COMPUTER HOLY WARS

HOLD IT RIGHT THERE, BUDDY.

THAT SCRUFFY BEARD... THOSE SUSPENDERS... THAT SMUG EXPRESSION...

YOU'RE ONE OF THOSE CONDESCENDING UNIX COMPUTER USERS!

HERE'S A NICKEL, KID. GET YOURSELF A BETTER COMPUTER.

OS
What is my role?
What is my role?

Multi-faceted:
• Instructor
• Assessment
• Guide
What is your role?
What is your role?

• Absorb material so that the key ideas stay with you for a long time
• Perform well in the assessments

Don’t be passive!
• Ask questions
• Do the reading and the work
• Challenge yourself
• Don’t be afraid to try things
  • or throw out code
In the beginning…

Uniprocessors

• No real OS … (machine-level) programs access hardware directly
• Execute one program at a time
• I/O very slow
• Program waits for I/O

americanhistory.si.edu
Uniprocessors

Imagine a program that must wait for every I/O operation.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time (µs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read one record from file</td>
<td>15</td>
</tr>
<tr>
<td>Execute 100 instructions</td>
<td>1</td>
</tr>
<tr>
<td>Write one record to file</td>
<td>15</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td><strong>31</strong></td>
</tr>
</tbody>
</table>

Percent CPU Utilization: 

\[
\text{Percent CPU Utilization} = \frac{1}{31} = 0.032 = 3.2\%
\]
CPU Utilization with I/O Bound Programs

(a) Uniprogramming

Program A

<table>
<thead>
<tr>
<th>Run</th>
<th>Wait</th>
<th>Run</th>
<th>Wait</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td>B</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.5  Multiprogramming Example
Multiprogramming

(b) Multiprogramming with two programs
Multiprogramming

Program A
- Run
- Wait
- Run
- Wait

Program B
- Wait
- Run
- Wait
- Run
- Wait

Program C
- Wait
- Run
- Wait
- Run
- Wait

Combined
- Run A
- Run B
- Run C
- Wait
- Run A
- Run B
- Run C
- Wait

Time

(c) Multiprogramming with three programs
Multiprogramming

In order to get this to work, we must have:

• A way to figure out which job to switch to next
• The memory space to fit the jobs being executed
• A mechanism that performs the switching between the jobs

These functions are provided by the OS
Processes

• A*process* is a program in execution:
  • It is a unit of work within the system.
  • Program is a passive entity, process is the active entity

• Process needs resources to accomplish its task
  • CPU, memory, I/O, files

• OS manages these resources
  • Process termination requires the OS to reclaim of any reusable resources
Processes

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

• Typically a system has many processes
  • Some user, some OS-related
  • These are running concurrently on one or more CPUs
Multi-Threading

Even more complicated systems support *multi-threaded processes*: a process has one program counter per thread

- Allows execution of many closely-linked tasks in parallel
Process Management Activities

The OS is responsible for:

• Creating and deleting both user and system processes
• Suspending and resuming processes
• Scheduling processes to have access to resources, including the CPU
• Providing mechanisms for process synchronization and deadlock handling
• Providing mechanisms for process communication
Making Efficient Use of a CPU

• **Multiprogramming:**
  • Switch between processes as CPU becomes idle (e.g., if a process is waiting for I/O)
  • Scheduling processes is relatively straight-forward

• **Multitasking:**
  • Switch quickly between processes automatically
    • Processes have a fixed upper bound of time before needed to wait
  • Allows processes to appear like they are responding in real time (at least to a user)
  • Scheduling processes and their memory use is a challenge
Protection with Processor Modes

Dual-mode operation allows the OS to protect itself and other system components

• Mode bit provided in the hardware:
  • User mode and kernel mode

• Provides ability to distinguish when system is running user code versus kernel code

• Some instructions designated as privileged and can only be executed in kernel mode

• Some hardware can only be manipulated in privileged mode
Protection with Processor Modes

- **System calls** change mode from user to kernel
  - Allow safe manipulation of kernel data structures and hardware
  - Return from call resets mode to user

- Increasingly, CPUs support multi-mode operations
  - For example: virtual machine manager (VMM) mode for guest VMs
System Calls

System calls allow a user program to request services from the kernel
  • Including I/O and process management services
Computer System Organization

Common bus structure:

- One or more CPUs, device controllers connect through a common bus that provides access to shared memory
- Concurrent execution of CPUs and devices
  - All can compete for memory cycles
Computer-System Operation

I/O devices and the CPU execute concurrently

• Each device controller is in charge of a particular device
• Data sent to or received from the device are stored in a local buffer
• CPU moves data between these local buffers and main memory
• When a device controller completes an I/O operation, it informs the CPU by causing an interrupt
Interrupts

An operating system is **interrupt driven**

• An interrupt transfers control from the currently executing program to the appropriate interrupt service routine

• Interrupt architecture must save the address of the interrupted instruction, as well as the state of the registers

• A **trap** or **exception** is a software-generated interrupt caused either by an error or a user request
Interrupt Timeline for I/O

