Stage 5
System Calls

Objectives

- APIs, POSIX, and the C Library
- Syscalls and Implementation
- System Call kernel Implementation

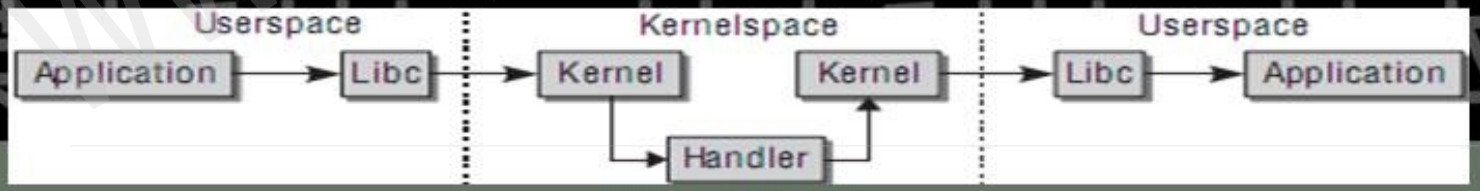
APIs, POSIX, and the C Library

- Applications are programmed against an Application Programming Interface (API), not directly to system calls. This is important, because no direct correlation is needed between the interfaces that applications make use of and the actual interface provided by the kernel. An API defines a set of programming interfaces used by applications.
- One of the more common application programming interfaces in the Unix world is based on the POSIX standard. Technically, POSIX comprises a series of standards from the IEEE that aim to provide a portable operating system standard roughly based on Unix. Linux strives to be POSIX and SUSv3 complaint where applicable.
- The system call interface in Linux, as with most Unix systems, is provided in part by the C library. The C library implements the main API on Unix systems, including the standard C library and the system call interface. The C library is used by all C programs and, because of C's nature, is easily wrapped by other programming languages for use in their programs.
  - relationship between applications, the C library, and the kernel with a call to printf().

- Lab
- #starce ./lab10

Syscalls

- Process Management
- Time Operations
- Signal Handling
- Scheduling
- Filesystem
- Memory Management
- Interprocess Communication and Network Functions
- System Information and Settings
- System Security and Capabilities
- arch/x86/kernel/syscall_table_32.S
- include/asm-generic/unistd.h
- include/asm-generic/errno-base.h
- Chronological sequence of a system call



Syscalls

- Parameter Passing
  - On IA-32 systems, the assembly language instruction int $0x80 raises software interrupt 128.This is a call gate to which a specific function is assigned to continue system call processing. The system call number is passed in register eax, while parameters are passed in registers ebx, ecx,edx, esi,and edi .(many ？stack)
  - 当一个系统调用所需的参数个数大于 5 时,执行 int 0x80 指令时仍需将系统调用功能号保存在寄存器 eax 中,所不同的只是全部参数应该依次放在一块连续的内存区域里,同时在寄存器ebx 中保存指向该内存区域的指针。系统调用完成之后,返回值仍将保存在寄存器 eax 中。由于只是需要一块连续的内存区域来保存系统调用的参数,因此完全可以像普通的函数调用一样使用栈(stack)来传递系统调用所需的参数。但要注意一点,Linux 采用的是 C 语言的调用模式,这就意味着所有参数必须以相反的顺序进栈,即最后一个参数先入栈,而第一个参数则最后入栈。如果采用栈来传递系统调用所需的参数,在执行 int 0x80 指令时还应该将栈指针的当前值复制到寄存器 ebx 中。

