CIS 505: Software Systems
OS Overview -- System Calls and Signals

Insup Lee
Department of Computer and Information Science
University of Pennsylvania

CIS 505, Spring 2007

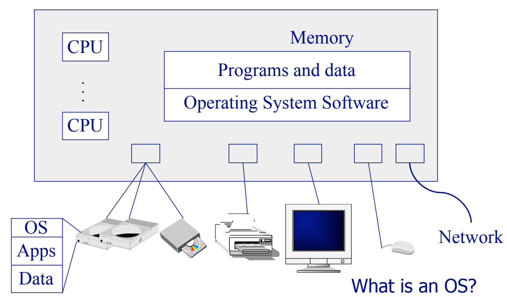
Selected Readings

- Some early systems
  - o E. W. Dijkstra, "The Structure of the THE Multiprogramming System," *Communications of the ACM*, Vol. 11, No. 5, May 1968, pp. 341–346.
  - o D. M. Ritchie and K. Thompson, "The UNIX Time-sharing System," *Bell System Technical Journal*, Vol. 57, No. 6, 1978, pp. 1905–1929.
  - o D. M. Ritchie, "The Evolution of the Unix Time-sharing System," *Bell System Technical Journal*, Vol. 63, No. 6, Part 2, October 1984, pp. 1577–1593.

Brief review on basic OS concepts

- What is an OS?
- System calls
- Signals
- Processes
- Threads
- Scheduling

A Typical Computer System



What is an OS?

What Is an OS?

"Code" that:

- Sits between programs & hardware
- Sits between different programs
- Sits betweens different users

But what does it do?

What Is an OS?

Resources
- Allocation
- Protection
- Reclamation
- Virtualization

Services
- Abstraction
- Simplification
- Convenience
- Standardization

Makes computers simpler

What Is an OS?

Resources
- Allocation
- Protection
- Reclamation
- Virtualization

Finite resources
Competing demands

Examples:
- CPU
- Memory
- Disk
- Network

What Is an OS?

Resources
- Allocation
- Protection
- Reclamation
- Virtualization

You can't hurt me
I can't hurt you

Implies some degree of safety & security

What Is an OS?

Resources
- Allocation
- Protection
- Reclamation
- Virtualization

The OS gives and
The OS takes away

Voluntary at run time
Implied at termination
Involuntary
Cooperative

What Is an OS?

Resources
- Allocation
- Protection
- Reclamation
- Virtualization

Illusion of infinite, private resources

Memory versus disk
Timeshared CPU

More extreme cases possible (& exist)

OS Service Examples

- System calls: file open, close, read and write
- Control the CPU so that users won't stuck by running `while ( 1 ) ;`
- Protection:
  - Keep user programs from crashing OS
  - Keep user programs from crashing each other
- Read time of the day

System calls

- A mechanism for user programs to obtain OS services
- Is it a procedure call?

System calls

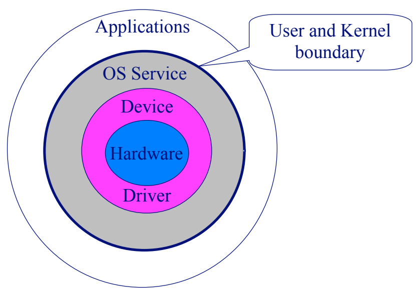
- A mechanism for user programs to obtain OS services
- Is it a procedure call?
  - User can't access kernel mode memory
  - Kernel can access user memory
- Kernel runs in privileged mode

>> kernel vs. OS, why privileged?

Kernel =? OS

- Kernel – "heart" of the operating system
  - Minimum set of mechanisms with universal applicability
- Operating system – usually includes more
  - Various libraries
  - Support programs

The Unix "Onion"



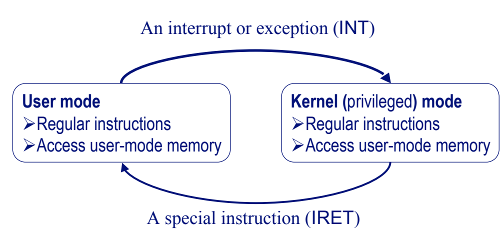
Why a Privileged Mode?

