Today: OS and Computer Architecture

• Architectural support can greatly simplify or complicate the OS.
• What the OS can do is dictated in part by the architecture.
• Basic Architecture Reminders
• Basic OS Functionality

An operating system is the interface between the user and the architecture.

— OS reacts to changes in hardware, and can motivate changes.
— History lesson in change.

Hardware

Physical Machine Interface

Virtual Machine Interface

User Applications

Last Class: Introduction to Operating Systems
OS as inverted: providing the illusion of a dedicated machine with infinite memory and CPU

OS as interface: 

OS as government: protecting users from each other, allocating resources efficiently and fairly, and providing security and safe

OS as complex system: keeping OS design and implementation as simple as possible is the key to getting the OS to work.

OS as history teacher: Learning from past to predict the future, i.e., OS

5. Distributed systems & networks: allow a group of workstations to work together on distributed hardware

4. Files: OS coordinates how disk space is used to store multiple files moving data between disk and main memory.

3. Memory management: OS coordinates allocation of memory and several users work at the same time on each has a private machine

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

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

Modern Operating System Functionality

Summary of Operating System Principles
Translation lookaside buffers
Atomic instructions
Interrupts or Memory-Mapping?
Trap instructions and trap vectors
Kernel/User mode

interrupts
CPU
System bus
Disk controller
Printer/tape drive
Protection

Virtual memory
Scheduling, error recovery, billing
I/O
System calls

Hardware Support

OS Service

Architectural Features Motivated by OS Services

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

Generic Computer Architecture
The architecture must also provide a way to return to user mode when finished.

- The architecture must permit the OS to verify the caller's parameters.
- The handler saves caller's state (PC, mode bit) so it can restore control to the user process.
- The trap handler passes the parameter to the system call to jump to the appropriate function.
- Causes a trap, which vectors (jumps) to the trap handler in the OS kernel.

1/0 (System call: OS procedure that executes privileged instructions (e.g.,

**Crossing Protection Boundaries**

- Protected instructions can only be executed in kernel mode.
- A status bit in a protected processor register indicates the mode.

The hardware must support at least kernel and user mode.

- But 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
  - use instructions that manipulate the state of memory (Page table pointers, TLB load)
  - address I/O directly

May not

**Kernel mode vs. User mode**: To protect the system from aberrant users...
On completion, the OS resumes execution of the process.

- Starts to execute at that address.
- then jumps to the address given in the vector, and
- Transfers control to appropriate trap handler (OS routine).
- Saves the state of the process (PC, stack, etc.)

On detecting a trap, the hardware...

- Example: page fault, write to a read-only file, overflow, systems call

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

---

**Memory Protection**

Ensuring it falls 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
- Archıtecture must provide support so that the OS can...
Interrupt
- CPU stops whatever it was doing and the OS processes the I/O device’s
  command.
- When the I/O device completes the command, it issues an interrupt
  autonomously.
- Each I/O device has a little processor inside it that enables it to run

**1/O Control**

Extra instructions into the code everywhere a special condition could arise.
- Traps are a performance optimization. A less efficient solution is to insert
  distributed VM, garbage collection, copy-on-write, etc.
- Modern OS use Virtual Memory trap for many functions: debugging,

<table>
<thead>
<tr>
<th>System Call</th>
<th>Trap Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>3:</td>
<td>0x00123010</td>
</tr>
<tr>
<td>2:</td>
<td>0x00100480</td>
</tr>
<tr>
<td>I:</td>
<td>0x00000100</td>
</tr>
<tr>
<td>0:</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>
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 I/O

Memory-Mapped I/O

01. Write the data directly into memory.

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

Video controller.

Device memory in that memory (e.g., all the bits for a video frame for a

PC's (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
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
     interrupts interfere with executing processes.
   - OS must be able to synchronize cooperating, concurrent processes.

**Synchronization**

### Disk I

<table>
<thead>
<tr>
<th>Disk</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>0:</td>
<td>0x2168b6810</td>
</tr>
<tr>
<td>1:</td>
<td>0x21600000</td>
</tr>
<tr>
<td>2:</td>
<td>0x21600b00</td>
</tr>
<tr>
<td>3:</td>
<td>0x21600000</td>
</tr>
</tbody>
</table>

**Interrupt Vector:**

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

**Timer**

**Timer & Atomic Instructions**
Features:
The OS and hardware combine to provide many useful and important additional functionality from the architecture.

OS provides an interface to the architecture, but also requires some additional software.

Keep your architecture book on hand.

Summary:

.In order for pieces of the program to be located and loaded without causing a major disruption to the program, the hardware provides a translation lookaside buffer to speed the lookup.
.

The OS must keep track of which pieces are in which parts of physical memory and which pieces are on disk.
.

Instead, pieces of the program are loaded as they are needed.

Virtual memory allows users to run programs without loading the entire program in memory at once.

Virtual Memory: