System Calls

Invoking Kernel Operations

Dr Andrew Scott

a.scott@lancaster.ac.uk

1

System Calls

- User programs can't directly
  - Communicate with (most) hardware
  - Read/ write kernel data/ data structures

  …unrestricted access would allow any program to read/ change anything
- System Calls
  - Offer standard interface to kernel/ OS functions
  - Provide access control mechanism
    - Single point of entry that can check all parameters

2

Unix man pages

1. User Commands
2. System Calls
3. C Library Functions
4. Devices and Special Files
5. File Formats and Conventions
6. Games etc.
7. Miscellaneous
8. System Administration tools and Daemons

3

Simple I/O Example

```
int
main ( ) {

   char * msg = "Hello World\n";

   int    fd  = 1;     // stdout: normal output stream

   int    len = (int) strlen(msg);

   int    retval;

   retval = write(fd, msg, len);  // Needs kernel privileges!!!

   printf ("write(fd, msg, len) reported %d characters written\n",
           retval);
}
```

4

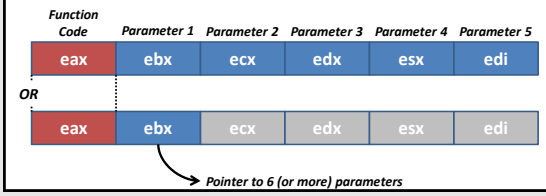
Making a System Call on x86

- Fill registers (EAX, EBX, …) with
  - Function code (for read, write, …)
  - Parameters

128 decimal

46 decimal

- Execute software interrupt: INT 80 (Unix) or INT 2E (WinNT)



5

Making Unix System Calls

```
#       Build with: as syscall.s -o syscall.o
#                   ld syscall.o -o syscall

.section      .rodata
MSG:    .string       "Hello World\n"

.set    LEN, .-MSG            # calculate length of message string

.text
.global       _start
_start:
        mov     $4,     %eax    # write: System call #4
        mov     $1,     %ebx    # fd = 1 : stdout
        mov     $MSG,   %ecx    # msg : char * msg
        mov     $LEN,   %edx    # number of characters to output
        int     $0x80           # syscall: int write(fd, msg, len)

        mov     $1,     %eax    # exit: System call #1
        mov     $0,     %ebx    # code = 0 (success)
        int     $0x80           # syscall: exit(0)
```

6

Direct System Call from C

```
int
main ( ) {
   char * msg = "Hello World\n";

   int    fd  = 1;     // stdout
   int    len = (int) strlen(msg);
   int    retval;

   asm volatile (
        "mov    $4,     %%eax; # write: System call #4"
        "int    $0x80;         # syscall: int write(fd, msg, len)"
        "movl   %%eax,  %0     # get return value from function"

      : "=rm" (retval)     // %0 refers to return variable retval
      : "b" (fd), "c" (msg), "d" (len)  // Place in EBX, ECX, EDX
      : "%eax"                          // Clobbered/ overwritten
   );

   printf ("write(fd, msg, len) reported %d characters written\n",
           retval);
}
```

Remember inline assembler allows us to directly specify the x86 registers variables should be passed in, e.g., to pass fd in reg. EBX we use "b" (fd) -- this really simplifies the code

7

Kernel System Call Entry Point

- Software Interrupts appear as any other interrupt
  - Normal ISR entry point
  - Simply check for Unix (0x80) or NT (0x2E) 'call' convention
    - Note difference, one kernel could support both Unix and Windows

```
void
interrupt_handler(
      struct cpu_registers regs, uint32_t irq, uint32_t code,
      uint32_t eip, uint32_t cs, uint32_t flags,
      uint32_t esp, uint32_t ss ) {

   switch (irq) {

   case 0x80:              // INT 0x80   (80 hex = 128 decimal)
      systemcall ( &regs );
      break;
```

8