Memory Management

Memory management efficiently allocates RAM and prevents programs from interfering with each other's memory space. It ensures smooth program execution and effective use of this resource.

Logical and Physical Address Space

Logical Address Space (Virtual Address Space):

- Set of all logical addresses generated by the CPU for a program.
- **Virtual:** Doesn't directly correspond to physical memory locations.
- **Program-centric view:** Designed for programmer convenience, independent of physical memory layout.
- **Example:** A program might reference data at logical address 1000.

Physical Address Space:

- Set of all physical addresses in the main memory (RAM) of a computer system.
- **Real addresses:** Directly correspond to memory locations.
- **Hardware-centric view:** Reflects the actual memory organization.
- **Example:** The physical location of the data referenced at logical address 1000 might be 2048 in physical memory.

Key Differences:

- **Visibility:** Users work with logical addresses, while physical addresses are hidden by the Memory Management Unit (MMU).
- **Mapping:** MMU translates logical addresses to physical addresses before memory access.
- **Portability:** Logical addresses are portable across different systems, while physical addresses are specific to the hardware.

Swapping in Memory Management

- **Concept:** A memory management technique used in multiprogramming environments to run more processes than can fit in physical memory (RAM) at once.
- **Process:**
    - **Swap Out:** Moving an inactive process from RAM to secondary storage (like a hard disk) to free up RAM for other processes.

    - **Swap In:** Transferring a swapped-out process back to RAM when it needs to be executed again.

Benefits:

- **Increases effective memory capacity:** Allows utilization of disk space as virtual memory, effectively expanding usable memory.
- **Improves system performance:** Frequently used processes stay in RAM for faster access, while less used ones are swapped out, minimizing RAM access delays.
- **Enhances multitasking:** Enables running more processes concurrently by freeing up RAM for new processes when needed.

Example:

Imagine you have two memory-intensive programs open (A and B). While working on A, program B is swapped out to the hard disk. When you switch to B, the operating system swaps B back into RAM and swaps out A (if needed) to provide B with the resources to run.

Contiguous Memory Allocation

Contiguous memory allocation is a memory management technique where the operating system allocates a single, continuous block of memory to a process. Here's a breakdown:

- **Concept:** Imagine a long stretch of land. Contiguous allocation assigns a house (process) a continuous block of that land (memory).

Allocation:

- The OS tracks available memory and assigns a free block large enough for the requesting process.
- There are two main approaches:
    - **Fixed Partitioning:** Memory is divided into equal-sized blocks beforehand. This is simple but may lead to internal fragmentation (wasted memory).
    - **Variable Partitioning:** Blocks can be of varying sizes, allowing for a better fit but requiring more complex management.

Example:

- Consider three processes: P1 (100 KB), P2 (50 KB), and P3 (75 KB).

- With fixed partitioning (say 100 KB blocks), each process gets a block regardless of its size, leading to some internal fragmentation.
- Variable partitioning allows assigning a tighter fit, reducing fragmentation but requiring the OS to track free blocks of various sizes.

Advantages:

- Simple to implement and manage.
- Faster memory access due to contiguous data storage.
- Easy to track free memory.

Disadvantages:

- External fragmentation: Over time, free memory becomes scattered, making it harder to find contiguous blocks for new processes (like holes appearing in the land).
- Memory wastage due to internal fragmentation (fixed partitioning).
- Not suitable for systems with dynamic memory requirements (processes whose memory needs change frequently).

Multiple Partitions

- **Concept:** Divides main memory into fixed-size chunks called partitions.
- **Process Allocation:** Each partition holds a single process at a time.

Partitioning Scheme:

- Can be equal sized for simplicity.
- Can be unequal sized to accommodate processes with varying memory demands.

Advantages:

- Simple to implement and manage.
- Easy to protect memory space for the operating system.

Disadvantages:

- **Internal Fragmentation:** Wasted memory within partitions when a process is smaller than its allocated partition. (e.g., A 10KB process in a 16KB partition leaves 6KB unused)
- **External Fragmentation:** Free memory scattered across multiple partitions, but no single contiguous block large enough for a new process. (e.g., Three 4KB free partitions won't fit a 5KB process)

Example: Imagine a system with 4 partitions of 100KB each. It can run 4 processes at most. A 70KB process would suffer from internal fragmentation (30KB wasted). Even with free space available, fitting a new 120KB process might be impossible due to external fragmentation.

Modern OSes typically avoid fixed partitioning due to these limitations. They use more dynamic memory allocation schemes like variable partitioning or paging.

Fragmentation

It is a common issue in memory management that arises when free memory becomes scattered throughout the memory space. This happens as processes are loaded and unloaded, leaving behind small chunks of unusable memory. There are two main types of fragmentation:

- **External Fragmentation:** Occurs when there's enough total free memory to fit a new process, but it's divided into unusable small blocks. Imagine having enough Legos to build a car, but they're all single bricks scattered around.
- **Internal Fragmentation:** Occurs when allocated memory has unused space within it due to memory allocation strategies. Think of a Lego car with some empty spaces where specific pieces wouldn't fit perfectly.

