Chapter 2 – Hardware and Software Concepts

Outline

2.1 Introduction
2.2 Evolution of Hardware Devices
2.3 Hardware Components
2.3.1 Mainboards
2.3.2 Processors
2.3.3 Clocks
2.3.4 Memory Hierarchy
2.3.5 Main Memory
2.3.6 Secondary Storage
2.3.7 Buses
2.3.8 Direct Memory Access (DMA)
2.3.9 Peripheral Devices
2.4 Hardware Support for Operating Systems
2.4.1 Processor
2.4.2 Timers and Clocks
2.4.3 Bootstrapping
2.4.4 Plug and Play
2.5 Caching and Buffering
2.6 Software overview
2.6.1 Machine Language and Assembly Language
2.6.2 Interpreters and Compilers
2.6.3 High-Level Languages
2.6.4 Structured Programming
2.6.5 Object-Oriented Programming
2.7 Application Programming Interfaces (APIs)
2.8 Compiling, Linking and Loading
2.8.1 Compiling
2.8.2 Linking
2.8.3 Loading
2.9 Firmware
2.10 Middleware
Objectives

After reading this chapter, you should understand:
- hardware components that must be managed by an operating system.
- how hardware has evolved to support operating system functions.
- how to optimize performance of various hardware devices.
- the notion of an application programming interface (API).
- the process of compilation, linking and loading.

2.1 Introduction

An operating system is primarily a resource manager.
- Design is tied to the hardware and software resources the operating system must manage
  - processors
  - memory
  - secondary storage (such as hard disks)
  - other I/O devices
  - processes
  - threads
  - files
  - databases

2.2 Evolution of Hardware Devices

Most operating systems are independent of hardware configurations.
- Operating systems use device drivers to perform device-specific I/O operations
  - For example, plug-and-play devices when connected instruct the operating system on which driver to use without user interaction
2.2 Evolution of Hardware Devices

Figure 2.1 Transistor count plotted against time for Intel processors.

2.3 Hardware Components

- A computer’s hardware consists of:
  - processor(s)
  - main memory
  - input/output devices

2.3.1 Mainboards

- Printed Circuit Board
  - Hardware component that provides electrical connections between devices
  - The mainboard is the central PCB (printed circuit board) in a system
    - Devices such as processors and main memory are attached
    - Include chips to perform low-level operations (e.g., BIOS)
2.3.2 Processors

- A processor is hardware that executes machine-language
  - CPU executes the instructions of a program
  - Coprocessor executes special-purpose instructions
    - Ex., graphics or audio coprocessors
  - Registers are high-speed memory located on processors
    - Data must be in registers before a processor can operate on it
  - Instruction length is the size of a machine-language instruction
    - Some processors support multiple instruction lengths

2.3.3 Clocks

- Computer time is measured in cycles
  - One complete oscillation of an electrical signal
  - Provided by system clock generator
  - Processor speeds are measured in GHz (billions of cycles per second)
    - Modern desktops execute at hundreds of megahertz or several GHz
2.3.4 Memory Hierarchy

- The memory hierarchy is a scheme for categorizing memory
  - Fastest and most expensive at the top, slowest and least expensive at the bottom
    - Registers
    - L1 Cache
    - L2 Cache
    - Main Memory
    - Secondary and tertiary storage (CDs, DVDs and floppy disks)
  - Main memory is the lowest data referenced directly by processor
    - Volatile – loses its contents when the system loses power

Figure 2.3 Memory hierarchy.

- Assume that
  - L1 Cache: 10 ns (hit ratio 70%)
  - L2 Cache: 50 ns (hit ratio 80%)
  - Main Memory: 100 ns (hit ratio 90%)
  - HDD: 10 ms (hit ratio 100%)

- What would be the average memory access time?
  - \[0.7 \times 10\text{ns} + 0.3 \times (10\text{ns} + 0.8 \times 50\text{ns} + 0.2 \times (50\text{ns} + 0.9 \times 100\text{ns} + 0.1 \times (100\text{ns} + 10,000\text{ns})))\]
  - \[= 600,241\text{ ns}\]
2.3.5 Main Memory

- Main memory consists of volatile random access memory (RAM)
  - Processes can access data locations in any order
  - Common forms of RAM include:
    - dynamic RAM (DRAM) – requires refresh circuit
    - static RAM (SRAM) – does not require refresh circuit
    - SRAM is more expensive and faster than DRAM
  - RAM manufacturers try to narrow the gap between processor speed and memory-transfer speed
  - Bandwidth is the amount of data that can be transferred per unit of time

