OS Concepts 1.1: What Operating Systems Do

- **What is an OS?:** OS's vary widely in design and in function, but basically an OS is the software that sits between application programs and computer hardware. It provides an environment in which application programs are executed, by allocating physical resources like CPU, memory, and I/O devices.

OS Concepts 1.2: Computer-System Organization

1.2.1: Interrupts

- **Device Controller:** The I/O managing processor within a device.
- **Device Driver:** A component in the OS that understands how to communicate with its respective device controller and manages I/O to those devices.
- **Interrupts:** Interrupts are used in OS's to handle asynchronous events originating from outside the processor (interrupts originating from within the processor are called exceptions). Device controllers and hardware faults raise interrupts. Because interrupts are used so heavily for time-sensitive processing, efficient interrupt handling is necessary for good system performance.
- **Interrupt Vector:** A table of pointers stored in low memory that holds the addresses of the interrupt service routines.
- **Basic Interrupt Implementation:** The CPU hardware has a wire called the interrupt-request line that the CPU senses after executing every instruction. When the CPU detects a device controller has asserted a signal on the wire, it reads the interrupt number and jumps to the respective interrupt-handler routine by using the interrupt number as an index into the interrupt vector. It then saves the current state of whatever was interrupted, and starts execution of the interrupt-handler routine. Once the handler is finished executing, it performs a state restore and returns the CPU to the execution state prior to the interrupt.
- **Interrupt Terminology:** We say that the device controller **raises** an interrupt by asserting a signal on the interrupt request line, the CPU **catches** the interrupt and **dispatches** it to the interrupt handler, and the handler **clears** the interrupt by servicing the device.
- **More Sophisticated Interrupt Implementation:** We need the ability for the following:
  - Defer interrupt handling during critical processing
  - Efficiently dispatch to the correct interrupt-handler for a device
  - Multilevel interrupts, so that the OS can distinguish between high and low priority interrupts and respond with the appropriate level of urgency

To do this, most CPU's have two interrupt request lines: one is the nonmaskable interrupt, which is used for events such as unrecoverable memory errors, and the second is the maskable interrupt, which the CPU can turn off before the execution of critical instruction sequences that must not be interrupted. Device controllers use the maskable interrupt to request service.

1.2.2: Storage Structure

- **Firmware:** Software stored in ROM or EEPROM for booting the system and managing low level hardware.

1.2.3: I/O Structure

- **Direct Memory Access (DMA):** Interrupt-driven I/O as described in section 1.2.1 is fine for moving small amounts of data but can produce high overhead when used for bulk data movement, like when moving data to and from nonvolatile memory. DMA is used to avoid this overhead. The device controller sets up buffers, pointers, and counters for its I/O device, and transfers entire blocks of data to or from the device and main memory, with no intervention by the CPU. Only one interrupt is generated per block, to tell the device driver that the operation has completed, rather than the one interrupt per byte generated for low-speed devices. The CPU is able to perform other work while the device controller is performing these operations.

OS Concepts 1.3: Computer-System Architecture

1.3.1: Single-Processor Systems

- **CPU:** The hardware that executes instructions.
- **Processor:** A physical chip that contains one or more CPU's.
- **CPU Core:** The core is the component of the CPU that executes instructions and contains registers for storing data locally.
- **Single-Processor System:** A computer system with a single processor containing one CPU with a single processing core. These systems often also have other special-purpose processors as well, such as disk, keyboard, and graphics controllers. These special-purpose processors run a limited instruction set and do not run processes; their use is incredibly common and does not turn a single-processor system into a multiprocessor system.

1.3.2: Multiprocessor Systems

- **Multiprocessor Systems:** A computer system containing multiple processors. Traditionally contains two or more processors, each with a single-core CPU.

