Today we go into details about the structure and operation of the OS.

Forgive the formatting of my slides. For some reason I can’t convince PPT to allow different themes in a single slideshow to distinguish mine from the book authors!

So, I put ducks over the dinosaurs. Now you can tell them apart.
- I didn’t bother getting rid of the author info at the bottom though…

NOTE: Some of the material on the duck slides isn’t from the book.
- It’s mostly stuff I think is interesting or useful to know.
- Or places I think the book doesn’t explain as well as I’d like.
Last Time

- Last time we looked in a broad sense at what the OS is there to help out with.
  - Resource management
  - Process management

- These are abstract concepts. Today, we start making the details a bit more concrete.

- We’ll start with what I believe to be the most important parts of chapter 1.
  - Learn about interrupts: we’ve probably heard of them, but what are they?
  - Learn about CPU-level security enforcement.

- Then we’ll begin chapter 2 material.
Computer-System Operation

- I/O devices and the CPU can execute concurrently
- Each device controller is in charge of a particular device type
- Each device controller has a local buffer
- CPU moves data from/to main memory to/from local buffers
- I/O is from the device to local buffer of controller
- Device controller informs CPU that it has finished its operation by causing an interrupt
Common Functions of Interrupts

- Interrupt transfers control to the interrupt service routine generally, through the **interrupt vector**, which contains the addresses of all the service routines.
- Interrupt architecture must save the address of the interrupted instruction.
- Incoming interrupts are *disabled* while another interrupt is being processed to prevent a *lost interrupt*.
- A *trap* is a software-generated interrupt caused either by an error or a user request.
- An operating system is **interrupt driven**.
Interrupt Timeline

- CPU user process executing
- I/O interrupt processing
- I/O device idle transferring

- I/O request
- transfer done
- I/O request
- transfer done
Interrupt vectors

- Devices aren’t (or, rarely are) operating system aware.
- They provide a generic mechanism through which the OS informs them of a handle they use to call back to execute code when something interesting happens.
  - IE: USB device plugged in, or disk read completes.
- This can either be:
  - An index into a table within the OS, with the entry being the address of the code that handles the interrupt.
  - The interrupt handler address itself.
Interrupt handlers

- How does an interrupt handler work?
- The handler itself runs on the CPU, so it has to coexist with programs running there.
- It functions very similarly to what we will later see in dealing with multiple processes sharing a machine.

- First, the context of the program within the CPU is saved.
  - For a running program, this means the set of registers and program counter and any other important state.
- The interrupt handler then uses the CPU.
- When the handler finishes, the state of the program that was interrupted is restored and it continues on as though nothing had happened.
The most familiar interrupt.

- The clock!

- Ever wonder how the computer knows what time it is?
  - And why the clock doesn’t slow down when the machine is highly loaded?

- A clock device running at a relatively low frequency (compared to the CPU) causes a periodic interrupt.
  - The handler just increments a counter. The counter units are usually referred to as jiffies.
  - Typically measured in ms. (1ms = 1/1000 sec).
  - Common value is 4ms (1/250 sec).
Clocks and cycle timers

- Don’t confuse the timer device with the clock that drives the CPU.

- Timers for “time” are low frequency (Hz). The CPU clock is much faster (GHz).

- Most modern CPUs provide a non-interrupt driven method for accessing something called the “cycle counter”.
  - Typically a special purpose register on the CPU.
    - Each clock tick of the CPU it gets incremented automatically.
  - Useful for very high resolution timing for performance analysis, not driving a time-of-day clock.
  - Usually a special-purpose assembly instruction is provided to read the value of this special register.
Interrupts vs. Polling

- Depending on the device, sometimes it’s preferable to poll instead of wait for interrupts.

- Polling: while (“are we there yet?” == false) {} 
- Interrupt: “We’re there.”

- Polling may be better from a performance perspective if the device is quite busy and would otherwise overwhelm the CPU with interrupts.
  - Interrupts at high frequency = lots of contention with user programs.
- Polling can cause wasteful cycles to be spent when probing something slow.
  - Interrupts more appropriate in those cases so the CPU can do something useful while it waits for the event to occur.
Direct Memory Access Structure

- Used for high-speed I/O devices able to transmit information at close to memory speeds
- Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention
- Only one interrupt is generated per block, rather than the one interrupt per byte
Common Example of DMA

- Network cards
  - Allocate a buffer in RAM where you want data coming in over the network to be placed.
  - Network card streams data directly into this special place in RAM that a program can directly access.
  - Eliminates the need for an explicit copy from a device buffer to a program buffer.
    - Can get you into trouble though if you forget that the memory is DMA’d into by the card.
    - Example: Forgetting to check to see if a transfer is in progress and seeing bad data.
  - Interrupts are used to signal when DMA events occur.
  - Common in high performance networks. Not as common with slower ones.
Computer-System Architecture

- Most systems use a single general-purpose processor (PDAs through mainframes)
  - Most systems have special-purpose processors as well
- Multiprocessors systems growing in use and importance
  - Also known as parallel systems, tightly-coupled systems
  - Advantages include
    1. Increased throughput
    2. Economy of scale
    3. Increased reliability – graceful degradation or fault tolerance
  - Two types
    1. Asymmetric Multiprocessing
    2. Symmetric Multiprocessing
Symmetric Multiprocessing Architecture
Asymmetric multiprocessors

- Every participant isn’t identical.
- Some participants have special roles.
- For example, one CPU may act as a master and the other N-1 CPUs may act as workers.
- This structure is common in clusters and generic distributed systems.
Symmetric vs Asymmetric

- Often it is a question of complexity and scalability.

