Architectural Support for Operating Systems

Operating Systems
CS 4410/4411
Cornell University

The slides are the product of many rounds of teaching CS 4410 by Professors Sirer, Bracy, George, and Van Renesse.
Keyboard (1)

- 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.

4-bit encoder (16 to 4)
not all 16 wires shown

3-bit encoder (4 to 3)
• A latch can store the keystroke indefinitely
• The keyboard can then appear to the CPU as a special memory address
Device Interfacing Techniques

- **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**
  - Device communication goes over memory bus
  - Reads/Writes to special addresses converted into I/O operations by dedicated device hardware
  - Each device appears as if it is part of the memory address space
  - **Predominant device interfacing technique**
Polling vs. Interrupts

• In our design, CPU constantly reads the keyboard latch memory location to see if a key is pressed
  = Polling
    ▪ Inefficient

• Alternative: add extra circuitry so keyboard can alert CPU when there is a keypress
  = interrupt driven I/O
    → CPU and devices can perform tasks concurrently, increasing throughput
      • Only need a bit of circuitry + a few extra wires to implement “alert” operation
Interrupt Driven I/O

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

- Interrupt controller mediates between competing devices
  - Raises interrupt flag to get CPU’s attention
  - Identifies interrupting device
- Can disable (aka mask) interrupts if CPU 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
Exceptions

= any time control transfers to the OS

Hardware interrupts

Asynchronous

= caused by events external to CPU

Maskable

Can be turned off by CPU

Ex: alert from the network device that a packet just arrived, clock notifying CPU of clock tick

Software exceptions

Synchronous

= caused by CPU executing an instruction

Unmaskable

Cannot be ignored

Serious errors like alert from the power supply that electricity is about to go out
Software Exceptions

**Trap**
*Intentional*
Examples:
- **System call** (obtains service from the OS)
- **Breakpoint traps**
- **Privileged instructions**

**Fault**
*Unintentional but Possibly recoverable*
Examples:
- Division by zero
- Page fault

**Abort**
*Unintentional Not recoverable*
Examples:
- Parity error
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
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
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
Libraries provide the glue between user processes and the OS

- **libc** linked in with all C programs
- Provides `printf`, `malloc`, ... and other routines necessary for programs

```
printf(char * fmt, ...) {
  create the string to be printed
  SYSCALL 80
}
malloc() { ... }
strcmp() { ... }
main() {
  printf("HELLO WORLD");
  printf("GO BIG RED CS");
}
```
Full System Layout

OS:
• is omnipresent
• steps in to aid application execution
• typically resides in high memory

When an application needs to perform a privileged operation → needs to invoke the OS
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
SYSCALL instruction

• SYSCALL insn does atomic jump to a controlled location
  ▪ Sets the new privilege mode to “kernel mode”
  ▪ 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 (why?)
  ▪ Sets the new PC to the kernel syscall handler

• Kernel system call handler carries out desired system call
  ▪ Saves callee-save registers
  ▪ Examines the syscall number
  ▪ Checks arguments for sanity
  ▪ Performs operation
  ▪ Stores result in result register
  ▪ Restores callee-save registers
  ▪ Clears scratch registers

• “Return from SYSCALL” instruction
  ▪ restores the privilege mode, SP, and PC
Interrupts & Exceptions

• On an interrupt or exception
  ▪ Sets the new privilege mode to 1
  ▪ 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 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
  ▪ restores the privilege mode, SP and PC
Syscall vs. other exceptions

• Differences:
  ▪ how they are initiated
  ▪ how much state needs to be saved and restored

• Syscall requires less state saving
  ▪ Caller-save registers already saved by the application
  ▪ Does this mean less state restoring?

• Other exceptions typically require saving and restoring the full state of the processor
  ▪ because application got struck by lightning without anticipating the control transfer