2.3.6 Secondary Storage

- Secondary storage stores large quantities of persistent data at low cost
  - Accessing data on a hard disk is slower than main memory
    - Mechanical movement of read/write head
    - Rotational latency
    - Transfer time
  - Removable secondary storage facilitates data backup and transfer
    - CDs (CD-R, CD-RW)
    - DVDs (DVD-R, DVD+R)
    - Zip disks
    - Floppy disks
    - Flash memory cards
    - Tapes

2.3.7 Buses

- A bus is a collection of traces
  - Traces are thin electrical connections that transport information between hardware devices
  - A port is a bus that connects exactly two devices
  - An I/O channel is a bus shared by several devices to perform I/O operations
    - Handle I/O independently of the system’s main processors
    - Example, the frontside bus (FSB) connects a processor to main memory
2.3.8 Direct Memory Access (DMA)

• DMA improves data transfer between memory and I/O devices
  – Devices and controllers transfer data to and from main memory directly
  – Processor is free to execute software instructions
  – DMA channel uses an I/O controller to manage data transfer
  – Notifies processor when I/O operation is complete
  – Improves performance in systems that perform large numbers of I/O operations (e.g., mainframes and servers)

2.3.9 Peripheral Devices

• Peripheral devices
  – Any device not required for a computer to execute software instructions
  – Internal devices are referred to as integrated peripheral devices
    • Network interface cards, modems, sound cards
    • Hard disk, CD and DVD drives
  – Character devices transfer data one bit at a time
    • Keyboards and mice
  – Can be attached to a computer via ports and other buses
    • Serial ports, parallel ports, USB, IEEE 1394 ports and SCSI
### 2.3.9 Peripheral Devices

**Figure 2.5 Peripheral devices (1 of 2).**

<table>
<thead>
<tr>
<th>Device</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CD-RW drive</td>
<td>Reads data from, and writes data to, optical disks.</td>
</tr>
<tr>
<td>Zip drive</td>
<td>Transfers data to, and from a removable, durable magnetic disk.</td>
</tr>
<tr>
<td>Floppy drive</td>
<td>Reads data from, and writes data to, removable magnetic disks.</td>
</tr>
<tr>
<td>Mouse</td>
<td>Transmits the change in location of a pointer or cursor in a graphical user interface (GUI).</td>
</tr>
<tr>
<td>Keyboard</td>
<td>Transmits characters or commands that a user types.</td>
</tr>
<tr>
<td>Multifunction printer</td>
<td>Can print, copy, fax and scan documents.</td>
</tr>
<tr>
<td>Sound card</td>
<td>Converts digital signals to audio signals for speakers. Also can receive audio signals via a microphone and produce a digital signal.</td>
</tr>
</tbody>
</table>

**Figure 2.5 Peripheral devices (2 of 2).**

<table>
<thead>
<tr>
<th>Device</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Video accelerator</td>
<td>Displays graphics on the screen; accelerates two- and three-dimensional graphics.</td>
</tr>
<tr>
<td>Network card</td>
<td>Sends data to and receives data from other computers.</td>
</tr>
<tr>
<td>Digital camera</td>
<td>Records, and often displays, digital images.</td>
</tr>
<tr>
<td>Biometric device</td>
<td>Scans human characteristics, such as fingerprints and retinas, typically for identification and authentication purposes.</td>
</tr>
<tr>
<td>Infrared device</td>
<td>Communicates data between devices via a line-of-sight wireless connection.</td>
</tr>
<tr>
<td>Wireless device</td>
<td>Communicates data between devices via an omnidirectional wireless connection.</td>
</tr>
</tbody>
</table>

### 2.4 Hardware Support for Operating Systems

- Computer architectures contain:
  - Features that perform operating system functions quickly in hardware to improve performance.
  - Features that enable the operating system to rigidly enforce protection.
2.4.1 Processor

• A processor implements operating system protection mechanisms
  – Prevents processes from accessing privileged instructions or memory
  – Computer systems generally have several different execution modes:
    • User mode (user state or problem state)
      – User may execute only a subset of instructions
    • Kernel mode (supervisor state)
      – Processor may access privileged instructions and resources on behalf of processes

Memory protection and management

• Prevents processes from accessing memory that has not been assigned to them
• Implemented using processor registers modified only by privileged instructions

Interrupts and Exceptions

• Most devices send a signal called an interrupt to the processor when an event occurs
• Exceptions are interrupts generated in response to errors
• The OS can respond to an interrupt by notifying processes that are waiting on such events

