System Calls in KOS

CPSC 457

Jalal Kawash

General Advice

- Kernel code is overwhelming
- You do not need to understand everything
- Understand enough in order to make your code work
- Bit by bit, you will accumulate a more general understanding
- If it looks to complex, ignore details & focus on the big picture

What is KOS?

- KOS is a small OS written mainly in C++ (C & x86 assembly)
- It is educational
- It is relatively small
- It has a RAM-based file system
  - No disk
- Runs within qemu (emulator)

KOS directory structure

- KOS
  - cfg
  - ptaches
  - src
    - devices
    - extern
    - gdb
    - generic
    - include
    - kernel
    - machine
    - main
    - runtime
    - scripts
    - stage
    - ufiber
    - ulib
    - unit
    - user
    - world

main

- UserMain.h
  - Main user method that runs when KOS runs
  - Contains some tests (for debugging) and InitProcess()
  - InitProcess() calls your programs

```
static void UserMain() { // has some system tests; ignore
    InitProcess()
}
```

main

- Creating user programs
  - Created in *user*
  - *Mak*ing the file creates an executable in *user/exec*
  - *Example: systest* (executable in exec)
- Calling these programs
  - From InitProcess.cc (in *KOS*/*main)*

```
int InitProcess() {
    Process* p0 = knew<Process>();
    p0->exec("systest");
    return 0;
  }
```

kernel

- Kernel code
  - syscalls.cc
  - process.cc
  - memorymanager.cc
  - framemanager.cc
  - And more

machine

- Emulation of x86 processors
  - Processor.cc
  - CPU.cc
  - Paging.cc
  - And more

runtime

- Runtime environment
  - Thread.cc
  - Stack.cc
  - Scheduler.cc
  - And more

Adding a System Call

Assignment 1

Adding a system call: 1. the stub

- Stubs:
  - This is the wrapper or interface

```
extern "C" pid_t getpid() {
  return syscallStub(SyscallNum::getpid);
}
```

  - getpid(): wrapper name. This is the system call name as it appears to the user program
  - **getpid:** is the actual name of the implementation in *syscalls.cc*
  - They need nmot be the same name:

```
extern "C" pid_t getpid() {
  return syscallStub(SyscallNum::getpidImpl);
}
```

Adding a system call: 1. the stub

- Add a stub in *ulib/libKOS.cc*

```
extern "C" bool isEven(long n) {
  return syscallStub(SyscallNum::isEvenImpl);
}
```

Adding a system call: 2. declaration

- Declare it in *include/syscalls.h*

```
extern "C" bool isEven( long n ); // just an example
```

- System calls have to be given numbers for system identification
  - Also in *syscalls.h*

```
enum : mword {
  _exit = 0,
  open, //1
  close, // 2
  isEvenImpl, //3
…
};
```

Adding a system call: 3. implementation

- Write the implementation in *kernel/syscalls.cc*

```
extern "C" bool syscall_isEven( long n ) {
	return !(n % 2);
}
```

Adding a system call: 4. Testing

- Write a user program in *user*
  - even = isEven(3);
- *Make*
  - Exec goes to *user/exec*
- InitProcess() in main will call your user program:
  - Process p1 = knew<Process>();
  - P1->exec(“IsEven”);