System Calls

(Φροντιστήριο για τη 3η σειρά)

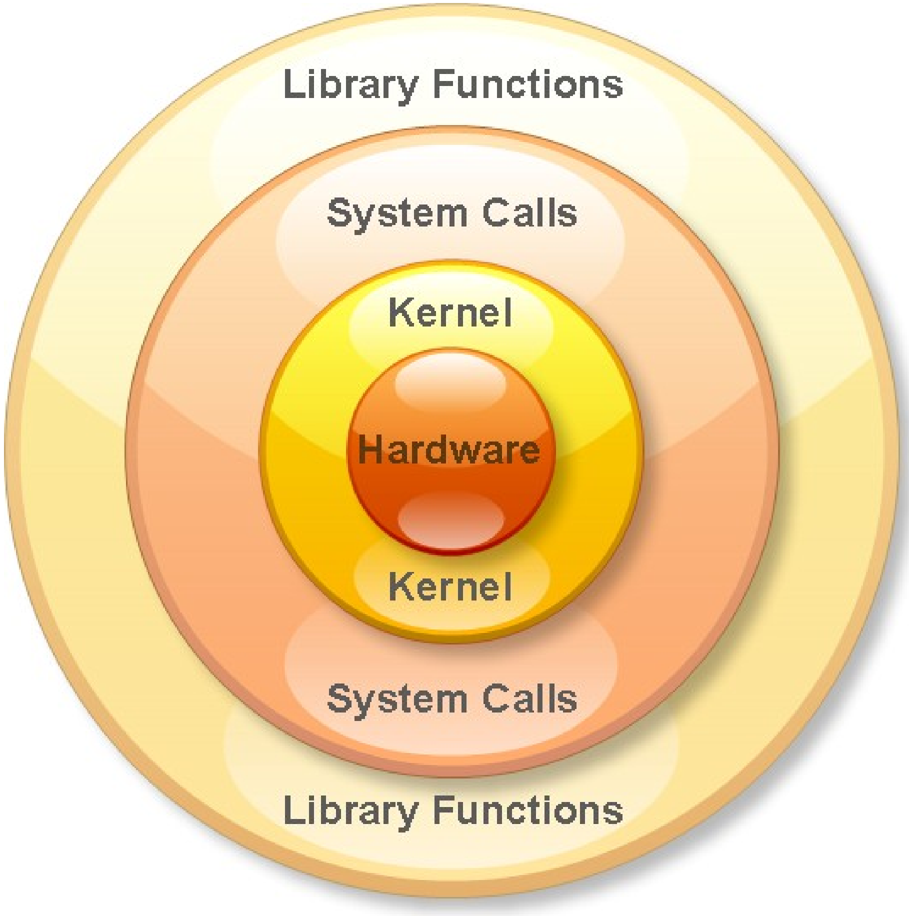
cs-345

Vangelis Ladakis
ladakis@csd.uoc.gr

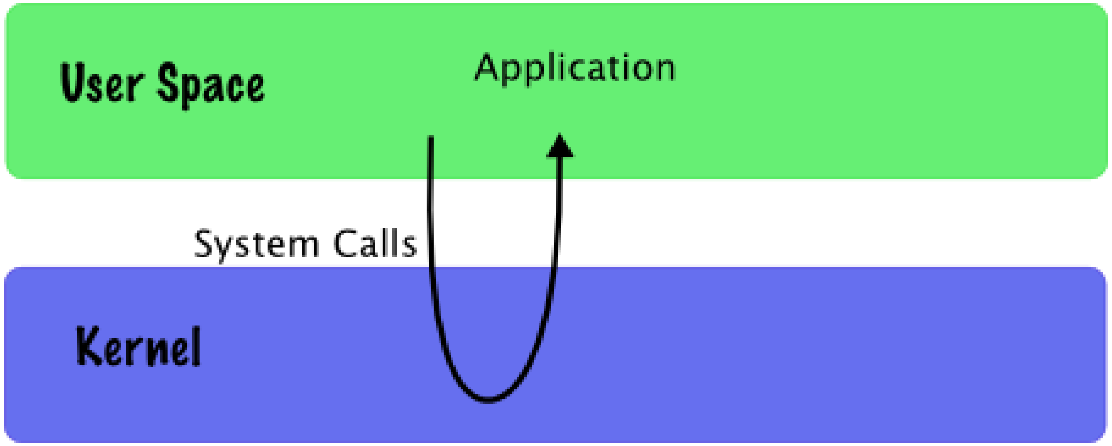
Τμήμα Επιστήμης Υπολογιστών
Πανεπιστήμιο Κρήτης

What is a System Call?

