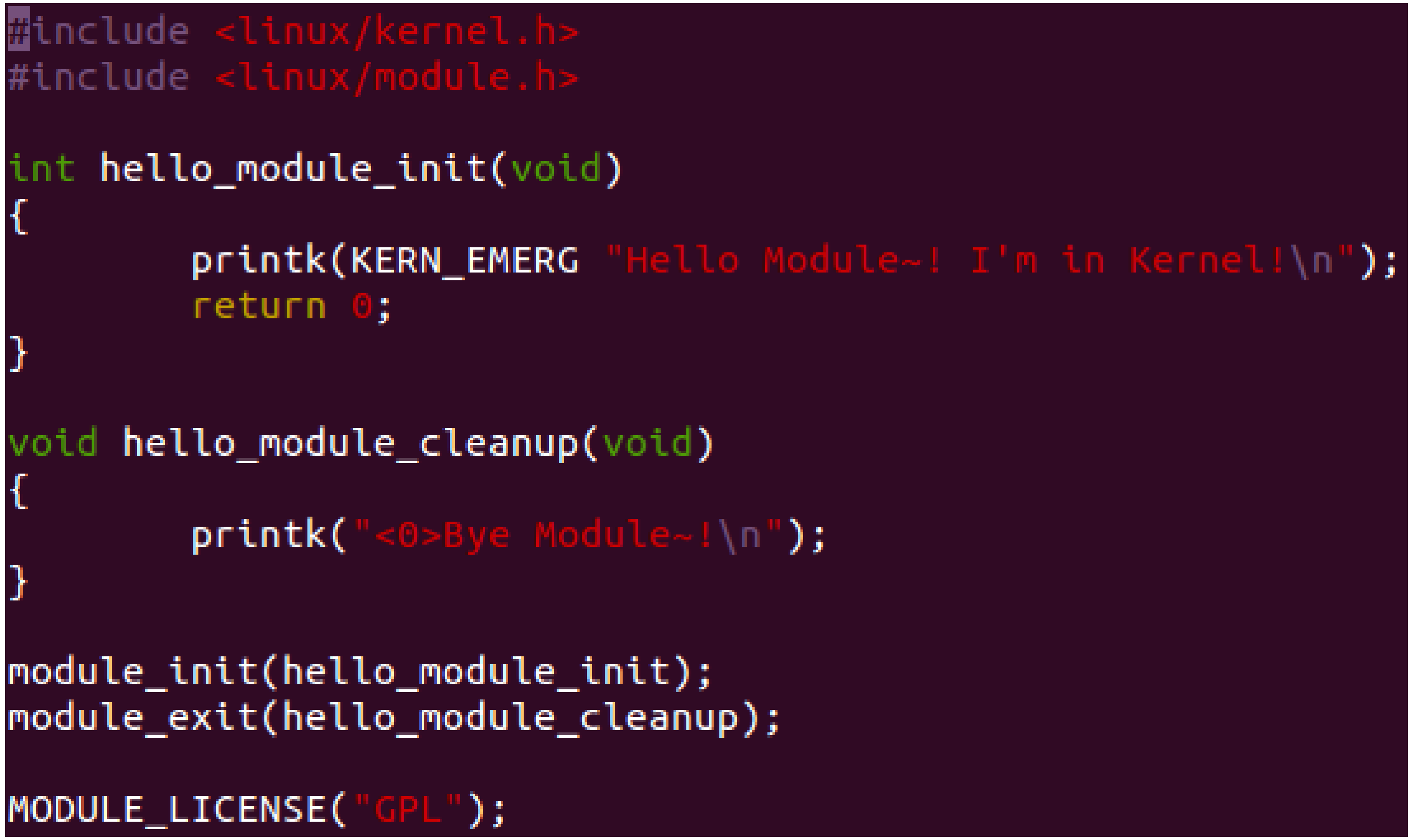


※ *Reference : Linux Device Drivers 2nd Edition*

Make a Kernel Module (Tutorial 2)

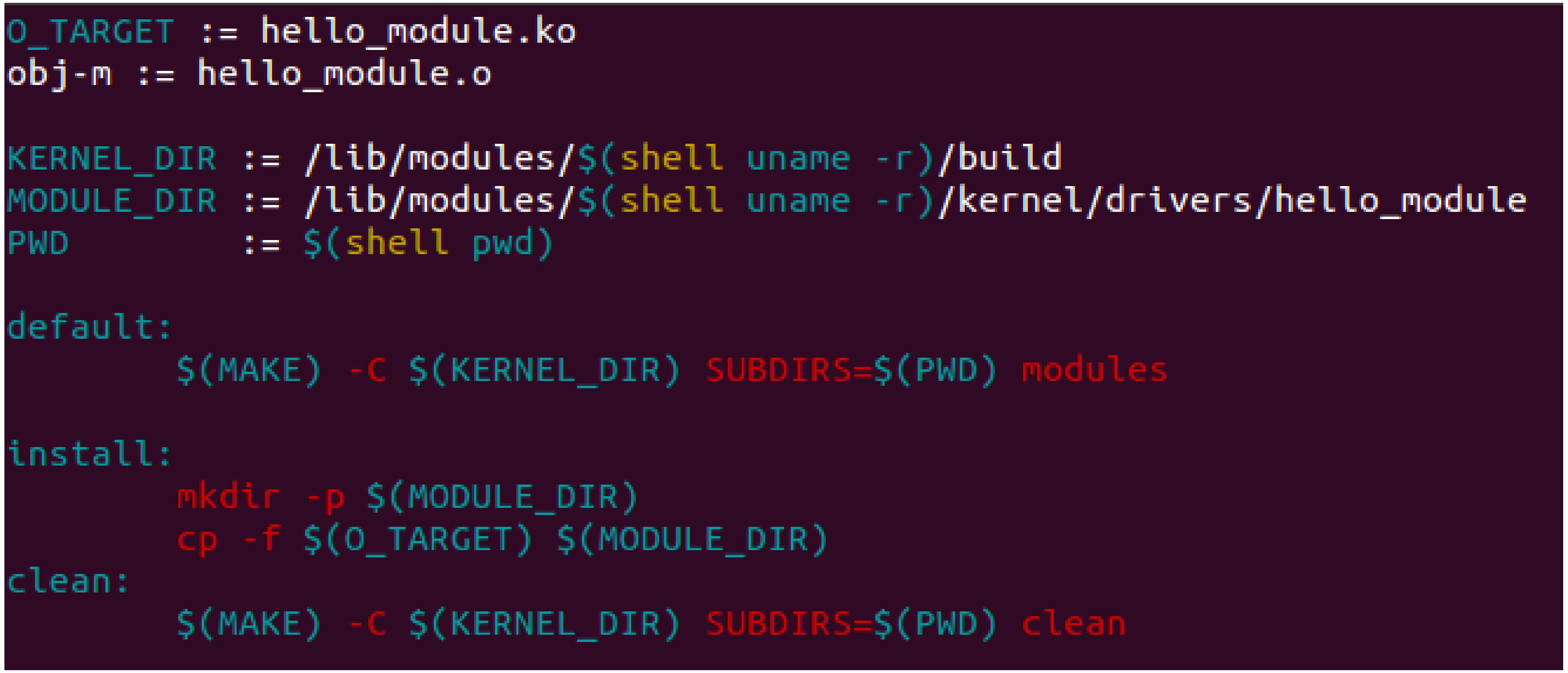
- Write a simple module code
- Write a Makefile
- Load the simple module to kernel