- Symmetric often requires agreement or consensus amongst the processors.
  - E.g.: The value of some common state.
  - Consensus algorithms often have superlinear complexity (N log N or N^2)

- Sometimes, you can get away with making one special processor the exclusive holder of this state.
  - Workers only have to ask master about it.
  - No negotiation amongst set of peers required.

- **Tradeoff**: Uniformity (*symmetric*) vs scalability (*asymmetric*).
A Dual-Core Design

- CPU core_0
  - registers
  - cache

- CPU core_1
  - registers
  - cache

- memory
Operating System Structure

- **Multiprogramming** needed for efficiency
  - Single user cannot keep CPU and I/O devices busy at all times
  - Multiprogramming organizes jobs (code and data) so CPU always has one to execute
  - A subset of total jobs in system is kept in memory
  - One job selected and run via **job scheduling**
  - When it has to wait (for I/O for example), OS switches to another job

- **Timesharing (multitasking)** is logical extension in which CPU switches jobs so frequently that users can interact with each job while it is running, creating **interactive** computing
  - **Response time** should be < 1 second
  - Each user has at least one program executing in memory ⇒ **process**
  - If several jobs ready to run at the same time ⇒ **CPU scheduling**
  - If processes don’t fit in memory, **swapping** moves them in and out to run
  - **Virtual memory** allows execution of processes not completely in memory
Memory Layout for Multiprogrammed System

- Operating system
- Job 1
- Job 2
- Job 3
- Job 4
Operating-System Operations

- **Interrupt** driven by hardware
- **Software** error or request creates **exception** or **trap**
  - Division by zero, request for operating system service
- Other process problems include infinite loop, processes modifying each other or the operating system
- **Dual-mode** operation allows OS to protect itself and other system components
  - **User mode** and **kernel mode**
  - **Mode bit** provided by hardware
    - Provides ability to distinguish when system is running user code or kernel code
    - Some instructions designated as **privileged**, only executable in kernel mode
    - System call changes mode to kernel, return from call resets it to user
Transition from User to Kernel Mode

- Timer to prevent infinite loop / process hogging resources
  - Set interrupt after specific period
  - Operating system decrements counter
  - When counter zero generate an interrupt
  - Set up before scheduling process to regain control or terminate program that exceeds allotted time
Recap: Responsibilities of OS

- Let’s review what we talked about in a rough sense last time.

- What are the responsibilities of the OS?
  - Process management
  - Resource management
  - Storage management
    - Arguably this is a special case of a resource.
    - I think the authors called it out as separate from resources that don’t just store bits, like CPUs.
A process is a program in execution. It is a unit of work within the system. Program is a *passive entity*, process is an *active entity*.

Process needs resources to accomplish its task
- CPU, memory, I/O, files
- Initialization data

Process termination requires reclaim of any reusable resources

Single-threaded process has one *program counter* specifying location of next instruction to execute
- Process executes instructions sequentially, one at a time, until completion

Multi-threaded process has one program counter per thread

Typically system has many processes, some user, some operating system running concurrently on one or more CPUs
- Concurrency by multiplexing the CPUs among the processes / threads
Process Management Activities

The operating system is responsible for the following activities in connection with process management:

- Creating and deleting both user and system processes
- Suspending and resuming processes
- Providing mechanisms for process synchronization
- Providing mechanisms for process communication
- Providing mechanisms for deadlock handling
Memory Management

- All data in memory before and after processing
- All instructions in memory in order to execute
- Memory management determines what is in memory when
  - Optimizing CPU utilization and computer response to users
- Memory management activities
  - Keeping track of which parts of memory are currently being used and by whom
  - Deciding which processes (or parts thereof) and data to move into and out of memory
  - Allocating and deallocating memory space as needed
Storage Management

- OS provides uniform, logical view of information storage
  - Abstracts physical properties to logical storage unit - file
  - Each medium is controlled by device (i.e., disk drive, tape drive)
    - Varying properties include access speed, capacity, data-transfer rate, access method (sequential or random)

- File-System management
  - Files usually organized into directories
  - Access control on most systems to determine who can access what
  - OS activities include
    - Creating and deleting files and directories
    - Primitives to manipulate files and dirs
    - Mapping files onto secondary storage
    - Backup files onto stable (non-volatile) storage media
Mass-Storage Management

