History of Computer Systems

- First generation 1945 – 1955
  - vacuum tubes and plug boards (no OS)
- Second generation 1955 – 1965
  - transistors, batch systems
- Third generation 1965 – 1980
  - ICs and multiprogramming
- Fourth generation 1980 – present
  - personal computers, hand-held devices, sensors

First Generation 1945-1955 (no OS)

ENIAC (from Wikipedia)
History of Operating Systems
(1955-65)

Early batch system
☐ Single user
☐ Secure
☐ Programmer/User as the operator
☐ But, low CPU utilization-slow mechanical I/O devices

History of Operating Systems
(1965-80)

☐ Multiprogramming system
  - Three jobs in memory - 3rd generation
  - Spooling - use disk as a very large buffer for input/output devices
  - Timesharing - quick response time
History of Operating Systems (1980-present)

- Mainframe operating systems
- Server operating systems
- Multiprocessor operating systems
- Personal computer operating systems
- Real-time operating systems
- Embedded operating systems
- Smart card operating systems

A Basic (Single CPU) Computer System
Typical PC (Intel-based) Computer Structure

Typical Memory-Storage Structure

<table>
<thead>
<tr>
<th>Typical access time</th>
<th>Typical capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 nsec</td>
<td>&lt;1 KB</td>
</tr>
<tr>
<td>2 nsec</td>
<td>1 MB</td>
</tr>
<tr>
<td>10 nsec</td>
<td>64-512 MB</td>
</tr>
<tr>
<td>10 msec</td>
<td>5-50 GB</td>
</tr>
<tr>
<td>100 sec</td>
<td>20-100 GB</td>
</tr>
</tbody>
</table>

☐ When you programming, have you thought about

■ Register?, Disk or Tape?
A Peek Into Unix

<table>
<thead>
<tr>
<th>Application</th>
<th>User space/level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Libraries</td>
<td>Kernel space/level</td>
</tr>
<tr>
<td>Portable OS Layer</td>
<td></td>
</tr>
<tr>
<td>Machine-dependent layer</td>
<td></td>
</tr>
</tbody>
</table>
Unix: Application

Application
(E.g., emacs)

Libraries

Portable OS Layer

Machine-dependent layer

Written by programmer
Compiled by programmer
Uses function calls

Unix: Libraries

Application

Libraries (e.g., stdio.h)

Portable OS Layer

Machine-dependent layer

Provided, pre-compiled
Defined in headers
Input to linker (compiler)
Invoked like functions
May be “resolved” when program is loaded
Typical Unix OS Structure

Application
Libraries
Portable OS Layer
Machine-dependent layer

system calls (read, open..)
All “high-level” code

Bootstrap
System initialization
Interrupt and exception
I/O device driver
Memory management
Kernel/user mode switching
Processor management
Discussion

- What will future Operating Systems look like?
  - 20-30 years from now
  - What are the problems for current OSes?
  - How can future OSes fix them?
  - What features will future OSes have?
  - What are the criteria to evaluate the OSes?

Why Kernel Mode?

- Services that need to be provided at kernel level
  - System calls: file open, close, read and write
  - Control the CPU so that users won’t stuck by running
  - Protection:
    - Keep user programs from crashing OS
    - Keep user programs from crashing each other

How?
How to Provide Kernel Mode?

- **CPU mode bit** added to computer hardware to indicate the current CPU mode: 0 (=kernel) or 1 (=user).
- When an interrupt occurs, CPU hardware switches to the kernel mode.
- Switching to user mode (from kernel mode) done by setting CPU mode bit (by an instruction).

Exception/Interrupt/Fault

```
  kernel
     ↓
  Set user mode
     ↓
  user
```

*Privileged instructions* can be executed only in kernel mode.

---

Three Interrupt Classes

- **Interrupts caused by hardware failures**
  - Power outage
  - Memory parity error
- **Interrupts caused by external events:**
  - Reset
  - I/O devices
- **Interrupts caused by executed instructions**
  - Exceptions
  - System calls