# System Calls

## What are they?

- Standard interface to allow the kernel to safely handle user requests
  - Read from hardware
  - Spawn a new process
  - Get current time
  - Create shared memory
- Message passing technique between
  - OS kernel (server)
  - User (client)

## Executing System Calls

- User program issues call
- Core kernel looks up call in syscall table
- Kernel module handles syscall action
- Module returns result of system call
- Core kernel forwards result to user

## Module is not Loaded...

- User program issues call
- Core kernel looks up call in syscall table
- Kernel module isn't loaded to handle action
- ...
- Where does call go?

## System Call Wrappers

- Wrapper calls system call if loaded
  - Otherwise returns an error
- Needs to be in a separate location so that the function can actually be called
  - Uses function pointer to point to kernel module implementation

# Adding System Calls

- You'll need to add and implement:
  - int start_elevator(void);
  - int issue_request(int, int, int);
  - int stop_elevator(void);
- As an example, let's add a call to printk an argument passed in:
  - int test_call(int);

# Adding System Calls

- Files to add (project files):
  - /usr/src/test_kernel/hello_world/test_call.c
  - /usr/src/test_kernel/hello_world/hello.c
  - /usr/src/test_kernel/hello_world/Makefile
- Files to modify (core kernel):
  - /usr/src/test_kernel/arch/x86/entry/syscalls/syscall_64.tbl
  - /usr/src/test_kernel/include/linux/syscalls.h
  - /usr/src/test_kernel/Makefile

# hello_world/test_call.c

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

/* System call stub */
long (*STUB_test_call)(int) = NULL;
EXPORT_SYMBOL(STUB_test_call);

/* System call wrapper */
asmlinkage long sys_test_call(int test_int) {
    if (STUB_test_call != NULL)
        return STUB_test_call(test_int)
    else
        return -ENOSYS;
}
```

Holds syscall pointer
Exports pointer for public use
Defines syscall wrapper

## hello_world/test_call.c

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

/* System call stub */
long (*STUB_test_call)(int) = NULL;
EXPORT_SYMBOL(STUB_test_call);

/* System call wrapper */
asmlinkage long sys_test_call(int test_int) {
    if (STUB_test_call != NULL)
        return STUB_test_call(test_int)
    else
        return -ENOSYS;
}
```

System Call Library

# hello_world/test_call.c

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

/* System call stub */
long (*STUB_test_call)(int) = NULL;
EXPORT_SYMBOL(STUB_test_call);

/* System call wrapper */
asmlinkage long sys_test_call(int test_int) {
    if (STUB_test_call != NULL)
        return STUB_test_call(test_int)
    else
        return -ENOSYS;
}
```

Function Pointer

# hello_world/test_call.c

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

/* System call stub */
long (*STUB_test_call)(int) = NULL;
EXPORT_SYMBOL(STUB_test_call);

/* System call wrapper */
asmlinkage long sys_test_call(int test_int) {
    if (STUB_test_call != NULL)
        return STUB_test_call(test_int)
    else
        return -ENOSYS;
}
```

Export pointer to access from other places

# hello_world/test_call.c

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

/* System call stub */
long (*STUB_test_call)(int) = NULL;
EXPORT_SYMBOL(STUB_test_call);

/* System call wrapper */
asmlinkage long sys_test_call(int test_int) {
    if (STUB_test_call != NULL)
        return STUB_test_call(test_int)
    else
        return -ENOSYS;
}
```

Wrapper Function

# hello_world/hello.c

```
extern long (*STUB_test_call)(int test_int);
long my_test_call(int test) {
    printk(“%s: Your int is %i\n”, __FUNCTION__, test);
    return test;
}
my_module_init() {
    STUB_test_call =& (my_test_call);
    return 0;
}
my_module_exit() {
    STUB_test_call = NULL;
}
```

Holds module code
Registers syscall pointer
Implements syscall behavior

# hello_world/hello.c

```
extern long (*STUB_test_call)(int test_int);
long my_test_call(int test) {
    printk("%s: Your int is %i\n", __FUNCTION__, test);
    return test;
}
my_module_init() {
    STUB_test_call =& (my_test_call);
    return 0;
}
my_module_exit() {
    STUB_test_call = NULL;
}
```

Get access to syscall pointer

## hello_world/hello.c

```
extern long (*STUB_test_call)(int test_int);
long my_test_call(int test) {
    printk("%s: Your int is %i\n", __FUNCTION__, test);
    return test;
}
my_module_init() {
    STUB_test_call =& (my_test_call);
    return 0;
}
my_module_exit() {
    STUB_test_call = NULL;
}
```

Implement syscall

# hello_world/hello.c

```
extern long (*STUB_test_call)(int test_int);
long my_test_call(int test) {
    printk("%s: Your int is %i\n", __FUNCTION__, test);
    return test;
}
my_module_init() {
    STUB_test_call =& (my_test_call);
    return 0;
}
my_module_exit() {
    STUB_test_call = NULL;
}
```

Point stub to implementation on load

# hello_world/hello.c

```
extern long (*STUB_test_call)(int test_int);
long my_test_call(int test) {
    printk(“%s: Your int is %i\n”, __FUNCTION__, test);
    return test;
}
my_module_init() {
    STUB_test_call =& (my_test_call);
    return 0;
}
my_module_exit() {
    STUB_test_call = NULL;
}
```

Reset stub to NULL on unload

## hello_world/Makefile

```
obj-m := hello_world.o
obj-y := test_call.o

KDIR := /lib/modules/4.2.0/build
PWD := $(shell pwd)

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

# hello_world/Makefile

```
obj-m := hello_world.o
obj-y := test_call.o

KDIR := /lib/modules/4.2.0/build
PWD := $(shell pwd)

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

Compile as a module

# hello_world/Makefile

