Architectural Support for Operating Systems

Prof. Sirer
CS 4410
Cornell University
Basic Computer Organization

CPU

Memory

?
Keyboard

Let’s build a keyboard

Reading input

- Mechanical switches
- Single pole, single throw
- Double pole, single throw

Digital processing

- Encoders, decoders, muxes, latches, tristate buffers, logic gates, ...
When a key is pressed, a 7-bit key identifier is computed.
A latch can store the keystroke indefinitely
The keyboard can then appear to the CPU as if it is a special memory address.
Device Interfacing Techniques

- **Memory-mapped I/O**
  - Device communication goes over the memory bus.
  - Reads/Writes to special addresses are converted into I/O operations by dedicated device hardware.
  - Each device appears as if it is part of the memory address space.

- **Programmed I/O**
  - CPU has dedicated, special instructions.
  - CPU has additional input/output wires (I/O bus).
  - Instruction specifies device and operation.

- Memory-mapped I/O is the predominant device interfacing technique in use.
Polling vs. Interrupts

- In our design, the CPU constantly needs to read the keyboard latch memory location to see if a key is pressed
  - Called polling
  - Inefficient

- An alternative is to add extra circuitry so the keyboard can alert the CPU when there is a keypress
  - Called interrupt driven I/O

- Interrupt driven I/O enables the CPU and devices to perform tasks concurrently, increasing throughput
  - Only needs a tiny bit of circuitry and a few extra wires to implement the “alert” operation
Interrupt Driven I/O

- An interrupt controller mediates between competing devices
  - Raises an interrupt flag to get the CPU’s attention
  - Identifies the interrupting device
- Can disable (aka mask) interrupts if the CPU so desires
Interrupt Driven I/O

- An interrupt controller mediates between competing devices
  - Raises an interrupt flag to get the CPU’s attention
  - Identifies the interrupting device
- Can disable (aka mask) interrupts if the CPU so desires
Interrupt Management

- Interrupt controllers manage interrupts
  - Maskable interrupts: can be turned off by the CPU for critical processing
  - Nonmaskable interrupts: signifies serious errors (e.g. unrecoverable memory error, power out warning, etc)

- Interrupts contain a descriptor of the interrupting device
  - A priority selector circuit examines all interrupting devices, reports highest level to the CPU

- Interrupt controller implements interrupt priorities
  - Can optionally remap priority levels
Interrupt-driven I/O summary

- Normal interrupt-driven operation with memory-mapped I/O proceeds as follows
  - CPU initiates a device operation (e.g. read from disk) by writing an operation descriptor to a device register
  - CPU continues its regular computation
  - The device asynchronously performs the operation
  - When the operation is complete, interrupts the CPU

- This would incur high-overhead for moving bulk-data
  - One interrupt per byte!
Direct Memory Access (DMA)

- Transfer data directly between device and memory
  - No CPU intervention required for moving bits

- Device raises interrupts solely when the block transfer is complete

- Critical for high-performance devices
Recap

- We now have a basic computer system to which devices can be connected.

- How do we execute applications on this system?
  - Applications are not necessarily trusted!
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)

- 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

- Privilege mode bit indicates if the current program can perform privileged operations
  - On system startup, privilege mode is set to 1, and the processor jumps to a well-known address
  - The operating system (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
Sample System Calls

- **Print character to screen**
  - Needs to multiplex the shared screen resource between multiple applications

- **Send a packet on the network**
  - Needs to manipulate the internals of a device whose hardware interface is unsafe

- **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
  - A potential alternative is to make OS code read-only, and allow applications to just jump to the desired system call routine. Why is this a bad idea?

- A SYSCALL instruction transfers control to a system call handler at a fixed address
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

- 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
Typical Process Layout

- Libraries provide the glue between user processes and the OS
  - libc linked in with all C programs
  - Provides printf, malloc, and a whole slew of other routines necessary for programs

```c
main() {
    printf("HELLO WORLD");
    printf("GO BIG RED CS");
}
```
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.
Exceptional Situations

- System calls are control transfers to the OS, performed under the control of the user application.

- Sometimes, need to transfer control to the OS at a time when the user program least expects it:
  - Division by zero,
  - Alert from the power supply that electricity is about to go out,
  - Alert from the network device that a packet just arrived,
  - Clock notifying the processor that the 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/exception
  - Sets the new privilege mode to 1
  - Sets the new PC to the kernel interrupt/exception handler

- 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, segmentation fault

- **Interrupt**
  - Asynchronous, device-initiated control transfer from user to the OS
  - e.g. Clock tick, network packet
Memory Protection

- Some memory addresses need protection
  - The OS text, data, heap and stack need to be protected from untrusted applications
  - Some devices should be out of reach of applications

- Memory Management Unit (MMU) aids with memory management
  - Provides a virtual to physical address translation
  - Examines every load/store/jump and ensures that applications remain within bounds using protection (RWX) bits associated with every page of memory

- Modern architectures use a Translation Lookaside Buffer (TLB) for keeping track of virtual to physical mappings
  - Software is invoked on a miss
TLB Operation

- TLB examines every virtual address uttered by the CPU, and if there is a match, and the permissions are appropriate, replaces the virtual page number with the physical page number.
Atomic Instructions

Hardware needs to provide special instructions to enable concurrent programs to operate correctly