

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 it is loaded
 - Otherwise it returns an error
- Needs to be in a separate location so that the function can actually be called
 - Uses a function pointer to point to the kernel module implementation
- You'll need to create a system call wrapper for each system call you add

Adding System Calls

- For project 2, 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 (module files):
 - /usr/src/test_kernel/example3_syscall/
 - /usr/src/test_kernel/example3_syscall/test_call.c
 - /usr/src/test_kernel/example3_syscall/hello.c
 - /usr/src/test_kernel/example3_syscall/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

example3_syscall/test_call.c

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

/* System call stub */
long (*STUB_test_call)(int)
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

example3_syscall/test_call.c

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

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

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



System Call
Library

```
/* 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;
```

```
}
```

example3_syscall/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;
```

```
}
```

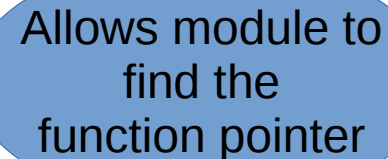


Function
Pointer

example3_syscall/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);
```



Allows module to
find the
function pointer

```
/* 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;
}
```

example3_syscall/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

example3_syscall/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;
}
```



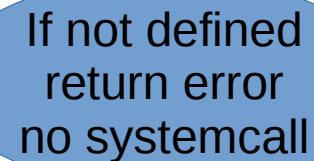
Execute if defined

example3_syscall/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;
}
```



If not defined
return error
no syscall


example3_syscall/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

example3_syscall/hello.c

```
extern long (*STUB_test_call) (int);  
long my_test_call(int test) {  
    printk("%s: Your int is %i\n", __FUNCTION__  
    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

example3_syscall/hello.c

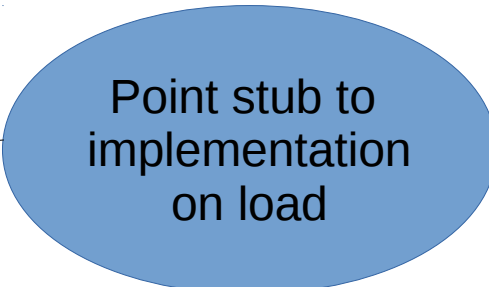
```
extern long (*STUB_test_call)(int);  
long my_test_call(int test) {  
    printk(KERN_NOTICE "%s: Your int is %d\n", __FUNCTION__, test);  
    return test;  
}  
my_module_init() {  
    STUB_test_call = my_test_call;  
    return 0;  
}  
my_module_exit() {  
    STUB_test_call = NULL;  
}
```



Syscall
Implementation

example3_syscall/hello.c

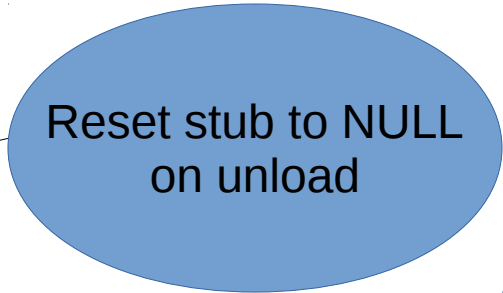
```
extern long (*STUB_test_call) (int);  
long my_test_call(int test) {  
    printk("%s: Your int is %i\n", __FUNCTION__, test);  
    return test;  
}  
my_module_init() {  
    STUB_test_call = my_test_call;  
    return 0;  
}  
my_module_exit() {  
    STUB_test_call = NULL;  
}
```



Point stub to
implementation
on load

example3_syscall/hello.c

```
extern long (*STUB_test_call) (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

example3_syscall/Makefile

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

```
obj-m := hello.o
```

```
PWD := $(shell pwd)
```

```
KDIR := /lib/modules/`uname -r`/build
```

```
default:
```

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

```
clean:
```

```
Rm -f *.o *.ko *.mod.* Module.* modules.*
```

example3_syscall/Makefile

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

```
obj-m := hello.o
```



Compile files
directly into kernel

```
PWD := $(shell pwd)
```

```
KDIR := /lib/modules/`uname -r`/build
```

```
default:
```

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

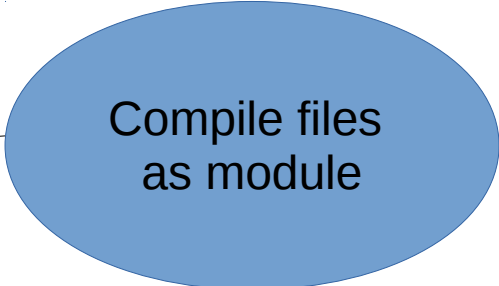
```
clean:
```

```
Rm -f *.o *.ko *.mod.* Module.* modules.*
```

example3_syscall/Makefile

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

```
obj-m := hello.o
```



Compile files
as module

```
PWD := $(shell pwd)
```

```
KDIR := /lib/modules/`uname -r`/build
```

```
default:
```