“The system call is the fundamental interface between an application and the Linux kernel.”



Why we need System Calls?



- A system calls provide an essential interface between a process and the operating system.
- A system call is how a program requests a service from an operating system's kernel.

What system calls can do?

- Process Control
  - exec, kill, wait...
- File management
  - create, delete, open, load...
- Device Management
  - request, release...
- Information Maintenance
  - get time, set time...
- Communication
  - Send/receive messages
  - create/destroy communication (sockets)...

Sounds Familiar?!

How we use them?

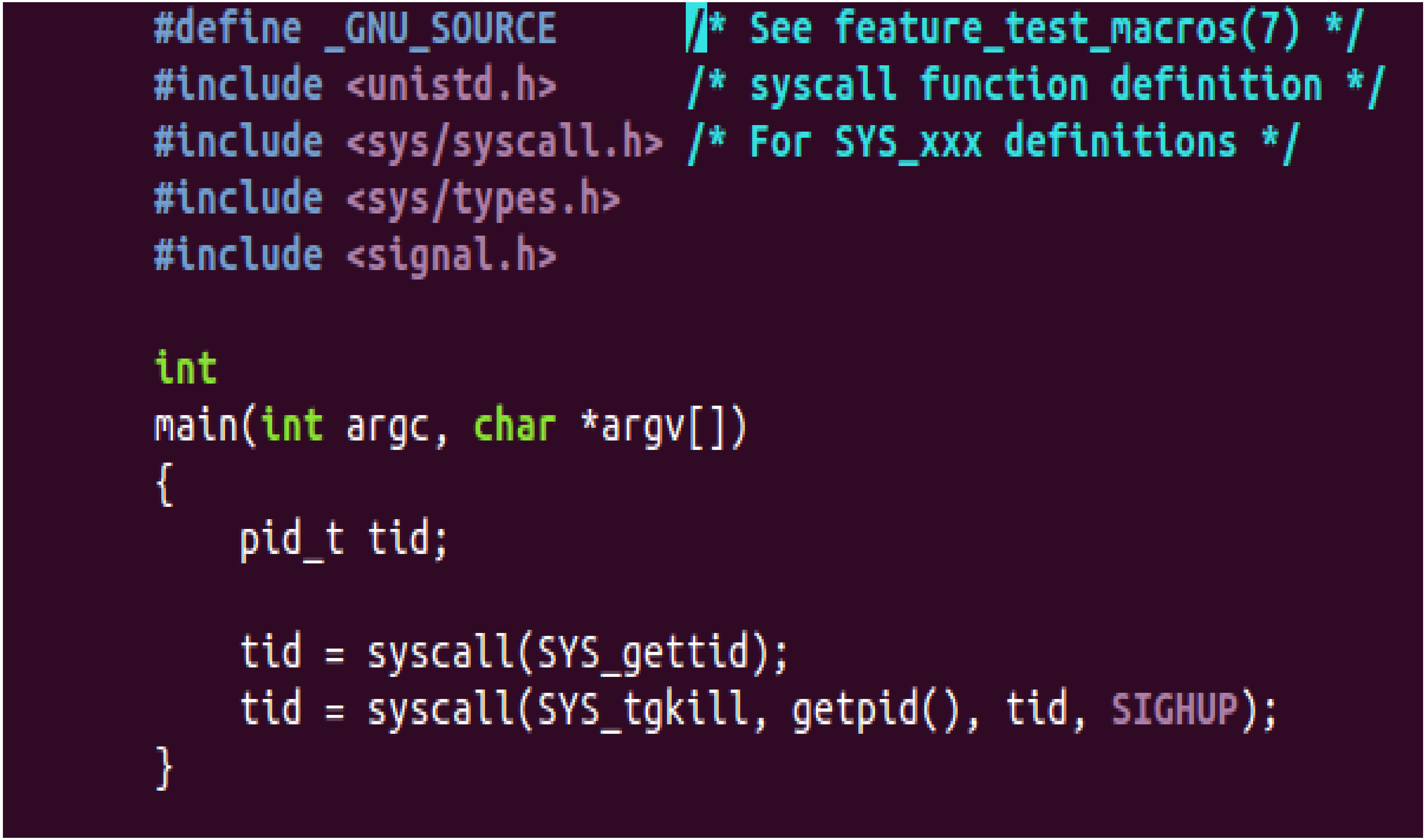
int syscall(int number, …)

