Input/Output

Operating Systems
Kartik Gopalan

Chapter 5, Modern Operating Systems — Tanenbaum
Chapter 13, OS Concepts — Silberschatz (optional)
Layering in the I/O subsystem

User Applications
How CPU and I/O devices communicate

A single-bus architecture

(a)

All addresses (memory and I/O) go here

Bus

A dual-bus memory architecture

(b)

CPU reads and writes of memory go over this high-bandwidth bus

This memory port is to allow I/O devices access to memory
Device Controllers

- I/O devices may have
  - mechanical components
  - electronic components

- Device controller is the electronic component
  - may be able to handle multiple devices

- Controller's tasks
  - Intermediary between I/O devices, CPU, and memory
  - convert serial bit stream to block of bytes
  - perform error correction as necessary
Programmed I/O

Printing a string: CPU transfers one byte at a time to device

```c
copy_from_user(buffer, p, count); /* p is the kernel buffer */
for (i = 0; i < count; i++) {
    while (*printer_status_reg != READY) ; /* loop on every character */
    *printer_data_register = p[i]; /* loop until ready */
    *printer_data_register = p[i]; /* output one character */
}
return_to_user();
```
Interrupts

- I/O events signaled by device to CPU
- Interrupt-request (IRQ) line triggered by controller to CPU
  - Checked by CPU after each instruction
- Interrupt vector table used to invoke the correct interrupt handler
<table>
<thead>
<tr>
<th>vector number</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>divide error</td>
</tr>
<tr>
<td>1</td>
<td>debug exception</td>
</tr>
<tr>
<td>2</td>
<td>null interrupt</td>
</tr>
<tr>
<td>3</td>
<td>breakpoint</td>
</tr>
<tr>
<td>4</td>
<td>INTO-detected overflow</td>
</tr>
<tr>
<td>5</td>
<td>bound range exception</td>
</tr>
<tr>
<td>6</td>
<td>invalid opcode</td>
</tr>
<tr>
<td>7</td>
<td>device not available</td>
</tr>
<tr>
<td>8</td>
<td>double fault</td>
</tr>
<tr>
<td>9</td>
<td>coprocessor segment overrun (reserved)</td>
</tr>
<tr>
<td>10</td>
<td>invalid task state segment</td>
</tr>
<tr>
<td>11</td>
<td>segment not present</td>
</tr>
<tr>
<td>12</td>
<td>stack fault</td>
</tr>
<tr>
<td>13</td>
<td>general protection</td>
</tr>
<tr>
<td>14</td>
<td>page fault</td>
</tr>
<tr>
<td>15</td>
<td>(Intel reserved, do not use)</td>
</tr>
<tr>
<td>16</td>
<td>floating-point error</td>
</tr>
<tr>
<td>17</td>
<td>alignment check</td>
</tr>
<tr>
<td>18</td>
<td>machine check</td>
</tr>
<tr>
<td>19–31</td>
<td>(Intel reserved, do not use)</td>
</tr>
<tr>
<td>32–255</td>
<td>maskable interrupts</td>
</tr>
</tbody>
</table>
Interrupt-Driven I/O

```c
int write_string_to_printer(const char *buffer, int count) {
    int i;
    for (i = 0; i < count; i++) {
        *printer_data_register = p[i];
        count = count - 1;
        i = i + 1;
        enable_interrupts();
        if (count == 0) {
            unblock_user();
        } else {
            *printer_data_register = p[i];
            count = count - 1;
            i = i + 1;
        }
        acknowledge_interrupt();
        return_from_interrupt();
    }
    disable_interrupts();
    return 0;
}
```

Writing a string to the printer using interrupt-driven I/O
(a) Code executed when print system call is made
   - Starts off by printing the first character
(b) Interrupt service procedure
   - called every time the printer completes printing one character and generates an interrupt
Direct Memory Access (DMA)

