Lec 22: Interrupts

Kavita Bala
CS 3410, Fall 2008
Computer Science
Cornell University

Announcements

- HW 3
- HW4: due this Friday
- PA 3 out Nov 14th
  - Due Nov 25th (feel free to turn it in early)
  - Demos and pizza party: Dec 1st or 2nd
- Prelim 2: Dec 4th
- Final project: distributed multicore ray tracer
  - Due exam week

Caches/TLBs/VM

Caches, TLBs, Virtual Memory all understood by examining how they deal with the four questions

1. Where can block be placed?
   - Cache: direct, n-way set
   - TLB: fully assoc
   - VM: direct? Fully assoc?
2. What block is replaced on miss?
   - LRU? Random?
3. How are writes handled?
   - Write-back (fast, block at time)
   - Write-through (simple, reason about consistency)

Virtual Memory Design Parameters

	L1	Paged Memory	TLB
Size (blocks)	1/4k to 4k	16k to 1M	64 to 4k
Size (kB)	16 to 64	1M to 4G	2 to 16
Block size (B)	16-64	4k to 64k	4-32
Miss rates	2%-5%	10^{-4} to 10^{-5}%	0.01% to 2%
Miss penalty	10-25	10M-100M	10-1000

Hardware/Software Boundary

- Virtual to physical address translation is assisted by hardware
- Need hardware and software support
- Software
  - Page table storage, fault detection and updating
    - Page faults result in interrupts that are then handled by the OS
    - Must update appropriately Dirty and Reference bits (e.g., ~LRU) in the Page Tables

Hardware/Software Boundary

- OS has to keep TLB valid
- Keep TLB valid on context switch
  - Flush TLB when new process runs (x86)
  - Store process id (MIPs)

- Also, store pids with cache to avoid flushing cache on context switches

- Hardware support
  - Page table register
  - Process id register

Hardware/Software Boundary

- Hardware support for exceptions
  - Exception program counter
  - Cause register
  - Special instructions to load TLB
    - Only do-able by kernel
- Precise and imprecise exceptions
  - In pipelined architecture
    - Have to correctly identify PC of exception
    - MIPS and modern processors support this

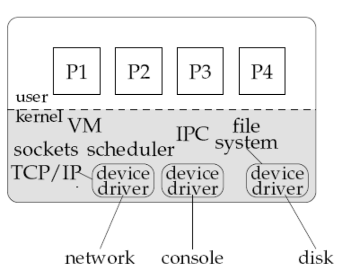
Hardware/Software Boundary

- Hardware guarantees
  - Previous instructions complete
  - Later instructions are flushed
  - EPC and cause register are set
  - Jump to prearranged address in OS
  - When you come back, restart instruction
  - Disable exceptions while responding to one
    - Otherwise can overwrite EPC and cause

Privileged Mode, Exceptions and Interrupts

Privilege Levels

- Some processor functionality cannot be made accessible to untrusted user applications
  - e.g. HALT, change MMU settings, set clock, reset devices, manipulate device settings, …
- Need to have a designated mediator between untrusted/untrusting applications
  - The operating system (OS)



Privilege Mode

- Need to delineate between untrusted applications and OS code
  – Use a “privilege mode” bit in the processor
  – 0 = Untrusted = user, 1 = Trusted = OS
- Privilege mode bit indicates if the current program can perform privileged operations
  – On system startup, privilege mode is set to 1, and processor jumps to a well-known address
  – The OS boot code resides at this address
  – The OS sets up the devices, initializes the MMU, loads applications, and resets the privilege bit before invoking the application
- Applications must transfer control back to OS for privileged operations

Terminology

- Trap
  – Any kind of a control transfer to the OS
- Syscall
  – Synchronous, program-initiated control transfer from user to the OS to obtain service from the OS
  – e.g. SYSCALL
- Exception
  – Asynchronous, program-initiated control transfer from user to the OS in response to an exceptional event
  – e.g. Divide by zero, TLB miss, Page fault
- Interrupt
  – Asynchronous, device-initiated control transfer from user to the OS
  – e.g. Network packet, I/O complete

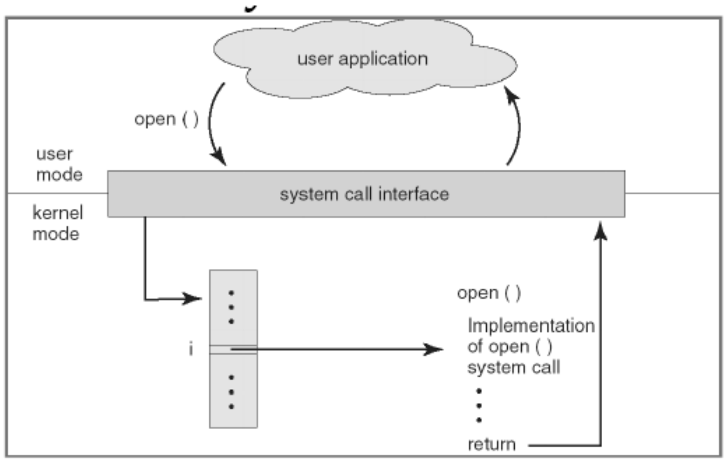
Sample System Calls

- Print character to screen
  - Needs to multiplex the shared screen resource between multiple applications
- Send a packet on the network
  - Need to manipulate the internals of a device
- Allocate a page
  - Needs to update page tables & MMU

