

HY345 - Assignment 3 Tutorial

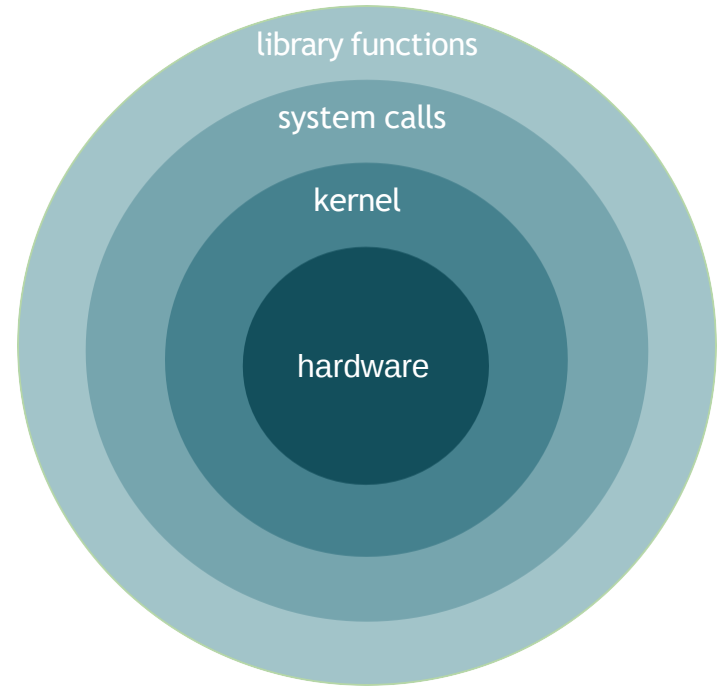
System calls

Outline

- Linux kernel
- System calls
- Emulator
- Implementing a new system call
- Notes

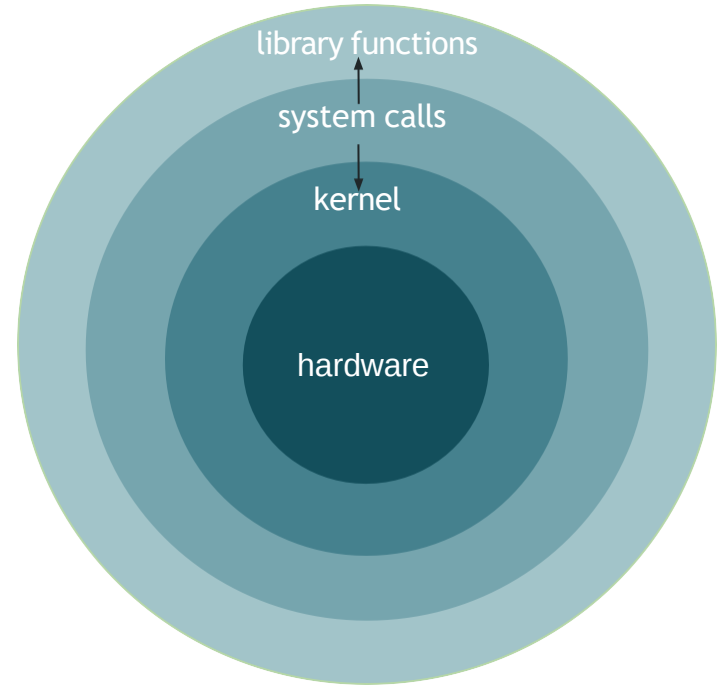
Kernel

- core of the operating system
- interface between **resources** and **user processes**
- what the kernel does:
 - memory management
 - process management
 - device drivers
 - **system calls**



System calls

- the interface between a process and the operating system
- how a program **requests a service from the kernel**



