Chapter 21: Beyond Physical Memory: Mechanisms

This chapter explores the mechanisms that enable an Operating System to provide the illusion of a virtual address space that is much larger than the available physical memory. This is achieved by using a slower, larger storage device, like a hard disk, as an overflow area for memory pages.

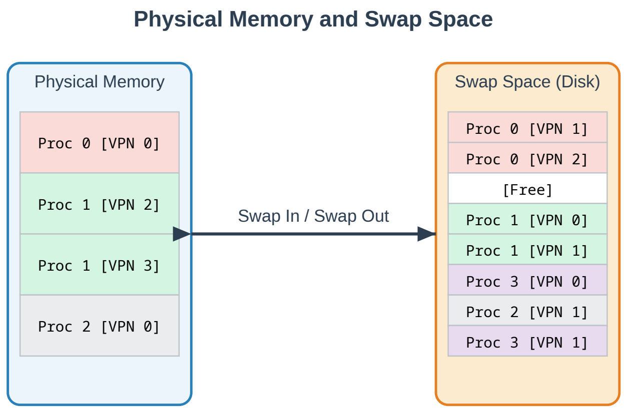
> **The Crux of the Problem: How to Go Beyond Physical Memory?**
>
> How can the OS make use of a larger, slower device to transparently provide the illusion of a large virtual address space?

The ability to support large address spaces is a major convenience for programmers, who no longer need to manually manage moving code and data in and out of memory (a technique known as **memory overlays**). It also enables efficient **multiprogramming**, as the OS can run more processes than can fit entirely into physical memory at once.

1. Swap Space

To support address spaces larger than physical memory, the OS needs a place on a larger, slower storage device (like a hard disk or SSD) to store pages that are not currently in memory.

- **Definition:** This reserved area on disk is called **swap space**.
- **Function:** The OS **swaps out** pages from physical memory to swap space to free up frames, and **swaps in** pages from swap space back to memory when they are needed.
- **OS Responsibility:** The OS must keep track of the disk address for each swapped-out page.



2. The Present Bit

To support swapping, the hardware and OS need a way to know whether a page is in physical memory or on disk. This is accomplished by adding a **present bit** to each Page Table Entry (PTE).

- **Mechanism:**
  - If the **present bit is 1**, the page is in physical memory. The PTE contains the valid Physical Frame Number (PFN), and address translation proceeds normally.
  - If the **present bit is 0**, the page is *not* in physical memory; it resides in swap space. Accessing this page triggers a special type of trap to the OS known as a **page fault**.

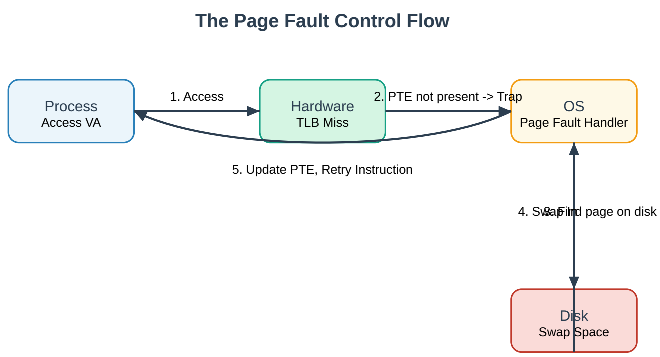
---

3. The Page Fault

A page fault is not necessarily an error. It is the mechanism that allows the OS to transparently bring pages into memory from the disk on demand.

- **Trigger:** A page fault occurs when a program tries to access a valid page that is marked as *not present* (present bit = 0).
- **Handling:** The hardware traps to the OS, which executes a special **page-fault handler** to service the fault. The handler performs the following steps:
  1. It uses the information in the PTE (where the disk address of the page is now stored) to locate the page in swap space.
  2. It finds a free physical frame to load the page into.
  3. If no frames are free, the OS must run a **page-replacement policy** to evict a page from memory, potentially writing it to swap space if

it has been modified.
4. It issues an I/O request to read the desired page from disk into the now-free physical frame.
5. While the slow disk I/O is in progress, the process is put in the **blocked** state, and the OS schedules another process to run.
6. When the I/O completes, the OS updates the page table: it sets the present bit to 1 and updates the PFN to point to the new physical frame.
7. Finally, the OS retries the instruction that caused the fault. This time, the translation will succeed (though it may cause a TLB miss first).



4. Page Replacement

A critical part of handling a page fault is deciding which page to evict if memory is full.

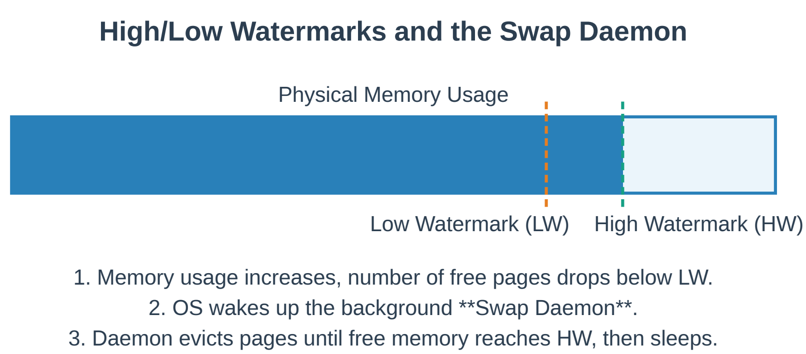
- **Page-Replacement Policy:** The algorithm the OS uses to select a "victim" page to remove from memory.
- **Importance:** A good policy is crucial for performance. Evicting a page that will be needed again soon will cause another page fault, dramatically slowing down the system. This topic is so important it will be covered in detail in a later chapter.

---

5. Proactive Page Replacement: Swap Daemon

Waiting until memory is completely full to evict a page is inefficient. Most modern OSes are more proactive.

- **High and Low Watermarks:** The OS defines two thresholds for memory usage: a **low watermark (LW)** and a **high watermark (HW)**.
- **Swap Daemon:** A background thread in the OS, also known as the **page daemon**, monitors the amount of free memory.
  - When the number of free pages drops below the `LW`, the daemon wakes up.
  - It evicts pages from memory until the number of free pages rises to the `HW`.
  - It then goes back to sleep.
- **Benefits:**
  - Ensures there are always free pages available for page faults, reducing latency.
  - Allows the OS to **cluster** or group writes to the swap disk, which is much more efficient than writing out single pages one at a time.



6. Summary

The OS creates the illusion of a vast virtual memory by using a slower, larger disk as **swap space**.

- The **present bit** in each PTE indicates whether a page is in physical memory or on disk.
- An access to a non-present page triggers a **page fault**, a trap to the OS.
- The OS **page-fault handler** brings the required page from disk into memory.
- If memory is full, a **page-replacement policy** decides which page to evict.
- Modern systems use background **swap daemons** and **watermarks** to proactively manage free memory.

All of these complex mechanisms work together **transparently**. From the process's perspective, it is simply accessing its own private, contiguous virtual memory, unaware that pages are being moved between memory and disk behind the scenes.