```
obj-m := hello_world.o
obj-y := test_call.o

KDIR := /lib/modules/4.2.0/build
PWD := $(shell pwd)


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

Compile as a kernel built-in

# arch/x86/entry/syscalls/syscall_64.tbl

```
316     common  renameat2           sys_renameat2
317     common  seccomp             sys_seccomp
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 test_call            sys_test_call

#
# 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
```

Line:332

Remember syscall numbers for userspace applications

# include/linux/syscalls.h

```
asmlinkage long sys_kcmp(pid_t pid1, pid_t pid2, int type,
                         unsigned long idx1, unsigned long idx2);
asmlinkage long sys_finit_module(int fd, const char __user *uargs, int flags);
asmlinkage long sys_seccomp(unsigned int op, unsigned int flags,
                            const char __user *uargs);
asmlinkage long sys_getrandom(char __user *buf, size_t count,
                              unsigned int flags);
asmlinkage long sys_bpf(int cmd, union bpf_attr *attr, unsigned int size);

asmlinkage long sys_execveat(int dfd, const char __user *filename,
                        const char __user *const __user *argv,
                        const char __user *const __user *envp, int flags);

asmlinkage long sys_test_call(int test_int);

#endif
```

Line:887

Defines syscall prototype

# ./Makefile

```
...
ifeq ($(KBUILD_EXTMOD),)
core-y        += kernel/ mm/ fs/ ipc/ security/ crypto/ block/ hello_world/

vmlinux-dirs    := $(patsubst %/,%,$(filter %/, $(init-y) $(init-m) \
                   $(core-y) $(core-m) $(drivers-y) $(drivers-m) \
                   $(net-y) $(net-m) $(libs-y) $(libs-m)))

vmlinux-alldirs := $(sort $(vmlinux-dirs) $(patsubst %/,%,$(filter %/, \
                   $(init-) $(core-) $(drivers-) $(net-) $(libs-))))

init-y        := $(patsubst %/, %/built-in.o, $(init-y))
core-y        := $(patsubst %/, %/built-in.o, $(core-y))
drivers-y     := $(patsubst %/, %/built-in.o, $(drivers-y))
net-y         := $(patsubst %/, %/built-in.o, $(net-y))
libs-y1       := $(patsubst %/, %/lib.a, $(libs-y))
libs-y2       := $(patsubst %/, %/built-in.o, $(libs-y))
libs-y        := $(libs-y1) $(libs-y2)
...
```

Line:889

Need to always include syscall wrapper

## User-space Program

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/syscall.h>
#define __NR_TEST_CALL  323

int test_call(int test) {
    return syscall(__NR_TEST_CALL, test);
}

int main() {
    int test = 5;
    long ret = test_call(test);
    if (ret < 0)
        perror(“system call error”);
    else
        printf(“Function successful, returned %i\n”, ret);
    return 0;
}
```

## User-space Program

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/syscall.h>
#define __NR_TEST_CALL  323

int test_call(int test) {
    return syscall(__NR_TEST_CALL, test);
}

int main() {
    int test = 5;
    long ret = test_call(test);
    if (ret < 0)
        perror(“system call error”);
    else
        printf(“Function successful, returned %i\n”, ret);
    return 0;
}
```

Definition of syscall()

# User-space Program

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/syscall.h>
#define __NR_TEST_CALL  323

int test_call(int test) {
    return syscall(__NR_TEST_CALL, test);
}

int main() {
    int test = 5;
    long ret = test_call(test);
    if (ret < 0)
        perror(“system call error”);
    else
        printf(“Function successful, returned %i\n”, ret);
    return 0;
}
```

Syscall Number

# User-space Program

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/syscall.h>
#define __NR_TEST_CALL  323

int test_call(int test) {
    return syscall(__NR_TEST_CALL, test);
}

int main() {
    int test = 5;
    long ret = test_call(test);
    if (ret < 0)
        perror(“system call error”);
    else
        printf(“Function successful, returned %i\n”, ret);
    return 0;
}
```

Wrapper Function

# User-space Program

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/syscall.h>
#define __NR_TEST_CALL  323

int test_call(int test) {
    return syscall(__NR_TEST_CALL, test);
}

int main() {
    int test = 5;
    long ret = test_call(test);
    if (ret < 0)
        perror("system call error");
    else
        printf("Function successful, returned %i\n", ret);
    return 0;
}
```

System Call

## User-space Program

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/syscall.h>
#define __NR_TEST_CALL  323

int test_call(int test) {
    return syscall(__NR_TEST_CALL, test);
}

int main() {
    int test = 5;
    long ret = test_call(test);
    if (ret < 0)
        perror(“system call error”);
    else
        printf(“Function successful, returned %i\n”, ret);
    return 0;
}
```

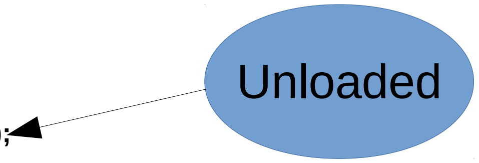
Wrapper Call

## User-space Program

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/syscall.h>
#define __NR_TEST_CALL  323

int test_call(int test) {
    return syscall(__NR_TEST_CALL, test);
}

int main() {
    int test = 5;
    long ret = test_call(test);
    if (ret < 0)
        perror(“system call error”);
    else
        printf(“Function successful, returned %i\n”, ret);
    return 0;
}
```



# User-space Program

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/syscall.h>
#define __NR_TEST_CALL  323

int test_call(int test) {
    return syscall(__NR_TEST_CALL, test);
}

int main() {
    int test = 5;
    long ret = test_call(test);
    if (ret < 0)
        perror("system call error");
    else
        printf("Function successful, returned %i\n", ret);
    return 0;
}
```