System Calls, Interrupt

ECE595, Jan 11

Y. Charlie Hu

Roadmap

- System calls
- Interrupt

[lec1] What is an OS?

- Resource manager
  - Manage shared resources (CPU, mem, I/O, networking)
- Extended (abstract) machine

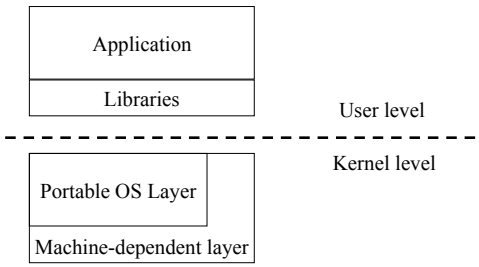
Dual-Mode Operation (ECE437)

- OS manages shared resources
- OS protects programs from other programs

➔ OS needs to be “privileged”

- Dual-mode operation of hardware
  - Kernel mode – can run *privileged* instructions
  - User mode – can only run *non-privileged* instructions

Typical Unix OS Structure



5

Dual-Mode Operation

- OS manages shared resources
- OS protects programs from other programs

➔ OS needs to be privileged

- If OS manages shared resources, how does a user program request for accessing shared resources (e.g. hard drive)?

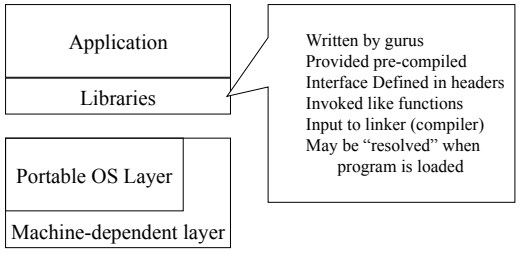
6

System calls

- Interface between a *process* and the operating system kernel
  - Kernel manages shared resources & exports interface
  - Process requests for access to shared resources
- Generally available as assembly-language instructions
- Directly invoked in many languages (C, C++, Perl)
  - Who is helping out here?

7

Typical Unix OS Structure



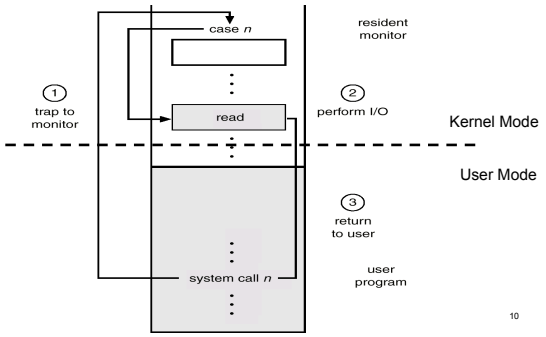
8

Example

- Given the I/O instructions are privileged, how does the user program perform I/O?
  - open()
  - read()
  - write()
  - close()

9

Use of a system call to perform I/O



10

System calls

- Categories
  - Process management
  - Memory management
  - File management
  - Device management
  - Networking

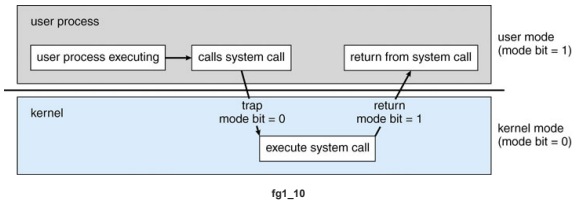
11

System calls in Linux (man syscalls)

- SYSCALLS(2) Linux Programmerâs Manual SYSCALLS(2)
- NAME
- none - list of all system calls
- SYNOPSIS
- Linux 2.4 system calls.
- DESCRIPTION
- The system call is the fundamental interface between an application and
- the Linux kernel. As of Linux 2.4.17, there are 1100 system calls
- listed in /usr/src/linux/include/asm-*/unistd.h. This man page lists
- those that are common to most platforms (providing hyperlinks if you
- read this with a browser).
- _llseek(2), _newselect(2), _sysctl(2), accept(2), access(2), acct(2),
- adjtimex(2), afs_syscall, alarm(2), bdflush(2), bind(2), break, brk(2),
- cacheflush(2), capget(2), capset(2), chdir(2), chmod(2), chown(2),
- chown32, chroot(2), clone(2), close(2), connect(2), creat(2), cre-
- ate_module(2), delete_module(2), dup(2), dup2(2), execve(2), exit(2),
- fchdir(2), fchmod(2), fchown(2), fchown32, fcntl(2), fcntl64, fdata-
- .....