Simple Module Code



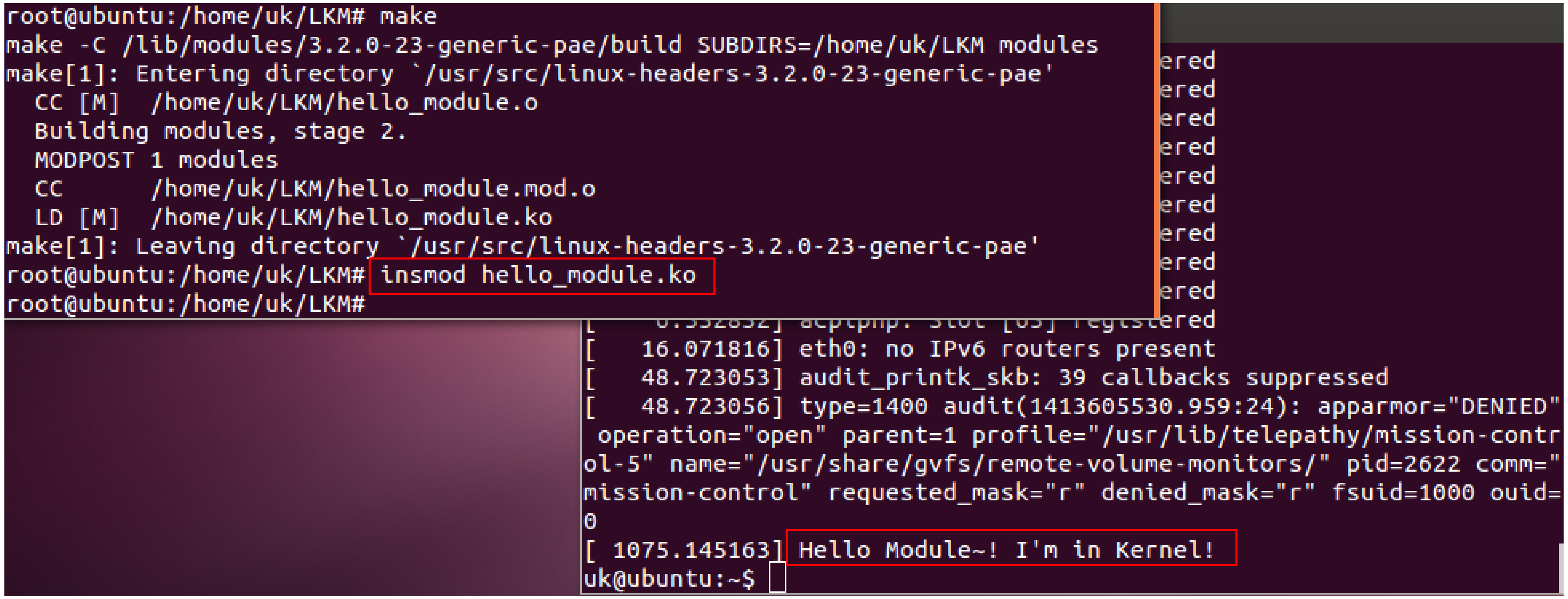
```
#include <linux/kernel.h>
#include <linux/module.h>

int hello_module_init(void)
{
        printk(KERN_EMERG "Hello Module~! I'm in Kernel!\n");
        return 0;
}

void hello_module_cleanup(void)
{
        printk("<0>Bye Module~!\n");
}

module_init(hello_module_init);
module_exit(hello_module_cleanup);

MODULE_LICENSE("GPL");
```

Makefile



```
O_TARGET := hello_module.ko
obj-m := hello_module.o

KERNEL_DIR := /lib/modules/$(shell uname -r)/build
MODULE_DIR := /lib/modules/$(shell uname -r)/kernel/drivers/hello_module
PWD        := $(shell pwd)

default:
        $(MAKE) -C $(KERNEL_DIR) SUBDIRS=$(PWD) modules

install:
        mkdir -p $(MODULE_DIR)
        cp -f $(O_TARGET) $(MODULE_DIR)
clean:
        $(MAKE) -C $(KERNEL_DIR) SUBDIRS=$(PWD) clean
```

Load the Module

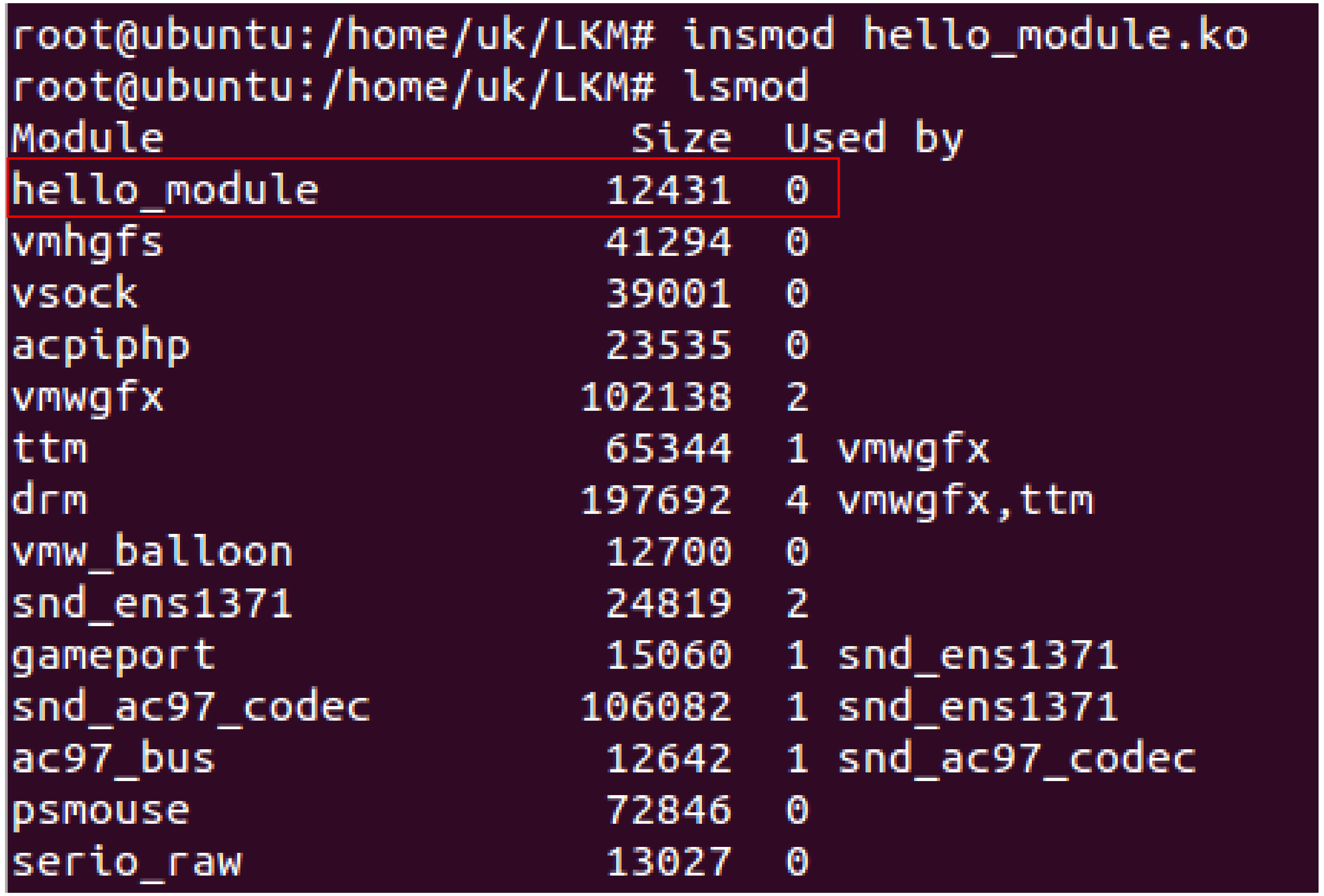


```
root@ubuntu:/home/uk/LKM# make
make -C /lib/modules/3.2.0-23-generic-pae/build SUBDIRS=/home/uk/LKM modules
make[1]: Entering directory `/usr/src/linux-headers-3.2.0-23-generic-pae'
  CC [M]  /home/uk/LKM/hello_module.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/uk/LKM/hello_module.mod.o
  LD [M]  /home/uk/LKM/hello_module.ko
make[1]: Leaving directory `/usr/src/linux-headers-3.2.0-23-generic-pae'
root@ubuntu:/home/uk/LKM# insmod hello_module.ko
root@ubuntu:/home/uk/LKM#
```

```
ered
ered
ered
ered
ered
ered
ered
ered
ered
[    0.332832] acpiphp: Slot [03] registered
[   16.071816] eth0: no IPv6 routers present
[   48.723053] audit_printk_skb: 39 callbacks suppressed
[   48.723056] type=1400 audit(1413605530.959:24): apparmor="DENIED"
 operation="open" parent=1 profile="/usr/lib/telepathy/mission-contr
ol-5" name="/usr/share/gvfs/remote-volume-monitors/" pid=2622 comm="
mission-control" requested_mask="r" denied_mask="r" fsuid=1000 ouid=
0
[ 1075.145163] Hello Module~! I'm in Kernel!
uk@ubuntu:~$
```

