Adding a “new” System call to Kernel

You can do it in two ways:

1. Easy and smart approach:
   Add the new system call by modifying an **existing** kernel file
2. Generic approach:
   Add the new system call by creating a **new** file in kernel

In following slides, generic approach is used to add a simple system call

Generic Approach

Step 1:

Create an entry for the system call in the kernel's syscall_table

- This file is located at ~/linux-4.2.5/source/arch/x86/entry/syscall/**syscall_64.tbl**
- Add a new entry for this new system call under "**64-bit system call numbers and entry vectors**" section. The number assigned for this system call must be unique.

  In syscall_64.tbl for linux-4.2.5 version, you will notice 0 to 322 numbers have already been used by other system calls, so add an entry with a new number.

  e.g.

  323 common myfoo sys_myfoo

Adding a new entry in syscall_64.tbl

Adding a new entry at the end of the list of system calls under "**64-bit system call numbers and entry vectors**" heading e.g.

323 common myfoo sys_myfoo

```
318     common  getrandom               sys_getrandom
319     common  memfd_create            sys_memfd_create
320     common  kexec_file_load         sys_kexec_file_load
321     common  bpf                     sys_bpf
322     64      execveat                stub_execveat
323     common  myfoo                   sys_myfoo

#
# x32-specific system call numbers start at 512 to avoid cache impact
# for native 64-bit operation.
#
512     x32     rt_sigaction            compat_sys_rt_sigaction
513     x32     rt_sigreturn            stub_x32_rt_sigreturn
514     x32     ioctl                   compat_sys_ioctl
515     x32     readv                   compat_sys_readv
516     x32     writev                  compat_sys_writev
```

Generic Approach

Step 2.a:

Write the system call code as a kernel function i.e. implementation of this new system call

- To do that (In generic approach), create a hello.c with following content and place this hello.c file in ~/linux-4.2.5/kernel/ folder:

```
#include  <linux/linkage.h>
#include  <linux/export.h>
#include  <linux/time.h>
#include  <asm/uaccess.h>
#include  <linux/printk.h>
#include  <linux/slab.h>
asmlinkage int sys_myfoo(void){
   printk(KERN_ALERT "Hello World!\n");
   return 0;
}
EXPORT_SYMBOL(sys_myfoo);
```

Generic Approach

Step 2.b:

Update the “Makefile” in ~/linux-4.2.5/kernel/ folder so “hello.c” gets compiled when you give command to compile the whole kernel.

- To update this Makefile: just add “hello.o” at the end of the list of object file names.

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

obj-y     = fork.o exec_domain.o panic.o \
            cpu.o exit.o softirq.o resource.o \
            sysctl.o sysctl_binary.o capability.o ptrace.o user.o \
            signal.o sys.o kmod.o workqueue.o pid.o task_work.o \
            extable.o params.o \
            kthread.o sys_ni.o nsproxy.o \
            notifier.o ksysfs.o cred.o reboot.o \
            async.o range.o smpboot.o hello.o

obj-$(CONFIG_MULTIUSER) += groups.o

ifdef CONFIG_FUNCTION_TRACER
# Do not trace debug files and internal ftrace files
```

Adding a “new” System call to Kernel

Step 3.a:

- Compile your kernel i.e. follow the instructions given in this link:

  http://www.cs.binghamton.edu/~kartik/cs350/lab_slides/kernel_compilation.html

  If you have already successfully compiled it before adding this new system call, then follow the instructions from step 7, else follow the instructions from step 1.
- After successful compilation, you will see “hello.o” object file is created in ~/linux-4.2.5/kernel/ folder.
- After compiling and rebooting your machine with your new image successfully, you can use this new system call in user space (see next slide).

Invoke your new handler with syscall

Step 3.b:

- Use the syscall() library function as explained in the system_calls.pdf slides

```
#include<stdio.h>
#include<unistd.h>
#include<linux/unistd.h>
#include<linux/time.h>
int main(){
        int y = 2;
        y = syscall(323);
        printf("syscall return value :%d\n",y);//negative value of y will indicate a failure
        return 0;
}
```

- Successful call will print “Hello World!” in /var/log/kern.log file
- Run “dmesg” command to check the content of the file