The Hardware Interfaces

• Modes of Instruction Execution
  – Some machine instructions should not be executed by ordinary applications
  – Examples:
    • Control I/O devices: may change the contents of the network packets
    • Set system registers or flags in the CPU: may cause a false arithmetic error that leads to terminating a process of interest
    • Manipulate system tables: may change the process table or page table to cause a false memory page fault
    • Modify external timer: may seize the control of the entire system by changing the value of the timer to disable a process scheduler
The Hardware Interfaces

- Modes of Instruction Execution
  - Only trusted system programs execute such instructions
  - The instruction is partitioned into multiple classes
    - Privileged instructions
      - Represented by a special bit in the CPU
      - An attempt to execute a privileged instruction causes a
        trap to handle the violation
    - Nonprivileged instructions
  - Examples
    - MS-DOS & 8088: no protection in modes
    - Windows 2000 & Pentium: use of dual modes

The Hardware Interfaces

- The lowest interface of the OS is between the hardware and the software
- Applications and OS compiled into machine instructions

- Interrupt
  - A hardware signal issued to the CPU from some external device
  - When an interrupt occurs, the CPU transfers control to a
    predetermined location (interrupt handler) ← a special routine of
    the OS
  - Interrupt handler analyzes the cause of the interrupt and takes
    appropriate actions
  - Interrupts are transparent to the application

Principles of Interrupts and Traps
The Hardware Interfaces

- Interrupts and Traps allow OS to seize control
  - A trap is a software-generated interrupt caused either by an error or a user request.
- Two main applications of the interrupt mechanism
  - Process management (time-sharing)
    - Time-out interrupt by a timer device
    - Scheduler is called to reevaluate the overall situation and decides which tasks to run next
  - Device management (I/O completion)
    - Most I/O devices are very slow compared with the CPU
    - When a device completes its operation, it generates an interrupt to let OS know that

The Hardware Interfaces

- Trap
  - While interrupts are caused by external devices, traps are triggered internal to the CPU
  - Handled by trap handler
  - Eg: overflow or underflow
    - Usually, the causing application cannot proceed beyond this point and it is suspended or terminated
  - Eg: memory management
    - When a page/segment fault occurs, trap is set and loading of pages are handled by traps
  - Eg: I/O operation
    - Voluntary, explicit transfer of the control to the OS
    - Supervisor Call (SVC)
      - Special machine instruction to set a trap and transfer the control to the OS
      - Implement all kernel calls and form the basic interface between the OS kernel and the rest of the software

Interrupt Time Line For a Single Process Doing Output
2.4.2 Timers and Clocks

• Timers
  – An interval timer periodically generates an interrupt
  – Operating systems use interval timers to prevent processes from monopolizing the processor

• Clocks
  – Provide a measure of continuity
  – A time-of-day clock enables an OS to determine the current time and date

2.4.3 Bootstrapping

• Bootstrapping: loading initial OS components into memory
  – Performed by a computer's Basic Input/Output System (BIOS)
  • Initializes system hardware
  • Loads instructions into main memory from a region of secondary storage called the boot sector
  – If the system is not loaded, the user will be unable to access any of the computer's hardware

Bootstrapping

• Bootstrapping (Booting)
  – A person attempting to pull himself by his own shoe laces or boot straps
  – ROM holds the initial startup software
2.4.3 Bootstrapping

Figure 2.6 Bootstrapping.

2.4.4 Plug and Play

- Plug and Play technology
  - Allows operating systems to configure newly installed hardware without user interaction
  - To support plug and play, a hardware device must:
    - Uniquely identify itself to the operating system
    - Communicate with the OS to indicate the resources and services the device requires to function properly
    - Identify the driver that supports the device and allows software to configure the device (e.g., assign the device to a DMA channel)

2.5 Caching and Buffering

- Caches
  - Relatively fast memory
  - Maintain copies of data that will be accessed soon
  - Increase program execution speed
  - Examples include:
    - L1 and L2 processor caches
    - Main memory can be viewed as a cache for hard disks and other secondary storage devices
2.5 Caching and Buffering

- Buffers
  - Temporary storage area that holds data during I/O transfers
  - Primarily used to:
    - Coordinate communications between devices operating at different speeds
    - Store data for asynchronous processing
    - Allow signals to be delivered asynchronously

- Spooling
  - Buffering technique in which an intermediate device, such as a disk, is interposed between a process and a low-speed I/O device
  - Allows processes to request operations from a peripheral device without requiring that the device be ready to service the request

2.6 Software Overview

- Programming languages
  - Some are directly understandable by computers, others require translation
  - Classified generally as either:
    - Machine language
    - Assembly language
    - High-level language

2.6.1 Machine Language and Assembly Language

- Machine language
  - Defined by the computer's hardware design
  - Consists of streams of numbers (1s and 0s) that instruct computers how to perform elementary operations
  - A computer can understand only its own machine language

- Assembly language
  - Represents machine-language instructions using English-like abbreviations
  - Assemblers convert assembly language to machine language
  - Speeds programming, reduces potential for bugs
2.6.2 Interpreters and Compilers

- High-level languages
  - Instructions look similar to everyday English
  - Accomplish more substantial tasks with fewer statements
  - Require compilers and interpreters

- Compiler
  - Translator program that converts high-level language programs into machine language

- Interpreter
  - Program that directly executes source code or code that has been reduced to a low-level language that is not machine code

2.6.3 High-level languages

- Popular high-level languages
  - Typically are procedural or object-oriented
  - Fortran
    - Used for scientific and engineering applications
  - COBOL
    - For business applications that manipulate large volumes of data
  - C
    - Development language of the UNIX OS
  - C++/Java
    - Popular object-oriented languages
  - C#
    - Object-oriented development language for the .NET platform

2.6.4 Structured programming

- Disciplined approach to creating programs
  - Programs are clear, provably correct and easy to modify
  - Structured programming languages include:
    - Pascal
      - Designed for teaching structured programming
    - Ada
      - Developed by the US Department of Defense
    - Fortran
### 2.6.5 Object-Oriented Programming

- **Objects**
  - Reusable software unit (any noun can be represented)
  - Easy to modify and understand
  - Have properties (e.g., color) and perform actions (e.g., moving)

- **Classes**
  - Types of related objects
  - Specify the general format of an object and the properties and actions available to it

- **Object-oriented programming**
  - Focuses on behaviors and interactions, not implementation
  - C++, Java and C# are popular object-oriented languages

### 2.7 Application Programming Interfaces (APIs)

- A set of routines
  - Programmers use routines to request services from the operating system
  - Programs call API functions, which may access the OS by making system calls
  - Examples of APIs include:
    - Portable Operating System Interface (POSIX) standard
    - Windows API

---

**Figure 2.7** Application programming interface (API).
The Programming Interfaces

• Invoking System Services
  – Library call (nonprivileged)
    • Eg: mathematical or string manipulation functions
  – Kernel call (privileged)
    • Eg: most of the I/O operations
    • The specific function sets provided by a given OS kernel and
      its libraries vary greatly between different OSs
    • MS Windows: kernel calls are completely hidden from the
      applications
    • UNIX: both library functions and kernel calls are known to the
      applications

• Class of system calls (kernel and library)
  – Process management
    • Create/destroy, suspend/activate process
    • Send signal or message to process
    • Wait for event or signal
  – Memory management
    • Allocate/deallocate memory space
    • Increase/decrease the size of allocated memory region
  – File management
    • Create/destroy, open/close, move, rename a file
    • Append/concatenate files
    • Display a file or directory
  – I/O
    • Read/write from a file or device
    • Control/check status of a device
  – Miscellaneous
    • Get current date and time
    • Mathematical and string manipulation functions
    • Various system diagnostics
2.8 Compiling, Linking and Loading

Before a high-level-language program can execute, it must be:
- Translated into machine language
- Linked with various other machine-language programs on which it depends
- Loaded into memory

Compiler, Assembler, Linker, and Loader

- **Compiler**
  - Generating relocatable machine code or assembly code

- **Assembler**
  - Generating relocatable machine code
  - Two pass assembly
    - **Pass one**:
      - Creating a symbol table to assign storage locations to identifiers
      - Names of data fields and program labels and their relative location
      - Determines the amount of code to be generated
    - **Pass two**:
      - Translating each operation code into the machine code
      - Translating each identifier into the address given for that identifier in the symbol table

- **Linker**
  - Making a single program from several files of relocatable machine code

- **Loader**
  - Altering the relocatable addresses
  - Placing the altered instructions and data in memory at the proper locations
Binding of Instructions and Data to Memory

Address binding of instructions and data to memory addresses can happen at three different stages.

- **Compile time:**
  - If memory location known a priori, absolute code can be generated;
  - must recompile code if starting location changes.

- **Load time:**
  - Must generate relocatable code if memory location is not known at compile time.