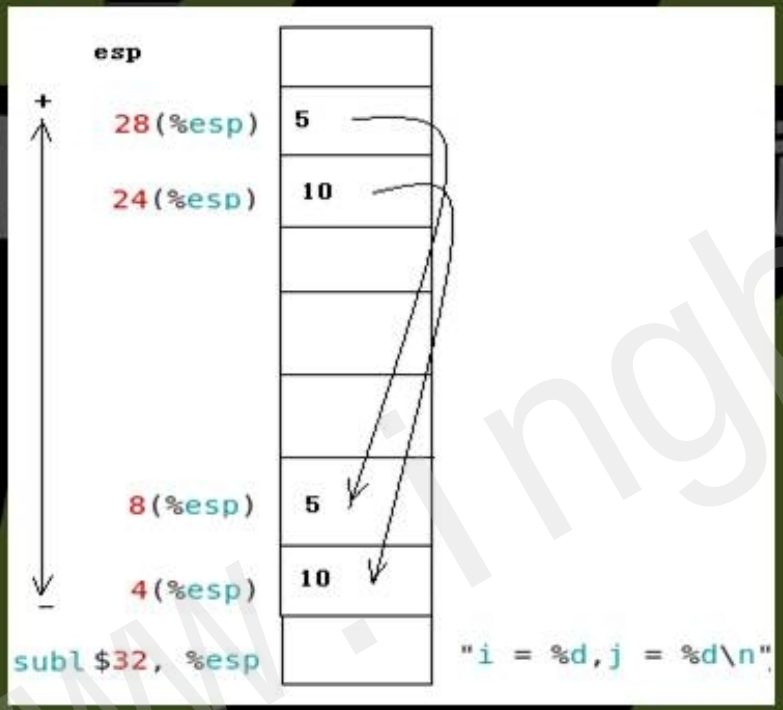
Syscalls

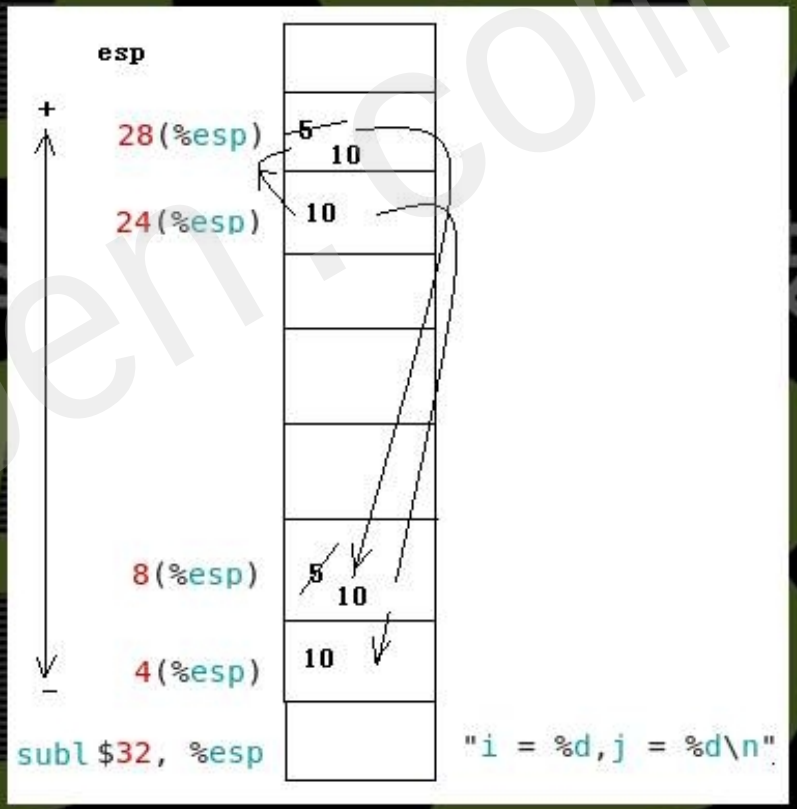
- Alpha processors provide a privileged system mode (PAL, privileged architecture level)in which various system kernel routines can be stored. v0 is used to pass the system call number, and the five possible arguments are held in a0 to a4.
- PowerPC processors feature an elegant assembly language instruction called sc (system call). Register r3 holds the system call number,while parameters are held in registers r4 to r8 inclusive.
- The AMD64 architecture also has its own assembly language instruction with the revealing name of syscall to implement system calls. The system call number is held in the raw register, parameters in rdi, rsi, rdx, r10, r8,and r9.

- Lab
  - lab11
    - # as lab11.s -o lab11.o
    - #ld lab11.o -o lab11
    - # ./lab11

Syscalls

- Lab
  - Lab12
    - #gcc -o lab12 lab12.c
    - gcc -S lab12.c





- Access to Userspace
  - To make sure that the kernel complies with this convention, userspace pointers are labeled with the __user attribute to support automated checking by C check tools.

System Call kernel Implementation

- The first step in implementing a system call is defining its purpose. What will it do? The syscall should have exactly one purpose.
- Multiplexing syscalls (a single system call that does wildly different things depending on a flag argument, Look at ioctl() as an example) is discouraged in Linux.
- What are the new system call's arguments, return value, and error codes? The system call should have a clean and simple interface with the smallest number of arguments possible.
- When you write a system call, it is important to realize the need for portability and robustness, not just today but in the future.
- Lab
  - Lab15
    - #mknod /dev/mycdrv c 700 2
    - #insmod ./lab15.ko
    - #./test
    - #dmesg

System Call kernel Implementation

- Lab
  - Lab13
    - #gcc -o mycall mycall.c
  - Note:
    - 1)/root/workspace/kernel/linux-2.6.38.2/mycall
      mycall.c and Makefile
    - 2) vi /root/workspace/kernel/linux-2.6.38.2/Makefile
      core-y += kernel/ mm/ fs/ ipc/ security/ crypto/ block/ mycall/
    - 3)vim /root/workspace/kernel/linux-2.6.38.2/arch/x86/kernel/syscall_table_32.S
      - .long sys_mycall /* 341 for ingben test */
- Lab
  - Lab14
    - #insmod ./lab14.ko
    - #gcc –o test test.c
  - Note
    - 1) vim /root/workspace/kernel/linux-2.6.38.2/arch/x86/include/asm/unistd_32.h
    - 2) /root/workspace/kernel/linux-2.6.38.2/arch/x86/kernel/syscall_table_32.S
    - 3) /root/workspace/kernel/linux-2.6.38.2/kernel/sys.c
    - 4)#make && make modules && make modules_install && make install && depmod 2.6.38.2