OS & Syscall Basics 2: System Calls

1. Operating Systems
2. **System Calls**
   - **system calls API**
   - **syscall wrapper functions**
   - **documentation**
3. Filesystems and Memory
4. Processes, IPC, Signals

System Calls API

Since the operating system controls access to all of a computer's resources, users and application programs must make requests to the OS to use of any of these resources.

The *interface* (API) between programs and the OS consists of a set of functions known as **system calls**.

Each system call can also be considered as an **entry point** into the functionality of the OS kernel.

The system calls of an OS define the set of services that the OS can provide to programs.

Linux has slightly over 300 defined system calls.

System Calls API (contd.)

OS system calls typically provide the following services:

- **process/thread** creation and control
- memory allocation/deallocation
- file creation, reading, writing, etc
- directory creation and manipulation
- **interprocess communication**, including network communication
- process synchronization and file locking
- **signal** handling (software interrupts)
- device I/O

Syscall Wrapper Functions

Exactly how a system call is invoked will depend on the processor architecture, as it involves switching from user to kernel mode.

Often it requires that the **system call code** (number) be placed in a register, argument addresses be set on the (user) stack, and a TRAP or similar **interrupt** type instruction be executed, after which the kernel code **dispatches** to the correct code based on the syscall code.

Luckily, the ugly elements of this process are hidden from user programs because an interface to the syscalls using **C wrapper functions** is part of the C standard library (libc).

In fact, this is how the calls are defined in the POSIX/SUS standards (i.e., as C functions).

Syscall Wrapper Functions (contd.)

For example, the `open` system call *wrapper function* is defined as:

```
int open(const char *pathname, int flags)
int open(const char *pathname, int flags, mode_t mode)
```

A C program will call one of these functions, and the *library code* that implements the function will do the required setup and use the correct architecture-specific method to initiate the appropriate kernel system call.

The Linux kernel `open` system call is syscall 5, named `sys_open` in the kernel code.

Syscall Wrapper Functions (contd.)

It is defined in the file `fs/open.c`:

```
asmlinkage long sys_open(const char __user *filename,
                         int flags, int mode)
{
        long ret;

        if (force_o_largefile())
                flags |= O_LARGEFILE;

        ret = do_sys_open(AT_FDCWD, filename, flags, mode);
        /* avoid REGPARM breakage on x86: */
        prevent_tail_call(ret);
        return ret;
}
```

Syscall Wrapper Functions (contd.)

The wrapper "system calls" in libc do not always have a one-to-one mapping to kernel syscalls.

There may be no equivalent kernel syscall, or multiple wrapper syscalls may invoke the same kernel syscall.

For example, there is no kernel syscall that directly implements the SUS calls `opendir()` and `seekdir()`.

Since *directories* are simply *specially formatted files*, most of the directory syscall functions can be implemented using file syscalls like `open` and `lseek`.

On the other hand, there is a kernel syscall for wrapper `readdir()`.

Syscall Wrapper Functions (contd.)

Another example is the wrapper syscalls `wait()` and `waitpid()`, which both invoke the Linux kernel syscall `sys_waitpid`.

While `wait()` and `waitpid()` are both required by SUS, Linux is free to implement them using kernel syscalls in any way it chooses, and `wait()` is basically a simplified version of `waitpid()`.

Perhaps the most extreme example of there not being direct mappings from wrapper calls to kernel syscalls is the kernel syscall `socketcall`, which implements all of the socket-related wrapper calls: `socket()`, `bind()`, `accept()`, `connect()`, etc.

Documentation

Further documentation on Linx system calls can be found by:

- `man 2 syscalls`
- `man 2 syscall` (call to invoke a system call by number)
- `more /usr/src/linux/arch/x86/kernel/syscall_table_32.S` (or similar)
- `nm -D /lib/libc-2.7.so` (or similar)