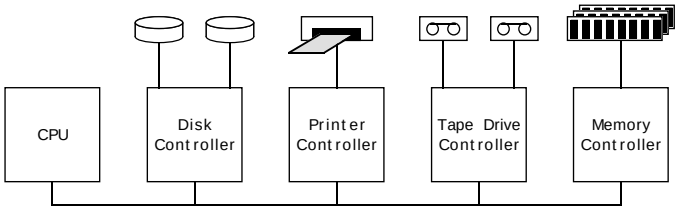
Input / Output (I/O)



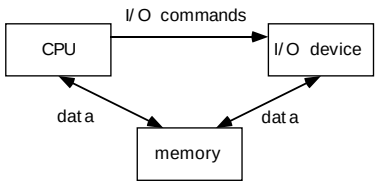
- CPU and device controllers all use a common bus for communication
- Software-polling synchronous I/O
  - CPU starts an I/O operation, and continuously *polls* (checks) that device until the I/O operation finishes
  - Device controller contains registers for communication with that device
    - Input register, output register — for data
    - Control register — to tell it what to do
    - Status register — to see what it's done
  - Why not connect all I/O devices directly to CPU? Why not… memory…?

Input / Output (I/O) (cont.)

- Terminology
  - *Synchronous* I/O — CPU execution waits while I/O proceeds
  - *Asynchronous* I/O — I/O proceeds concurrently with CPU execution
- Interrupt-based asynchronous I/O
  - Device controller has its own processor, and executes asynchronously with CPU
    - Device controller puts an interrupt signal on the bus when it needs CPU's attention
  - When CPU receives an interrupt:
    1. It saves the CPU state and invokes the appropriate interrupt handler using the *interrupt vector* (addresses of OS routines to handle various events)
    2. Handler must save and restore software state (e.g., registers it will modify)
    3. CPU restores CPU state

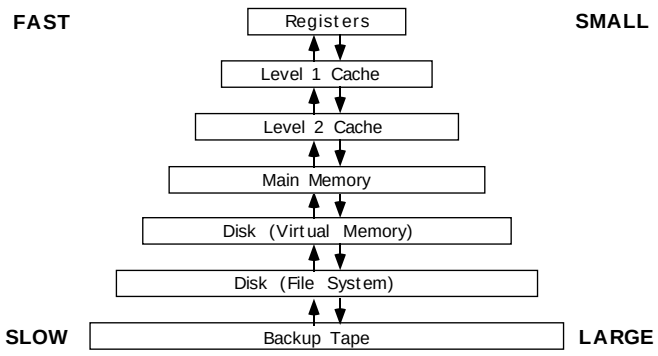
Input / Output (I/O) (cont.)

- Memory-mapped I/O
  - Uses direct memory access (DMA) — I/O device can transfer block of data to / from memory without going through CPU



  - OS allocates buffer in memory, tells I/O device to use that buffer
  - I/O device operates asynchronously with CPU, interrupts CPU when finished
  - Used for most high-speed I/O devices (e.g., disks, communication interfaces)

Storage Structures



- At a given level, memory may not be as big or as fast as you'd like it be
  - Tradeoffs between size and speed
- *Principle of Locality of Reference* leads to *caching*
  - When info is needed, look on this level
  - If it's not on this level, get it from the next slower level, and save a copy here in case it's needed again sometime soon

Magnetic Disks

- Provide secondary storage for system (after main memory)
- Technology
  - Covered with magnetic material
  - Read / write head "floats" just above surface of disk
  - Hierarchically organized as platters, tracks, sectors (blocks)
- Devices
  - Hard (moving-head) disk — one or more platters, head moves across tracks
  - Floppy disk — disk covered with hard surface, read / write head sits on disk, slower, smaller, removable, rugged
  - CDROM — uses laser, read-only, high-density
    - Optical —read / write

Protection

- Multiprogramming and timesharing require that the memory and I/O of the OS and user processes be **protected** against each other
  - Note that most PCs do not support this kind of protection
- Provide protection via two modes of CPU execution: *user mode* and *kernel mode*
  - In kernel / privileged / supervisor mode, *privileged instructions* can:
    - Access I/O devices, control interrupts
    - Manipulate the state of the memory (page table, TLB, etc.)
    - Halt the machine
    - Change the mode
  - Requires architectural support:
    - Mode bit in a protected register
    - Privileged instructions, which can only be executed in kernel mode

I/O Protection

- To prevent illegal I/O, or simultaneous I/O requests from multiple processes, perform all I/O via privileged instructions
  - User programs must make a *system call* to the OS to perform I/O
- When user process makes a system call:
  - A *trap* (software-generated interrupt) occurs, which causes:
    - The appropriate trap handler to be invoked using the trap vector
    - Kernel mode to be set
  - Trap handler:
    - Saves state
    - Performs requested I/O (if appropriate)
    - Restores state, sets user mode, and returns to calling program

Memory Protection

- Must protect OS's memory from user programs (can't overwrite, can't access)
  - Must protect memory of one process from another process
  - Must not protect memory of user process from OS
- Simplest and most common technique:
  - *Base register* —smallest legal address
  - *Limit register* — size of address range
  - Base and limit registers are loaded by OS before running a particular process
  - CPU checks each address (instruction & data) generated in user mode
    - Any attempt to access memory outside the legal range results in a trap to the OS
- Additional hardware support is provided for virtual memory

CPU Protection

- Use a timer to prevent CPU from being hogged by one process (either maliciously, or due to an infinite loop)
  - Set timer to interrupt OS after a specified period (small fraction of a second)
  - When interrupt occurs, control transfers to OS, which decides which process to execute for next time interval (maybe the same process, maybe another one)
- Also use timer to implement time sharing
  - At end of each time interval, OS switches to another process
  - *Context switch* = save state of that process, update Process Control Block for each of the two processes, restore state of next process

Computer Architecture & OS

- Need for OS services often drives inclusion of architectural features in CPU:

OS Service	Hardware Support
I/O	interrupts memory-mapped I/O caching
Data access	memory hierarchies file systems
Protection	system calls kernel & user mode privileged instructions interrupts & traps base & limit registers
Scheduling & Error recovery	timers