Check the Module List

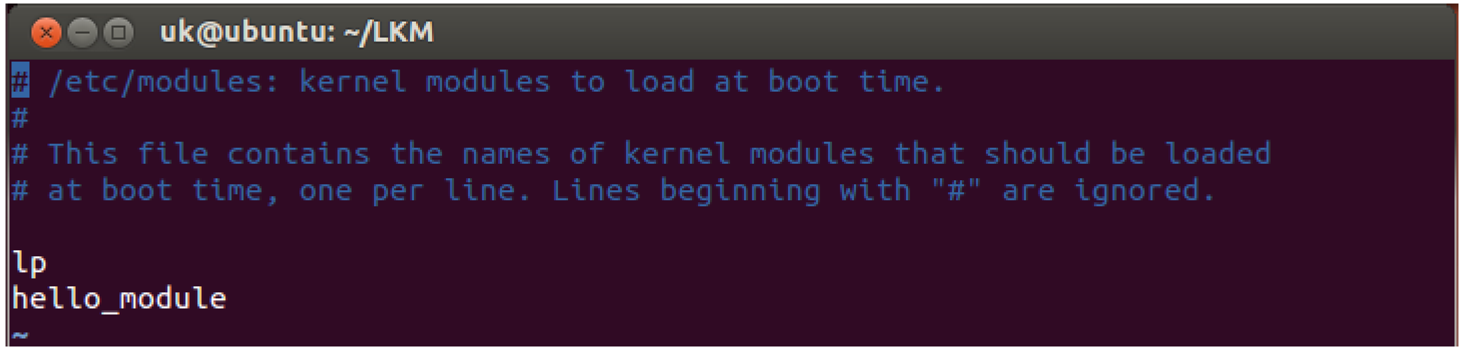
- $ lsmod
- $ cat /proc/modules



```
root@ubuntu:/home/uk/LKM# insmod hello_module.ko
root@ubuntu:/home/uk/LKM# lsmod
Module                  Size  Used by
hello_module           12431  0
vmhgfs                 41294  0
vsock                  39001  0
acpiphp                23535  0
vmwgfx                102138  2
ttm                    65344  1 vmwgfx
drm                   197692  4 vmwgfx,ttm
vmw_balloon            12700  0
snd_ens1371            24819  2
gameport               15060  1 snd_ens1371
snd_ac97_codec        106082  1 snd_ens1371
ac97_bus               12642  1 snd_ac97_codec
psmouse                72846  0
serio_raw              13027  0
```

Load a Module at the Boot Time

- $ make install
- Add the name of module to the /etc/modules



uk@ubuntu: ~/LKM

```
# /etc/modules: kernel modules to load at boot time.
#
# This file contains the names of kernel modules that should be loaded
# at boot time, one per line. Lines beginning with "#" are ignored.

lp
hello_module
~
```

- $ depmod

- Reboot

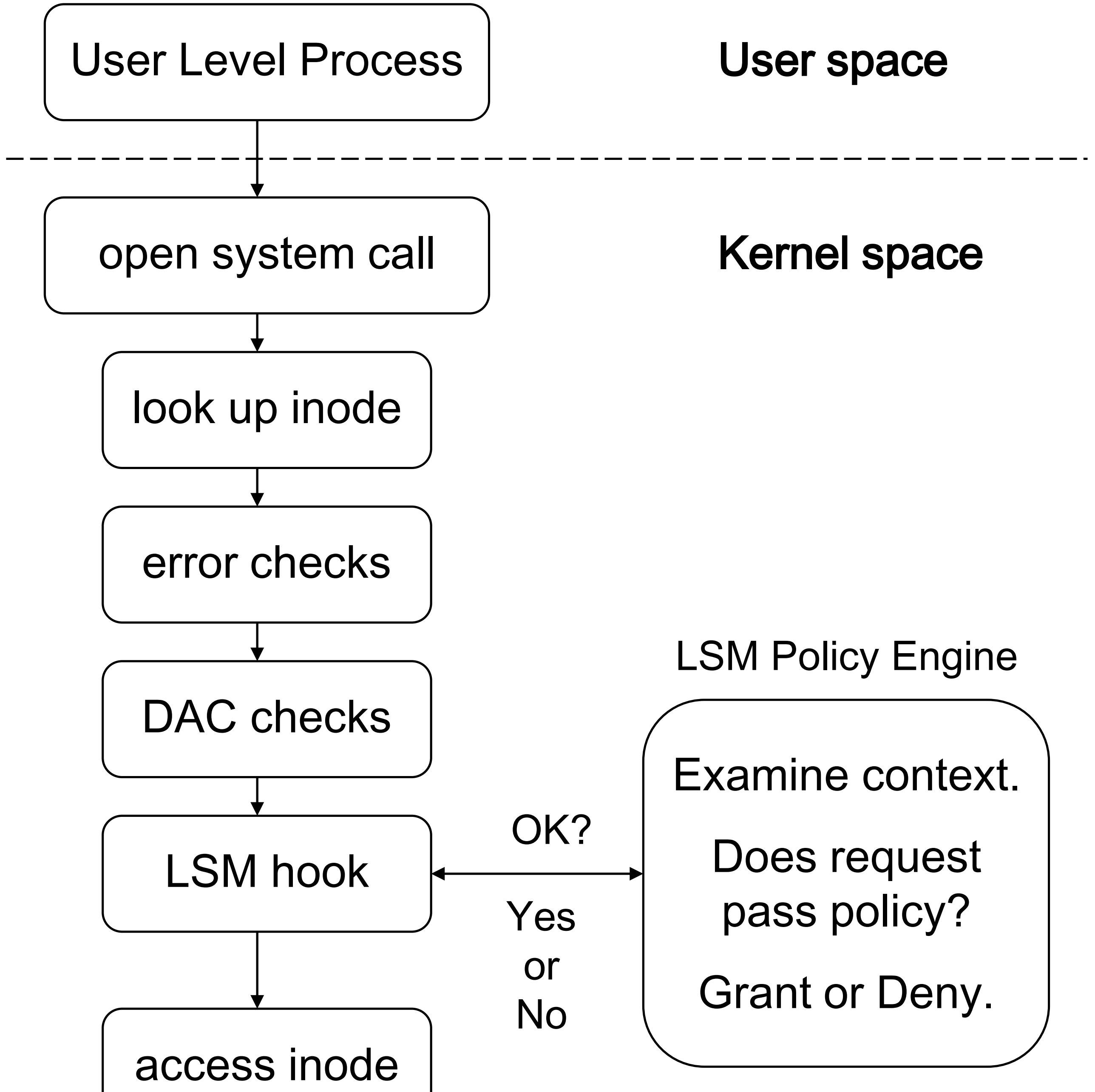
Linux Security Module

What is Linux Security Module?