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

```
clean:
```

```
Rm -f *.o *.ko *.mod.* Module.* modules.*
```

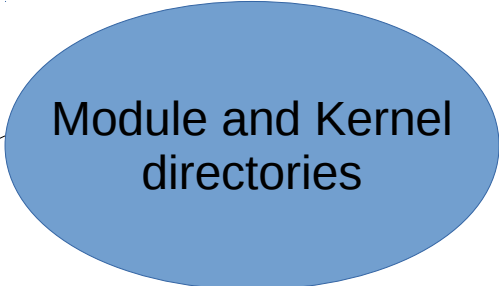
example3_syscall/Makefile

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

```
obj-m := hello.o
```

```
PWD := $(shell pwd)
```

```
KDIR := /lib/modules/`uname -r`/build
```



Module and Kernel
directories

```
default:
```

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

```
clean:
```

```
Rm -f *.o *.ko *.mod.* Module.* modules.*
```

example3_syscall/Makefile

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

```
obj-m := hello.o
```

```
PWD := $(shell pwd)
```

```
KDIR := /lib/modules/`uname -r`/build
```

```
default:
```

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

```
clean:
```

```
    Rm -f *.o *.ko *.mod.* Module.* modules.*
```



Compiles this module

example3_syscall/Makefile

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

```
obj-m := hello.o
```

```
PWD := $(shell pwd)
```

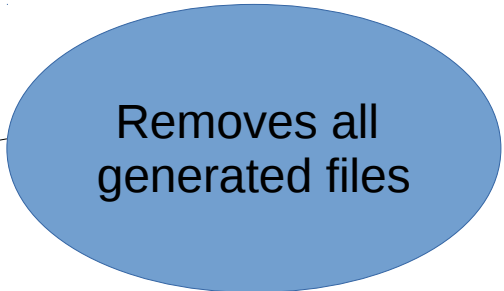
```
KDIR := /lib/modules/`uname -r`/build
```

```
default:
```

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

```
clean:
```

```
rm -f *.o *.ko *.mod.* Module.* modules.*
```



Removes all
generated files

arch/x86/entry/syscalls/syscall_64.tbl

327	64	preadv2	sys_preadv2
328	64	pwritev2	sys_pwritev2
329	common	pkey_mprotect	sys_pkey_mprotect
330	common	pkey_alloc	sys_pkey_alloc
331	common	pkey_free	sys_pkey_free
332	common	statx	sys_statx
333	common	test_call	sys_test_call

Line:342

System call table

Remember syscall numbers
for userspace applications

```
#  
# 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
```

include/linux/syscalls.h

```
Asmlinkage long sys_copy_file_range(int fd_in, loff_t __user *off_in,  
                                     int fd_out, loff_t __user *off_out,  
                                     size_t len, unsigned int flags);  
  
asmlinkage long sys_mlock2(unsigned long start, size_t len, int flags);  
  
asmlinkage long sys_pkey_mprotect(unsigned long start, size_t len, unsigned  
                                   long prot, int pkey);  
asmlinkage long sys_pkey_alloc(unsigned long flags, unsigned long init_val);  
asmlinkage long sys_pkey_free(int pkey);  
asmlinkage long sys_statx(int dfd, const char __user *path, unsigned flags,  
                          unsigned mask, struct statx __user *buffer);  
  
asmlinkage long sys_test_call(int test_int);  
  
#endif
```

End of Document

Defines syscall
prototype

./Makefile

Directories that have
files to be built
directly into the kernel

```
...
ifeq ($(KBUILD_EXTMOD),)
core-y          += kernel/ certs mm/ fs/ ipc/ security/ crypto/ block/ example3_syscall/

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

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

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         := $(filter-out %.a, $(patsubst %/, %/built-in.o, $(libs-y)))
virt-y          := $(patsubst %/, %/built-in.o, $(virt-y))
...
```

User-space Program

```
#include <sys/syscall.h>
#define __NR_TEST_CALL 333

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

int main(int argc, char **argv) {
    if argc != 2)
        perror("wrong number of args");

    int test = atoi(argv[1]);
    long ret = test_call(test);

    int test = atoi(argv[1]);
    long ret = test_call(test);
    if (ret < 0)
        perror("system call error");
    else
        printf("Function successful, passed in %d, returned %d\n", test, ret);
    return 0;
}
```

