Distributed Systems

Christian J. Rudder

January 2025

Contents

This page is left intentionally blank.

Big thanks to **Professor Ioannis Liagouris**
for teaching CS351: Distributed Systems
at Boston University [1].

All illustration contain original assets.

Prerequisites

Introduction

1.1 High-level Computer Architecture Overview

System Review

To understand distributed systems, we must first review the architecture of a single computer.

Definition 1.1: Turing Machine

Conceptualized by Alan Turing in 1936, a Turing machine is a mathematical model of computation that defines an abstract machine that manipulates symbols on a strip of tape according to a table of rules. Despite its simplicity, the machine can simulate the logic of any computer algorithm.

Definition 1.2: Vaun Neumann Architecture

The Von Neumann architecture, also known as the Princeton architecture, is a design architecture for an electronic digital computer with these components:

- **A processing unit** that contains an arithmetic logic unit and a control unit.

- **A memory unit** that stores data and instructions.

- **Input and output mechanisms**.

Fast forward, modern computers have the following components:

Definition 1.3: Modern Computer Components

- **CPU:** Central Processing Unit. The brain of the computer that performs instructions.

- **Memory:** Stores data and instructions.

- **Storage:** Hard drives, SSDs, etc.

- **Network Interface:** Connects the computer to the network.

- **Input/Output Devices (I/O):** Keyboard, mouse, monitor, etc.

- **Motherboard:** The central printed circuit board that interconnects all of the computer's components, including the CPU, storage devices, and I/O devices.

Before diving deeper into the inner workings of a single computer, let's define a distrusted system:

Definition 1.4: Distributed System

A distributed system is a system whose components are located on different networked computers, which communicate and coordinate their actions by passing messages to one another. The components interact with one another in order to achieve a common goal.

In the words of Andrew S. Tanenbaum,

"A set of nodes, connected by a network, which appear to its users as a single coherent system."

---

or in the words of Leslie Lamport,

A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable.

Tip: **Andrew S. Tanenbaum** is a computer scientist and professor emeritus at the Vrije Universiteit Amsterdam in the Netherlands who is best known for his books on computer science. **Leslie Lamport** is an American computer scientist known for his work in distributed systems and as the initial developer of the document preparation system LaTeX.

CPU and Memory Orchestration Review

Now at a high-lever, we discuss how the a system interacts with all its components to perform tasks.

Definition 1.5: CPU (Central Processing Unit)

The CPU is made of the following components:

- **ALU (Arithmetic Logic Unit):** Performs arithmetic and logical operations.

- **Control Unit:** Manages the execution of instructions.

- **Registers:** Small, fast storage locations in the CPU that temporarily hold data and instructions.

Definition 1.6: Memory Segments

A program's memory is typically divided into several segments:

- **Text Segment:** Contains the executable code.

- **Data Segment:** Stores global and static variables.

- **System Stack:** A memory region that manages temporary data related to function calls in a first-in-last-out manner.

- **System Heap:** A memory region that dynamically allocates references to data from the stack memory.

Definition 1.7: Instruction Execution Cycle

The instruction execution cycle, also known as the *fetch-decode-execute* cycle, is the process by which the CPU processes instructions. In each cycle:

1. **Fetch:** The CPU retrieves an instruction from memory using the *instruction pointer* (or program counter).

2. **Decode:** The instruction is interpreted to determine what action is required.

3. **Execute:** The CPU performs the instruction's operation, which may involve arithmetic calculations, memory accesses, or I/O operations.

The CPU performs instructions via the following steps:

Definition 1.8: CPU Registers

Registers are small, high-speed storage locations within the CPU that temporarily hold data, instructions, and control information. Key registers include:

- **Instruction Pointer (Program Counter):** Holds **addresses**, which are the locations of the next instruction to fetch.

- **Stack Pointer:** Points to the top of the current stack in memory.

- **General-Purpose Registers:** Used for arithmetic operations and temporary data storage.

Definition 1.9: RAM and Volatile Memory

RAM (Random Access Memory) is a type of volatile memory used to store data and instructions that are actively used by the CPU. Since it is volatile, its contents are lost when the computer is powered off.

Definition 1.10: Physical Storage and I/O Devices

Physical storage refers to non-volatile memory devices such as hard drives and SSDs, which retain data without power. Many of these devices are accessed via input/output (I/O) operations and are thus considered part of the system's I/O mechanism.