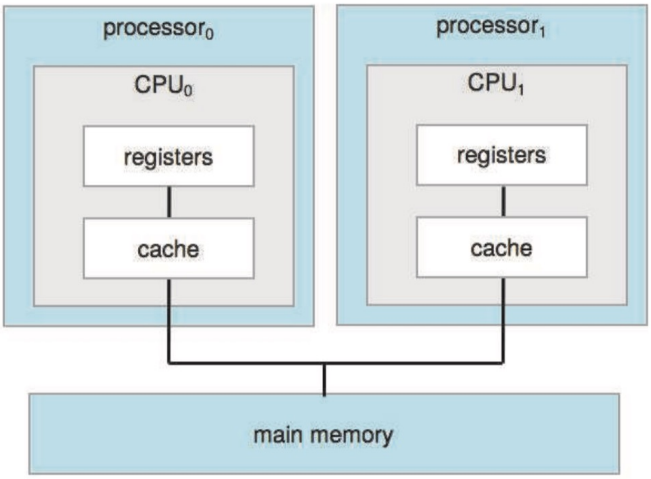

Figure 1: Symmetric *multiprocessing* architecture

- **Multiprocessor Advantages (Increased Throughput):** Primary advantages of multiprocessor systems is increased throughput. The speed-up ratio with N processors is not N, however; it is less than N because there is overhead incurred and contention for shared resources when dealing with multiple processors.
- **Multicore Systems:** A computer system containing multiple cores on the same processor chip. Such systems can be more efficient than multiple chips with single cores because on-chip communication is faster than between-chip communication. Additionally, one chip with multiple cores uses significantly less power than multiple single-core chips, an issue especially important for mobile devices.
- **Multiprocessor Bottleneck:** Adding additional CPU's to a multiprocessor system increases computing power, but does not scale very well. Once too many CPU's are added, contention for the system bus becomes a bottleneck and performance begins to degrade.
- **Non-uniform Memory Access (NUMA):** To avoid bottleneck performance degradation arising from system bus contention, we can provide each CPU with its own local memory that is accessed via a small and fast local bus. The CPU's are connected by a shared system interconnect, so that all CPU's share one physical address space. The advantage is that when a CPU accesses its local memory, not only is it fast, but there is also no contention over the system interconnect. Thus, NUMA systems can scale more effectively as more processors are added.
- **NUMA Drawbacks (Increased Latency):** A potential drawback is increased latency when a CPU must access remote memory across the system interconnect (accessing the local memory of another CPU). OS's can minimize this NUMA penalty through careful CPU scheduling and memory management.

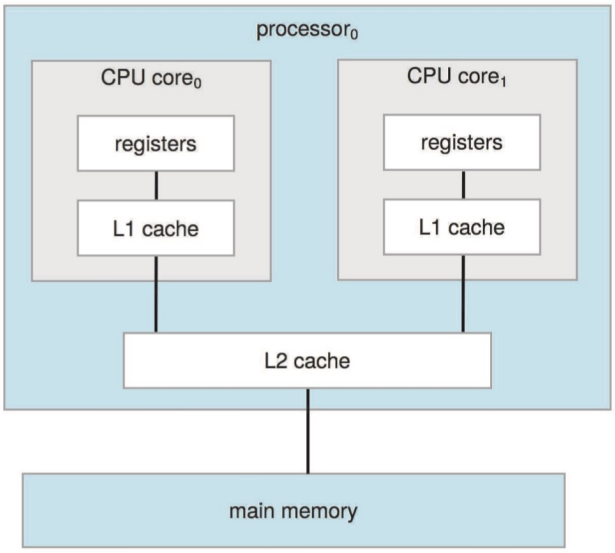

Figure 2: *Multicore* architecture

OS Concepts 1.4: Operating-System Operations

- **Bootstrap Program:** The initial program that is run when a computer starts running for the first time. Typically a very simple program and stored in firmware. The program must know how to load the OS kernel into memory and start executing it.
- **System Daemons:** Services provided outside of the kernel that are loaded into memory at boot time, which run the entire time the kernel is running.

1.4.1: Multiprogramming and Multitasking

- **Multiprogramming:** Increase CPU utilization by organizing programs so that the CPU always has one to execute. Execute a process until it needs to wait on some task like I/O, then switch to another process. Keep switching between processes such that the CPU is never idle.
- **Process:** In a multiprogrammed system, a program in execution is termed a process.
- **Multitasking:** The logical extension of multiprogramming. In multitasking systems, the CPU executes multiple processes by switching among them incredibly frequently. This is how interactive I/O like keyboard input works: rather than letting the CPU sit idle in the time between the keystrokes of the user, the OS will rapidly switch to another process in the meantime.

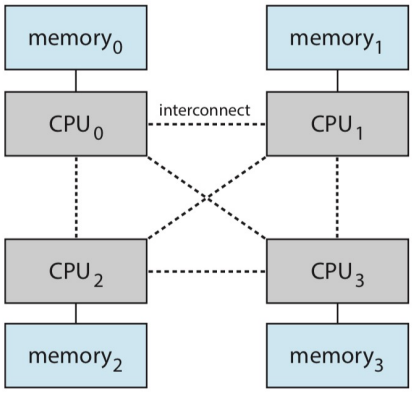

Figure 3: Numa architecture

1.4.2: Dual-Mode and Multimode Operation

- **Modes of Execution:** A properly designed OS must ensure that an incorrect (or malicious) program cannot cause other programs - or the OS itself - to execute incorrectly. To do this, we need to distinguish between the execution of OS code and user-defined code. Most computer systems provide hardware support that allows differentiation among various modes of execution.

- **User Mode and Kernel Mode:** At the very least, we need two separate modes of operation: user mode and kernel mode (also called supervisor mode, system mode, or privileged mode). A bit, called the mode bit, is added to the hardware of the computer to indicate the current mode: kernel (0) or user (1). Whenever the OS gains control of the computer, it is in kernel mode. The system always switches back to user mode before passing control to a user program. The concept of modes can be extended beyond two modes (e.g. the four protection rings in Intel processors).

- **Privileged Instructions:** Some machine instructions that may cause can only be executed in kernel mode. The instruction to switch to kernel mode is an example of a privileged instruction.

1.4.3: Timer

- **Timer:** We must ensure that the OS maintains control over the CPU: we allow user programs to execute, but they must eventually relinquish control to the OS (avoid situations like user program infinite loops or not calling system services and thus not returning control to the OS). To accomplish this, we can use a timer that is set to raise an interrupt after a specified amount of time. If the timer interrupts, control transfers automatically to the OS, which may treat the interrupt as a fatal error or may give the program more time. Instructions that modify the timer are clearly privileged.