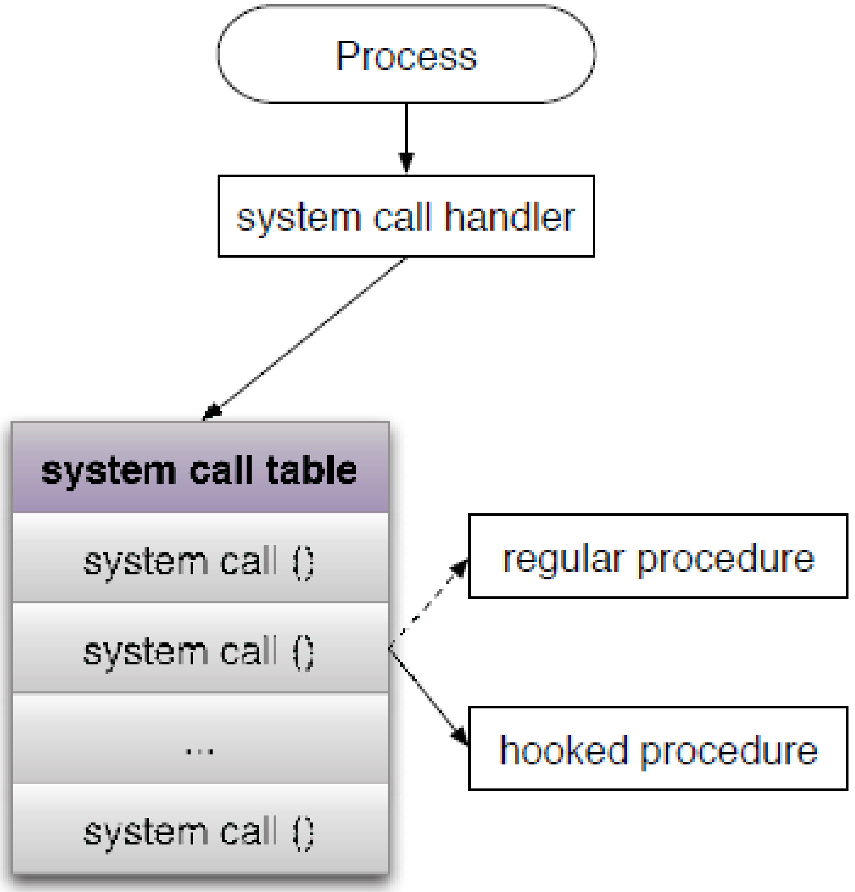
- **LSM(Linux Security Module)** is a framework that allows the Linux kernel to support a variety of computer security models while avoiding favoritism toward any single security implementation.
- The framework is standard part of the Linux kernel since Linux 2.6.
- **AppArmor, SELinux, Smack and TOMOYO** Linux are the currently accepted modules in the official kernel.
- LSM doesn't provide any security rather **it add security fields to kernel and provide interface** to manage these fields for maintaining security attributes.

Design of LSM

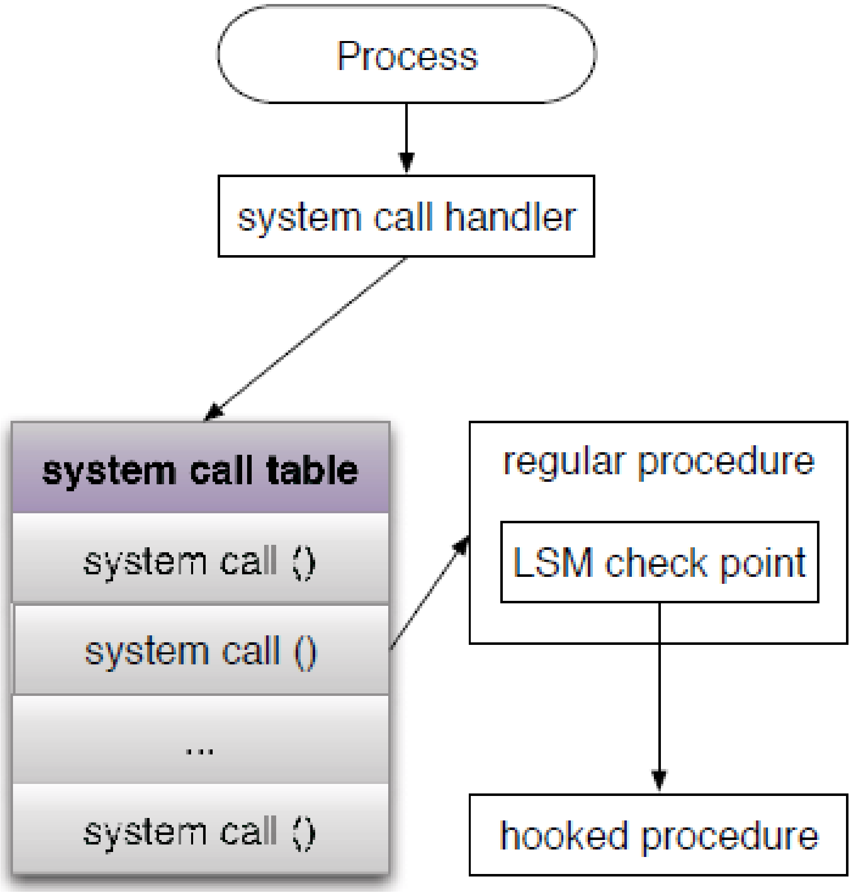
- LSM is to mediate access to internal kernel objects
- By placing **hooks** in kernel code just before the access
- LSM module provides the functions to be called by these hooks



