An Introduction to System Integration

Gemini

Software Team

September 4, 2025

Outline

Chapter 1: The Heart of the Machine

- **Objective:** Understand the primary components of a single computer and how they interact.

Topics: Core Architecture

- **The Von Neumann Architecture:**
  - Central Processing Unit (CPU)
  - Main Memory (RAM)
  - Input/Output (I/O) Systems
- **A Deeper Look at the CPU:**
  - Control Unit (CU), Arithmetic Logic Unit (ALU), Registers
- **The Memory Hierarchy:**
  - L1/L2/L3 Cache
  - RAM (Random Access Memory)
  - Permanent Storage (SSDs, HDDs)

Diagram: Von Neumann Architecture

```
graph TD
    subgraph Computer
        CPU -- "fetches/stores" --> Memory;
        CPU -- "reads/writes" --> IO_Devices;
        IO_Devices -- "data" --> Memory;
    end
    CPU <--> CU & ALU & Registers;
    subgraph IO_Devices
        direction LR
        Input_Device --> CPU;
        CPU --> Output_Device;
    end
```

Diagram: Memory Hierarchy

```
graph TD
    A[CPU Registers] --> B(L1 Cache);
    B --> C(L2 Cache);
    C --> D(L3 Cache);
    D --> E(Main Memory - RAM);
    E --> F(Permanent Storage - SSD/HDD);
```

Real-World Application: Launching an App

- When you double-click an icon:
- The OS finds the program on your **Permanent Storage (SSD)**.
- It loads the program into **Main Memory (RAM)**.
- The **CPU** fetches instructions from RAM into its **Caches** to execute the program.

- **Objective:** Explore how modern processors handle multiple tasks and data streams simultaneously.

Topics: Flynn's Taxonomy

- **SIMD (Single Instruction, Multiple Data):**
  - One instruction is applied to many different data points at once.
  - **Analogy:** A drill sergeant telling a whole platoon to "turn left".
- **MIMD (Multiple Instruction, Multiple Data):**
  - Multiple processors execute different instructions on different data streams.
  - **Analogy:** A workshop with multiple craftspeople working on different projects.

Diagram: SIMD vs MIMD

```
graph TD
    subgraph SIMD
        direction LR
        Instruction_SIMD --> Data1;
        Instruction_SIMD --> Data2;
        Instruction_SIMD --> Data3;
    end
    subgraph MIMD
        direction LR
        Instruction1_MIMD --> DataA;
        Instruction2_MIMD --> DataB;
        Instruction3_MIMD --> DataC;
    end
```

Real-World Application: SIMD vs MIMD

- **SIMD:** Applying a brightness filter in a photo editor. The same instruction ("increase brightness") is applied to every pixel at once.
- **MIMD:** A web server handling multiple user requests simultaneously. Each core processes a different request.

Chapter 3: Connecting the Dots

- **Objective:** Understand how different computer systems communicate with each other over a network.

Topics: The TCP/IP Model

A 5-Layer View for simplifying the complexity of networking:

- Layer 5: Application (HTTP, DNS)
- Layer 4: Transport (TCP, UDP)
- Layer 3: Network (IP)
- Layer 2: Data Link (Ethernet, Wi-Fi)
- Layer 1: Physical (Cables, Radio Waves)

Diagram: TCP/IP Model

```
graph TD
    subgraph Your Computer
        A[Application] --> B(Transport);
        B --> C(Network);
        C --> D(Data Link);
        D --> E(Physical);
    end
    subgraph Web Server
        F[Application] --> G(Transport);
        G --> H(Network);
        H --> I(Data Link);
        I --> J(Physical);
    end
    E -- The Internet --> J;
```

- **Objective:** Trace a single, common action from start to finish to see how all the concepts interact.
- **Scenario:** "Loading google.com in a web browser."

Diagram: Loading google.com

```
sequenceDiagram
    participant User
    participant Browser
    participant OS
    participant DNS_Server
    participant Google_Server

    User->>Browser: Enters "google.com"
    Browser->>OS: Need IP for "google.com"
    OS->>DNS_Server: Where is "google.com"?
    DNS_Server-->>OS: IP is 142.250.190.78
    OS-->>Browser: Here is the IP
    Browser->>Google_Server: HTTP GET request
    Google_Server-->>Browser: HTTP 200 OK
    Browser->>User: Renders the webpage
```

Chapter 5: Network Communication

- **Objective:** Explore common methods and patterns for communication between systems.

Topics: Communication Patterns

- **HTTP Communication:** Request/Response model.
- **Sockets:** Low-level, bidirectional communication.
- **Web Servers:** Role of servers like Nginx.
- **Message Queues:** Decoupling systems.
- **Publish-Subscribe Pattern:** Scalable messaging.

Diagram: Message Queue vs Pub/Sub

```
graph TD
    subgraph Message Queue
        Producer --> Queue((Queue));
        Queue --> Consumer;
    end
    subgraph Publish-Subscribe
        Publisher --> Broker((Topic));
        Broker --> Subscriber1;
        Broker --> Subscriber2;
    end
```

- **Objective:** Understand different models for executing multiple tasks at the same time.

Topics: Concurrency Models

- **Concurrent Processing (Threads):** Independent execution paths in one process. Feels simultaneous.
- **Parallel Processing:** Truly simultaneous execution on multiple cores.
- **Asynchronous Processing (Async):** Non-blocking operations.

Diagram: Concurrency vs Parallelism

```
graph TD
    subgraph Concurrency (1 Core)
        direction LR
        Core1 -- Task A --> Switch;
        Switch -- Task B --> Switch;
        Switch -- Task A --> ...;
    end
    subgraph Parallelism (2 Cores)
        direction LR
        CoreA -- Task A --> DoneA;
        CoreB -- Task B --> DoneB;
    end
```

- **Objective:** Gain a foundational understanding of the role of the Operating System.

Topics: OS Core Functions

- Process Management
- Memory Management
- File Systems
- I/O Handling
- Windows vs. Linux
- Hyper-Threading

Diagram: OS Kernel

```
graph TD
    subgraph OS Kernel
        Scheduler --> P1(Process 1);
        Scheduler --> P2(Process 2);
        MemoryManager --> P1_Mem;
        MemoryManager --> P2_Mem;
    end
    subgraph Hardware
        CPU & RAM & Disk;
    end
    OS_Kernel -- "manages" --> Hardware;
```

Chapter 8: Virtualization

- **Objective:** Understand how we create virtual environments to run software.

Topics: Virtualization Types

- **Virtual Machines (VMs):** Emulating an entire computer system (hardware + OS).
- **Containers:** OS-level virtualization, packaging an application and its dependencies (e.g., Docker).
- Key differences: Overhead, startup time, density.

Diagram: VMs vs Containers

```
graph TD
    subgraph Physical Server
        HW[Hardware] --> HostOS;
        subgraph VM Approach
            HostOS --> Hypervisor;
            Hypervisor --> GuestOS_A;
            Hypervisor --> GuestOS_B;
            GuestOS_A --> App_A;
            GuestOS_B --> App_B;
        end
        subgraph Container Approach
            HostOS --> ContainerEngine;
            ContainerEngine --> App_C;
            ContainerEngine --> App_D;
        end
    end
```

Thank You