OS and Architecture Overview
What is an operating system?

application (user)  
-------------------  
operating system  
-------------------  
hardware

- **A software layer**
  - between hardware and application programs/users,
  - provides a *virtual machine* interface
    - easy to use (hides complexity)
    - safe (prevents and handles errors)

- **Acts as resource manager**
  - allows programs/users to share hardware resources
  - in a protected way: fair and efficient
Operating System Definition

- A program that acts as an intermediary between a user of a computer and the computer hardware

Operating System Goals
- Execute user programs and make solving user problems easier
- Make the computer system convenient to use
- Use the computer hardware in an efficient manner
How does an OS work?

- Receives requests from the application: system calls
- Satisfies the requests: may issue commands to hardware
- Handles hardware interrupts: may upcall the application
- OS complexity: synchronous calls + asynchronous events
Storage Device Hierarchy

- magnetic tapes
- optical disk
- magnetic disk
- electronic disk
- main memory
- cache
- registers
### Performance of various levels of storage

<table>
<thead>
<tr>
<th>Level</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td>Registers</td>
<td>Cache</td>
<td>Main memory</td>
<td>Disk storage</td>
</tr>
<tr>
<td><strong>Typical size</strong></td>
<td>&lt; 1KB</td>
<td>&gt; 16MB</td>
<td>&gt; 16 GB</td>
<td>&gt; 100GB</td>
</tr>
<tr>
<td><strong>Access time (ns)</strong></td>
<td>0.25-0.5</td>
<td>0.5-25</td>
<td>80-250</td>
<td>5000</td>
</tr>
<tr>
<td><strong>Bandwidth (MB/sec)</strong></td>
<td>20,000-100,000</td>
<td>5000-10,000</td>
<td>1000-5000</td>
<td>20-150</td>
</tr>
<tr>
<td><strong>Managed by</strong></td>
<td>compiler</td>
<td>hardware</td>
<td>Operating system</td>
<td>Operating system</td>
</tr>
</tbody>
</table>
Caching

- Important principle, performed at many levels in a computer (in hardware, operating system, software)
- Information in use copied from slower to faster storage temporarily
- 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 there
- Cache smaller than storage being cached
- Cache management important design problem
  - Cache size and replacement policy
Caching

- Why does cache work?
  - Temporal Locality: a program is likely to access data it has just recently accessed
  - Spatial Locality: a program is likely to access data that are close to what have just been accessed
- Requires a *cache management* policy
- Caching introduces another level in storage hierarchy.
  - This requires data that are simultaneously stored in more than one level to be *consistent*
Questions!

- How does the application use the OS services? Is it a synchronous or asynchronous?
- How does hardware interact with OS? Is it synchronous or asynchronous?
Hardware Protection

- Dual-Mode Operation
- I/O Protection
- Memory Protection
- CPU Protection
Dual-Mode Operation

- OS requires hardware support to differentiate between at least two modes of operations
  1. *User mode* – execution done on behalf of a user
  2. *Kernel/monitor mode* – execution done on behalf of operating system
Dual-Mode Operation (Cont.)

- *Mode bit* added to computer hardware to indicate the current mode:
  - kernel (0) or user (1)

- When an interrupt or fault occurs hardware switches to kernel mode

  Interrupt/fault

  ![Diagram](Diagram)

  - kernel
  - user
  - set user mode

*Privileged instructions* can be issued only in kernel mode
I/O Protection

- All I/O instructions are privileged instructions
- Must ensure that a user program could never gain control of the computer in kernel mode
  - For example, a user program that, as part of its execution, stores a new address in the interrupt vector)
Use of A System Call to Perform I/O

1. trap to monitor
2. perform I/O
3. return to user

system call n

resident monitor

read

...
Memory Protection

- Must provide memory protection at least for the interrupt vector and the interrupt service routines
- In order to have memory protection, at a minimum add two registers that determine the range of legal addresses a program may access:
  - **Base register** – holds the smallest legal physical memory address
  - **Limit register** – contains the size of the range
- Memory outside the defined range is protected
Use of A Base and Limit Register
Hardware Address Protection

