Distributed Systems

Christian J. Rudder

January 2025

Contents

This page is left intentionally blank.

Big thanks to **Professor Ioannis Liagouris**
and **Dr. Anna Arpaci-Dusseau** for teaching CS351: Distributed Systems
at Boston University [2].

All illustration contain original assets.

Disclaimer: These notes are my personal understanding and interpretation of the course material. They are not officially endorsed by the instructor or the university. Please use them as a supplementary resource and refer to the official course materials for accurate information.

Prerequisites

This text assumes the reader has a basic understanding of computer science and programming. It will also assume they are somewhat familiar with computer architecture and operating systems at a high level. The text will review these concepts briefly for completeness, but it will not try to teach them from scratch or provide a full understanding of these topics.

The main focus will be on distributed systems, and will touch on:

- **Concurrency and Parallelism**
  - Concurrency, Parallelism, Threads
- **Consistency and Fault Tolerance**
  - Consistency, Fault-tolerance, Atomicity
- **Distributed Systems and Coordination**
  - Asynchrony, Coordination, Logical Time, Snapshots
- **Consensus Algorithms**
  - Raft, Paxos, Consensus
- **Replication and Data Management**
  - Replication, Sharding, Cluster
- **Protocols and Computing Models**
  - RPC, 2PC, Broadcast
- **Technologies and Tools**
  - MapReduce, Spanner, Dynamo, GFS, TLA+, Golang

— 1 —

Introduction

1.1 Virtual Memory*

1.1.1 Problem Space

This section is a quick aside to understand the problem we are going to solve in the next section when we talk about distributing shared memory [1]. Virtual memory solves three core problems:

- **Not enough memory, Memory fragmentation, and Security**

Definition 1.1: Not Enough Memory

Back then, computer memory were expensive, and many computers had very little memory (e.g., 4–1 GB or even less). CPUs could only support up to 4 GB of memory, as CPUs were 32-bit (2^{32} addresses $= 2^{32} bytes = 4GB$). On the other hand, 64-bit CPUs can support up to 2^{64} addresses = 16 million TB of memory.

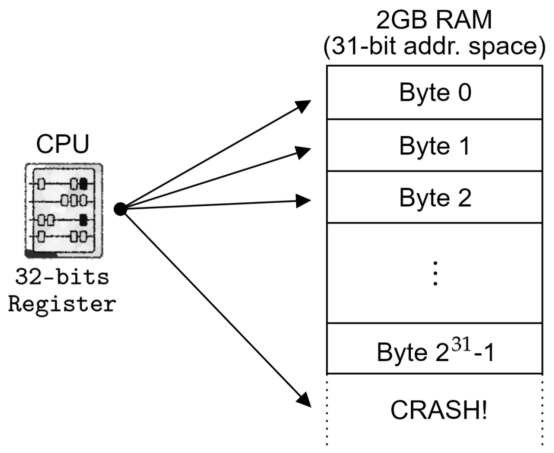

Figure 1.1: A 32-bit CPU accessing 2 GBs of RAM, where a crash happens when trying to access beyond the 31-bit address space.

The next problem deals with multiple processes allocating and deallocating memory:

Definition 1.2: Memory Fragmentation

Memory can be thought of as a big array, where each cell is a resource a program can use. We want memory usage to be contiguous (i.e., no gaps or holes). So say we have an array

$$[O, O, O, O]$$

Where O represents free space in our array, each cell 1 GB of space. If we have processes A and B take 1 and 2 GBs respectively, we might have a memory layout of:

$$[A, B, B, O,]$$

If we then free process A, we might have a memory layout of:

$$[O, B, B, O]$$

Now, if another process C needs 2 GBs of memory, it will not be able to find a contiguous space of 2 GBs, even though we have 2 GBs of free space. This is called **memory fragmentation**.

Now finally we have the problem of protecting memory from other processes:

Definition 1.3: Memory Security

In a multi-process system, processes may have collisions when trying to access the same memory space. For example, if process A is a weather service and process B is some finance service, we don't want the weather service to overwrite the same memory space where the finance service is storing critical data. This is called **memory security**.

So in theory we want to give each process it s own portion of memory, to solve overlapping access:

Definition 1.4: Virtual Memory

To solve process memory collisions, we give each process its own fictional view of memory, called **virtual memory**. Though for this to work, each virtual view is mapped to an actual place in the original memory we call **physical memory**.

Virtual and physical addresses are the cells spaces themselves.