  - “man syscall” for details

- It's a small library which invokes the system call that corresponds to the “number”.
  - The symbol of “...” corresponds to the rest of the arguments (just like printf).

Let's see an example...

Call a system call
Example



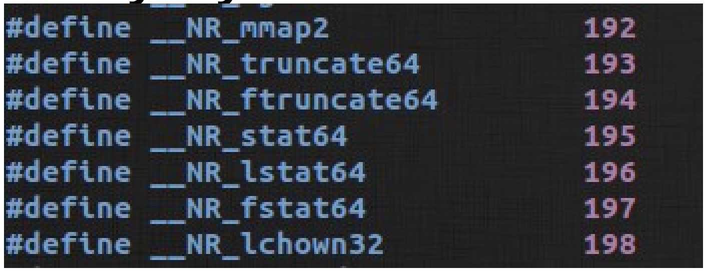
```c
#define _GNU_SOURCE      /* See feature_test_macros(7) */
#include <unistd.h>      /* syscall function definition */
#include <sys/syscall.h> /* For SYS_xxx definitions */
#include <sys/types.h>
#include <signal.h>

int
main(int argc, char *argv[])
{
    pid_t tid;

    tid = syscall(SYS_gettid);
    tid = syscall(SYS_tgkill, getpid(), tid, SIGHUP);
}
```

How can we write a new system call?

1. Define system call number
2. Define function pointer
3. Define function
4. Implementation

Define System Call Number

- Every system call has an invocation number



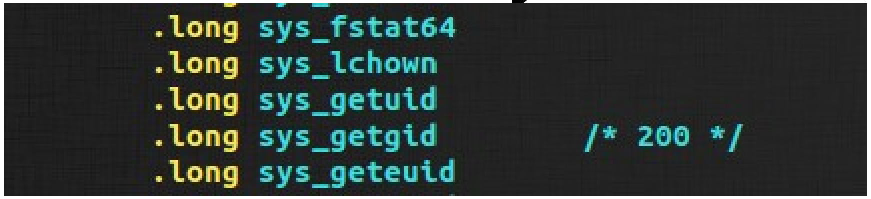
```
#define __NR_mmap2              192
#define __NR_truncate64         193
#define __NR_ftruncate64        194
#define __NR_stat64             195
#define __NR_lstat64            196
#define __NR_fstat64            197
#define __NR_lchown32           198
```

- Edit: *linux-2.6.38.1/arch/x86/include/asm/unistd_32.h*
  - Define at the bottom of the list your own system call number
  - Update the number of syscalls

#define __NR_dummy_sys 341

Define function Pointer

- Kernel needs to have a function pointer pointing to the new system call



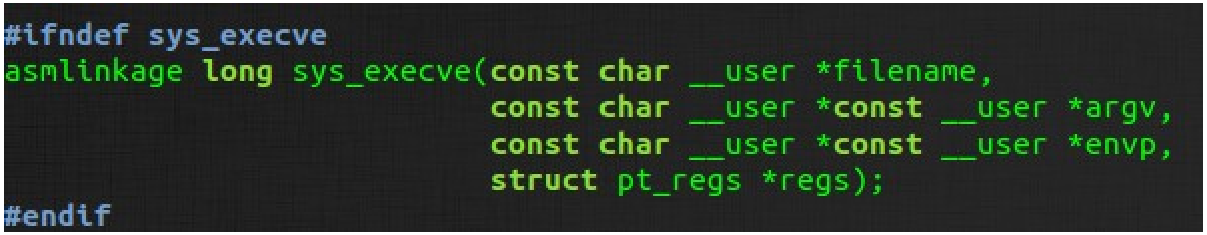
```
        .long sys_fstat64
        .long sys_lchown
        .long sys_getuid
        .long sys_getgid         /* 200 */
        .long sys_geteuid
```