System Calls

- A system call is a controlled transfer of execution from unprivileged code to the OS
  - An alternative is to make OS code read-only, and allow applications to just jump to the desired system call routine (less clean)

- A SYSCALL instruction transfers control to a system call handler at a fixed address
  - On the MIPS, v0 holds the syscall number, which specifies the operation the application is requesting



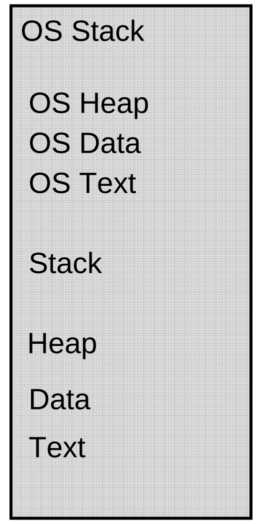
Where does OS live?

- In its own address space?
  – But then syscall would have to switch to a different address space
  – Also harder to deal with syscall arguments passed as pointers

- So in the same address space as process
  – Use protection bits to prevent user code from writing kernel
  – Higher part of VM, lower part of physical memory

Full System Layout

- Typically all kernel text, most data
  - At same VA in every address space
  - Map kernel in contiguous physical memory when boot loader puts kernel into physical memory
- The OS is omnipresent and steps in where necessary to aid application execution
  - Typically resides in high memory
- When an application needs to perform a privileged operation, it needs to invoke the OS



SYSCALL instruction

- SYSCALL instruction does an atomic jump to a controlled location
  - Switches the sp to the kernel stack
  - Saves the old (user) SP value
  - Saves the old (user) PC value (= return address)
  - Saves the old privilege mode
  - Sets the new privilege mode to 1
  - Sets the new PC to the kernel syscall handler

SYSCALL instruction

- Kernel system call handler carries out the desired system call
  - Saves callee-save registers
  - Examines the syscall number
  - Checks arguments for sanity
  - Performs operation
  - Stores result in v0
  - Restores callee-save registers
  - Performs a “return from syscall” instruction, which restores the privilege mode, SP and PC

Libraries and Wrappers

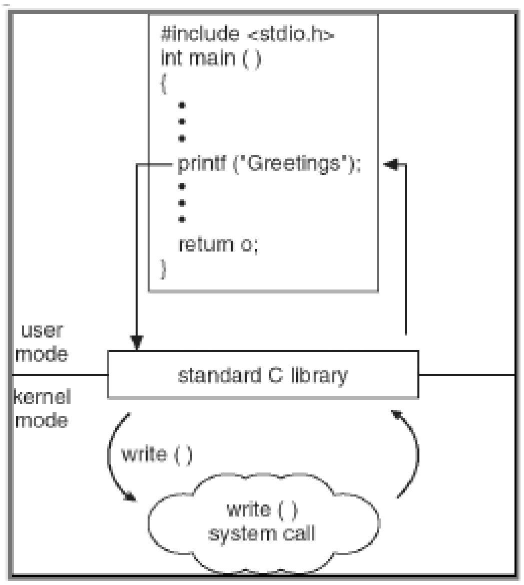
- Compilers do not emit SYSCALL instructions
  - They do not know the interface exposed by the OS
- Instead, applications are compiled with standard libraries, which provide “syscall wrappers”
  - printf() -> write(); malloc() -> sbrk(); recv(); open(); close(); …

- Wrappers are:
  - written in assembler
  - internally issue a SYSCALL instruction
  - pass arguments to kernel
  - pass result back to calling application

© Kavita Bala, Computer Science, Cornell University

Advantages?

- Portability



Exceptions

- System calls are control transfers to the OS, performed under the control of the user program
- Sometimes, need to transfer control to the OS at a time when the user program least expects it
  - Division by zero,
  - Alert from power supply that electricity is going out
  - Alert from network device that a packet just arrived
  - Clock notifying the processor that clock just ticked
- Some of these causes for interruption of execution have nothing to do with the user application
- Need a (slightly) different mechanism, that allows resuming the user application

Interrupts & Exceptions

- On an interrupt or exception
  - Switches the sp to the kernel stack
  - Saves the old (user) SP value
  - Saves the old (user) PC value
  - Saves the old privilege mode
  - Saves cause of the interrupt/privilege
  - Sets the new privilege mode to 1
  - Sets the new PC to the kernel interrupt/exception handler

Interrupts & Exceptions

- Kernel interrupt/exception handler handles the event
  - Saves all registers
  - Examines the cause
  - Performs operation required
  - Restores all registers
  - Performs a “return from interrupt” instruction, which restores the privilege mode, SP and PC

Syscall vs. Interrupt

- The differences lie in how they are initiated, and how much state needs to be saved and restored
- Syscall requires much less state saving
  - Caller-save registers are already saved by the application
- Interrupts typically require saving and restoring the full state of the processor
  - Because the application got struck by a lightning bolt without anticipating the control transfer

Terminology

- Trap
  - Any kind of a control transfer to the OS
- Syscall
  - Synchronous, program-initiated control transfer from user to the OS to obtain service from the OS
  - e.g. SYSCALL
- Exception
  - Asynchronous, program-initiated control transfer from user to the OS in response to an exceptional event
  - e.g. Divide by zero
- Interrupt
  - Asynchronous, device-initiated control transfer from user to the OS
  - e.g. Clock tick, network packet

© Kavita Bala, Computer Science, Cornell University