Review: Summary of Chapter 4

° Mono-programming -> Multi-programming

° Swapping

° Virtual Memory
  • Paging
  • Page replacement algorithms
  • Design and implementation issues

° Segmentation

° More reading: Chapter 4.1 - 4.9
Chapter 5: Input/Output

5.1 Principles of I/O hardware
5.2 Principles of I/O software
5.3 I/O software layers
5.4 Disks
5.5 Clocks
5.6 Character-oriented terminals
5.7 Graphical user interfaces
5.8 Network terminals
5.9 Power management

I/O of OS

Main Function: Control all the computer’s I/O devices
- Issue commands to the devices, catch interrupts, and handle errors
- Provide interface between the devices and the rest of system that is simple and easy to use

What we care I/O devices most as software designers and programmers?
The interface presented to us:
(1) The commands the hardware accepts
(2) The functions it carries out
(3) The errors that can be reported back
Principles of I/O Hardware: I/O Devices

<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>Telephone channel</td>
<td>8 KB/sec</td>
</tr>
<tr>
<td>Dual ISDN lines</td>
<td>16 KB/sec</td>
</tr>
<tr>
<td>Laser printer</td>
<td>100 KB/sec</td>
</tr>
<tr>
<td>Scanner</td>
<td>400 KB/sec</td>
</tr>
<tr>
<td>Gigabit Ethernet</td>
<td>1.25 MB/sec</td>
</tr>
<tr>
<td>USB (Universal Serial Bus)</td>
<td>1.5 MB/sec</td>
</tr>
<tr>
<td>Digital camcorder</td>
<td>4 MB/sec</td>
</tr>
<tr>
<td>IDE disk</td>
<td>5 MB/sec</td>
</tr>
<tr>
<td>40x 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>EIDE (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>XDA Monitor</td>
<td>60 MB/sec</td>
</tr>
<tr>
<td>SONET OC-12 network</td>
<td>75 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>Ultrium tape</td>
<td>320 MB/sec</td>
</tr>
<tr>
<td>PCI bus</td>
<td>528 MB/sec</td>
</tr>
<tr>
<td>Sun Gigaspace X8 backplane</td>
<td>20 GB/sec</td>
</tr>
</tbody>
</table>

Some typical device, network, and data base rates

Device Controllers

- I/O devices have components:
  - mechanical component
  - electronic component

- The electronic component is the **device controller**
  - may be able to handle multiple but identical devices

- Controller's tasks
  - convert serial bit stream to block of bytes
  - perform error correction as necessary
  - make blocks available to main memory

Standard interface: IDE, SCSI
CPU→I/O: Special I/O Instructions

- How the CPU communicates with the control registers and the device data buffers? Two methods are available.

- Method I: Special (assembly) I/O Instructions (IBM 360)
  - Each control register is assigned an I/O port number (device #)

  ```assembly
  IN REG, PORT
  OUT PORT, REG
  ```

  - The memory address space and I/O address space are different

  ```assembly
  IN R0, 4 ; port 4
  MOV R0, 4 ; memory word 4
  ```

  I/O operations require assembly language programming

CPU→I/O: Memory-mapped I/O

- Method II: Memory-mapped I/O (Pentium)
  - Each register is assigned a unique memory address to which no memory is assigned; read/write to addresses are I/O operations
  - How it works?

![Diagram of CPU→I/O: Memory-mapped I/O](image)
Advantages of Memory-mapped I/O

- An I/O device driver can be written in C, instead of assembly
- Protection is easy (by address translation); user programs are prevented from issuing I/O operations directly
  - That I/O-mapped space not in any user’s virtual addr. space

Disadvantages of Memory-mapped I/O

- Caching a device control register would be disastrous
- Both memory and all I/O devices must examine all memory references to see which ones to respond to, not easy if bus hierarchy

How to overcome the disadvantages?

(a) Separate I/O and memory space; (b) Memory-mapped I/O; (c) Hybrid (PDP-11) (Pentium)
**CPU→I/O: Memory-mapped I/O (4)**

- PCI bridge chip filters addresses to different buses

![Diagram of a large Pentium system](attachment:image.png)

**I/O→CPU: Device Notifying the CPU**

- The OS needs to know when:
  - The I/O device has completed an operation
  - The I/O operation has encountered an error

- This can be accomplished in two different ways:
  - **Polling (Busy Waiting):**
    - The I/O device puts information in a status register
    - The CPU periodically or continuously checks the status register
  - **I/O Interrupt:**
    - Whenever an I/O device needs attention from the processor, it interrupts the processor from what it is currently doing.

**In real-time systems, a hybrid approach is often used**

- Use a clock to periodically interrupt the CPU, at which time the CPU polls all I/O devices
Data Transfer: Programmed I/O (Busy Waiting)

Advantage:
- Simple: the processor is totally in control and does all the work

Disadvantage:
- Polling overhead can consume a lot of CPU time

Data Transfer: Interrupt Driven

Advantage:
- User program progress is only halted during actual transfer

Disadvantage, special hardware is needed to:
- Cause an interrupt (I/O device)
- Detect an interrupt (processor) (EPC, etc)
- Save the proper states to resume after the interrupt (processor)
Direct Memory Access (DMA)

- DMA controller has access to system bus independent of CPU
  - A memory address register
  - A byte count register
  - A control register (direction, transfer unit, and transfer mode)
  - Multiple reg. sets if multiple transfer at once

DMA Transfer Modes

- Cycle stealing mode: word-at-a-time, sneaks in and steals an occasional bus cycle from the CPU once in a while
- Burst mode: asks I/P device to acquire the bus, issue a series of transfers, then release the bus
- Tradeoff: I/O efficiency and CPU blocked

Why need an internal buffer at the disk controller?
Interrupts Revisited

- One way of how I/O notify CPU
  - Interrupt vector and interrupt service procedures
  - Priority if multiple simultaneous interrupts
  - CPU delays ACK until it is ready to handle the next interrupt is a way to avoid race conditions involving multiple almost simultaneous interrupts

Chapter 5: Input/Output

5.1 Principles of I/O hardware

5.2 Principles of I/O software

5.3 I/O software layers

5.4 Disks

5.5 Clocks

5.6 Character-oriented terminals

5.7 Graphical user interfaces

5.8 Network terminals

5.9 Power management
### Goals of I/O Software

- **Device independence**
  - Write programs that can access any I/O device without specifying device in advance
    - (floppy, hard drive, or CD-ROM)

- **Uniform naming**
  - name of a file or device a string or an integer (e.g., path name)
  - not depending on which machine / device
  - UNIX Mount System

- **Error handling**
  - handle as close to the hardware as possible, i.e., controller first then device driver next

### Goals of I/O Software (2)

- **Synchronous vs. asynchronous transfers**
  - blocked transfers vs. interrupt-driven
  - Logically synchronous to programmers and physically asynchronous

- **Buffering**
  - data coming off a device cannot be stored directly in final destination, e.g., network-I/O operations
    - For checksum and timing constraints

- **Sharable vs. dedicated devices**
  - disks are sharable
  - tape drives would not be
  - dedication introduces deadlock

**What are three fundamental different ways of performing I/O?**
Programmed I/O

- **Polling (busy waiting)**
  - CPU continuously polls the device (register) to see if it is ready
  - Simple but CPU time wasteful

Steps in printing a string

(a) User space: String to be printed
(b) Printed page: ABCD EFGH
(c) Printed page: Next

Programmed I/O (2)

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

Writing a string to the printer using programmed I/O

What if the printer can print 100 character/sec?

- CPU would be idle for 10ms each loop iteration!
- Why not switching to another process?
Interrupt-Driven I/O

° A way to allow the CPU to do something else while waiting for the printer to become ready

Still, an interrupt occurs every character, what to do?