- Special Instructions
  - Mapping, TLB, etc
  - Device registers
  - I/O channels, etc.
- Mode Bits
  - Processor features
- Device access

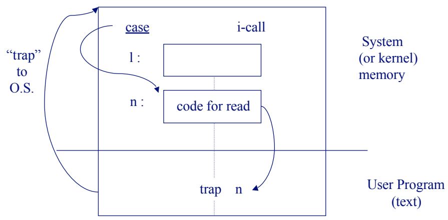
Protection Issues

- I/O protection
  - Prevent users from performing illegal I/Os
- Memory protection
  - Prevent users from modifying kernel code and data structures
- CPU protection
  - Prevent a user from using the CPU for too long

Support in Modern Processors: User ⇔ Kernel



User vs. System Mode



Special mode-bit set in PSW register:

mode-bit = 0 => user program executing

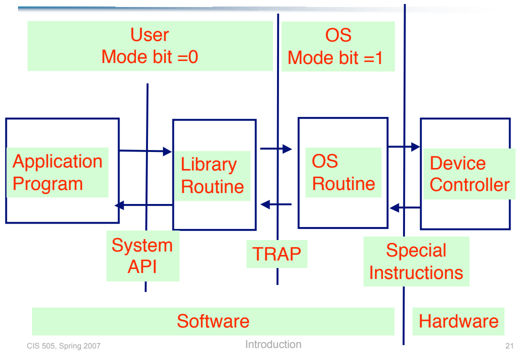
mode-bit = 1 => system routine executing

Privileged instructions possible only when mode-bit = 1!

System call

- User prospective
  - o Similar to function call
  - o But runs in kernel mode
- Difference from library routines?

System Calls



Steps in System Call

- User program pushes parameters to read on stack
- User program executes CALL instruction to invoke library routine **read** in assembly language
- Read routine sets up the register for system call number
- Read routine executes TRAP instruction to invoke OS
- Hardware sets the mode-bit to 1, saves the state of the executing read routine, and transfers control to a fixed location in kernel
- Kernel code, using a table look-up based on system call number, transfers control to correct system call handler

Steps in System Call (cont)

- OS routine copies parameters from user stack, sets up device driver registers, and executes the system call using privileged instructions
- OS routine can finish the job and return, or decide to suspend the current user process to avoid waiting
- Upon return from OS, hardware resets the mode-bit
- Control transfers to the read library routine and all registers are restored
- Library routine terminates, transferring control back to original user program
- User program increments stack pointer to clear the parameters

Memory allocation functions

- Sbrk(size)
  - o Increase size of heap by size
- Malloc(size)
  - o Allocate size byte on heap
- Both allocates memory
- Which is a system call?

Memory allocation functions

- Sbrk – allocates “pages” – hw protection
- Programs use malloc( ) – fine grained
- Kernel doesn’t care about small allocs
  - Allocates pages to library
  - Library handles malloc/free

Library Benefits

- Call overhead
  - Chains of alloc/free don’t go to kernel
- Flexibility – easy to change policy
  - Fragmentation
  - Coalescing, free list management
- Easier to program

Which is a System Call and Why

- read(int d, void *buf, size_t nbytes)
- fread(void *ptr, size_t size, size_t nmemb, FILE *stream)
- Both do the same thing, right?
- Buffered read

Feedback to the Program

- System calls and libraries are programs to OS
- What about other direction?
  - Various exceptional conditions
  - General information, like screen resize
- When would this occur?

Answer: signals

Polling Versus Interrupts

- Polling
  - Check “constantly”
  - Wastes resources – why?
  - Simpler design
- Interrupts
  - Controller free to do other work
  - More mechanism needed

When Are They Appropriate?

- Polling
  - Low cost systems
  - Low-delay environments
  - High-performance systems
  - Example: Real-time systems
- Interrupts
  - Multiprogrammed systems
  - Power-conscious environments

Why Interrupts for Syscalls?

- Interrupts have to exist
  - Hardware communication (IRQs)
  - Must be delivered to OS
  - Have to be in privileged mode
- Software interrupts for syscalls
  - Same infrastructure
  - Similar requirements

Interrupts and Exceptions

- Interrupt Sources
  - Hardware (by external devices)
  - Software: INT n
- Exceptions
  - Program error: faults, traps, and aborts
  - Software generated: INT 3, to debugger
  - Machine-check exceptions

Signals

- Notification mechanism to program
  - Used by OS/hardware to alert program
  - Asynchronous – like an interrupt
- What can program do?
  - Default action (signal-specific)
  - Ignore it
  - Perform some other action
- Man: signal, sigprocmask

Some Signals

SIGHUP	terminate process	terminal line hangup
SIGINT	terminate process	interrupt program
SIGILL	create core image	illegal instruction
SIGFPE	create core image	floating-point exception
SIGKILL	terminate process	kill program
SIGSEGV	create core image	segmentation violation
SIGPIPE	terminate process	write on a pipe with no reader
SIGALRM	terminate process	real-time timer expired
SIGURG	discard signal	urgent condition present on socket
SIGSTOP	stop process	stop (cannot be caught or ignored)
SIGCONT	discard signal	continue after stop