- DMA allows I/O controllers to directly transfer data to/from the main memory.
- Frees up the CPU for other tasks

```c
#include <linux/dma.h>

struct my_dma {
    void *buffer;
    int count;
    struct task_struct *p;
};

void print_job_completed(struct task_struct *p)
{
    struct my_dma *dma = to_dma(p);

    copy_from_user(dma->buffer, p, dma->count);
    set_up_DMA_controller(dma);
    scheduler();

    interrupt_request(dma->buffer);
    unblock_user();
    return_from_interrupt();
}
```

(a) Code executed by print system call

(b) Interrupt service routine (ISR) called when the entire print job is completed
Device Classification

• Character (char) devices
  • byte-stream abstraction
  • E.g. keyboard, mouse

• block devices
  • reads/writes in fixed block granularity
  • E.g. hard disks, CD drives

• network devices
  • message abstraction
  • send/receive packets of varying sizes
  • E.g. network interface cards

• others
  • USB, SCSI, Firewire, I2O
  • Can (mostly) be used to implement one or more of the above three classes
Hard Disks
Hard disk organizes data in tracks and sectors

- **Seeking**: Disk head moves across tracks
- **Rotation**: Disk rotates to bring the correct sector under disk head

Figure 5-18. (a) Physical geometry of a disk with two zones. (b) A possible virtual geometry for this disk.
Factors affecting read/write latency

- **Seek time**
  - Time to move magnetic head from one track to next.
  - Average 5 to 10 milliseconds.

- **Rotational delay**
  - Time to rotate the disk to reach the sector containing the data
  - Depends on the rotational speed of the disk
  - E.g. A disk with 15,000 rotations per minute (rpm) will have 1 rotation every 4 milliseconds, or an average of 2 millisecond rotational latency (assuming that average latency is one-half of a rotational period).

- **Actual transfer time**
  - Time to transfer data from the disk sector to memory
  - In microseconds

- **Seek time dominates**
  - For most practical purposes, rotational and transfer times are ignored.
Disk Arm Scheduling Algorithms

Shortest Seek First (SSF) disk scheduling algorithm
- High I/O throughput, but some I/O requests may starve

Elevator (SCAN) algorithm for scheduling disk requests
- Reasonable I/O throughput and No starvation
Disk I/O Throughput

- Seek Latency dominates

- Sequential I/O throughput is much higher than random I/O throughput
  - Sequential access involves fewer seeks

- I/O Operations per second (IOPS)
  - Better measure of disk throughput
  - Ignores the number of bytes in each I/O operation.

Disk Buffer

- Hard disk controllers have a small amount of buffer memory (RAM)
  - 8MB to 128MB

- Used to stage data moving between the physical disk and main memory

- Hides seek latency in two ways:
  - Read-ahead
    - Read extra data near the requested sector and store in disk buffer. Return from disk buffer when OS requests the extra data.

- Write Acceleration:
  - When OS writes data, store data in disk buffer and return completion interrupt. Commit the data to physical disk “later”.
  - If power fails, data in disk buffer could be lost!

Solid State Drives (SSD)

SSD

• NAND-based flash memory to store persistent data
  • Plus a controller performs Error Correction, Wear Leveling, Caching, etc.
• No mechanical components
• Works with various interfaces
  • SATA, SAS, PCI, USB, Fibre Channel, IDE
• Disks in various form factors
  • Regular, Thumb, embedded
• SSDs may also be used as buffer in Hard disks to improve reliability
• Or DRAM could be used as buffer to improve SSD access latencies

SSD vs Hard Disks
• Lower Access Latencies
  • Random access ~ 0.1ms
• Higher throughput
• Reliability
  • No moving parts
  • But repeated writes to the same cell makes the cell unreliable.
  • Wear leveling: write to a different cell each time
• Silent operation
• Low power consumption
• Smaller capacities
• More expensive
Network Attached Storage (NAS)

• A storage server accessed over a Local Area Network (LAN)
Storage Area Network (SAN)

- A dedicated network to access a collection of storage media
  - High speed network interconnect
    - Fiber Channel, iSCSI, AoE etc
  - Virtualized storage
    - Consolidated block-level access