# Announcements

- Program #0
  - its due next Friday

- Reading
  - Chapter 2
  - Chapter 3 (for Tuesday)

System Calls

- **Provide the interface between application programs and the kernel**

- **Are like procedure calls**
  - take parameters
  - calling routine waits for response

- **Permit application programs to access protected resources**

register r0

load r0, x
system call 10

Code for
sys call 10

User Program

Operating System
(kernel)
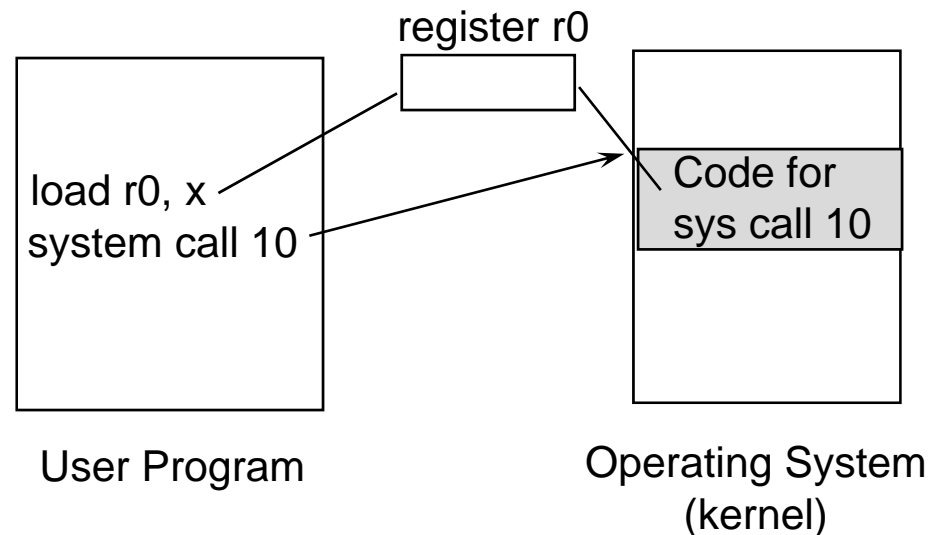
System Call Mechanism

- Use numbers to indicate what call is made
- Parameters are passed in registers or on the stack
- Why do we use indirection of system call numbers rather than directly calling a kernel subroutine?
  - provides protection since the only routines available are those that are export
  - permits changing the size and location of system call implementations without having to re-link application programs
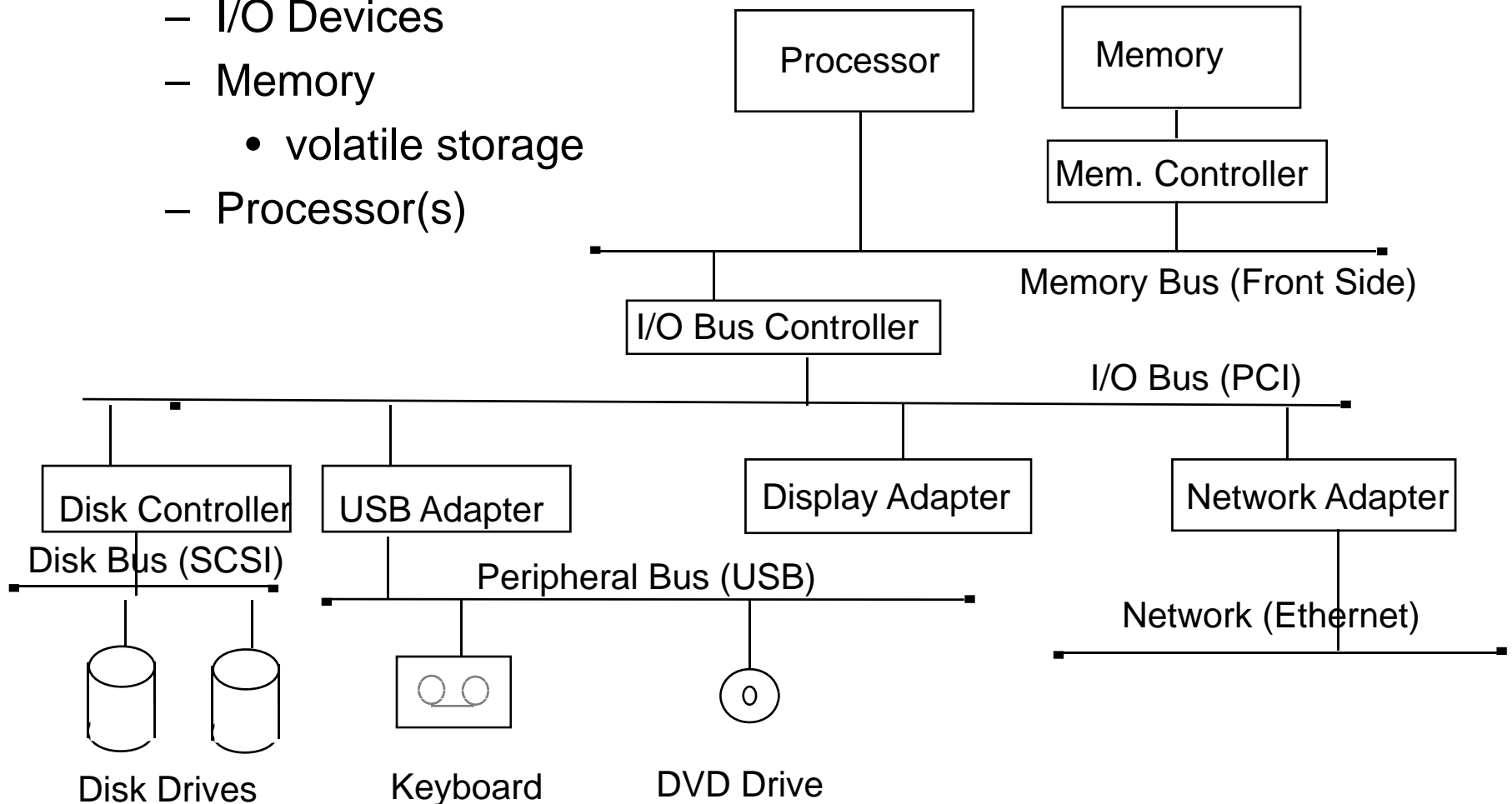
Types of System Calls