CPU
- user process executing

I/O device
- idle
- transferring

I/O request
- transfer done
- I/O request
- transfer done
I/O Structure

• User program does not have direct access to the devices (it is prevented explicitly!)
• Instead, a request for access is made to the OS through the use of a system call
  • Special function that is able to access the kernel-level data structures and I/O system
• After I/O starts, control returns to user program without waiting for I/O completion
Storage Definitions
Storage Definitions

- Bit: contains a value of 0 or 1
- Byte: 8-bits. Fundamental unit of memory
- Word: multiple bytes (system dependent)
  - In modern laptops: 8 bytes
- $2^{10}$ bytes: kilobyte
- $2^{20}$ bytes: megabyte
- $2^{30}$ bytes: gigabyte
- $2^{40}$ bytes: terabyte
Storage Types
Storage Types (some)

• Main memory – only large storage media that the CPU can access directly
  • Random access, typically volatile

• Secondary storage – extension of main memory that provides large nonvolatile storage capacity
  • Hard disks – rigid metal or glass platters covered with magnetic recording material
  • Disk surface is logically divided into tracks, which are subdivided into sectors

• Solid-state disks – faster than hard disks, nonvolatile
  • Various technologies
  • Expensive relative to hard disks
## Performance of Various Levels of Storage

<table>
<thead>
<tr>
<th>Level</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>registers</td>
<td>cache</td>
<td>main memory</td>
<td>solid state disk</td>
<td>magnetic disk</td>
</tr>
<tr>
<td>Typical size</td>
<td>&lt; 1 KB</td>
<td>&lt; 16MB</td>
<td>&lt; 64GB</td>
<td>&lt; 1 TB</td>
<td>&lt; 10 TB</td>
</tr>
<tr>
<td>Implementation technology</td>
<td>custom memory with multiple ports CMOS</td>
<td>on-chip or off-chip CMOS SRAM</td>
<td>CMOS SRAM</td>
<td>flash memory</td>
<td>magnetic disk</td>
</tr>
<tr>
<td>Access time (ns)</td>
<td>0.25 - 0.5</td>
<td>0.5 - 25</td>
<td>80 - 250</td>
<td>25,000 - 50,000</td>
<td>5,000,000</td>
</tr>
<tr>
<td>Bandwidth (MB/sec)</td>
<td>20,000 - 100,000</td>
<td>5,000 - 10,000</td>
<td>1,000 - 5,000</td>
<td>500</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>
<td>operating system</td>
</tr>
<tr>
<td>Backed by</td>
<td>cache</td>
<td>main memory</td>
<td>disk</td>
<td>disk</td>
<td>disk or tape</td>
</tr>
</tbody>
</table>
Storage-Device Hierarchy

- registers
- cache
- main memory
- solid-state disk
- hard disk
- optical disk
- magnetic tapes
Storage Hierarchy

• Storage systems organized in hierarchy. Each level involves trade-offs:
  • Speed
  • Cost
  • Volatility

• **Caching** – copying information into faster storage system
  • Allows faster access to and alterations of data
  • Main memory can be viewed as a cache for secondary storage
Caching

Information in use copied from slower to faster storage temporarily

• Important principle, performed at many levels in a computer (in hardware, operating system, software)

• Faster storage (cache) checked first to determine if information is there
  • If it is, information used directly from the cache (fast)
  • If not, data copied to cache and used from there

• Cache management is an important design choice
  • Including: cache size and replacement policy
Direct Memory Access

- 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
Data Flow in a Modern Computer

![Diagram of data flow in a modern computer](image)
Computer-System Architecture

• 15 years ago: most systems used a single general-purpose processor
  • Most systems (even today) also have special-purpose processors

• Multiprocessor systems have grown in use and importance
  • Also known as parallel systems, tightly-coupled systems
  • Advantages include:
    • Increased throughput
    • Economy of scale
    • Increased reliability – graceful degradation or fault tolerance

• Two types:
  • Asymmetric Multiprocessing – each processor performs specialized tasks
  • Symmetric Multiprocessing – each processor performs all tasks
Multiprocessing Architectures

Symmetric Multiprocessor: loosely coupled, multiple chips

Multi-Core Processors: tightly coupled, single chip
Clusters

Cluster: large number of coordinated computers

- Programs can execute in parallel across multiple computers
  - Number of computers can scale with demand
  - High Performance Computing (HPC) clusters: 1000s of nodes
- A single computer can potentially be used by many programs
  - More efficient use of hardware
- Provides redundancy in the face of hardware failure
Storage Management

• OS provides uniform, logical view of information storage
  • Abstracts physical properties to logical storage unit
  • These physical 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 directories
    • Mapping files onto secondary storage
    • Backup files onto stable (non-volatile) storage media
Memory Management

• To execute a program all (or part) of the instructions must be in memory
• All (or part) of the data that is needed by the program must be in memory.
• Memory management determines what is brought into memory and when
  • Optimizing CPU utilization and computer response to users
Memory Management Task

