CSCI-GA.2250-001

Operating Systems

I/O

Hubertus Franke
frankeh@cs.nyu.edu
External devices that engage in I/O with computer systems can be grouped into three categories:

**Human readable**
- suitable for communicating with the computer user
- printers, terminals, video display, keyboard, mouse

**Machine readable**
- suitable for communicating with electronic equipment
- disk drives, USB keys, sensors, controllers

**Communication**
- suitable for communicating with remote devices
- modems, digital line drivers
A Simple Definition

• The main concept of I/O is to move data from/to I/O devices to the processor using some modules and buffers.
• This is the way the processor deals with the outside world
• Examples: mouse, display, keyboard, disks, scanners, speakers, etc.
The OS and I/O

• The OS controls all I/O devices
  – Issue commands to devices
  – Catch interrupts
  – Handle errors

• Provides an interface between the devices and the rest of the system
I/O Devices: Challenges

• Very diverse devices
  — behavior (i.e., input vs. output vs. storage)
  — partner (who is at the other end?)
  — data rate

• I/O Design affected by many factors (expandability, resilience)

• Performance:
  — access latency
  — throughput
  — connection between devices and the system
  — the memory hierarchy
  — the operating system

• A variety of different users
I/O Devices

• Block device
  – Stores information in fixed-size blocks
  – Each block has its own address
  – Transfers in one or more blocks
  – Example: Hard-disks, CD-ROMs, USB sticks

• Character device
  – Delivers or accepts stream of character
  – Is not addressable
  – Example: mice, printers, network interfaces
<table>
<thead>
<tr>
<th>Device</th>
<th>Data rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keyboard</td>
<td>10 bytes/sec</td>
</tr>
<tr>
<td>Mouse</td>
<td>100 bytes/sec</td>
</tr>
<tr>
<td>56K modem</td>
<td>7 KB/sec</td>
</tr>
<tr>
<td>Scanner</td>
<td>400 KB/sec</td>
</tr>
<tr>
<td>Digital camcorder</td>
<td>3.5 MB/sec</td>
</tr>
<tr>
<td>802.11g Wireless</td>
<td>6.75 MB/sec</td>
</tr>
<tr>
<td>52x CD-ROM</td>
<td>7.8 MB/sec</td>
</tr>
<tr>
<td>Fast Ethernet</td>
<td>12.5 MB/sec</td>
</tr>
<tr>
<td>Compact flash card</td>
<td>40 MB/sec</td>
</tr>
<tr>
<td>FireWire (IEEE 1394)</td>
<td>50 MB/sec</td>
</tr>
<tr>
<td>USB 2.0</td>
<td>60 MB/sec</td>
</tr>
<tr>
<td>SONET OC-12 network</td>
<td>78 MB/sec</td>
</tr>
<tr>
<td>SCSI Ultra 2 disk</td>
<td>80 MB/sec</td>
</tr>
<tr>
<td>Gigabit Ethernet</td>
<td>125 MB/sec</td>
</tr>
<tr>
<td>SATA disk drive</td>
<td>300 MB/sec</td>
</tr>
<tr>
<td>Ultrium tape</td>
<td>320 MB/sec</td>
</tr>
<tr>
<td>PCI bus</td>
<td>528 MB/sec</td>
</tr>
</tbody>
</table>
Devices change !!!

- Network speed doubling every 18-24 month
- Approaching 100Gbps / 40Gbps
I/O Units

Mechanical component

The Device Itself

Electronic Component

Device Controller

Diagram showing various components connected through a bus, including:
- CPU
- Memory
- Video controller
- Keyboard controller
- USB controller
- Hard disk controller
- Monitor
- Keyboard
- USB printer
- Hard disk drive
Controller and Device

• Each controller has few registers used to communicate with CPU
• By writing/reading into/from those registers, the OS can control the devices.
• There are also data buffers in the device that can be read/written by the OS
How does CPU communicate with control registers and data buffers?

Two main approaches
- I/O port space
- Memory-mapped I/O
I/O Port Space

- Each control register is assigned an I/O port number
- The set of all I/O ports form the I/O port space
- I/O port space is protected
Memory-Mapped I/O

- Map control registers into the memory space
- Each control register is assigned a unique memory address
Advantages of Memory-Mapped I/O

- Device drivers can be written entirely in C (since no special instructions are needed)
- No special protection is needed from OS, just refrain from putting that portion of the address space in any user’s virtual address space
- Every instruction that can reference memory can also reference control registers
Disadvantages of Memory-Mapped I/O

- Caching a device control register can be disastrous
- Hardware complications
Direct Memory Access (DMA)

- It is not efficient for the CPU to request data from I/O one byte at a time.
- DMA controller has access to the system bus independent of the CPU.
Interrupts

• When an I/O device has finished the work given to it, it causes an interrupt.
Precise Interrupts

• Makes handling interrupts much simpler
• Has 4 properties
  – The program counter (PC) is saved in known place
  – All instructions before the one pointed by PC have fully executed
  – No instruction beyond the one pointed by PC has been executed
  – The execution state of the instruction pointed to by the PC is known
Precise Interrupt

(a)