System Call Hook vs. LSM Hook

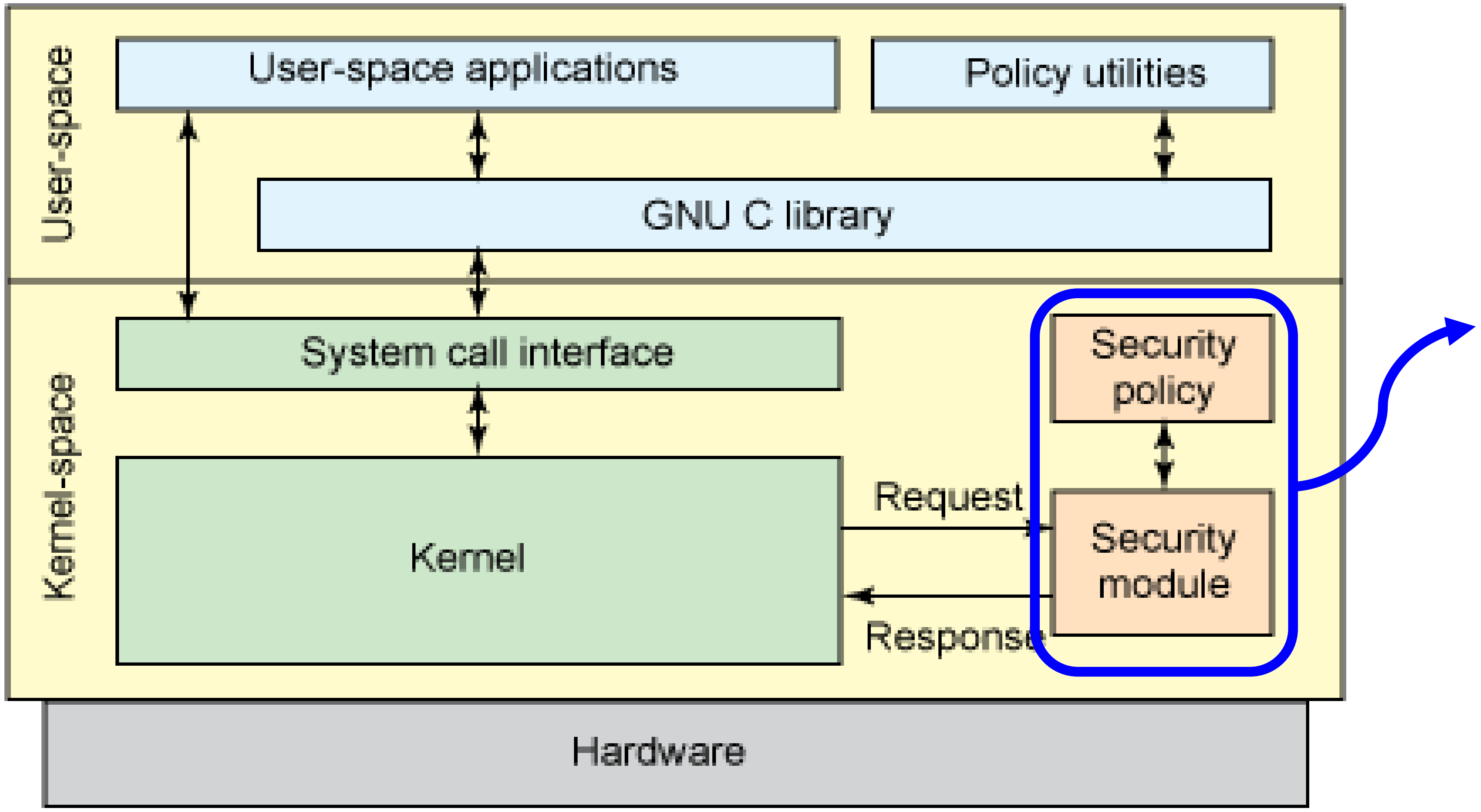


(a) sys_call_table export



(b) LSM

Linux Security Module Architecture



Security Policy
+
Security Module
=
LSM Framework

Make a Security Module (Tutorial 3)

- Write a simple security module code
- Write a Makefile
- Modify the kernel
- Load the security module to kernel

Simple Security Module Code

LSM hook Registration (1)

1. security_operation Structure Definition

sample.c

```
static struct security_operations sample_ops = {

.socket_connect  =  sample_socket_connect,
← hooking function   is defined for socket_connect

.inode_link =         sample_inode_link,
.inode_unlink =       sample_inode_unlink,
.inode_symlink =      sample_inode_symlink,
.inode_mkdir =        sample_inode_mkdir,
………… 생략 ………………
}
```

2. security_operation Structure Registration

Hooking function pointer is defined in security.h

include/linux/security.h

```
struct security_operations {
  char name[SECURITY_NAME_MAX + 1];

  int (*socket_connect) (struct socket *sock,  struct
                          sockaddr *address, int addrlen);
………….. omitted ………..
}
```

- define a hook to security operations structure
- When we want to add a hook, we can refer to security_operations in include/linux/security.h

sample.c

```
sample_socket_connect()
{
        //check permission
        check_perm(&perm);
}
```

Define a security check function when the connect() is called.

Simple Security Module Code

LSM hook Registration (2)

2. security_operation Structure Registration

security/sample/sample.c

```
static __init int sample_init(void)
{
        reset_security_ops();
        if (register_security (&sample_ops)) {
                printk("Sample: Unable to register with kernel.₩n");
                return 0;
         }

        printk(KERN_INFO "Sample:  Initializing.₩n");
        return 0;
}
```

- Register register_security () function using a predefined structure to register security_operations
- Defined previously registered sample_ops

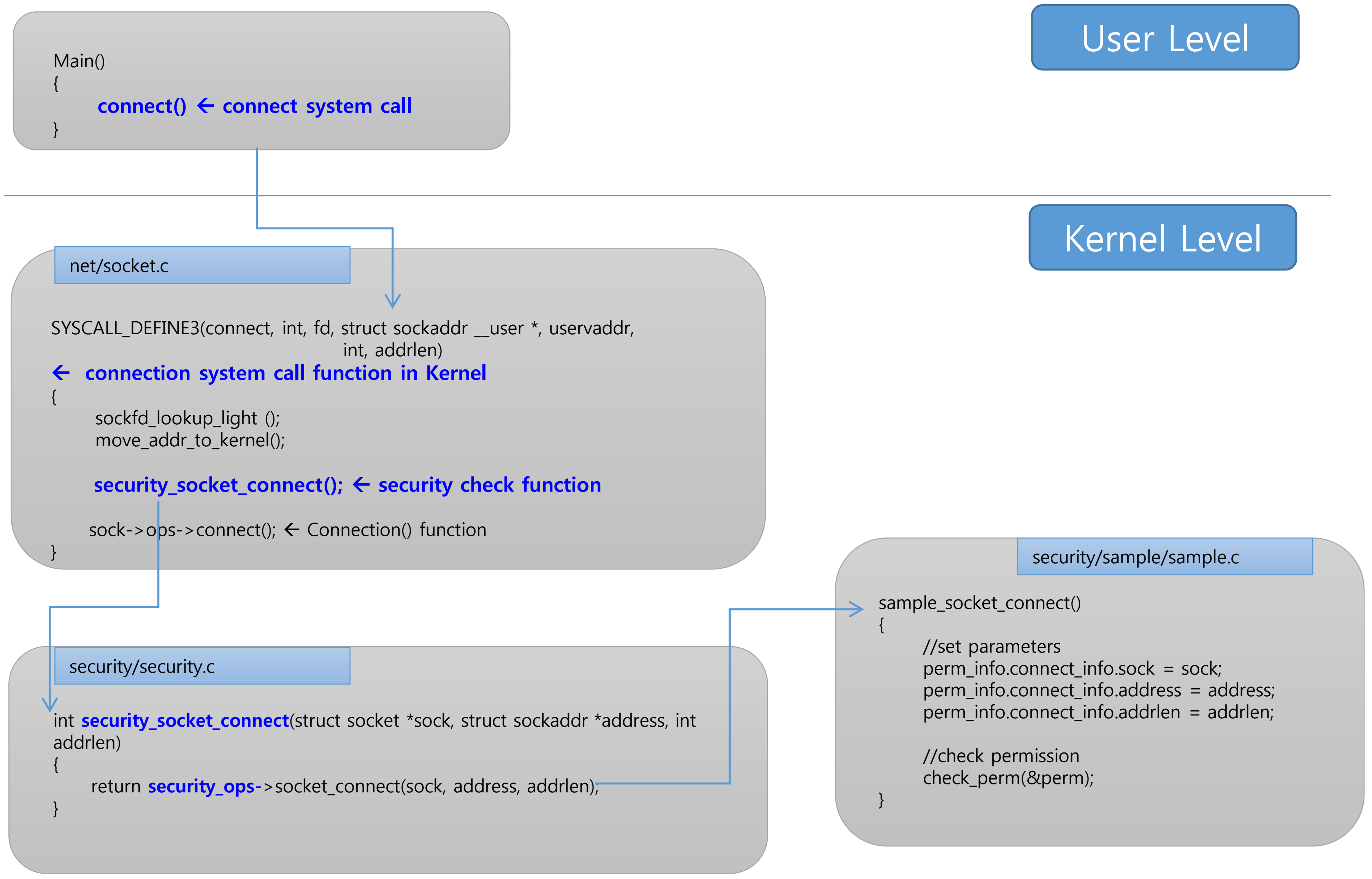
security/security.c

```
int register_security(struct security_operations *ops)
{
   .......... Omitted ............

     if (security_ops != &default_security_ops)
                 return -EAGAIN;
    security_ops = ops;
   ← sample_ops is registered as a security function
     return 0;
}
```

