Chapter 18: Paging: Introduction

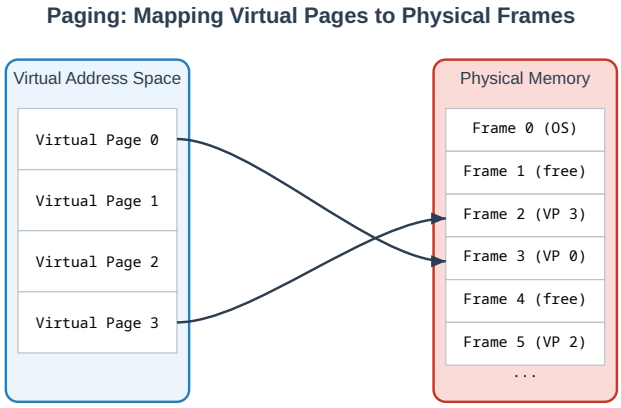
This chapter introduces **paging**, a memory management technique that forms the foundation of virtually all modern virtual memory systems. Unlike segmentation, which divides memory into variable-sized logical chunks, paging divides memory into fixed-sized units, solving many of the problems inherent in segmentation.

> **The Crux of the Problem: How to Virtualize Memory with Pages?**
>
> How can we virtualize memory with pages, so as to avoid the problems of segmentation? What are the basic techniques? How do we make those techniques work well, with minimal space and time overheads?

1. The Core Idea: Fixed-Sized Units

The fundamental idea of paging is to divide memory into fixed-sized pieces.

- **Page:** The virtual address space of a process is chopped up into fixed-sized blocks called **pages**.
- **Page Frame:** Physical memory is viewed as an array of fixed-sized slots called **page frames**.
- **Core Principle:** Each virtual page from a process's address space can be mapped to *any* physical page frame in memory. This allows for great flexibility, as the physical placement of a process's pages does not need to be contiguous.



Advantages of Paging

- **Flexibility:** The system can effectively support any usage pattern of an address space (e.g., large, sparse heaps) because only the pages that are actually used need to be placed in physical memory.

- **Simple Free-Space Management:** Physical memory is just a list of free page frames. There is no **external fragmentation**, as all units (pages and frames) are the same size. Allocating memory is as simple as taking a free frame from the list.

---

2. The Page Table: Storing Translations

To manage these mappings, the OS uses a per-process data structure called a **page table**.

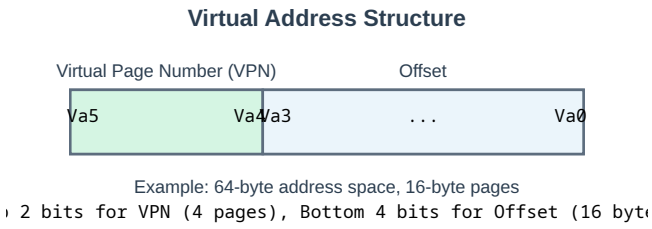
- **Purpose:** The page table stores the virtual-to-physical address translations for each page of a process's address space.
- **Function:** It maps a **Virtual Page Number (VPN)** to a **Physical Frame Number (PFN)**.
- **Example:** For the system shown above, the page table would contain entries like:
  - (VPN 0 → PFN 3)
  - (VPN 1 → PFN 7)
  - (VPN 2 → PFN 5)
  - (VPN 3 → PFN 2)

---

3. The Address Translation Process

When a program generates a memory reference, the hardware and OS work together to translate the virtual address into a physical address.

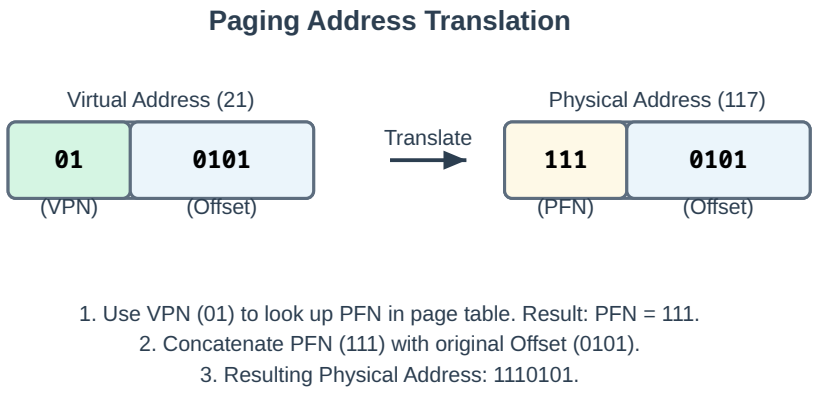
1. **Split the Virtual Address:** The hardware splits the virtual address into two parts:
   - **Virtual Page Number (VPN):** The high-order bits of the address, which identify the page.
   - **Offset:** The low-order bits of the address, which identify the byte within that page.



2. **Use VPN to Index the Page Table:** The hardware uses the VPN as an index into the process's page table to find the corresponding **Page Table**

Entry (PTE).

3. **Extract PFN from PTE:** The hardware extracts the Physical Frame Number (PFN) from this PTE.
4. **Construct the Physical Address:** The hardware constructs the final physical address by combining the PFN with the original offset. The PFN forms the high-order bits of the physical address, and the offset forms the low-order bits.



4. Page Table Structure and Contents

4.1. Where are Page Tables Stored?

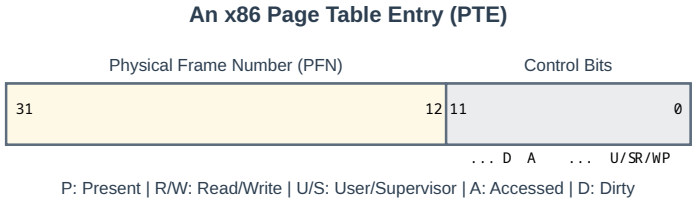
- Page tables can be very large. For a 32-bit address space with 4KB pages, the page table would have 2^{20} (over a million) entries. At 4 bytes per entry, this is 4MB *per process*.
- Because of their size, page tables are **stored in main memory**, managed by the OS. The hardware only needs to know the physical address of the start of the current process's page table, which is typically stored in a special CPU register called the **page-table base register**.

4.2. What's in a Page Table Entry (PTE)?

A Page Table Entry (PTE) is more than just a PFN. It contains several control bits that enable protection and more advanced memory management.

- **Valid Bit:** Indicates whether this page table entry is valid. If a process tries to access a page with an invalid PTE (e.g., the unused space between the heap and stack), it triggers a trap to the OS. This is crucial for supporting sparse address spaces efficiently.
- **Protection Bits:** Specify the permissions for the page (e.g., Read, Write, Execute). An attempt to perform an unauthorized action (like writing to a read-only page) will trap to the OS.

- **Present Bit:** Indicates whether the page is currently in physical memory or has been temporarily moved (swapped out) to disk.
- **Dirty Bit:** Indicates if the page has been modified since it was loaded into memory.
- **Reference Bit (Accessed Bit):** Tracks whether the page has been recently accessed. This is useful for page replacement algorithms.



5. The Problem: Paging is Too Slow

While flexible, this simple paging mechanism introduces a significant performance problem.

- **The Overhead:** To perform a single memory access (e.g., `movl 21, %eax`), the system must now perform *multiple* memory accesses:
  1. **First, an access to the page table** to fetch the PTE containing the translation.
  2. **Then, a second access to physical memory** to fetch the actual data at the translated address.
- **Impact:** This effectively doubles the number of memory accesses, potentially halving the speed of the program. This is unacceptably slow.

---

6. Summary

Paging is a powerful memory virtualization technique that offers great **flexibility** and solves the **external fragmentation** problem by dividing memory into fixed-sized pages. It works by using a per-process **page table** to translate virtual addresses into physical addresses.

However, this basic mechanism introduces two major challenges:

1. **Performance:** The extra memory access required for translation on every instruction fetch, load, and store makes the system too slow.
2. **Space:** Page tables themselves can consume a large amount of memory.

The next chapters will introduce the hardware and software techniques, most notably the **Translation Lookaside Buffer (TLB)**, designed to overcome these performance and space overheads and make paging a practical and efficient solution.