Definition 1.11: Virtual Memory and Address Translation

Virtual memory is a memory management technique that provides an abstraction of a large, contiguous memory space. It works by mapping virtual addresses used by programs to physical addresses in RAM via structures such as page tables, which are managed by the Memory Management Unit (MMU).

Definition 1.12: CPU Cores

A CPU core is a physical processing unit within a central processing unit (CPU) responsible for executing instructions and performing computations. Modern CPUs often contain multiple cores, enabling them to handle multiple tasks at the same time.

Definition 1.13: Task, Job, and Process

- A **Task** is a single unit of work in various states (waiting, running, completed).

- A **Job** is a high-level operation comprising multiple tasks.

- A **Process** is an executing instance of a program that manages system resources instructing the CPU to execute tasks.

Definition 1.14: Threads: Concurrency & Parallelism

A **thread** is a unit of logic (a segment of code) to be executed by a CPU core. **A core can only run one thread at a time**. The core itself is called the **hardware-thread**, while our units of logic are called **software-threads** or **OS-threads**.

The OS system scheduler manages the hardware-threads, and assigns software-threads to them. Switching between software-threads on a hardware-thread is called **context switching**. Context switching is expensive, as it requires saving the current state of the software-thread and loading the state of the new software-thread. Though it provides the illusion of tasks running simultaneously, called **concurrency**.

With multiple cores come **multi-threading**, where multiple threads run on other cores simultaneously. This true simultaneity is called **parallelism**.
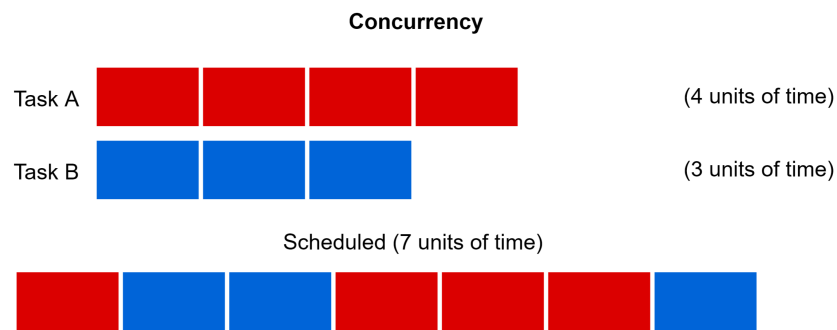

Concurrency

| Task A | | (4 units of time) |
| Task B | | (3 units of time) |

Scheduled (7 units of time)

Figure 1.1: Concurrency: Multiple software-threads running on a single hardware-thread.

In reality, many tasks preform I/O operations, which don't concern the CPU:

Concurrency

| Task A | 1/2 | (4 units of time) |
| Task B | 1/2 1/2 1/2 | (3 units of time) |

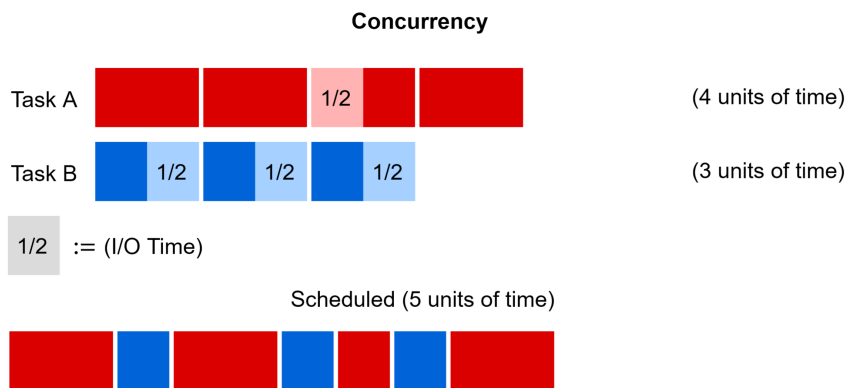1/2 := (I/O Time)

Scheduled (5 units of time)

Figure 1.2: Concurrency with I/O: Multiple software-threads running on a single hardware-thread.

Now since the CPU isn't idle on I/O operations, the overall time between tasks is cut significantly.