- security_operations is registered as a security function

Simple Security Module Code

Connect System Call



```
Main()
{
        connect() ← connect system call
}
```

net/socket.c

```
SYSCALL_DEFINE3(connect, int, fd, struct sockaddr __user *, uservaddr,
                                int, addrlen)
← connection system call function in Kernel
{
        sockfd_lookup_light ();
        move_addr_to_kernel();

        security_socket_connect(); ← security check function

        sock->ops->connect(); ← Connection() function
}
```

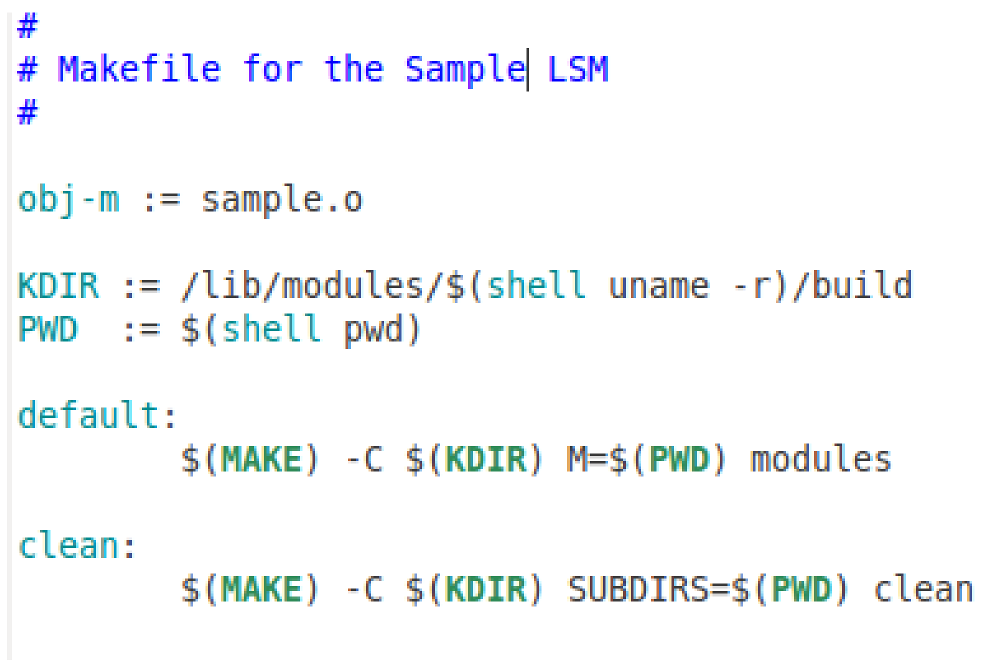
security/security.c

```
int security_socket_connect(struct socket *sock, struct sockaddr *address, int
addrlen)
{
        return security_ops->socket_connect(sock, address, addrlen);
}
```

security/sample/sample.c

```
sample_socket_connect()
{
        //set parameters
        perm_info.connect_info.sock = sock;
        perm_info.connect_info.address = address;
        perm_info.connect_info.addrlen = addrlen;

        //check permission
        check_perm(&perm);
}
```

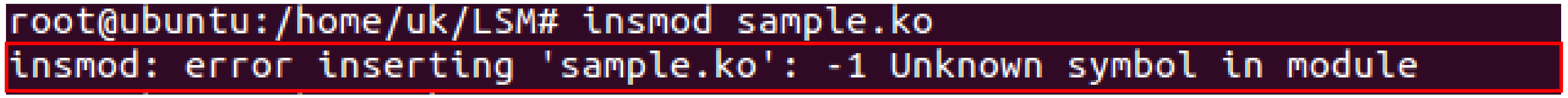
Write a Makefile

Makefile	Explanation
	• **obj-m :=** name of object file • **KDIR :=** kernel source directory(symbolic) • **PWD :=** module source directory • **-C** : change the directory to next parameter • **M=** module location (SUBDIRS is same meaning with M)

```
#
# Makefile for the Sample LSM
#

obj-m := sample.o

KDIR := /lib/modules/$(shell uname -r)/build
PWD  := $(shell pwd)

default:
        $(MAKE) -C $(KDIR) M=$(PWD) modules

clean:
        $(MAKE) -C $(KDIR) SUBDIRS=$(PWD) clean
```

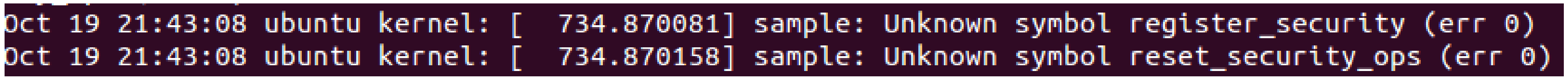
Have you got this error message?

Try to load the sample security module



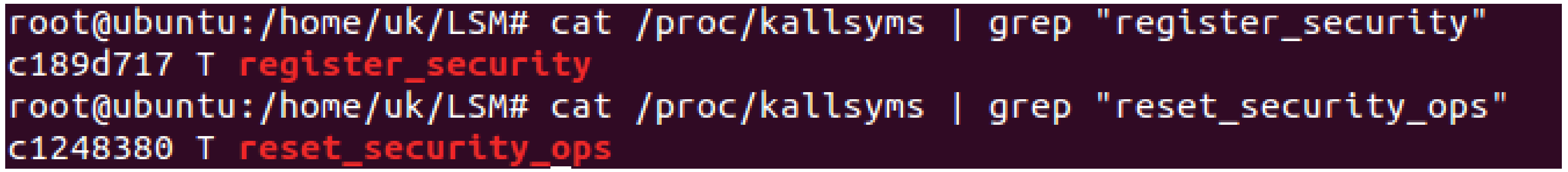
```
root@ubuntu:/home/uk/LSM# insmod sample.ko
insmod: error inserting 'sample.ko': -1 Unknown symbol in module
```

$ tail –f /var/log/syslog



```
Oct 19 21:43:08 ubuntu kernel: [  734.870081] sample: Unknown symbol register_security (err 0)
Oct 19 21:43:08 ubuntu kernel: [  734.870158] sample: Unknown symbol reset_security_ops (err 0)
```

