An operating system is the interface between the user and the hardware. It manages system resources, provides common services, and enables users to interact with the computer system.

- OS reads to changes in hardware, and can motivate changes.
- History lesson in change.

Announcements

Last Class: Introduction to Operating Systems
work together on distributed hardware

5. Distributed systems & networks: allow a group of workstations to

4. Files: OS coordinates how disk space is used to store multiple files

3. Memory management: OS coordinates allocation of memory and

2. I/O devices: let the CPU work while a slow I/O device is working

active concurrency

● Threads (unit of OS control) - one thread on the CPU at a time, but many threads
  ● Several users work at the same time as if each has a private machine
  ● Multiple programs, etc.

1. Concurrency: Doing many things simultaneously (I/O, processing,

Modern Operating System Functionality

Architectural support can greatly simplify or complicate the OS.

What the OS can do is dictated in part by the architecture.

Basic Architecture Reminder

Basic OS Functionality

Today: OS and Computer Architecture
peripherals

- System bus: communication medium between CPU, memory, and
- Memory: RAM containing data and programs used by the CPU
- I/O devices: terminal, disks, video board, printer, etc.
- CPU: the processor that performs the actual computation

**Generic Computer Architecture**

**Summary of Operating System Principles**

- **Design tradeoffs change with technology**
- **OS as History Teacher:** Learning from past to predict the future, i.e., OS
- **OS as Complex System:** Keeping OS design and implementation as simple as possible is the key to getting the OS to work.
- **OS as Communication:** Resources efficiently and fairly, and providing secure and safe communication
- **OS as Government:** Protecting users from each other, allocating memory and CPU
- **OS as Illustrator:** Providing the illusion of a dedicated machine with infinite
Protocol instructions can only be executed in kernel mode.
A status bit in a protocol register indicates the mode.

The hardware must support at least kernel and user mode.

In kernel mode, the OS can do all these things:
- halt the machine
- disable and enable interrupts
- set the mode bits that determine user or kernel mode
- etc.
- use instructions that manipulate the state of memory (page table pointers, TLB load,
  "dirty"

may not

and processors, some instructions are restricted to use only by the OS. Users

Kerne/; User mode: To protect the system from aberrant users

<table>
<thead>
<tr>
<th>Protected Services</th>
<th>OS Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>Translation Lookaside Buffers</td>
<td>Virtual Memory</td>
</tr>
<tr>
<td>Atomic Instructions</td>
<td>Synchronization</td>
</tr>
<tr>
<td>Timer</td>
<td>Scheduling, error recovery, billing</td>
</tr>
<tr>
<td>Interrupts or Memory-Mapping</td>
<td>I/O</td>
</tr>
<tr>
<td>Trap Instructions and Trap Vectors</td>
<td>System calls</td>
</tr>
<tr>
<td>Interrupt Vectors</td>
<td>Interrupts</td>
</tr>
<tr>
<td>Base and Limit Registers</td>
<td>Protection</td>
</tr>
<tr>
<td>Protocol Instructions</td>
<td>Protocol</td>
</tr>
<tr>
<td>Kernel/User mode</td>
<td>OS Service</td>
</tr>
</tbody>
</table>
ensuring it fails between the base and limit register values.

- The CPU checks each user reference (instruction and data addresses),
- Base and limit registers are loaded by the OS before starting a program.
- The simplest technique is to use base and limit registers.
- Protect the OS from user programs.
- Protect user programs from each other, and
- Architecture must provide support so that the OS can

Memory Protection

Crossing Protection Boundaries

System call: OS procedure that executes privileged instructions (e.g.,

1/0)

Trap Handler
System Service Routine
User Programs
Program C
Program B
Program A
Trap to Kernel Mode

Kernel Mode
User Mode

Base Register
Limit Register

Process

Trap Handler
Program B
Program A
Trap to Kernel Mode

Process

Trap to Kernel Mode
Traps are a performance optimization. A less efficient solution is to insert extra instructions into the code everywhere a special condition could arise.

- Modern OS use virtual memory traps for many functions: debugging, distributed VM, garbage collection, copy-on-write, etc.

<table>
<thead>
<tr>
<th>System Call</th>
<th>Trap Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00123010</td>
<td>3:</td>
</tr>
<tr>
<td>0x000430</td>
<td>2:</td>
</tr>
<tr>
<td>0x000100</td>
<td>1:</td>
</tr>
<tr>
<td>0x000000</td>
<td>0:</td>
</tr>
</tbody>
</table>

On detecting a trap, the hardware
- example: page fault, write to a read-only page, overflow, systems call
- Transfers control to appropriate trap handler (OS routine)
- Saves the state of the process (PC, stack, etc.)
- On completion, the OS resumes execution of the process
- Starts to execute at the address
- Then jumps to the address given in the vector, and
- The CPU indexes the memory-mapped trap vector with the trap number.

**Traps:** Special conditions detected by the architecture

**Traps:**
writing the data directly into memory.

Access to the device then becomes almost as fast and convenient as

- video controller.
- device manager in that memory (e.g., all the bits for a video frame for a
  PCIe (no virtual memory), reserve a part of the memory and put the
  I/O code and data into memory)

Enables direct access to I/O controller (vs. being required to move the

| Memory-Mapped I/O |

Interrupt

- CPU stops whatever it was doing and the OS processes the I/O device's
  - CPU issues commands to I/O devices and continues
  - Each I/O device has a little processor inside it that enables it to run

| I/O Control |
Interrupt Vector:

- At each timer interrupt, the CPU chooses a new process to execute.
- Say every 100 microseconds.
- CPU protocol from being hogged using timer interrupts that occur at
  - Accounting and billing
  - Time of Day

Timer & Atomic Instructions

1. Save critical CPU state (hardware state).
2. Disable interrupts.
3. Save state that interrupt handler will modify (software state).
4. Invoke interrupt handler using the in-memory interrupt vector.
5. Restore software state.
6. Enable interrupts.
7. Restore hardware state, and continue execution of interrupted process.

CPU takes an interrupt.

Device puts an interrupt signal on the bus when it is finished.

- Asynchronous with the main CPU.
- Device controller has its own small processor which executes

Interrupt based asynchronous 1/0
translating lookaside buffer to speed the lookup. In order for pieces of the program to be located and loaded without memory and which pieces are on disk. The OS must keep track of which pieces are in which parts of physical instead, pieces of the program are loaded as they are needed. Program in memory at once. Virtual memory allows users to run programs without loading the entire

**Virtual Memory**

2. A special instruction that executes atomically (e.g., test&set). 1. Architecture mechanism to disable interrupts before sequence, execute instructions (e.g., read-modify-write) execute atomically. Two solutions: Architecture must provide a guarantee that short sequences of OS must be able to synchronize cooperating concurrent processes. Interrupts interfere with executing processes.

**Synchronization**
The OS and hardware combine to provide many useful and important features. Additionally, the OS provides an interface to the architecture, but also requires some user-architecture book on hand.

Summary