Definition 1.15: Kernel

The kernel is the central component of the operating system. It manages hardware resources—including the CPU, memory, and I/O devices—and provides core services such as process management, memory management, and device control.
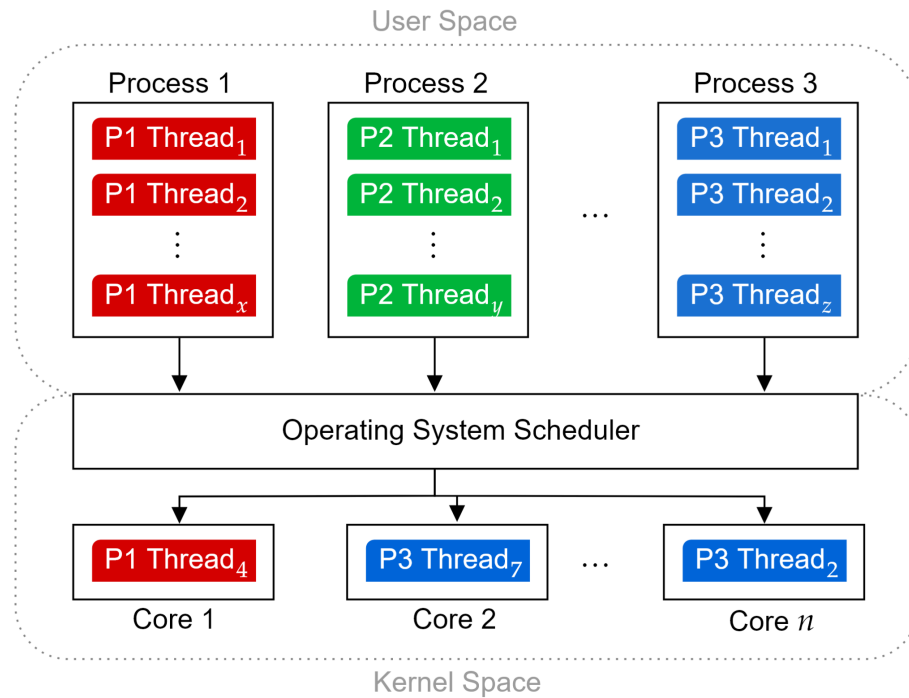
Figure 1.3: Process threads scheduled by the OS kernel and processed by the CPU.

Motivation for Distributed Systems

Distributed systems cover a vast and diverse range of applications, including:

- **Offloading Computation:** Perhaps a system A offloads a heavy computation to system B.

- **Fault Tolerance:** If a critical system A fails, an almost identical system B can take over.

- **Load Balancing:** Say a system A is overwhelmed with requests, it can distribute the load to system B, acting as one system, from the requests point of view.

In todays market there are numerous applications of distributed systems, such as: Cloud Computing, Social Networks, E-commerce, Streaming Services, Search Engines, Renting Computation (AI training), etc.

Let's begin to define the problem space. Say there be two individuals Alice and Bob, who wish to communicate:

Figure 1.4: Alice sends a letter m_1 overseas to Bob.

How does Alice know that her message m_1 was received by Bob? Bob would have to send a message back to Alice, acknowledging the receipt of m_1.
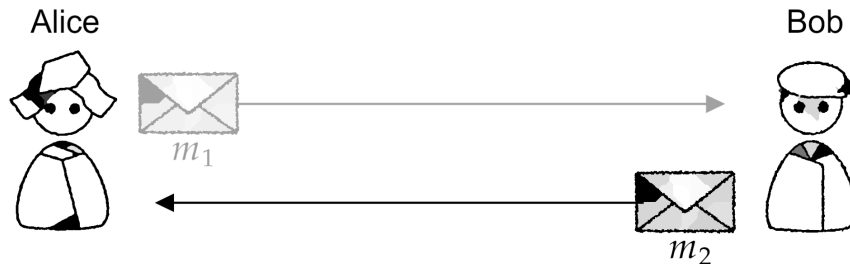

Figure 1.5: Bob sends an acknowledgment letter back to Alice.

Though problems can arise, what if Alice's letter gets lost in the mail, what if Bob receives multiple letters from Alice, how does Bob know which letter is the most recent? These are the fundamental problems of distributed systems.

Bibliography

[1] Ioannis Liagouris. Cs351: Distributed systems. Lecture notes, Boston University, Spring Semester, 2025. Boston University, CS Department.