Check the kernel symbol table

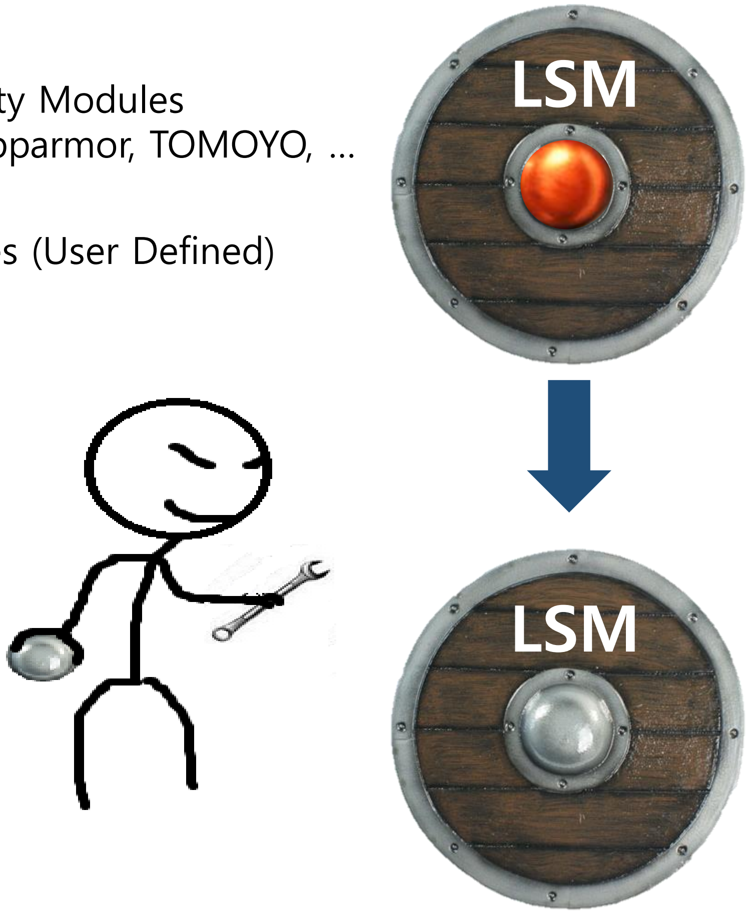
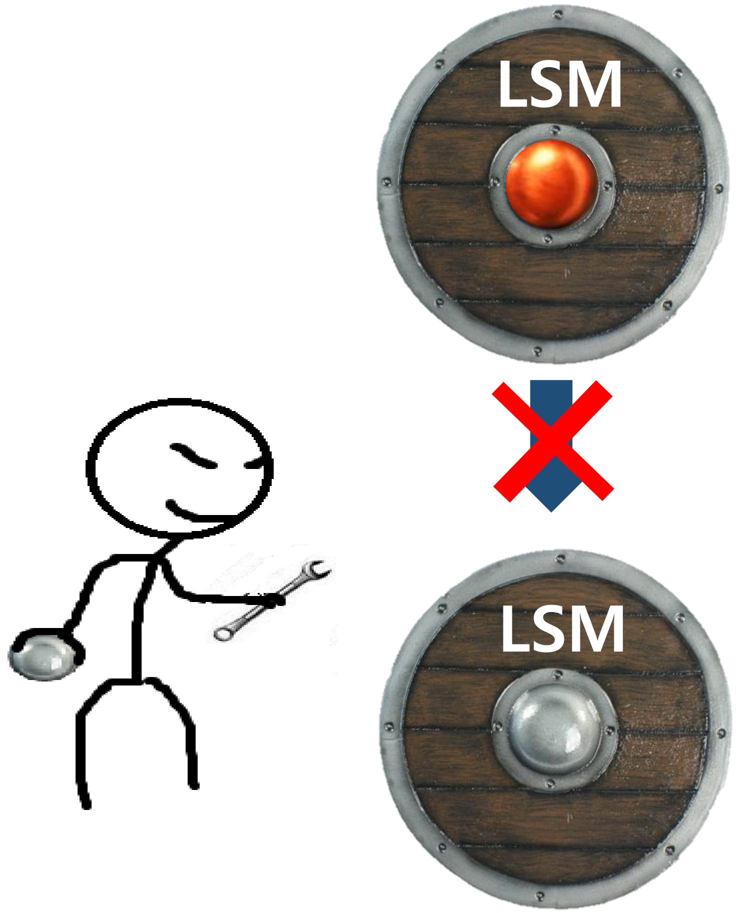


```
root@ubuntu:/home/uk/LSM# cat /proc/kallsyms | grep "register_security"
c189d717 T register_security
root@ubuntu:/home/uk/LSM# cat /proc/kallsyms | grep "reset_security_ops"
c1248380 T reset_security_ops
```

Why???

● : Built-In Security Modules
ex) SELinux, Apparmor, TOMOYO, …

● : Other Modules (User Defined)

Befor Linux v.2.6.24	After Linux v.2.6.24
	

We Need to Modify the Kernel

Basis on Linux Kernel v3.2.0

/include/linux/security.h

Line

Remove "__init" from

void __init security_fixup_ops(struct security_operations *ops) 1663

/security/security.c

Remove "__init" from

static inline int __init verify(struct security_operations *ops) 34

int __init register_security(struct security_operations *ops) 113

Add "EXPORT_SYMBOL()" at bottom line

```
EXPORT_SYMBOL(register_security);
EXPORT_SYMBOL(reset_security_ops);
```

/security/capability.c

Remove "__init" from

void __init security_fixup_ops(struct security_operations *ops) 875

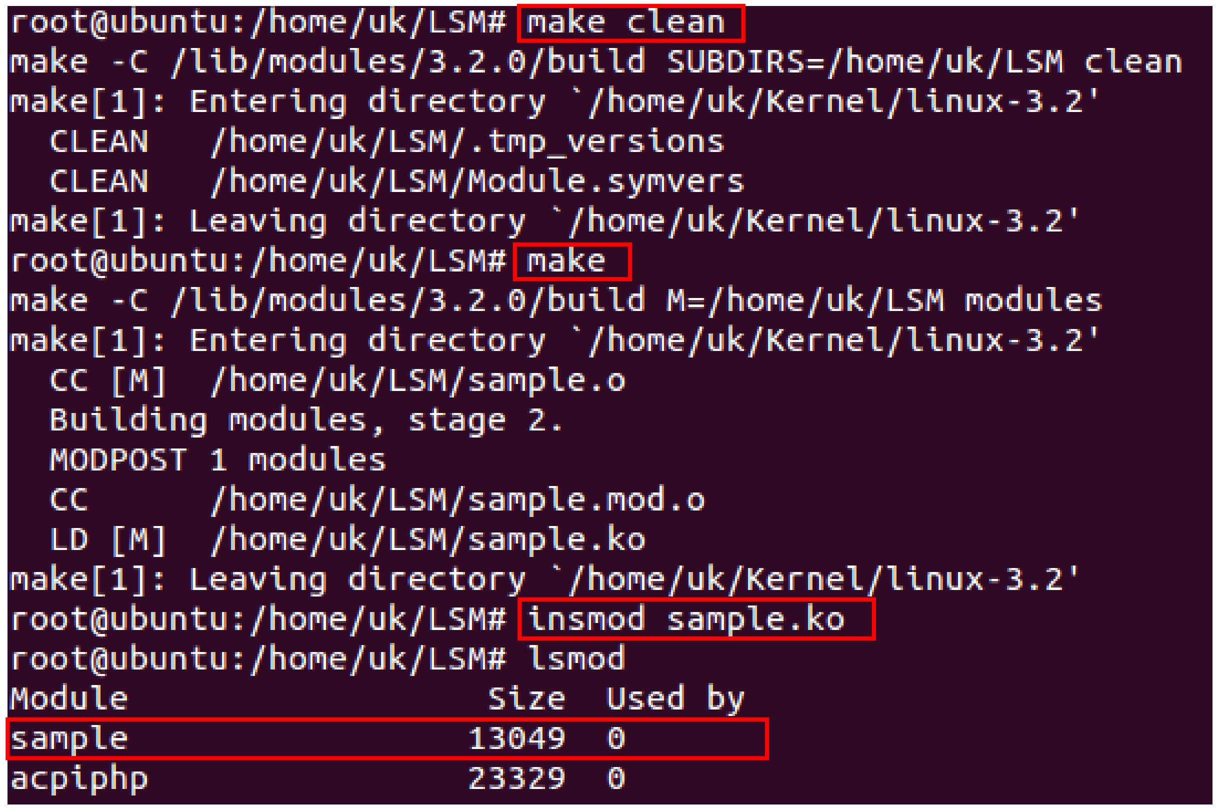
Kernel Compile and Rebooting

- Follow the below commands to re-compile the kernel
  1. make mrproper
  2. make menuconfig
  3. make clean
  4. make bzImage
  5. make install

