Computer Science 432/563
Operating Systems

The College of Saint Rose
Spring 2016

Topic Notes: Input/Output

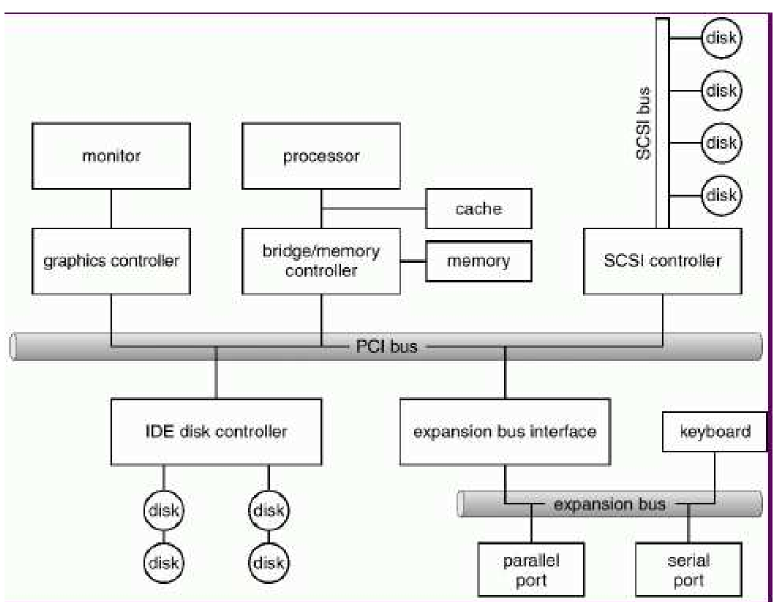
One of the primary functions of an OS is I/O management. We have looked at disk structures and file system management, but there is more that the OS needs to be concerned with.

- wide variety of devices to be handled
- provide convenient interface
- manage wide variety of device speeds – range from a few bytes per second from a keyboard to several gigabytes per second on a fast network interface
- organization (files/file system from a disk)
- error handling
- deliver I/O to the correct process
- protection and security

We will not spend much time in class going over the background – just highlights.

There are two common ways to connect an I/O device:

- port (serial, parallel) ex: mouse, modem, printer, joystick
- bus (SCSI, USB, PCI) ex: disks, tapes



Note the presence of *device controllers* – hardware that connects directly to the main bus on behalf of actual devices.

The OS provides I/O instructions to control devices. Control may be by

- *direct* I/O instructions – here, we read or write chunks of data to or from a port or bus with special machine instructions.
- *memory-mapped* I/O – interact with a device by reading/writing memory in special locations assigned to a device. This has the advantage that we can use regular machine instructions to read and write the portion of memory reserved for communication with each device.

Accessing devices

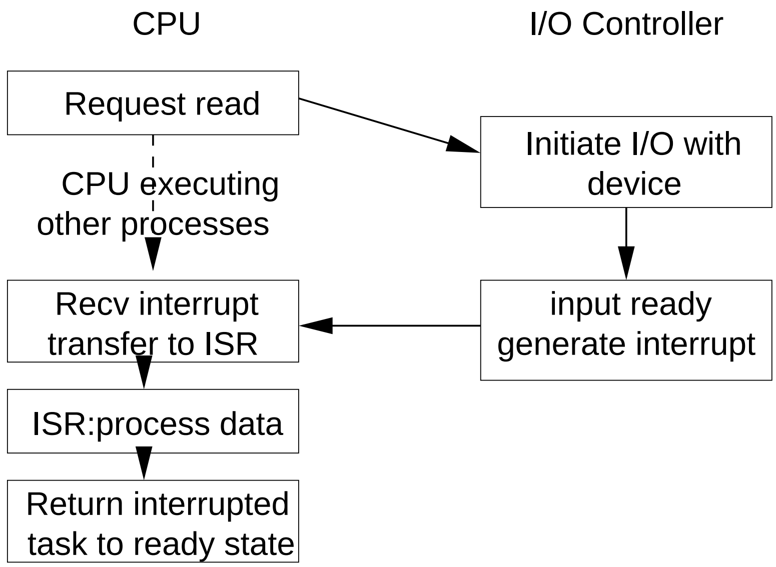
Either way, how do we know when the device needs the attention of the CPU? Remember that "I/O Wait" means the process is not in the ready queue or on the CPU, ideally.

- *Polling* – When OS has control of the CPU, it queries the device. This could end up being a busy-wait if the data will not remain available for long, such as on a port.

  We saw an example of this with the Commodore 64 manual, where we could examine a memory location to get, for example, the state of a joystick controller.
- *Interrupts* –

  We have seen interrupts before – this is what makes a process leave the CPU when a time quantum expires.