- Usually disks used to store data that does not fit in main memory or data that must be kept for a “long” period of time
- Proper management is of central importance
- Entire speed of computer operation hinges on disk subsystem and its algorithms
- OS activities
  - Free-space management
  - Storage allocation
  - Disk scheduling
- Some storage need not be fast
  - Tertiary storage includes optical storage, magnetic tape
  - Still must be managed
  - Varies between WORM (write-once, read-many-times) and RW (read-write)
Performance of Various Levels of Storage

- Movement between levels of storage hierarchy can be explicit or implicit

<table>
<thead>
<tr>
<th>Level</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>registers</td>
<td>cache</td>
<td>main memory</td>
<td>disk storage</td>
</tr>
<tr>
<td>Typical size</td>
<td>&lt; 1 KB</td>
<td>&gt; 16 MB</td>
<td>&gt; 16 GB</td>
<td>&gt; 100 GB</td>
</tr>
<tr>
<td>Implementation</td>
<td>custom memory with</td>
<td>on-chip or off-chip</td>
<td>CMOS DRAM</td>
<td>magnetic disk</td>
</tr>
<tr>
<td>technology</td>
<td>multiple ports, CMOS</td>
<td>CMOS SRAM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Access time (ns)</td>
<td>0.25 – 0.5</td>
<td>0.5 – 25</td>
<td>80 – 250</td>
<td>5,000,000</td>
</tr>
<tr>
<td>Bandwidth (MB/sec)</td>
<td>20,000 – 100,000</td>
<td>5000 – 10,000</td>
<td>1000 – 5000</td>
<td>20 – 150</td>
</tr>
<tr>
<td>Managed by</td>
<td>compiler</td>
<td>hardware</td>
<td>operating system</td>
<td>operating system</td>
</tr>
<tr>
<td>Backed by</td>
<td>cache</td>
<td>main memory</td>
<td>disk</td>
<td>CD or tape</td>
</tr>
</tbody>
</table>

CMOS = Complementary metal-oxide semiconductor
Migration of Integer A from Disk to Register

- Multitasking environments must be careful to use most recent value, no matter where it is stored in the storage hierarchy.

- Multiprocessor environment must provide cache coherency in hardware such that all CPUs have the most recent value in their cache.

- Distributed environment situation even more complex:
  - Several copies of a datum can exist
  - Various solutions covered in Chapter 17
Abstractions provided by OS

Because the OS hides details about storage devices from you, clever tricks can be played.

- One example of this is memory mapping of files.

Say you have a file that you want to work with. You can either:

- Allocate a buffer, explicitly read file into buffer.
- Define a segment of virtual memory that, when read, results in File I/O operations behind the scenes.
  - Acts like bits in memory. In reality, it’s being read from disk.

This is an example of why abstractions provided by OS’s are nice.

- More on this when we talk about virtual memory.
I/O Subsystem

- One purpose of OS is to hide peculiarities of hardware devices from the user
- I/O subsystem responsible for
  - Memory management of I/O including buffering (storing data temporarily while it is being transferred), caching (storing parts of data in faster storage for performance), spooling (the overlapping of output of one job with input of other jobs)
  - General device-driver interface
  - Drivers for specific hardware devices
Protection and Security

- **Protection** – any mechanism for controlling access of processes or users to resources defined by the OS
- **Security** – defense of the system against internal and external attacks
  - Huge range, including denial-of-service, worms, viruses, identity theft, theft of service
- Systems generally first distinguish among users, to determine who can do what
  - User identities (**user IDs**, security IDs) include name and associated number, one per user
  - User ID then associated with all files, processes of that user to determine access control
  - Group identifier (**group ID**) allows set of users to be defined and controls managed, then also associated with each process, file
  - **Privilege escalation** allows user to change to effective ID with more rights
Into Chapter 2
Operating System Services

- One set of operating-system services provides functions that are helpful to the user:
  - User interface - Almost all operating systems have a user interface (UI)
    - Varies between Command-Line (CLI), Graphics User Interface (GUI), Batch
  - Program execution - The system must be able to load a program into memory and to run that program, end execution, either normally or abnormally (indicating error)
  - I/O operations - A running program may require I/O, which may involve a file or an I/O device
  - File-system manipulation - The file system is of particular interest. Obviously, programs need to read and write files and directories, create and delete them, search them, list file Information, permission management.
### A View of Operating System Services

<table>
<thead>
<tr>
<th>user and other system programs</th>
</tr>
</thead>
<tbody>
<tr>
<td>user interfaces</td>
</tr>
<tr>
<td>system calls</td>
</tr>
<tr>
<td>program execution</td>
</tr>
<tr>
<td>I/O operations</td>
</tr>
<tr>
<td>file systems</td>
</tr>
<tr>
<td>communication</td>
</tr>
<tr>
<td>resource allocation</td>
</tr>
<tr>
<td>accounting</td>
</tr>
<tr>
<td>protection and security</td>
</tr>
<tr>
<td>services</td>
</tr>
<tr>
<td>error detection</td>
</tr>
<tr>
<td>operating system</td>
</tr>
<tr>
<td>hardware</td>
</tr>
</tbody>
</table>