Consider the figure below showing how virtual memory works in theory:

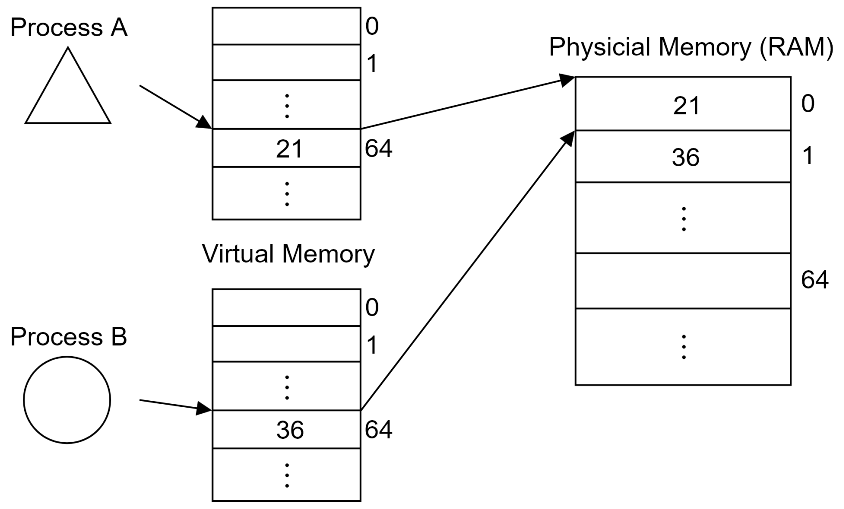

Figure 1.2: Processes A and B write to memory cell 64 in their view of memory, but in reality they map to different physical memory cells (0 and 1 respectively).

A quick aside:

Theorem 1.1: Physical Memory the CPU Accesses

In reality the CPU can access the physical memory of many other devices (e.g., hard drives, SSDs, etc.). In addition, the OS takes up some of the physical memory as well. The rest is left to programs to use. The program allocated space is the memory we refer to going forward as the **physical memory**.

Virtual memory solves the three problems we mentioned before:

Definition 1.5: Virtual Memory & Not Enough Memory (Swapping)

The physical memory can be much smaller than what a program thinks it has in virtual memory. When a program tries to access memory it does not have, the OS will **swap** physical memory to external storage to free up space. This means, while a program is not using a portion of memory at a given time, the OS can swap it in and out depending on system needs.

Memory that is swapped out is called **swap-memory**. Every time a we try to access such absent memory in our mappings, it's called a **page fault** (more on this later).

Definition 1.6: Virtual Memory & Fragmentation

Virtual memory allows programs to think they have contiguous memory. This simplifies their memory management, while in reality the OS manages the split memory mappings.

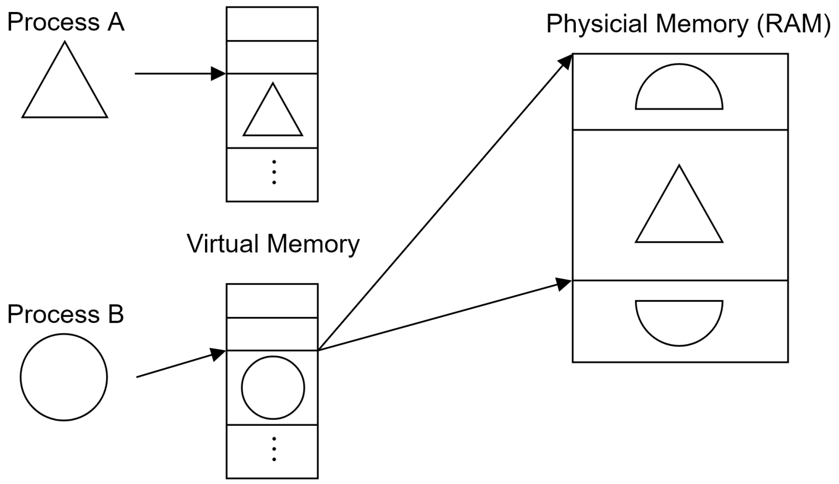

Figure 1.3: Here two processes A and B are using the same physical memory space. Both think they have contiguous memory, but in reality, B is split into two parts in the physical memory.

Definition 1.7: Virtual Memory & Security

Memory cells are collision safe as memory mappings are not shared in physical memory; However, this would be inefficient if say A and B depend on a secondary process C. In this case, A and B may share the same mapping to C's memory, but not to each other.

1.1.2 Virtual Memory Implementation (Page Tables)

We discuss the mapping mechanism further in linking virtual to physical memory.

Definition 1.8: Page Tables

The OS keeps a **page table**, where each entry is a mapping of a virtual memory cell to a physical one, called a **Page Table Entry (PTE)**. CPUs work with words (32 bits = 4 bytes), so the page table has one entry for every word (address) in the virtual memory space. If there are 2^{32} addresses, then there are 2^{30} words, and thus ≈ 1 billion PTEs (4 GB per table).

Bibliography

[1] Computerphile. But, what is virtual memory?, 2023. Accessed: 2025-04-18.

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