Operating Systems:
Basic Concepts and History

What is an Operating System?

- A program and an interface
  - An abstract virtual machine
  - A set of abstractions that simplify application design
    - Files instead of "bytes on a disk"
- For any OS area (CPU scheduling, file systems, memory management), begin by asking two questions
  - What’s the hardware interface? (The Physical Reality)
  - What is the application interface? (The Nicer Interface)
- Key questions:
  - Why is the application interface defined the way it is?
  - Should we push more functionality into applications, the OS, or the hardware?
  - What are the tradeoffs between programmability, complexity, and flexibility?

Operating System Functions

- **Service provider**
  - Provide standard facilities
    - File system
    - Standard libraries
    - Window system
    - ...
- **Coordinator:** three aspects
  - Security: prevent jobs from interfering with each other
  - Communication: enable jobs to interact with each other
  - Resource management: facilitate sharing of resources across jobs
- **Examples**
  - Single-function devices (embedded controllers, Nintendo, ...)
    - OS provides a collection of standard services
  - Multi-function/application devices (workstations and servers)
    - OS manages application interactions

Why do we need operating systems?

- **Convenience**
  - Provide a high-level abstraction of physical resources
  - Enable the construction of more complex software systems
  - Enable portable code
- **Efficiency**
  - Share limited or expensive physical resources
  - Provide protection
**Evolution of Operating Systems**

- Why do operating systems change?
  - Key functions: hardware abstraction and coordination
  - Principle: Design tradeoffs change as technology changes
  - Underlying technology has changed immensely over the past two decades!!

- Comparing computing systems from 1981 and 2000

<table>
<thead>
<tr>
<th></th>
<th>1981</th>
<th>2000</th>
<th>Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIPS</td>
<td>1</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>$/SPECint</td>
<td>$100K</td>
<td>$2</td>
<td>50000</td>
</tr>
<tr>
<td>DRAM size</td>
<td>128KB</td>
<td>256MB</td>
<td>2000</td>
</tr>
<tr>
<td>Disk size</td>
<td>10MB</td>
<td>10GB</td>
<td>1000</td>
</tr>
<tr>
<td>Net BW</td>
<td>9600 bps</td>
<td>100 Mb/s</td>
<td>10000</td>
</tr>
<tr>
<td>Address bits</td>
<td>16</td>
<td>64</td>
<td>4</td>
</tr>
<tr>
<td>Users/machine</td>
<td>100</td>
<td>1</td>
<td>100</td>
</tr>
</tbody>
</table>

**History of Operating Systems: Phases**

- Phase 1: Hardware is expensive, humans are cheap
  - User at console: single-user systems
  - Batching systems
  - Multi-programming systems

- Phase 2: Hardware is cheap, humans are expensive
  - Time sharing: Users use cheap terminals and share servers

- Phase 3: Hardware is very cheap, humans are very expensive
  - Personal computing: One system per user
  - Distributed computing: Lots of systems per user

- Phase 4: Richer services
  - Real-time operating systems

**A Brief History of Operating Systems**

Hand programmed machines (45-55)

- Single user systems
- OS = loader + libraries of common subroutines
- Problem: low utilization of expensive components

\[
\text{time device busy} = \frac{\text{observation interval}}{} \times \text{utilization}
\]
**Batch/Off-line processing ('55-'65)**

- Batch processing v. sequential execution of jobs

Card Reader: Read Batch 1
CPU: Execute Batch 1
Printer: Print Batch 1

Card Reader: Read Batch 2
CPU: Execute Batch 2
Printer: Print Batch 2

Card Reader: Read Batch 3
CPU: Execute Batch 3
Printer: Print Batch 3

---

**Batch processing ('55-'65)**

- Operating system = loader + sequencer + output processor

Card Reader → CPU → Printer

---

**Multiprogramming ('65-'80)**

- Keep several jobs in memory and multiplex CPU between jobs

User Program 1
User Program 2
User Program n
System Software
Operating System

```
program P begin
  Read(var)
end P
```

System call Read():
- StartIO(input device)
- WaitIO(interrupt)
- EndIO(input device)

Simple, "synchronous" input: What to do while we wait for the I/O device?

