Device Management

Basic Issues

- I/O devices:
  - Communication devices
    - Input only (keyboard, mouse, joystick)
    - Output only (printer, display)
    - Input/output (network card)
  - Storage devices
    - Input/output (disk, tape)
    - Input only (CD-ROM)
Basic Issues

- Main tasks of I/O system:
  - Present logical (abstract) view of devices
    - Hide details of hardware interface
    - Hide error handling
  - Facilitate efficient use
    - Overlap CPU and I/O
  - Support sharing of devices
    - Protection when device is shared (disk)
    - Scheduling when exclusive access needed (printer)

A Hierarchical Model of I/O

Abstract I/O interface:
- Block devices, character devices, network

Device-independent software:
- Buffering, scheduling, caching

Device-dependent software:
- I/O drivers
  (supplied by device manufacturer)
Figure 11.1 Typical I/O Device Data Rates
I/O System Interface

- Block-Oriented Device Interface
  - Operations: open, read, write, close
  - File System and Virtual Memory System

- Stream-Oriented Device Interface
  - = “character-device” interface
  - Operations: get, put
    - Also, open & close to reserve & to release the exclusive access normally needed
    - (Tapes are both Block-Oriented and Stream-Oriented)

- Network Communications
  - Key abstraction: *socket*
  - Protocols

Network Communications

- Key abstraction: *socket*
- Protocols:
  - Connection-based (virtual circuits)
  - Connection-less (datagrams)
  - Operations: connect, accept, write/send, read/receive
I/O Devices

- Display monitors
  - Character or graphics oriented
  - Different data rates:
    - 25 x 80 characters vs 800 x 600 x 256
    - 30-60 times/sec

Figure 11-2

I/O Devices

- Keyboards
  - Most common: "QWERTY"
  - Very low data rate (<10 char/sec)

- Pointing devices
  - Mouse (optical, optical-mechanical)
  - Trackball
  - Joystick
  - Low data rate (hundreds of bytes/sec)
I/O Devices

- Printers
  - Line printers, dot-matrix, ink-jet, laser
  - Low data rates
  - Character-oriented

- Scanners
  - Digitize picture into bit map (similar to video RAM)
  - Low data rates

Floppy disks

- Surface, tracks/surface, sectors/track, bytes/sector
- All sectors numbered sequentially 0..(n-1)
  (physical location vs logical numbering)

Figure 11-3(a) Physical

Figure 11-3(b) Logical
I/O Devices

- Floppy disks
  - Track skew
    - Account for seek-to-next-track to minimize latency
  - Double-sided floppy
    - Tracks with same diameter: cylinder
    - Number sectors within cylinder consecutively to minimize seek

![Figure 11-3(c)](image1)
![Figure 11-13(d)](image2)

I/O Devices

- Hard disks
  - Multiple surfaces

- Higher densities and data rates than floppy

<table>
<thead>
<tr>
<th></th>
<th>floppy</th>
<th>hard disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>bytes/sec</td>
<td>512</td>
<td>512-4096</td>
</tr>
<tr>
<td>sec/track</td>
<td>9, 15, 18, 36</td>
<td>100-400</td>
</tr>
<tr>
<td>tracks/surf</td>
<td>40, 80, 160</td>
<td>1000-10,000</td>
</tr>
<tr>
<td># surf</td>
<td>1-2</td>
<td>2-24</td>
</tr>
<tr>
<td>seek</td>
<td>30-100 ms</td>
<td>5-12 ms</td>
</tr>
<tr>
<td>rotation</td>
<td>400-700 rpm</td>
<td>3600-10,000 rpm</td>
</tr>
</tbody>
</table>
I/O Devices

- Optical disks
  - CD-ROM, CD-R (WORM), CD-RW
  - Originally designed for music
  - Data stored as continuous spiral, subdivided into sectors
  - Constant linear speed (200-530 rpm)
  - Higher storage capacity than magnetic disks: 0.66 GB/surface

I/O Devices

- Data transfer rates of disks
  - Sustained: continuous data delivery
  - Peek: transfer once read/write head is in place
    - Depends on rotation speed and data density
    - 1 revolution passes over all sectors of 1 track
  - Example: 7200 rpm, 100 sect/track, 512 B/sect
    - 7200 rpm: 60,000/7200 = 8.3 ms/rev
    - 8.3/100 = 0.083 ms/sector
    - 512 bytes transferred in 0.083 ms: ~6MB/s
I/O Devices

- Magnetic tapes (reel or cartridge)
  - Large storage capacity (GB)
  - Data transfer rate: ~2 MB/sec
- Networks (interface card)
  - Ethernet, token ring, slotted ring
    - Controller implements protocol to accept, transmit, receive packets
  - Modem
    - Convert between analog and digital (phone lines)
    - Character-oriented (like printer and keyboard)

Device Drivers

- Accept command from application
  - get/put character, read/write block, send/receive packet
- Interact with (hardware) device controller to carry out command
- Typical device controller interface: set of registers
- Example: serial or parallel port on PC
  - Generic driver reads/writes characters to registers

Figure 11-6
Device Drivers

- Memory-mapped vs Explicit device interface
  - Similar idea to memory-mapped files

- Explicit: Special I/O instruction:
  \[
  \text{io\_store \ cpu\_reg, dev\_no, dev\_reg}
  \]