System calls - Examples

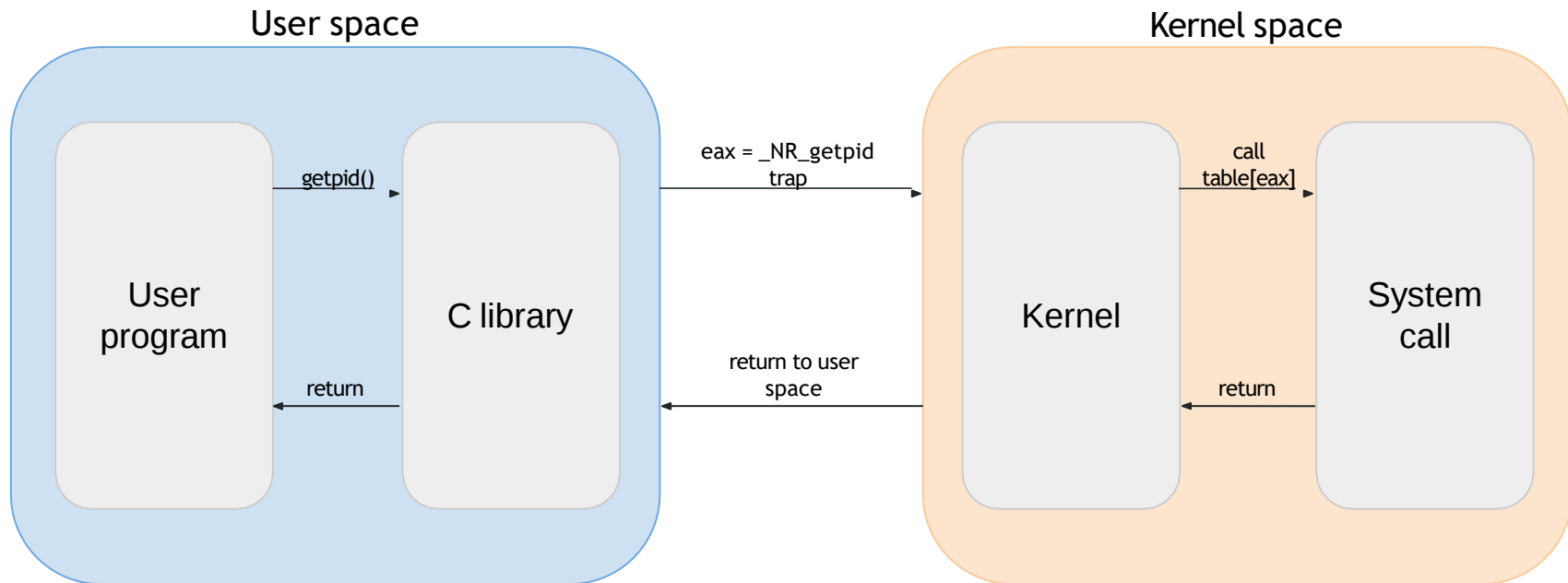
- Process control: fork, exit, wait
- File manipulation: open, read, close
- Device manipulation: ioctl, release
- Information: getpid, gettid
- Communication: pipe, socket
- Security: chmod, chown

System calls

- How do we make a system call in a C program?
`syscall(long number, ...);`
 - number: the number that corresponds to the system call
 - '...': the arguments we want to pass to the system call
- System call numbers can be found in `<sys/syscall.h>`

System calls

```
printf( "The process ID is %d\n", getpid() );
```



Assignment 3

- Introduce 2 new fields for each process:
 - **deadline:** the deadline of the process from now in seconds
 - **est_runtime:** the expected execution time, in milliseconds
- Implement 2 new system calls
 - `set_proc_info(...)`
 - `get_proc_info(...)`
- Support for a new scheduling policy
 - will be implemented in Assignment 4

Linux Kernel

Getting the source code:

```
$ cd spare
$ mkdir <username>
$ chmod 700 <username>
$ cd <username>
$ cp ~/hy345/qemu-linux/linux-2.6.38.1-patched.tar.bz2 .
$ tar -jxvf linux-2.6.38.1-patched.tar.bz2
```

Linux Kernel

Compiling it:

```
$ cd linux-2.6.38.1  
$ cp ~hy345/qemu-linux/.config .
```

<Implement additional functionality>

```
$export PATH="/home/misc/courses/hy345/gcc-4.9.2-standalone/bin/:$PATH"  
$export  
PATH="/home/misc/courses/hy345/gcc-4.9.2-standalone/libexec/gcc/x86_64-unknown  
-linux-gnu/4.9.2/:$PATH"  
  
$ make ARCH=i386 bzImage
```

Emulator

- Load the image and start the guest OS

```
$ cp ~hy345/qemu-linux/hy345-linux.img .  
$ qemu-system-i386 -hda hy345-linux.img -display curses
```

- Load the image and start the guest OS **with the new kernel**

```
$ qemu-system-i386 -hda hy345-linux.img -append "root=/dev/hda"  
-kernel linux-2.6.38.1/arch/x86/boot/bzImage -display curses
```

Implementing a new system call

1. Define a system call number
2. Define a function pointer
3. Define a function
4. Implement the system call

Example: Implement the system call **dummy_sys**. Takes one integer as an argument, prints something and returns the integer multiplied by 2.