- Edit: *arch/x86/kernel/syscall_table_32.S*
  - Define at the bottom of the list the function pointer

.long sys_dummy_sys / 341 */*

Define function

- At this point we have to define the function signature at the syscalls.h



```
#ifndef sys_execve
asmlinkage long sys_execve(const char __user *filename,
                           const char __user *const __user *argv,
                           const char __user *const __user *envp,
                           struct pt_regs *regs);
#endif
```

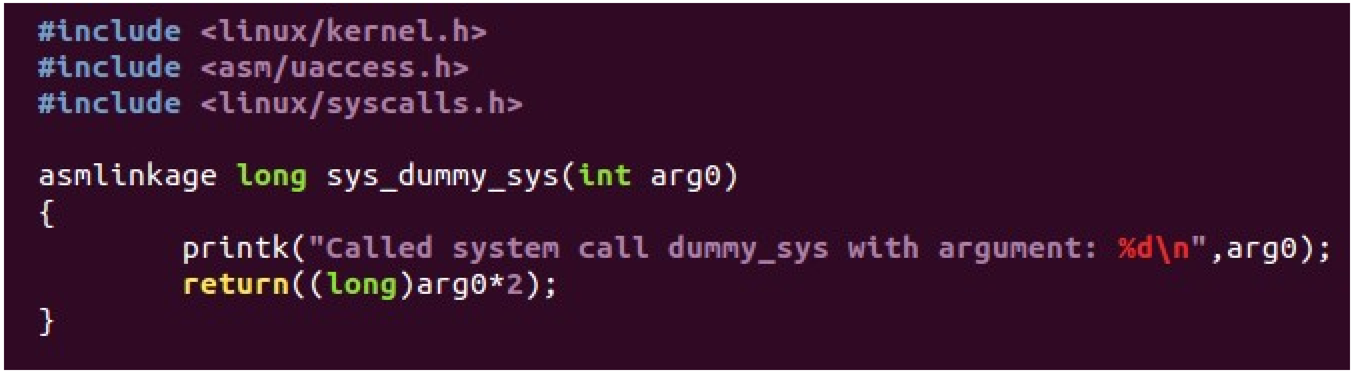
- Edit: include/asm-generic/syscalls.h

asmlinkage long sys_dummy_sys(int arg0);

Implement syscall 1/2

- Add the source code inside the kernel
  - Add new file at: kernel/dummy_sys.c
  - Edit the Makefile

The new system call may look as follow:



```
#include <linux/kernel.h>
#include <asm/uaccess.h>
#include <linux/syscalls.h>

asmlinkage long sys_dummy_sys(int arg0)
{
        printk("Called system call dummy_sys with argument: %d\n",arg0);
        return((long)arg0*2);
}
```

Implement syscall 2/2

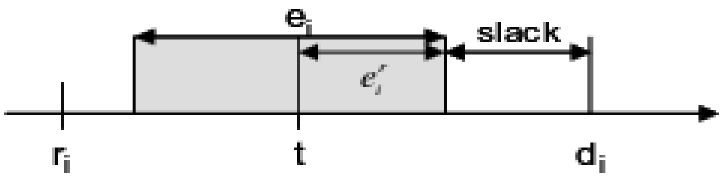
- Notice that now you are programming in kernel space
  - No segmentation faults will occur, but Black screens of Death
  - ***Printf, malloc*** etc are for user-space instead you have to use ***printk, kmalloc*** etc
    - Messages of ***printk*** you may see them by typing ***dmesg*** to command prompt or ***cat /var/log/messages***
  - Debugging may be a pain