CPU

address

≥

yes

≥

no

<

no

yes

trap to operating system monitor—addressing error

memory
Hardware Protection

- When executing in the kernel mode, the operating system has unrestricted access to both kernel and user’s memory.
- The load instructions for the base and limit registers are privileged instructions.
CPU Protection

- **Timer** – interrupts computer after specified period to ensure operating system maintains control
  - Timer is decremented every clock tick
  - When timer reaches the value 0, an interrupt occurs
- Timer commonly used to implement time sharing
- Timer also used to compute the current time
- Load-timer is a privileged instruction
Can CPU directly access the following devices?

(A) Disk;
(B) Memory;
(C) Register;
(D) Network;
(E) CD-ROM;
OS Structure

- Monolithic
- Layered
- Microkernel
Monolithic

- user applications
- OS: procs+data
- hardware

user/kernel boundary
Layered Approach

- The operating system is divided into a number of layers (levels), each built on top of lower layers.
- The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface.
- With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers.
Layered Operating System
Microkernel System Structure

- Moves as much from the kernel into “user” space
- Communication takes place between user modules using message passing
- **Advantages:**
  - Easier to extend a microkernel
  - Easier to port the operating system to new architectures
  - More reliable (less code is running in kernel mode)
  - More secure
- **Disadvantages:**
  - Performance overhead of user space to kernel space communication
Microkernel in action

- System services at the same level as user process
- System call crosses user/kernel boundary many times
## Comparison

<table>
<thead>
<tr>
<th></th>
<th>Performance</th>
<th>Extensibility</th>
<th>Reliability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monolithic</td>
<td>✓</td>
<td>❌</td>
<td>❌</td>
</tr>
<tr>
<td>Layered</td>
<td>❌</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>Microkernel</td>
<td>—</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

- good
- In between
- bad
MS-DOS System Structure

- MS-DOS – written to provide the most functionality in the least space
  - Not divided into modules
  - Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated
MS-DOS Execution (Single Process OS)

At System Start-up

Running a Program
MS-DOS Layer Structure

- Application program
- Resident system program
- MS-DOS device drivers
- ROM BIOS device drivers
UNIX Running Multiple Programs

<table>
<thead>
<tr>
<th>process D</th>
</tr>
</thead>
<tbody>
<tr>
<td>free memory</td>
</tr>
<tr>
<td>process C</td>
</tr>
<tr>
<td>interpreter</td>
</tr>
<tr>
<td>process B</td>
</tr>
<tr>
<td>kernel</td>
</tr>
</tbody>
</table>
Communication Models

Communication may take place using either message passing or shared memory.

Message Passing

Shared Memory
UNIX system structure

- **UNIX**
  - Limited hardware functionality -> the original UNIX operating system had limited structuring
  - UNIX OS consists of two separable parts
- **Systems programs**
- **The kernel**
  - Everything below the system-call interface and above the physical hardware
  - Provides
    - File system
    - CPU scheduling
    - Memory management
    - Etc.
  - A large number of functions for one level
### UNIX System Structure

<table>
<thead>
<tr>
<th>the users</th>
<th>shells and commands</th>
<th>compilers and interpreters</th>
<th>system libraries</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>system-call interface to the kernel</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>signals terminal handling</td>
<td>file system</td>
<td>CPU scheduling page replacement</td>
</tr>
<tr>
<td></td>
<td>character I/O system</td>
<td>swapping block I/O system</td>
<td>demand paging</td>
</tr>
<tr>
<td></td>
<td>terminal drivers</td>
<td>disk and tape drivers</td>
<td>virtual memory</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>kernel interface to the hardware</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>terminal controllers</td>
<td>device controllers</td>
<td>memory controllers</td>
</tr>
<tr>
<td></td>
<td>terminals</td>
<td>disks and tapes</td>
<td>physical memory</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Mac OS X Structure

- Hybrid architecture
  - Layered structure + Mach microkernel