- **Execution time:**
  - Binding delayed until run time if the process can be moved during its execution from one memory segment to another. Need hardware support for address maps (e.g., base and limit registers). (most OSs use this method)
Preparing Program for Execution

- **Program Transformations**
  - Translation (Compilation)
  - Linking
  - Loading

![Diagram](image)

**Address Binding**

- **Loading a program into main memory**
  - What needs to be adjusted
    - Store, Load instructions (memory references)
    - Branch instructions (memory references)
  - **Address binding**
    - Assigning actual physical addresses to program instructions and data
    - Can take place at different points during a program life cycle
    - Before binding, the programmer, the compiler, the linker, or the loader work within an assumed logical address space
      - Programmers: symbolic names
      - Compiler, linker, loader: numerical values assuming a hypothetical starting address like 0

**Address Binding**

- **Assign Physical Addresses = Relocation**
  - Relocatable program: can be moved into different areas of memory without any transformations
  - Think about how we can copy a program from one computer to another and can run it without any transformation
- **Static binding**
  - Programming time
    - At the time of writing a program
    - Some assemblers and compilers permit the specification of an absolute (physical) address
    - Used relatively infrequently
      - The lowest kernel level of an OS
      - Real-time and embedded systems
      - Controlling a special hardware
Address Binding

- Loading time
  - More flexibility by separating the linker from the loader
  - The starting address needs not be known at the time of linking
  - When the program is actually loaded into the main memory, memory references are translated into physical memory addresses
  - Still not dynamically relocatable

• Dynamic binding
  - Execution time binding
  - The loader does not transform the program
  - All memory references are still the logical addresses
  - Actual translation is performed each time an instruction is fetched from memory and each time an operand is read or written to memory
  - Must be supported by the hardware for efficiency
Dynamic Address Binding using Relocation Register

Dynamic Binding = At Execution Time

Relocation Register = 1000 \leftarrow added to the addresses at address_map

Logical vs. Physical Address Space

- The concept of a logical address space that is bound to a separate physical address space is central to proper memory management.
  - Logical address – generated by the CPU; also referred to as virtual address.
  - Physical address – address seen by the memory unit.

- Logical and physical addresses are the same in compile-time and load-time address-binding schemes; logical (virtual) and physical addresses differ in execution-time address-binding scheme.

Memory-Management Unit (MMU)

- Hardware device that maps virtual to physical address.
- In MMU scheme, the value in the relocation register is added to every address generated by a user process at the time it is sent to memory.
- The user program deals with logical addresses; it never sees the real physical addresses.
2.8.1 Compiling

- Translating high-level code to machine code
  - Accepts source code as input and returns object code
  - Compilation phases include:
    - Lexer
      - Separates the characters of a program’s source into tokens
    - Parser
      - Groups tokens into syntactically correct statements
    - Intermediate code generator
      - Converts statements into a stream of simple instructions
    - Optimizer
      - Improves code execution efficiency and memory requirements
    - Code generator
      - Produces the object file containing the machine-language
2.8.2 Linking

- **Linkers**
  - Create a single executable unit
  - Integrate precompiled modules called libraries referenced by a program
  - Assign relative addresses to different program or data units
  - Resolve all external references between subprograms
  - Produce an integrated module called a load module
  - Linking can be performed at compile time, before loading, at load time or at runtime
2.8.2 Linking

Figure 2.11 Symbol resolution.

2.8.3 Loading

- Loaders
  - Convert relative addresses to physical addresses
  - Place each instruction and data unit in main memory

- Techniques for loading a program into memory
  - Absolute loading
    - Place program at the addresses specified by programmer or compiler (assuming addresses are available)
  - Relocatable loading
    - Relocate the program’s addresses to correspond to its actual location in memory
  - Dynamic loading
    - Load program modules upon first use

Figure 2.12 Loading.
2.8.3 Loading

Figure 2.13 Compiling, linking and loading.

2.9 Firmware

- Firmware contains executable instructions stored in persistent memory attached to a device
  - Programmed with microprogramming
    - Layer of programming below a computer’s machine-language
    - Microcode
      - Simple, fundamental instruction necessary to implement all machine-language operations

2.10 Middleware

- Middleware is software for distributed systems
  - Enables interactions among multiple processes running on one or more computers across a network
  - Facilitates heterogeneous distributed systems
  - Simplifies application programming
  - Example, Open Database Connectivity (ODBC)
    - Permits applications to access databases through middleware called an ODBC driver