(a) Code executed when print system call is made

(b) Interrupt service procedure

```
copy_from_user(buffer, p, count);
if (count == 0) {
    enable_interrupts();
    Unblock_user();
    } else {
    *printer_status_reg != READY ;
    *printer_data_reg = p[i];
    count = count - 1;
    i = i + 1;
    }
    acknowledge_interrupt();
    return_from_interrupt();
```

Writing a string to the printer using interrupt-driven I/O

I/O Using DMA

° A way to reduce the number of interrupts from one per character to one per buffer printed

° Transfers a block of data directly to or from memory
° An interrupt is sent when the task is complete
° The processor is only involved at the beginning and end of the transfer

```
copy_from_user(buffer, p, count);
set_up_DMA_controller();
acknowledge_interrupt();
unblock_user();
scheduler();
return_from_interrupt();
```

° Printing a string using DMA

° code executed when the print system call is made
° interrupt service procedure
I/O Software Hierarchy

One objective: as little of OS as possible knows about the interrupts

<table>
<thead>
<tr>
<th>User-level I/O software</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device-independent operating system software</td>
</tr>
<tr>
<td>Device drivers</td>
</tr>
<tr>
<td>Interrupt handlers</td>
</tr>
<tr>
<td>Hardware</td>
</tr>
</tbody>
</table>

Layers of the I/O Software System

How about the following jobs:
1. Check if the user is permitted to use the device
2. Writing commands to the device registers
3. Formatting the output data for printing

Interrupt Handlers

- Interrupt handlers are best hidden
  - have driver starting an I/O operation block until interrupt notifies of completion

- Interrupt procedure does its task, handling interrupts
  - then unblocks driver that started it

- But actually, many OS steps must be performed in software after *hardware interrupt* has completed
  1. Save regs not already saved by interrupt hardware
  2. Set up context for interrupt service procedure
  3. ...... (Page 288)
Device Drivers

- Device drivers: provided by device's manufacturer, device-specific code for controlling the device (via device controller registers)

Logical position of device drivers is shown here
- Communications between drivers and device controllers goes over the bus

Device Drivers Installation

How to install device drivers to OS kernel?
- Static installation and recompiling (Unix)
  - Recompile the kernel with the new driver

- Static registration and rebooting (early Windows)
  - Make an entry in an OS file and reboot the system

- Dynamic installation (Windows)
  - On-the-fly loading into the system during the execution
Device-Independent I/O Software

- The basic function is to perform the I/O functions that are common to all devices and to provide a uniform interface to the user-level software

<table>
<thead>
<tr>
<th>Uniform interfacing for device drivers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffering</td>
</tr>
<tr>
<td>Error reporting</td>
</tr>
<tr>
<td>Allocating and releasing dedicate devices</td>
</tr>
<tr>
<td>Providing a device-independent block size</td>
</tr>
</tbody>
</table>

Functions of the device-independent I/O software

Uniform Interfacing for Device Drivers

- The device-independent I/O software takes care of mapping symbolic device names onto the proper driver
  - UNIX i-node; (major device number, minor device number)
  - Programmers see i-number instead (another level of mapping)

(a) Without a standard driver interface;
(b) With a standard driver interface
Buffering

- (a) Unbuffered input: doesn’t it have too many interrupts?
- (b) Buffering in user space: What happens if the buffer is *paged out* (to disk) when a character arrives? Go pinning?
- (c) Buffering in the kernel followed by copying to user space
  - What happens to characters that arrive while the page with the user buffer is being brought (in memory) from the disk?
- (d) Double buffering in the kernel; two buffers take turns.

Disadvantages of Buffering

- Why the process does not copy the data to the wire directly from kernel memory?
  - But how about the performance?
    - Why not bit-by-bit from the kernel buffer to the network?

Networking may involve many copies of a packet, and all steps must happen sequentially.
### User-Space I/O Software

- Some of I/O software consists of user-space libraries
  - The standard I/O library contains a number of procedures that involve with I/O and all run as part of user programs
    - E.g., scanf(), printf() for I/O formatting

```
Layer         I/O reply              I/O functions
User processes Make I/O call; format I/O; spooling
Device-independent software Naming, protection, blocking, buffering, allocation
Device drivers Set up device registers; check status
Interrupt handlers Wake up driver when I/O completed
Hardware      Perform I/O operation
```

Layers of the I/O system and the main functions of each layer

### Hard Disks

- A stack of platters, a surface with a magnetic coating
- Magnetic disks are organized into cylinders, for S/R/W
- Overlapped seeks: one disk controller can seek two or more drives simultaneously
  - Transfer between the disk and the controller’s buffer can be simultaneous
  - But one transfer between the controller and the main memory
### Disk Hardware Characteristics

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IBM 360-KB floppy disk</th>
<th>WD 18300 hard disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of cylinders</td>
<td>40</td>
<td>10601</td>
</tr>
<tr>
<td>Tracks per cylinder</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>Sectors per track</td>
<td>9</td>
<td>281 (avg)</td>
</tr>
<tr>
<td>Sectors per disk</td>
<td>720</td>
<td>35742000</td>
</tr>
<tr>
<td>Bytes per sector</td>
<td>512</td>
<td>512</td>
</tr>
<tr>
<td>Disk capacity</td>
<td>360 KB</td>
<td>18.3 GB</td>
</tr>
<tr>
<td>Seek time (adjacent cylinders)</td>
<td>6 msec</td>
<td>0.8 msec</td>
</tr>
<tr>
<td>Seek time (average case)</td>
<td>77 msec</td>
<td>6.9 msec</td>
</tr>
<tr>
<td>Rotation time</td>
<td>200 msec</td>
<td>8.33 msec</td>
</tr>
<tr>
<td>Motor stop/start time</td>
<td>250 msec</td>
<td>20 sec</td>
</tr>
<tr>
<td>Time to transfer 1 sector</td>
<td>22 msec</td>
<td>17 µsec</td>
</tr>
</tbody>
</table>

Disk parameters for the original IBM PC floppy disk and a Western Digital WD 18300 hard disk

---

### Disk Hardware – Multiple Zones

- **“Constant” bit density**: record more sectors on the outer tracks
  - Originally, all tracks have the same number of sectors
  - Controller maps \((x, y, z)\) to \((\text{cylinder, head, sector})\)

  - Left: physical geometry of a disk with two zones
  - Right: A possible virtual geometry (to OS) for this disk
RAID

- Redundant Array of Inexpensive/Independent Disks
  - Used with proper data distribution schemes for parallel-I/O

RAID levels 0 through 2
- Backup and parity drives are shaded

RAID (2)

- Raid levels 3 through 5
- Backup and parity drives are shaded
Optical Disks (Self-Reading)

- CD-ROM, CD-R/W, DVD

Recording structure of a CD or CD-ROM

Three-Stage Disk D/W Process

- Three-stage Disk R/W process: time required to read or write a disk block determined mainly by 3 factors
  1. Seek time
  2. Rotational delay
  3. Actual transfer time

- Seek time dominates

Disk Access Time = Seek time + Rotational Latency + Transfer time + Controller Time + Queueing Delay

How to optimize seek time?
- Disk arm scheduling algorithms
SSF Disk Arm Scheduling Algorithm

How to optimize seek time if a table available, indexed by cylinder number, with all the pending disk requests for each cylinder chained in a linked list headed by the table entries?

- Example: 11 (c), 1, 36, 16, 34, 9, 12 by FCFS and SSF
- SSF cuts average seek time of FCFS almost in half

The Elevator Disk Arm Scheduling Algorithm

Elevator algorithm: keep moving in the same direction until there are no more outstanding requests in that direction

- One current direction bit: UP and Down
- Property: given any collection of requests, the upper bound on the total motion is fixed: twice the number of cylinders
- Example: 11 (c), 1, 36, 16, 34, 9, 12 by FCFS and SSF
Error Handling

- Two general approaches to deal with bad blocks/sectors
  - disk controller based vs. OS based

- Controller does sector substitution
  - A disk track with a bad sector.
  - Substituting a spare for the bad sector
  - Shifting all the sectors to bypass the bad one

- OS does step (c)

Stable Storage (Self-Reading)

- When a disk write is issued, the disk either correctly writes the data or it does nothing, leaving the existing data intact.

- Analysis of the influence of crashes on stable writes
Clocks (Timers)

- Clock hardware: generate interrupts at known intervals
  - A programmable clock: its interrupt frequency can be controlled by software
  - A 500 MHz crystal with a 32-bit register can generate interrupt every 2ns to 8.6s

Clock Software

- Clock Software (clock driver):
  - Maintaining the time of day
  - support time-shared scheduling
  - Accounting CPU usage
  - Handling the *alarm* system call
  - ...
Maintaining the Time of Day

- How about incrementing a counter at each clock tick?
  - Given a clock rate of 60 Hz, a 32-bit counter last about 2 years

(a) Using 64-bits counter
(b) Recording time in terms of seconds with a second counter in ticks
(c) Relative to system boot time, instead of a fixed external moment

Multiple Timers with a Single Clock

- How to simulate multiple virtual clocks with a single physical clock?