Compaction

It is a technique used to address external fragmentation. It involves shuffling the allocated memory blocks in the main memory to create one larger contiguous block of free space. This is similar to defragmenting a hard drive on your computer.

Here's a breakdown of the key points in Fragmentation and Compaction:

- **Fragmentation:**
  - Makes it difficult to allocate memory to new processes, even if there's enough free space in total.
  - Reduces memory utilization efficiency.
- **Compaction:**
  - Rearranges allocated memory blocks to consolidate free space.
  - Requires moving processes in memory, which can be time-consuming

Example:

Imagine memory is divided into 100 units. Process A takes 20 units, Process B takes 30 units, and

Process C takes 10 units. After some time, Process B finishes and leaves a hole of 30 units. Now, even though there's 40 units free (30 units from B + 10 units remaining), a new process of 45 units cannot be loaded due to fragmentation. Compaction would move Process A and C to one end, creating a contiguous block of 70 units (30 + 40) to fit the new process.

Note: Compaction is not always a practical solution due to the overhead of moving processes. Other memory management techniques like paging and segmentation are often used to prevent fragmentation in the first place.

Paging

- **Concept:** Paging is a memory management technique that divides both physical memory (RAM) and processes into fixed-size blocks called pages and frames, respectively. Unlike contiguous memory allocation, pages can be stored anywhere in physical memory as long as a frame is available.

Process Division:

- Each process is broken down into equal-sized pages.
- Pages can hold code, data, or a combination of both.

Memory Division:

- Physical memory is divided into equal-sized frames.
- A page from a process is loaded into an available frame in memory.

Logical vs. Physical Address:

- The CPU generates logical addresses for memory access within a process.
- A Memory Management Unit (MMU) translates the logical address into a physical address using a page table.

Page Table:

- The page table is a data structure maintained by the OS.
- It maps logical page numbers to physical frame numbers.

Benefits of Paging:

- **Non-contiguous Allocation:** Processes don't require contiguous memory blocks, allowing for better memory utilization.

- **Simplified Memory Management:** OS can load only the required pages of a process into memory, improving efficiency.
- **Protection:** Page tables can define access permissions for each page, enhancing memory protection.

Example:

- Consider a process with 10KB of code and data divided into 2KB pages.
- Physical memory is also divided into 2KB frames.

Virtual Memory Management

- **Concept:** An illusion of having more RAM than physically available. It utilizes secondary storage (hard disk) to extend the capacity of primary memory (RAM).

Benefits:

- **Run Larger Programs:** Allows programs exceeding RAM size to run by loading parts as needed.
- **Increased Multiprogramming:** Enables running multiple processes simultaneously, even if they don't entirely fit in RAM.
- **Memory Protection:** Isolates processes, preventing them from interfering with each other's memory space.

Implementation:

- **Division:** Divides virtual memory (program view) and physical memory (RAM) into fixed-size blocks called pages and frames respectively.
- **Translation:** A Memory Management Unit (MMU) in the CPU translates virtual addresses used by programs to physical addresses in RAM.
- **Demand Paging:** Popular technique. Loads only the required program pages into RAM when accessed, reducing initial load time. Triggers a page fault (OS intervention) to swap data between RAM and disk when needed.

Example:

Imagine a program with 10 pages. RAM can only hold 6 pages at once. The OS loads the first 6 pages. When the program tries to access data in page 8 (not in RAM), a page fault occurs. The OS swaps a less used page in RAM with page 8 from the disk, allowing the program to continue.

Demand Paging

- **Memory Management Technique:** Improves memory utilization by loading program sections (pages) into RAM only when needed.
- **Lazy Loading:** Only brings required pages from secondary storage (like hard disk) on-demand, reducing upfront memory usage.
- **Page Fault:** When a program tries to access a page not in RAM, it triggers a page fault.
    - OS then swaps a less used page from RAM to disk and loads the requested page.

Benefits:

- Runs larger programs on limited RAM.
- Enables efficient multitasking by allowing more processes in memory.

Example:

Imagine a program with code for editing text and displaying images. Demand paging would load the text editing code when you start typing, and the image processing code only when you open an image.

Drawbacks:

- **Page Faults:** Can slow down program execution if they occur frequently.
- **Overhead:** Managing page tables and swapping pages adds some processing overhead.

Page Replacement Algorithms

- In virtual memory management using paging, these algorithms decide which memory pages to swap out to secondary storage (disk) when a page fault occurs.
- A page fault happens when a requested page isn't present in main memory.
- The goal is to minimize page faults, reducing the I/O wait time caused by loading pages back into memory.

Common Algorithms:

1. First In, First Out (FIFO):

- Simplest algorithm, treats memory like a queue.
- The oldest page (at the queue's front) gets replaced.
- Easy to implement, but may not be optimal for programs with frequently used pages accessed later.

2. Least Recently Used (LRU):

- Replaces the page that hasn't been used for the longest time.
- Better than FIFO for most access patterns, approximates future usage.
- Requires keeping track of recent page accesses, adding some complexity.

3. Optimal Page Replacement:

- (Not practical for real systems) Replaces the page that won't be used for the longest time.
- Minimizes page faults but requires knowing the entire reference string in advance, unrealistic in practice.