---

**Multiprogramming ('65-'80)**

- Keep several jobs in memory and multiplex CPU between jobs

User Program 1
User Program 2
User Program n
System Software
Operating System

```
main {
  read() startIO()
  waitIO()
}
```

Program 1 OS I/O Device
### History of Operating Systems: Phases

<table>
<thead>
<tr>
<th>Phase 1: Hardware is expensive, humans are cheap</th>
</tr>
</thead>
<tbody>
<tr>
<td>User at console: single-user systems</td>
</tr>
<tr>
<td>Batch systems</td>
</tr>
<tr>
<td>Multi-programming systems</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Phase 2: Hardware is cheap, humans are expensive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time sharing: Users use cheap terminals and share servers</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Phase 3: Hardware is very cheap, humans are very expensive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Personal computing: One system per user</td>
</tr>
<tr>
<td>Distributed computing: lots of systems per user</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Phase 4: Richer services</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real-time operating systems</td>
</tr>
</tbody>
</table>

---

### Multiprogramming ('65-'80)

- Keep several jobs in memory and multiplex CPU between jobs

| User Program 1 |
| User Program 2 |
| User Program 1 |
| “System Software” |

| Operating System |

---

### Timesharing ('70- )

- A timer interrupt is used to multiplex CPU among jobs

| User Program n |
| User Program 2 |
| User Program 1 |
| “System Software” |

| Operating System |
**Operating Systems for PCs**

- **Personal computing systems**
  - Single user
  - Utilization is no longer a concern
  - Emphasis is on user interface and API
  - Many services & features not present

- **Evolution**
  - Initially: OS as a simple service provider
    (simple libraries)
  - Now: Multi-application systems with support for coordination

**Distributed Operating Systems**

- Typically support distributed services
  - Sharing of data and coordination across multiple systems
- Possibly employ multiple processors
  - Loosely coupled vs. tightly coupled systems
- High availability & reliability requirements

**History of Operating Systems: Phases**

- **Phase 1**: Hardware is expensive, humans are cheap
  - User at console: single-user systems
  - Batch systems
  - Multi-programming systems

- **Phase 2**: Hardware is cheap, humans are expensive
  - Time sharing: Users use cheap terminals and share servers

- **Phase 3**: Hardware is very cheap, humans are very expensive
  - Personal computing: One system per user
  - Distributed computing: lots of systems per user

- **Phase 4**: Richer services
  - Real-time operating systems

**Richer Operating Systems**

- A system with a dual notion of correctness
  - Logical correctness — “it does the right thing”
  - Temporal correctness — “it does it on time”

- A system wherein predictability is more important than performance

*Example: Digital video playout*

```c
/* Main processing loop */
loop
  data = read( network)
  video_frame = decompress(data)
  write( frame_buffer, video_frame)
end loop
```

Timing constraint: Execute loop once every 33 ms.
**Real-time Operating Systems: Issues**

Digital video processing loop:
```c
/* Main processing loop */
loop
    data = read(network);
    video_frame = decompress(data);
    display(video_frame);
end loop
```

Timing constraint: Execute loop once every 33 ms.

- **Dedicated system** — real-time performance \( \text{iff} \) \( \text{time(loop)} \leq 33 \text{ ms} \)
  - Real-time computing is a programming problem
    - (“Just buy a faster processor!”)
- **Multi-programmed system** — real-time performance \( \text{iff} \ldots ??? \)
  - Real-time computing is an operating systems problem

---

**Course Overview**

- OS Structure, Processes and Process Management
- Threads and concurrent programming
  - Thread coordination, mutual exclusion, monitors
  - Deadlocks
- CPU scheduling
- Memory management
- Secondary storage management & file systems
- Distributed systems & networking