  1. CPU Interrupt request line triggered by I/O device
  2. CPU switched to interrupt handler
  3. Maskable to ignore or delay some interrupts (consider an interrupt being interrupted!)
  4. Interrupt vector to dispatch interrupt to correct handler (Interrupt Service Routine - ISR)



Interrupt service can be complicated on modern machines – consider pipelined execution (see text).

- *Direct Memory Access* – I/O controller gets data and puts the results directly *in memory* at a specified location

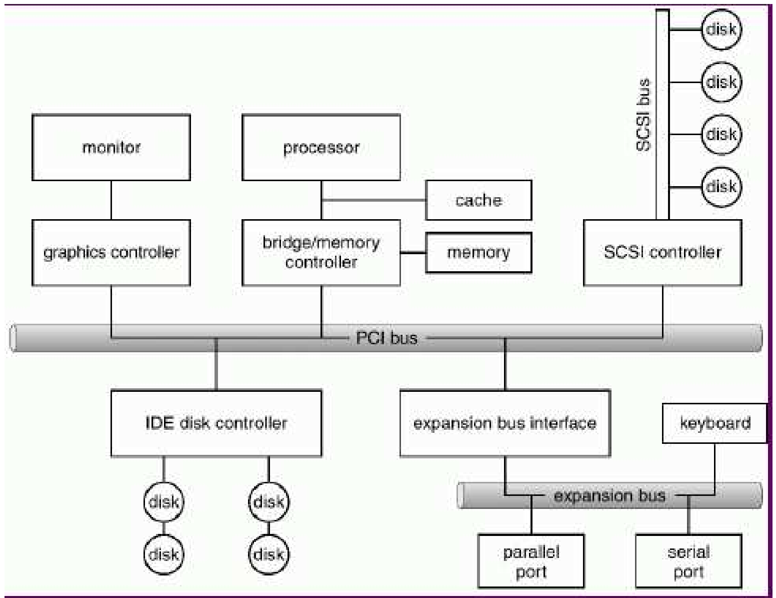
  This is used to avoid programmed I/O for large data movement, but requires a DMA-capable controller.

  Here, we bypass the CPU to transfer data directly between I/O device and memory. Hence, the CPU can be doing other things, and we avoid the need to copy data from a device buffer into user space.

Application I/O Interface

I/O system calls encapsulate device behaviors using a common interface for a wide variety of devices.

Device drivers hide differences among I/O controllers from the kernel. The same kernel routines can call functions in specific device drivers without worrying about the details of the device.



A device driver needs to be aware of the characteristics of a particular device.

- Character-stream or block
- Sequential or random-access
- Sharable or dedicated
- Speed of operation
- Read-write, read only, or write only

Block devices:

- "large" blocks of data read/written at once
- include disk drives
- `read`, `write`, `seek` operations
- Raw I/O (kernel) or file-system (user) access

Character devices:

- include keyboards, mice, serial ports, printers, modems
- `get`, `put` operations on individual characters

Network devices:

Similar to block and character, but different enough to be unique

- socket interface to separate network protocol from network operation
- other options: pipes, FIFOs, streams, queues, mailboxes
- need to deal with large amounts of data rapidly as well as short interactive traffic

---

Blocking vs. Nonblocking I/O

- *Blocking* – the requesting process suspended (removed from ready/run) until I/O completed
  - Easy to use and understand
  - Insufficient for some needs
- *Nonblocking* – I/O call returns as much as available
  - User interface, data copy (buffered I/O)
  - Implemented via multithreading
  - Returns quickly with count of bytes read or written
- *Asynchronous* – process runs (doing something else) while I/O executes
  - Difficult to use
  - I/O subsystem signals process when I/O completed
  - need to wait for the data explicitly
  - overlap I/O with other computation

Kernel I/O Subsystem

Beyond just the interface, the kernel manages devices to improve. efficiency

- *Scheduling*
  - some I/O request ordering via per-device queue
  - consider a series of disk access requests – what order to use? efficiency? priorities?
- *Buffering* – may be needed because of
  - device speed mismatch (disk to printer, modem to disk)

- device transfer size mismatch (gather network packets)

- *Caching* – fast memory holding copy of data
  - always just a copy
  - key to performance
  - has come up before and will come up again
- *Spooling* – hold output for a device
  - if device can serve only one request at a time
  - printing, maybe tape I/O
- *Device reservation* – provides exclusive access to a device
  - system calls for allocation and deallocation
  - deadlock avoidance/detection/recovery
- *Error Handling*
  - OS can recover from disk read, device unavailable, transient write failures (retry)
  - switch to another device, if possible
  - return failure code when I/O request fails
  - error logs
- *Bookkeeping and kernel data structures*
  - state info (open files, network connections, etc.)
  - complex data structures (i.e., Unix buffer cache)

Performance

I/O performance is a critical factor in overall system performance:

- reduce number of context switches
- reduce data copying
- reduce interrupts by using large transfers, smart controllers, polling
- use DMA
- balance CPU, memory, bus, and I/O performance for highest throughput