Chapter 13: I/O Systems

Monday, November 19, 2007
3:48 PM

- I/O subsystem separates the rest of the kernel from complexity of I/O devices.
- **Device driver** gives uniform interface to various I/O devices to the I/O subsystem.

- I/O Hardware
  - **Port** = a connection point. For examples, serial port, parallel port, USB port, etc.
  - **Bus** = a set of wires and a protocol that dictates how to send information over the wires.
  - **Controller** = circuits that operate a port, a bus, or a device.
    - Has one or more registers for data and control signals.
    - CPU communicates with controller by reading and writing these registers.
  - **Memory mapped I/O**
    - CPU executes I/O request by writing to memory address.
    - Vulnerable to accidental modification
  - I/O port registers
    - Status
    - Control
    - Data-in
    - Data-out

- Communicating with I/O devices
  - Polling (busy-waiting)
    - CPU repeatedly check status bits, and doing the righting when the status updates.
    - Wastes time if the wait is long.
  - Interrupt
    - CPU sense the **interrupt request line** after it finishes every instruction.
    - Control transfers to the **interrupt handler** routine, whose address is fixed in memory, when the interrupt request line is set high.
    - When communicate using interrupts, CPU issues command to the I/O device and goes to do other tasks. When the I/O device has something to tell the CPU, it fires an interrupt, and the CPU responds.
  - Direct memory access (DMA)
    - Handled by a special purpose circuit call the **DMA controller**.
    - The CPU can write DMA command, source, and destination of the data transfer to the DMA controller.
    - The DMA controller than controls the bus, transferring data from I/O device on the CPU's behalf.
    - After the transfer finishes, the DMA informs the CPU by an interrupt.
    - DMA controller's control of the bus can make CPU access to memory slower.

- Application I/O interface
  - **Block device** transfer data in the unit of blocks.
    - Examples: Hard disks.
    - read(), write(), seek()
  - **Character stream device** transfer data one character at a time.
    - Example: Keyboard
    - get() or put() one character.
  - **Socket**
    - For inter-process communication, especially across network.
    - Can be thought of as an interface to network adapters, thus an I/O interface.
    - read(), and write() still works with socket.
    - Most socket interface have send() and recv(), which is just the same as write() and read(), but users can specify more options, such as receiving from multiple clients.
  - **Clocks and timers**
    - Three main tasks
      - Give the current time.
      - Give the elapsed time.

        - Set a timer to fire an interrupt at some interval.
      - CPU don't have enough clocks to use for all the timing requests.
      - Virtual clocks
        - Kernel maintains a list of requests, sorted by time.
        - Kernel sets the timer to the first earliest time.
        - Once interrupted, kernel removes the earliest request from the list, and sets the timer to the next earliest time.
  - **Blocking and Nonblocking I/O**
    - Blocking
      - the execution of the application is suspended until I/O completes.
      - The application can do other things while waiting, so it's kind of wasteful.
      - However, blocking calls are easier to understand.
    - Nonblocking
      - Call returns immediately with whatever is available at the moment.
      - Application has to call nonblocking calls again to finish its job.
    - Asynchronous
      - Call returns immediately with nothing the calling application can use.
      - Once the I/O finishes, the I/O subsystem interrupts the application.
    - select() system call in Unix
      - Input
        - A list of I/O devices or sockets to watch for.
        - Condition that we're interested in for each device.
        - How long do we want to wait for the responses.
      - Output
        - The devices that are ready for us to do something.
      - Can use select to implement non-blocking I/O.
      - Nevertheless, has to call read() or write() later.

- I/O Subsystem
  - I/O scheduling
    - Determines a good order to execute I/O requests.
    - How to implement: Maintain a wait queue for each device.
  - Buffering
    - Store data while it is being transferred between devices.
    - Why?
      - Speed mismatch between producer and consumer.
        - Keyboard -> harddisk.
      - Different data transfer size.
        - Network packets -> memory
      - Copy semantics
        - A block in memory that is going to be written to disk should be the one at the time when the write command was issued, not the one when the write is actually taking place.
        - Can be implemented by copying the block that is going to be written to the kernel's buffer cache.
        - But can be implemented by copy-on-write page protection as well.
  - Caching
    - **Cache** = fast memory that holds a copy of data stored elsewhere.
    - A buffer may hold the only copy of the data. But the cache holds things that are already somewhere else.
  - Spooling
    - **Spool** = buffer that holds data for device that cannot accept interleaved data.
    - How to implement: Intercept all data to the printer. Create spool file for each job. Put jobs into a queue, and feed the jobs to the printer one by one.
  - Error hadling
    - I/O system call usually returns one-bit status indicating whether the I/O actually succeeded.
    - In Unix, I/O system calls return error number (errno).
  - I/O protection
    - Prevent users from performing illegal I/O.

        - All I/O instructions must be privileged instructions.
    - Prevent users from performing illegal memory mapped I/O.
        - This is the topic of memory management.
- Implementation
    - Unix: File system access to everything. Possible because of the user of virtual file system, which is OO concept in action.
    - Windows NT: Message-passing interface for I/O. Lots of overhead, but lots of flexibility as well.

- Performance
    - Context switch is an expensive process.
    - Optimization
        - Use **front-end processors** to reduce interrupt burden.
            - **terminal concentrator** multiplex traffic from remote terminals to one port on a large computer.
            - **I/O channels** = special purpose CPU that offloads I/O work from main CPU.
        - Reduce number of context switches.
        - Reduce number of times a data must be copied in memory.
        - Reduce the frequency of interrupts by using large transfers, smart controllers, or polling.
        - Increase concurrency. For example, use DMA.
        - Move processing primitive to hardware. Again, see DMA or I/O channel.

- Engineering concern: Where should I/O functionality be implemented?
    - Implement experimental I/O algorithm in software.
        - Allows for great flexibility.
        - Fault is contained.
    - Move algorithm that's worth it to kernel.
        - Improve performance.
        - Have to get it right. Kernel might fail because of your module.
        - Tanenbaum might not be happy about this.
    - If you really need performance, implement it in hardware. Or use an appliance.