Imprecise Interrupt

(b)
I/O Software

• Device independence
• Uniform naming
• Error handling
  – Should be handled as close to the hardware as possible
• Synchronous vs asynchronous (interrupt-driven)
• Buffering
• Sharable verses dedicated devices
Three Ways of Doing I/O

• Programmed I/O
• Interrupt-driven I/O
• I/O Using DMA
<table>
<thead>
<tr>
<th></th>
<th>No Interrupts</th>
<th>Use of Interrupts</th>
</tr>
</thead>
<tbody>
<tr>
<td>I/O-to-memory transfer through processor</td>
<td>Programmed I/O</td>
<td>Interrupt-driven I/O</td>
</tr>
<tr>
<td>Direct I/O-to-memory transfer</td>
<td></td>
<td>Direct memory access (DMA)</td>
</tr>
</tbody>
</table>
Programmed I/O

- CPU does all the work
- Busy-waiting (polling)

Example:

```c
int copy_from_user(buffer, p, count);
for (i = 0; i < count; i++) {
    while (*printer_status_reg != READY);
    *printer_data_register = p[i];
}
return_to_user();
```
Interrupt-Driven I/O

• Waiting for a device to be ready, the process is blocked and another process is scheduled.

• When the device is ready it raises an interrupt.
I/O Using DMA

- DMA does the work instead of the CPU
- Let the DMA do its work and then interrupts
DMA

(a) Single-bus, detached DMA

Alternative

(b) Single-bus, Integrated DMA-I/O

Configurations

(c) I/O bus
OS Software Layers for I/O

- Hardware
- Interrupt handlers
- Device drivers
- Device-independent operating system software
- User-level I/O software
Interrupt Handlers
Device Drivers

- Device specific code for controlling the device
  - Read device registers from controller
  - Write device registers to issue commands
- Usually supplied by the device manufacturer
- Can be part of the kernel or at user-space (with system calls to access controller registers)
- OS defines a standard interface that drivers for block devices must follow and another standard for driver of character devices
Device Drivers

User process

User program

Rest of the operating system

Kernel space

Printer driver
Camcorder driver
CD-ROM driver

Hardware

Printer controller
Camcorder controller
CD-ROM controller

Devices
Device Drivers

• **Main functions:**
  – Receive abstract read/write from layer above and carry them out
  – Initialize the device
  – Log events
  – Manage power requirements
• Drivers must be *reentrant*
• Drivers must deal with events such as a device removed or plugged
### Device Independent I/O Software

<table>
<thead>
<tr>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniform interfacing for device drivers</td>
</tr>
<tr>
<td>Buffering</td>
</tr>
<tr>
<td>Error reporting</td>
</tr>
<tr>
<td>Allocating and releasing dedicated devices</td>
</tr>
<tr>
<td>Providing a device-independent block size</td>
</tr>
</tbody>
</table>
Device Independent
I/O Software

• Uniform interfacing for device drivers
  – Trying to make all devices look the same
  – For each class of devices, the OS defines a set of functions that the driver must supply.
  – This layer of OS maps symbolic device names onto proper drivers
Device Independent I/O Software

Interrupt with every character

Buffering n char.

Very inefficient

What if buffer is paged out?

The need for buffering
Example of Devices
Clocks

Oscillators
Clock Hardware

- **Old**: tied to the power line and causes an interrupt on every voltage cycle
- **New**: Crystal oscillator + counter + holding register

![Diagram]

- Crystal oscillator
- Counter is decremented at each pulse
- Holding register is used to load the counter
Clock Software

• Maintaining the time of the day
• Preventing processes from running longer than they are allowed to
• Accounting for CPU usage
• Handling alarm system call made by user processes
• Providing watchdog timers for parts of the system itself
• Doing profiling, monitoring, and statistics gathering
User Interfaces

- Keyboard
- Mouse
- Monitor

As examples, we will take a closer look at the keyboard and mouse.
Keyboards

- Contains simplified embedded processor that communicates through a port with the controller at the motherboard
- An interrupt is generated whenever a key is struck and a second one whenever a key is released
- Keyboard driver extracts the information about what happens from the I/O port assigned to the keyboard
- The number in the I/O port is called the scan code (7 bits for code + 1 bit for key press/release)
Keyboards

Microprocessor of the keyboard

Key matrix
Keyboards

- There are two philosophies for programs dealing with keyboards
  1. The programs get a raw sequence of ASCII codes (raw mode, or noncanonical mode)
  2. Driver handles all the intraline editing and just delivered corrected lines to the use programs (cooked mode or canonical mode)
- Either way, a buffer is needed to store characters
Mouse

• Mouse only indicates changes in position, not absolute position (delta_x and delta_y)
I/O Software Layer: Principle

- Interrupts are facts of life, but should be hidden away, so that as little of the OS as possible knows about them.
- The best way to hide interrupts is to have the driver starting an IO operation block until IO has completed and the interrupt occurs.
- When interrupt happens, the interrupt handler handles the interrupt.
- Once the handling of interrupt is done, the interrupt handler unblocks the device driver that started it.
- This model works if drivers are structured as kernel processes with their own states, stacks and program counters.
Conclusions

• The OS provides an interface between the devices and the rest of the system.
• The I/O part of the OS is divided into several layers.
• The hardware: CPU, programmable interrupt controller, DMA, device controller, and the device itself.
• OS must expand as new I/O devices are added