I/O Systems

CS 256/456
Dept. of Computer Science, University of Rochester

Input/Output System Structure

- Principles of I/O hardware
- Principles of I/O software
- I/O software layers

Principles of I/O Hardware

<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 Kbps/sec</td>
</tr>
<tr>
<td>Telephone channel</td>
<td>9.6 Kbps/sec</td>
</tr>
<tr>
<td>Dual ISDN lines</td>
<td>192 Kbps/sec</td>
</tr>
<tr>
<td>Laser printer</td>
<td>180 Kbps/sec</td>
</tr>
<tr>
<td>Scanner</td>
<td>400 Kbps/sec</td>
</tr>
<tr>
<td>Ethernet</td>
<td>1.25 MB/sec</td>
</tr>
<tr>
<td>USB (universal serial bus)</td>
<td>1.6 MB/sec</td>
</tr>
<tr>
<td>Digital camcorder</td>
<td>4 MB/sec</td>
</tr>
<tr>
<td>UHF clock</td>
<td>9.6 MB/sec</td>
</tr>
<tr>
<td>4x CD-ROM</td>
<td>6 MB/sec</td>
</tr>
<tr>
<td>Fast Ethernet</td>
<td>12.5 MB/sec</td>
</tr>
<tr>
<td>ISA bus</td>
<td>16.7 MB/sec</td>
</tr>
<tr>
<td>IDE (ATA-2) disk</td>
<td>16.7 MB/sec</td>
</tr>
<tr>
<td>FireWire (IEEE 1394)</td>
<td>50 MB/sec</td>
</tr>
<tr>
<td>XGA Monitor</td>
<td>60 MB/sec</td>
</tr>
<tr>
<td>SONET OC-3/PW network</td>
<td>75 MB/sec</td>
</tr>
<tr>
<td>SCSI Ultra 2 disk</td>
<td>10 MB/sec</td>
</tr>
<tr>
<td>Gigabit Ethernet</td>
<td>125 MB/sec</td>
</tr>
<tr>
<td>Ultrium tape</td>
<td>220 MB/sec</td>
</tr>
<tr>
<td>PCI bus</td>
<td>528 MB/sec</td>
</tr>
<tr>
<td>Sun Gigaspin X8 backup</td>
<td>30 GB/sec</td>
</tr>
</tbody>
</table>

Device Controllers

- I/O devices have mechanical component & electronic component
- The electronic component is the device controller
  - It contains control logic, command registers, status registers, and on-board buffer space
I/O Ports & Memory-Mapped I/O

I/O methods:
- Separate I/O and memory space; special I/O commands (IN/OUT)
- Memory-mapped I/O

Issues with them:
- Convenience/efficiency when use high-level language;
- Protection mechanisms;
- Special data access schemes: TSL
- Data caching

More on Memory-Mapped I/O

(a) A single-bus architecture
(b) A dual-bus memory architecture

Direct Memory Access (DMA)

- Are the addresses CPU sends to the DMA controller virtual or physical addresses?
- Can the disk controller directly read data into the main memory (bypassing the controller buffer)?

Interrupt Hardware
Goals of I/O Software

- Device independence
  - reuse software as much as possible across different types of devices
- Error handling
  - handle as close to the hardware as possible
- Synchronous vs. asynchronous transfers
  - blocked transfers vs. interrupt-driven
- Buffering
  - data coming off a device cannot be stored in final destination

Interrupt-Driven I/O

```c
void copy_from_user(char *buffer, int count);
void enable_interrups();
int *printer_status_reg;
int *printer_data_register;
scheduler();

for (i = 0; i < count; i++) {
  if (*printer_status_reg == READY) {
    *printer_data_register = buffer[i];
    scheduler();
  } else {
    *printer_data_register = buffer[i];
    count = count - 1;
    i = i + 1;
  }
}

void return_from_interrupt();
```

- Writing a string to the printer using interrupt-driven I/O
  - Code executed when print system call is made
  - Interrupt service procedure

Programmed I/O (polling-based)

```c
void copy_from_user(char *buffer, int count);
for (i = 0; i < count; i++) {
  while (*printer_status_reg == READY) {
    *printer_data_register = buffer[i];
    *printer_data_register = buffer[i];
    *printer_data_register = buffer[i];
    i = i + 1;
  }
  return_from_interrupt();
```

I/O Software Layers

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

Layers of the I/O Software System
### Interrupt Handlers

1. Save regs not already saved by interrupt hardware
2. Set up context for interrupt service procedure (switch from the user space to kernel space: TLB, page table, ...)
3. Set up stack for interrupt service procedure
4. Run service procedure; when safe, re-enable interrupts
5. Set up MMU context for process to run next
6. Load new process' registers
7. Start running the new process

### Device Drivers

- Device-dependent OS I/O software; directly interacts with controller hardware
- Interface to upper-layer OS code is standardized.

### Disclaimer

- Parts of the lecture slides contain original work of Abraham Silberschatz, Peter B. Galvin, Greg Gagne, Andrew S. Tanenbaum, and Gary Nutt. The slides are intended for the sole purpose of instruction of operating systems at the University of Rochester. All copyrighted materials belong to their original owner(s).