- Reboot

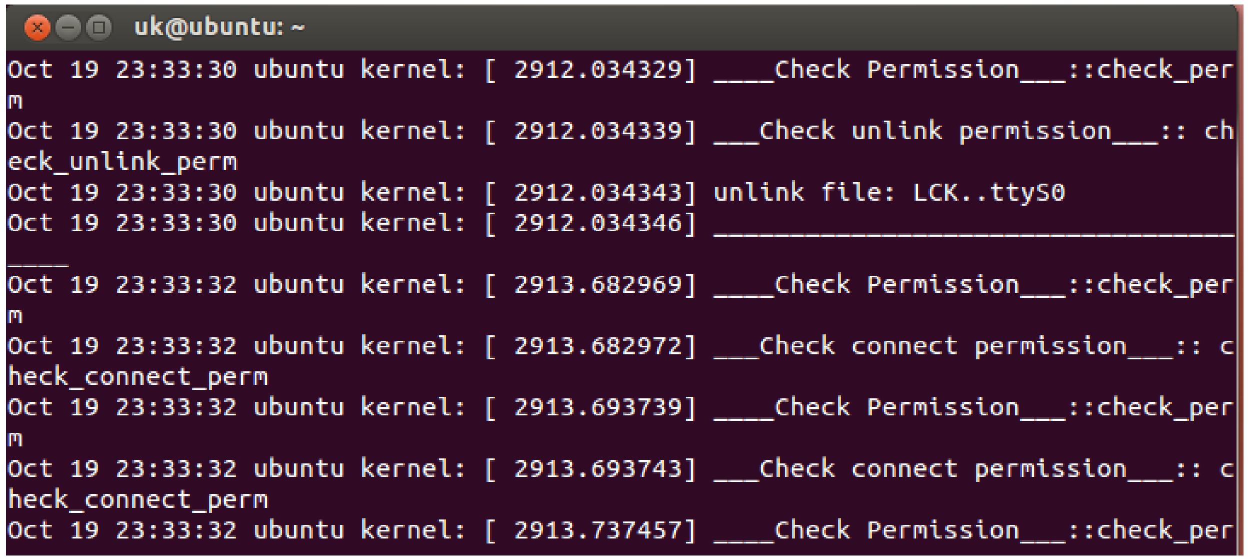
Load the Security Module to Kernel

Load the Security Module



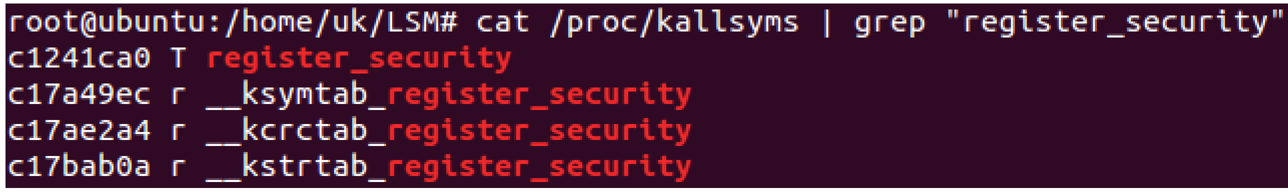
```
root@ubuntu:/home/uk/LSM# make clean
make -C /lib/modules/3.2.0/build SUBDIRS=/home/uk/LSM clean
make[1]: Entering directory `/home/uk/Kernel/linux-3.2'
  CLEAN   /home/uk/LSM/.tmp_versions
  CLEAN   /home/uk/LSM/Module.symvers
make[1]: Leaving directory `/home/uk/Kernel/linux-3.2'
root@ubuntu:/home/uk/LSM# make
make -C /lib/modules/3.2.0/build M=/home/uk/LSM modules
make[1]: Entering directory `/home/uk/Kernel/linux-3.2'
  CC [M]  /home/uk/LSM/sample.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/uk/LSM/sample.mod.o
  LD [M]  /home/uk/LSM/sample.ko
make[1]: Leaving directory `/home/uk/Kernel/linux-3.2'
root@ubuntu:/home/uk/LSM# insmod sample.ko
root@ubuntu:/home/uk/LSM# lsmod
Module                  Size  Used by
sample                 13049  0
acpiphp                23329  0
```

$ tail –f /var/log/syslog

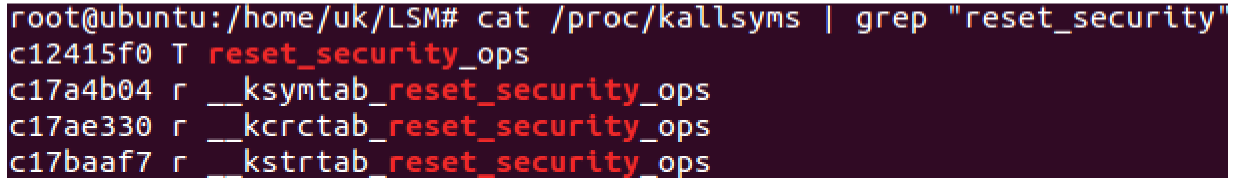


```
Oct 19 23:33:30 ubuntu kernel: [ 2912.034329] ____Check Permission___::check_per
m
Oct 19 23:33:30 ubuntu kernel: [ 2912.034339] ___Check unlink permission___:: ch
eck_unlink_perm
Oct 19 23:33:30 ubuntu kernel: [ 2912.034343] unlink file: LCK..ttyS0
Oct 19 23:33:30 ubuntu kernel: [ 2912.034346] __________________________________
____
Oct 19 23:33:32 ubuntu kernel: [ 2913.682969] ____Check Permission___::check_per
m
Oct 19 23:33:32 ubuntu kernel: [ 2913.682972] ___Check connect permission___:: c
heck_connect_perm
Oct 19 23:33:32 ubuntu kernel: [ 2913.693739] ____Check Permission___::check_per
m
Oct 19 23:33:32 ubuntu kernel: [ 2913.693743] ___Check connect permission___:: c
heck_connect_perm
Oct 19 23:33:32 ubuntu kernel: [ 2913.737457] ____Check Permission___::check_per
```

Check the kernel symbol table



```
root@ubuntu:/home/uk/LSM# cat /proc/kallsyms | grep "register_security"
c1241ca0 T register_security
c17a49ec r __ksymtab_register_security
c17ae2a4 r __kcrctab_register_security
c17bab0a r __kstrtab_register_security
```



```
root@ubuntu:/home/uk/LSM# cat /proc/kallsyms | grep "reset_security"
c12415f0 T reset_security_ops
c17a4b04 r __ksymtab_reset_security_ops
c17ae330 r __kcrctab_reset_security_ops
c17baaf7 r __kstrtab_reset_security_ops
```