Slack Time

- Every process will have
  - Deadline
  - Remaining time
- Slack comes from:

  deadline – remaining time – current time

  - It's the remaining spare time



Assignment 3

- You will have to implement two system calls that you will need for the next assignment

/ set to the process with the given pid the remaining time and deadline time */*

set_lst_parameters(*int pid, int remaining_computation_time, time_t deadline****);***

/ fill the struct lst_parms with the remaining time and the deadline time for the process with the given pid */*

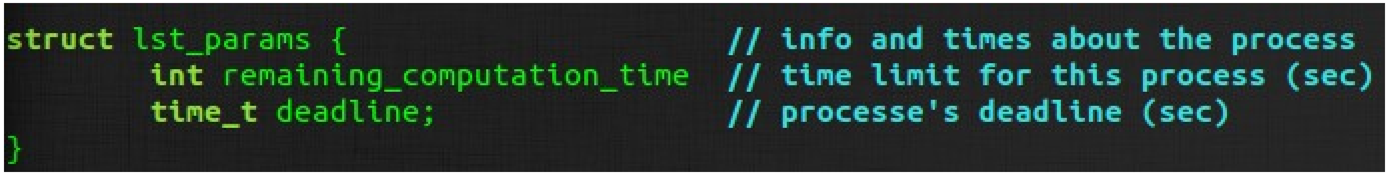
get_lst_parameters(*int pid, struct lst_params *lst_arguments****);***

Assignment in detail:

http://www.csd.uoc.gr/~hy345/assignments/2014/assign3/assignment3.html

- You will have to add some information to the ***task_struct***
  - Stores information for a process.
  - Defined in ***include/linux/sched.h***

For every process running you will have to add:



```
struct lst_params {                         // info and times about the process
        int remaining_computation_time  // time limit for this process (sec)
        time_t deadline;                    // processe's deadline (sec)
}
```

- The system calls will eventually set and get information for a process
- You will need them for the slack scheduler (next assignment)

Try to keep your code clean, you are messing with the kernel

Be careful with the memory space

- Arguments passed by value

- When you have memory references you have to pass the data from user-space to kernel-space
  - ***int access_ok(****type, address, size****)***
  - ***Unsigned long copy_from_user(****void* to, const void_user* from, unsigned long n****)***
  - ***Unsigned long copy_to_user(****void_user* to, const void* from, unsigned long n****)***

Functions are defined in: ***/linux/uaccess.h & /asm-generic/uaccess.h***

Qemu & Linux OS

- Qemu is pre-installed on CSD machines
  - Files are big!!! Work on spare directory. Details on the site
- Download from the course site the:
  - Linux source code
  - .config file for building the kernel
  - linux image

Source code:

http://www.csd.uoc.gr/~hy345/qemu-linux/linux-2.6.38.1.tar.bz2

.config:

http://www.csd.uoc.gr/~hy345/qemu-linux/.config

Linux Image:

http://www.csd.uoc.gr/~hy345/qemu-linux/hy345-linux.img

Load Image to Qemu

- In order to load Image
  - *qemu -hda hy345-linux.img*
- In order to compile the source code and load the new image

1) Download and place inside *linux-2.6.38.1* the config file .config

2) Edit .config, find CONFIG_LOCALVERSION="-hy345", and append to the kernel's version name your username and a revision number

3) make ARCH=i386 bzImage

4) qemu -hda hy345-linux.img -append "root=/dev/hda" -kernel linux-2.6.38.1/arch/x86/boot/bzImage

Useful Links

- Assignment 3: http://www.csd.uoc.gr/~hy345/assignments/2014/assign3/assignment3.html
- Qemu and Linux:

  http://www.csd.uoc.gr/~hy345/assignments/quemu_notes.html
- Adding a System call:

  http://www.csd.uoc.gr/~hy345/assignments/system_calls_notes.html
- Adding a system call:

  http://www.cs.rochester.edu/~sandhya/csc256/assignments/adding-a-system-call.html
- Adding a System call video:

  https://www.youtube.com/watch?v=5rr_VoQCOgE