- Memory-mapped: Regular CPU instruction:
  \[
  \text{store \ cpu\_reg, n} \quad (n \text{ is a memory address})
  \]

Programmed I/O with Polling

- CPU is responsible for
  - Moving every character to/from controller buffer
  - Detecting when I/O operation completed
  - Protocol to input a character:

  ![Diagram of Programmed I/O with Polling](image)
Programmed I/O with Polling

- Driver operation to input sequence of characters
  
  ```
  i = 0;
  do { write_reg(opcode, read);
       while (busy_flag == true) {...};
       mm_in_area[i] = data_buffer;
       increment i;
       compute;
  } while (…)
  ```

- What to do while waiting?
  - Idle (busy wait)
  - Some other computation
    - How frequently to poll?
  - Give up CPU
    - Device may remain unused for a long time

Figure 11-9
Programmed I/O with Interrupts

- CPU is responsible for
  - Moving characters to/from controller buffer, but
  - Interrupt signal informs CPU when I/O operation completes
- Protocol to input a character:

![Diagram of I/O operations with interrupts]

Figure 11-10

Programmed I/O with Interrupts

Compare Polling with Interrupts:

```c
i = 0;
do { write_reg(opcode, read);
  while (busy_flag == true) {...};
  mm_in_area[i] = data_buffer;
  increment i;
  compute;
} while (...)
```

```c
i = 0;
do { write_reg(opcode, read);
  block to wait for interrupt;
  mm_in_area[i] = data_buffer;
  increment i;
  compute;
} while (...)
```
Programmed I/O with Interrupts

Example: Keyboard driver

```c
i = 0;
    do { block to wait for interrupt;
        mm_in_area[i] = data_buffer;
        increment i;
        compute(mm_in_area[]);
    } while (data_buffer != ENTER)
```

- Timing of interrupt-driven I/O
- More OS overhead but better device utilization

![Figure 11-11](image)

DMA

- CPU only initiates operation
- DMA controller transfers data directly to/from main memory
- Interrupt when transfer completed
- Protocol to input data using DMA:

![Figure 11-12](image)
DMA

- Driver operation to input sequence of characters
  
  ```
  write_reg(mm_buf, m);
  write_reg(count, n);
  write_reg(opcode, read);
  block to wait for interrupt;
  ```

  - Writing `opcode` triggers DMA controller
  - DMA controller issues interrupt after n chars in memory

- I/O processor (channel)
  - Extended DMA controller
  - Executes I/O program in own memory

Device Management

- Device-independent techniques
- Reasons for buffering
  - Allows asynchronous operation of producers and consumers
  - Allows different granularities of data
  - Consumer or producer can be swapped out while waiting for buffer fill/empty

Figure 11-13
Device Management

- Single buffer operation
- Double buffer (buffer swapping)
  - Increases overlap
  - Ideal when: time to fill = time to empty = constant
  - When times differ, benefits diminish

![Diagram of single buffer operation]

When times differ, benefits diminish

![Figure 11-14(a,b)]

Device Management

- Circular Buffer
  - When average times to fill and empty are comparable but vary over time: circular buffer absorbs bursts
  - Producer and consumer each use an index
    - nextin gives position of next input
    - nextout gives position of next output
    - Both are incremented modulo n at end of operation

- Buffer Queue
  - Variable size buffer for more efficient use of memory
  - Depends on linked data structures and dynamic memory management. More (CPU) time consuming.

- Buffer Cache: pool of buffers for repeated access
Device Management

- Error handling
  - Persistent vs Transient, SW vs HW
  - Persistent SW error
    - Repair/reinstall program
  - Other errors: Build in defense mechanisms
  - Examples:
    - Transient SW errors:
      - Error correcting codes, retransmission
    - Transient HW errors:
      - Retry disk seek/read/write
    - Persistent HW errors:
      - Redundancy in storage media

Device Management

- Bad block detection and handling
  - Block may be defective as a manufacturing fault or during use (a common problem)
  - Parity bit is used to detect faulty block
  - The controller bypasses faulty block by renumbering
  - A spare block is used instead
  - Two possible remappings:
    - More work but contiguity of allocation preserved

Figure 11-17
Device Management

- Stable storage
  - Some applications cannot tolerate any loss of data (even temporarily)
  - Stable storage protocols:
    - Use 2 independent disks, A and B
    - Write: write to A; if successful, write to B
    - Read: read from A and B; if A!=B, go to Recovery
    - Recovery from Media Failure: A or B contains correct data; remap failed disk block
    - Recovery from Crash: if before writing A, B is correct; if after writing A, A is correct; recover from whichever is correct

RAID (Redundant Array of Independent Disks)

- Increased performance through parallel access
- Increased reliability through redundant data
- Maintain exact replicas of all disks
  - Most reliable but wasteful

- Maintain only partial recovery information
  - (e.g. error correcting codes)

Figure 11-19
Device Management

- Disk Scheduling
  - Minimize seek time and rotational delay
  - Requests from different processes arrive concurrently:
    - Scheduler must attempt to preserve locality
  - Rotational delay:
    - Order requests to blocks on each track in the direction of rotation: access in one rotation
    - Proceed with next track on same cylinder

Minimizing seek time: more difficult
- Read/write arm can move in two directions
- Minimize total travel distance
- Guarantee fairness
- FIFO: simple, fair, but inefficient
- SSTF: most efficient but prone to starvation
- (Elevator) Scan: fair, acceptable performance

Figure 11-20