• 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
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
Protection and Security

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
Kernel-Level Data Structures

Requirements

• Space efficient

• Time efficient
  • Many data structures exist over the lifetime of the system
  • Queries and small changes to the data structure must be quick

• Secure
  • Manipulated only in kernel mode
  • Changes must leave the data structure in a proper state
Kernel Data Structures

• Singly linked list

• Doubly linked list

• Circular linked list
Kernel Data Structures

• Linear list
  • Search performance is $O(n)$

• Binary search tree
  left $\leq$ right
  • Balanced binary search tree access is $O(\lg n)$
Kernel Data Structures

• Hash functions:
  • Translate some many-byte data structure into a short hash value
  • Small changes in the data structure mean substantial changes in the hashed value
  • These are typically one-way functions!

• Hash maps:
  • Associate a hash value with some other data structure
  • O(1) lookup and storage
  • Hash table must be large relative to the number of items stored
Kernel Data Structures

Bitmaps

• A word is composed of k bits
• If we need to store a set of Boolean values, we can map each to one of these bits
• Example: allocation table for k blocks on a hard disk
  • Each bit indicates whether the corresponding block is used by a file or is free to be allocated to new files
  • 0xC7: blocks 3, 4 and 5 are free to be used
Kernel Data Structures

In Linux (and other OSes), these data structures types are modularly implemented

• Used by different OS components
• Satisfy security requirements (hopefully)
• Well tested and debugged (hopefully)
Distributed Computing

- Collection of separate, possibly heterogeneous, systems networked together
- Goals: achieve the illusion of a single system
- Network is a communications path, TCP/IP most common protocol
  - Local Area Network (LAN)
  - Wide Area Network (WAN)
  - Metropolitan Area Network (MAN)
  - Personal Area Network (PAN)
Client-Server Computing

Remote server provides some service to many different clients
• File system: storage and retrieval of files
• Database
• Map services
• Image recognition
• Messaging
Peer-to-Peer Systems

P2P does not distinguish clients and servers

- All nodes are considered peers
- May each act as client, server or both
- Node must join P2P network
  - Registers its service with central lookup service on network, or
  - Broadcast request for service and respond to requests for service via discovery protocol
- Examples include Napster and Gnutella, Voice over IP (VoIP) such as Skype
Virtualization

• Allows operating systems to run applications within other OSes

• Vast and growing industry

• **Emulation** used when source CPU type different from target type (i.e. PowerPC to Intel x86)
  • Generally slowest method
  • When computer language not compiled to native code, *Interpretation* is required

• **Virtualization**: OS natively compiled for CPU, running guest OSes that are also natively compiled
  • VMware running WinXP guests, each running applications, all on native WinXP host OS
  • VMM (virtual machine Manager) provides virtualization services
Virtualization

• Use cases involve laptops and desktops running multiple OSes for exploration or compatibility
  • Apple laptop running Mac OS X host, Windows as a guest
  • Developing apps for multiple OSes without having multiple systems
  • QA testing applications without having multiple systems
  • Executing and managing compute environments within data centers

• VMM can run natively, in which case they are also the host
  • There is no general purpose host then (VMware ESX and Citrix XenServer)
Virtualization
Cloud Computing

• Delivers computing, storage, even apps as a service across a network

• Logical extension of virtualization because it uses virtualization as the base for its functionality

• Amazon EC2 has thousands of servers, millions of virtual machines, petabytes of storage available across the Internet
  • Users pay based on usage
Cloud Computing: Many Types

• Public cloud – available via Internet to anyone willing to pay
• Private cloud – run by a company for the company’s own use
• Hybrid cloud – includes both public and private cloud components
• Software as a Service (SaaS) – one or more applications available via the Internet (i.e., word processor)
• Platform as a Service (PaaS) – software stack ready for application use via the Internet (i.e., a database server)
• Infrastructure as a Service (IaaS) – servers or storage available over Internet (i.e., storage available for backup use)
Cloud Computing

• Cloud computing environments composed of traditional OSes, plus VMMs, plus cloud management tools
• Internet connectivity requires security like firewalls
• Load balancers spread traffic and load across multiple machines
Real-Time Embedded Systems

• Real-time embedded systems are the most prevalent form of computers
  • Vary considerably, special purpose, limited purpose OS
  • Real-time OS
  • Use expanding
• Many other special computing environments as well
  • Some have OSes, some perform tasks without an OS
• Real-time OS has well-defined fixed time constraints
  • Processing must be done within constraint
  • Correct operation only if constraints met
Open Source Operating Systems

Full source code is available for some OSes

- Individuals can make changes to the source & build their own OS version
- These changes can be integrated back to the main distribution
- Many “eyes” on the source code: improve quality of the code
  - Just discovered at Def Con (last week): malicious code was inserted into Linux component that allows administrator-level privileges under certain conditions
Next Week

Practicalities of writing and executing code
• System calls for I/O
• Linux environment
• Writing and compiling code
• Low-level data representation in C