- **File Related**
  - open, create
  - read, write
  - close, delete
  - get or set file attributes
- **Information**
  - get time
  - set system data (OS parameters)
  - get process information (id, time used)
- **Communication**
  - establish a connection
  - send, receive messages
  - terminate a connection
- **Process control**
  - create/terminate a process (including self)

Computer Systems

- **Computers have many different devices**
  - I/O Devices
  - Memory
    - volatile storage
  - Processor(s)

| Processor | Memory |

Mem. Controller

Memory Bus (Front Side)

I/O Bus Controller

I/O Bus (PCI)

| Disk Controller | USB Adapter | Display Adapter | Network Adapter |

Disk Bus (SCSI)

Peripheral Bus (USB)

Network (Ethernet)

Disk Drives   Keyboard   DVD Drive

I/O Systems

- **Many different types of devices**
  - disks
  - networks
  - displays
  - mouse
  - keyboard
  - tapes
- **Each have a different expectation for performance**
  - bandwidth
    - rate at which data can be moved
  - latency
    - time from request to first data back

Different Requirements lead to Multiple Buses

- Processor Bus (on chip)
    - Many Gigabytes/sec

- Memory Bus (on processor board)
    - Up to 100 Gigabyte per second

- I/O Bus (PCI & PCI-E)
    - ~1s gigabytes per second
    - buses are more complex than we saw in class
        - show PCI spec.

- Device Bus (SCSI, USB)
    - tens of megabytes per second

Issues In Busses

- Performance
  - increase the data bus width
  - have separate address and data busses
  - block transfers
    - move multiple words in a single request
- Who controls the bus?
  - one or more bus masters
    - a bus master is a device that can initiate a bus request
  - need to arbitrate who is the bus master
    - assign priority to different devices
    - use a protocol to select the highest priority item
      - daisy chained
      - central control

Disks

- Several types:
  - Hard Disks - rigid surface with magnetic coating
  - Floppy disks - flexible surface with magnetic coating
  - Optical (CDs and DVDs) - read only, write once, multi-write
  - Solid State (Flash) – fast seek times, limited number of writes
- Hard Disk Drives:
  - collection of platters
  - platters contain concentric rings called tracks
  - tracks are divided into fixed sized units called sectors
  - a cylinder is a collection of all tracks equal distant from the center of disk
  - Current Performance:
    - capacity: gigabytes to terabytes
    - throughput: sustained < 20 megabytes/sec
    - latency: mili-seconds

I/O Interfaces

- Need to adapt Devices to CPU speeds
- Moving the data
  - Programmed I/O
    - Special instructions for I/O
  - Mapped I/O
    - looks like memory only slower
  - DMA (direct memory access)
    - device controller can write to memory
    - processor is not required to be involved
    - can grab bus bandwidth which can slow the processor down

I/O Interrupts

- **Interrupt defined**
  - indication of an event
  - can be caused by hardware devices
    - indicates data present or hardware free
  - can be caused by software
    - system call (or trap)
  - CPU stops what it is doing and executes a handler function
    - saves state about what was happening
    - returns where it left off when the interrupt is done
- **Need to know what device interrupted**
  - could ask each device (slow!)
  - instead use an interrupt vector
    - array of pointers to functions to handle a specific interrupt