12

Transition from user to kernel mode



fg1_10

Project 1 (week 2)

- DLXOS setup
- Trap (drop into the kernel) implementation
  - Used in later projects
  - Adding a new system call (getpid)

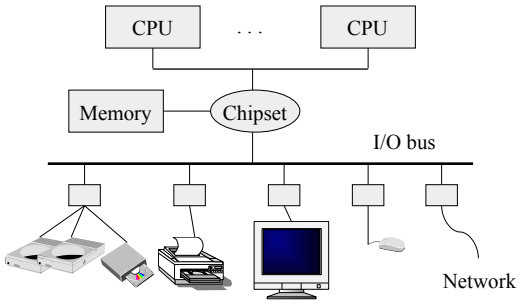
14

Roadmap

- System calls
- Interrupt

15

A Typical Computer from a Hardware Point of View



16

Concurrency & Unexpected Events

- How do human handle unexpected events?
  - Assume
    - mail delivered to my mailbox continously
    - I have a secretary
  - Do I have mail now? (need to be processed quickly)
- Poll vs. interrupt
- Which one is more efficient?
  - Assume 1 interrupt more costly than 1 poll
  - If I have one mail per day?
  - If I have lots of mail per delivery?

17

Interrupt (ECE437)

- A mechanism for
  - coordination between concurrently operating units of a computer system (e.g. CPU and I/O devices)
  - for responding to specific conditions within a computer
- Results in transfer of flow of control (to interrupt handler in the OS), forced by hardware

18

Two types of Interrupts

- *Hardware interrupts*
  - timers
  - I/O devices: keyboards, NICs, etc.
- *Software interrupts* (aka. *trap, exception*)
  - an error (floating point exception)
  - a *system call* requesting OS for special services (e.g. I/O)

19

Handling interrupts (ECE437)

- Incoming interrupts are disabled (at this and lower priority levels) while the interrupt is being processed to prevent a lost interrupt
- Interrupt architecture must save the address of the interrupted instruction
- Interrupt transfers control to the interrupt service routine
  - generally, through the interrupt vector, which contains the addresses of all the service routines
- If interrupt routine modifies process state (register values)
  - save the current state of the CPU (registers and the program counter) on the system stack
  - restore the state before returning
- Interrupts are re-enabled affter servicing current interrupt
- Resume the interrupted instruction

20

X86 Interrupt and Exceptions (1)

Vector #	Mnemonic	Description	Type
0	#DE	Divide error (by zero)	Fault
1	#DB	Debug	Fault/trap
2		Non-Maskable interrupt	Interrupt
3	#BP	Breakpoint	Trap
4	#OF	Overflow	Trap
5	#BR	BOUND range exceeded	Trap
6	#UD	Invalid opcode	Fault
7	#NM	Device not available	Fault
8	#DF	Double fault	Abort
9		Coprocessor segment overrun	Fault
10	#TS	Invalid TSS	

X86 Interrupt and Exceptions (2)

Vector #	Mnemonic	Description	Type
11	#NP	Segment not present	Fault
12	#SS	Stack-segment fault	Fault
13	#GP	General protection	Fault
14	#PF	Page fault	Fault
15		Reserved	Fault
16	#MF	Floating-point error (math fault)	Fault
17	#AC	Alignment check	Fault
18	#MC	Machine check	Abort
19-31		Reserved	
32-255		User defined	Interrupt

Vector 128 is for system calls

"Modern OSes are interrupt driven"

- "An OS is a giant interrupt handler!" (Def 3)
  - Timer interrupt → Context switches in multiprogramming
  - (unexpected) I/O interrupts
  - System calls (traps) to switch from user to supervisor mode
- At the lowest level an OS is just a bunch of interrupt service routings
  - Each routine simply returns to whatever was executing before it was interrupted
    - A user process
    - An OS process
    - Another interrupt routine
  - Else infinite wait loop

Reading Assignment

- We will dive in process management next week
- We will go through Project 1 on Friday
- Reading assignment: OSC Chapters 1-2