Simulating multiple timers with a single clock
Soft Timers

- There is a need of a second hardware clock for timer interrupts at whatever rate a program needs
  - specified by applications
  - no problems if interrupt frequency is low

- Two ways of managing I/O
  - Polling has an average latency of half the polling interval
  - Interrupt can be costly, say 4.45 $\mu$s on a 300 MHz PII
  - A Giga-bit Ethernet asks for a packet every 12 $\mu$s

- Soft timers avoid costly interrupts
  - kernel checks for soft timer expiration before it exits to user mode
  - how well this works depends on rate of kernel entries!

The combination of soft timers and a low-frequency hardware timer may be better than pure interrupt-driven I/O or pure polling

Character Oriented Terminals (Self-Reading)

- An RS-232 terminal communicates with computer 1 bit at a time
- Called a serial line – bits go out in series, 1 bit at a time
- Windows uses COM1 and COM2 ports, first to serial lines
- Computer and terminal are completely independent
Graphical User Interface (GUI)

- GUI is based on WIMP paradigm
  - Windows, icons, menus, and a pointing device
  - GUI-based programs are generally event-driven

Memory-mapped displays: driver writes directly into display's video RAM

How much RAM needed for a 1024 x 768 pixel 24-bit color bitmap? How much for a 25 line x 80 row character monochrome text screen?

Network Terminals: X Windows

Clients and servers in the M.I.T. X Window System
**Power Management**

- Power consumption is important to desktops and battery-powered computer

- **Two General approaches to battery conservation**
  - Turn off some parts, top 3 power eaters: display, hard disk, and CPU
  - Application uses less power, multiple states: on/off, sleep, hibernating

<table>
<thead>
<tr>
<th>Device</th>
<th>Li et al. (1994)</th>
<th>Lorch and Smith (1998)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display</td>
<td>68%</td>
<td>39%</td>
</tr>
<tr>
<td>CPU</td>
<td>12%</td>
<td>18%</td>
</tr>
<tr>
<td>Hard disk</td>
<td>20%</td>
<td>12%</td>
</tr>
<tr>
<td>Modem</td>
<td>6%</td>
<td></td>
</tr>
<tr>
<td>Sound</td>
<td>2%</td>
<td></td>
</tr>
<tr>
<td>Memory</td>
<td>0.5%</td>
<td>1%</td>
</tr>
<tr>
<td>Other</td>
<td>22%</td>
<td></td>
</tr>
</tbody>
</table>

Power consumption of various parts of a laptop computer

Power-aware computing in mobile and sensor networks, many tradeoffs!

---

**OS Issues: The Display**

- **On-off switching**
  - Display consist of some zones that can be independently powered up or down
**OS Issues: The Hard Disk**

- It takes substantial energy to keep it spinning at high speed, even if there are no disk accesses.
- But restarting a hard disk consumes considerable energy.
  - What is the break-even point?
  - $T_d$ [5s, 15s]
  - If the next disk access is expected some time $t$ in the future, compare $t$ to $T_d$ determines the appropriate disk action.
- Alternative: have a substantial disk cache in RAM.
- Alternative: OS keeps running programs informed about the disk states, so that some accesses can be delayed (and clustered).

Something can be done to CPU too!

**OS Issues: The CPU**

- Running at full clock speed (full voltage)
- Cutting voltage by two
  - Cuts clock speed by two
  - Cuts power by four (power consumption $\sim$ voltage$^2$)
OS Issues: Wireless Communication

- The radio transmitter and receiver are power-consuming
- Radio on-off model:
  - How to exploit the fact that mobile computers communicate with fixed base stations that have large memories and disks and no power constraints?

Summary

- OS responsibilities in I/O operations
  - Protection and Scheduling
  - CPU communicates with I/O devices
  - I/O devices notify OS/CPU

- I/O software hierarchy
  - Interrupt handlers
  - Device drivers
  - Buffering

- Storage Systems
  - Disk head scheduling algorithms

- Power Management

- More reading: textbook 5.1 - 5.11