User-space Program

```
#include <sys/syscall.h>
#define __NR_TEST_CALL 333
```

Definition of
syscall()



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

```
int main(int argc, char **argv) {
    if argc != 2)
        perror("wrong number of args");

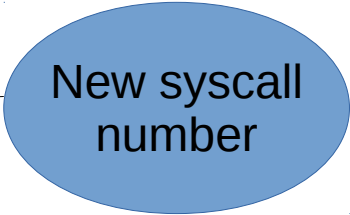
    int test = atoi(argv[1]);
    long ret = test_call(test);

    int test = atoi(argv[1]);
    long ret = test_call(test);
    if (ret < 0)
        perror("system call error");
    else
        printf("Function successful, passed in %d, returned %d\n", test, ret);
    return 0;
}
```

User-space Program

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

```
#define __NR_TEST_CALL 333
```



New syscall
number

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

```
int main(int argc, char **argv) {  
    if argc != 2)  
        perror("wrong number of args");  
  
    int test = atoi(argv[1]);  
    long ret = test_call(test);  
  
    int test = atoi(argv[1]);  
    long ret = test_call(test);  
    if (ret < 0)  
        perror("system call error");  
    else  
        printf("Function successful, passed in %d, returned %d\n", test, ret);  
    return 0;  
}
```

User-space Program

```
#include <sys/syscall.h>
#define __NR_TEST_CALL 333
```

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



Wrapper
function

```
int main(int argc, char **argv) {
    if argc != 2)
        perror("wrong number of args");

    int test = atoi(argv[1]);
    long ret = test_call(test);

    int test = atoi(argv[1]);
    long ret = test_call(test);
    if (ret < 0)
        perror("system call error");
    else
        printf("Function successful, passed in %d, returned %d\n", test, ret);
    return 0;
}
```


User-space Program

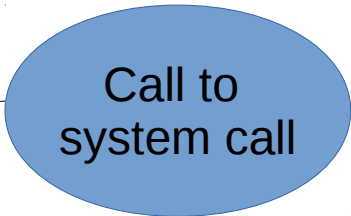
```
#include <sys/syscall.h>
#define __NR_TEST_CALL 333

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

int main(int argc, char **argv) {
    if argc != 2)
        perror("wrong number of args");

    int test = atoi(argv[1]);
    long ret = test_call(test);

    int test = atoi(argv[1]);
    long ret = test_call(test);
    if (ret < 0)
        perror("system call error");
    else
        printf("Function successful, passed in %d, returned %d\n", test, ret);
    return 0;
}
```



Call to
system call

User-space Program

```
#include <sys/syscall.h>
#define __NR_TEST_CALL 333

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

int main(int argc, char **argv) {
    if argc != 2)
        perror("wrong number of args");

    int test = atoi(argv[1]);
    long ret = test_call(test);

    int test = atoi(argv[1]);
    long ret = test_call(test);
    if (ret < 0)
        perror("system call error");
    else
        printf("Function successful, passed in %d, returned %d\n", test, ret);
    return 0;
}
```



Wrapper call

User-space Program

```
#include <sys/syscall.h>
#define __NR_TEST_CALL 333

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

int main(int argc, char **argv) {
    if argc != 2)
        perror("wrong number of args");

    int test = atoi(argv[1]);
    long ret = test_call(test);

    int test = atoi(argv[1]);
    long ret = test_call(test);
    if (ret < 0)
        perror("system call error");
    else
        printf("Function successful, passed in %d, returned %d\n", test, ret);
    return 0;
}
```



Unloaded

User-space Program

```
#include <sys/syscall.h>
#define __NR_TEST_CALL 333

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

int main(int argc, char **argv) {
    if argc != 2)
        perror("wrong number of args");

    int test = atoi(argv[1]);
    long ret = test_call(test);

    int test = atoi(argv[1]);
    long ret = test_call(test);
    if (ret < 0)
        perror("system call error");
    else
        printf("Function successful, passed in %d, returned %d\n", test, ret);
    return 0;
}
```



Loaded

Notes

- To use this example you'll need to edit various files by hand in addition to downloading the module
 - Edit all files outside of the module
- The system call number of this example is the same as that of `start_elevator` (since it's the next available slot)
 - So you may either want to add this after adding these elevator calls (and change the number)
 - Or go back and replace it with them later
- Adding new system calls (and anything to the kernel proper) requires recompiling and reinstalling the entire kernel
 - If you get unknown symbol errors, that's probably why