1. Define a system call number

- Each system call has an invocation number
- Edit *linux-2.6.38.1/arch/x86/include/asm/unistd_32.h*
 - Define a new system call number
`#define __NR_dummy_sys 341`
 - Increase the number of system calls by 1
`#define NR_syscalls 342`



```
#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_dummy_sys 341
```



```
#ifdef __KERNEL__

#define NR_syscalls 342

#define __ARCH_WANT_IPC_PARSE_VERSION
#define __ARCH_WANT_OLD_READDIR
#define __ARCH_WANT_OLD_STAT
#define __ARCH_WANT_STAT64
```

2. Define a function pointer

- The kernel needs to have a function pointer pointing to the new system call
- Edit `linux-2.6.38.1/arch/x86/kernel/syscall_table_32.S`
 - Add an entry at the bottom of the list
`.long sys_dummy_sys`



```
.long sys_dummy_sys /* 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_dummy_sys
```

3. Define a function

- We need to define a function signature
- Edit *linux-2.6.38.1/include/asm-generic/syscalls.h*

- At the bottom of the file add

```
#ifndef sys_dummy_sys
    asmlinkage long sys_dummy_sys(int arg0);
#endif
```

```
#ifndef sys_rt_sigsuspend
asmlinkage long sys_rt_sigsuspend(sigset_t __user *u
#endif

#ifndef sys_rt_sigaction
asmlinkage long sys_rt_sigaction(int sig, const struct
                                struct sigaction __user *oa
#endif

#ifndef sys_dummy_sys|
asmlinkage long sys_dummy_sys(int arg0);
#endif

#endif /* __ASM_GENERIC_SYSCALLS_H */
```

4. Implement the system call

- Create *linux-2.6.38.1/kernel/dummy_sys.c*

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

asmlinkage long sys_dummy_sys(int
    arg0){ printk("Called
    dummy_sys\n"); return ((long)
    arg0*2);
}
```

- Add to *linux-2.6.38.1/kernel/Makefile*:
obj-y += dummy_sys.o

Simple demo application

```
#include <stdio.h>
#include <unistd.h>
#include <errno.h>

#define __NR_dummy_sys 341

int main(void){

    printf("Trap to kernel level\n");
    syscall(__NR_dummy_sys, 42); /* you should check return value for errors */
    printf("Back to user level\n");

    return 0;
}
```

Test the new system call

- Start the VM with the new kernel
 - `$ qemu-system-i386 -hda hy345-linux.img -append "root=/dev/hda" -kernel linux-2.6.38.1/arch/x86/boot/bzImage -curses`
- Write a test application
 - `$ vi test.c`
- Compile the test application
 - `$ gcc -o demo.out test.c`
- Run the test
 - `$ ./demo.out`
- Check the kernel log
 - `$ dmesg | tail`

Wrapper function

- Macro

```
#define dummy_sys(arg1) syscall(341, arg1)
```

- Wrapper function

```
long dummy_sys(int arg1){  
    return syscall(341, arg1);  
}
```

Notes

Process Data

- Edit *linux-2.6.38.1/include/linux/sched.h*
 - Find the task_struct structure
 - Introduce the 2 new fields
- Your system calls will interact with those fields

Faster Compiling Using ccache

Στο directory που δουλεύετε για την άσκηση φτιάχνετε ένα subdirectory:

```
mkdir -p /spare/csdXXXX/ccache
```

Κάνετε export το path:

```
export PATH="/home/misc/courses/hy345/ccache-4.7.4-linux-x86_64/:$PATH"
```

Πλέον για να κάνετε build τον kernel χρησιμοποιείτε την εντολή:

```
CCACHE_DIR=/spare/csdXXXX/ccache/ make CC="ccache gcc" ARCH=i386  
bzImage
```

Printk()

- Prints messages to the kernel log
- Every time one of your system calls is executed, you should print a message
 - Your name, A.M. and the name of the system call
- You can view these messages from the user level
 - `dmesg`
 - `cat /var/log/messages`
- Very useful for debugging

Hints

Useful kernel functions:

- `for_each_process()`
- `get_current()`
- `access_ok()`
- `copy_to_user()`
- `copy_from_user()`

E-LEARN

What to submit:

- bzImage
- Modified or created source files
- Test programs and headers in Guest OS
- README



Good luck!

