**Study Material**

**Topic: Memory Management Hardware – Continued**

**1. Introduction**

In modern computer systems, the **CPU generates logical (virtual) addresses**, which need to be translated into **physical addresses** in RAM. This translation is not trivial because:

* Programs are often larger than available RAM.
* Multiple processes share the same physical memory.
* Protection and security must be enforced (one program should not overwrite another).

The hardware component responsible for this is the **Memory Management Unit (MMU)**.  
In earlier discussions, we studied **base/limit registers, paging, and segmentation**. Now, we go further into **virtual memory, page tables, TLB, page faults, and hybrid schemes**.

**2. Virtual Memory**

* **Definition:** Virtual memory is a memory management technique that gives each process the illusion of having its own large, continuous memory space, even though the actual physical memory may be smaller.
* **How it works:**
  + Only a part of the program is loaded into physical memory at a time.
  + The rest of the program resides in secondary storage (HDD/SSD).
  + When required, missing parts are brought into RAM dynamically.

**Key Benefits:**

1. Programs larger than RAM can execute.
2. Provides process isolation (each process sees its own address space).
3. Increases CPU utilization through multiprogramming.
4. Allows better security via controlled access.

**Example:**

* RAM = 4 GB, Program = 8 GB.
* Only currently needed parts (say 2 GB) are kept in RAM.
* Remaining 6 GB stored in disk, swapped in and out as needed.

**3. Page Tables**

A **page table** stores the mapping between **virtual pages** (from process address space) and **physical frames** (actual RAM locations).

**Structure of a Page Table Entry (PTE):**

* Frame number (where the page resides in RAM).
* Valid/Invalid bit (whether the page is in memory).
* Protection bits (R/W/X permissions).
* Dirty bit (whether the page was modified).
* Referenced bit (used in replacement algorithms).

**Types of Page Tables**

1. **Single-Level Page Table:**
   * Direct mapping from virtual pages → physical frames.
   * Disadvantage: Requires huge tables for large address spaces.
2. **Multi-Level Page Table:**
   * Breaks the mapping into multiple smaller tables.
   * Reduces memory usage since only needed portions of the page table are kept.
   * Example: 2-level, 3-level, or 4-level page tables in 64-bit systems.
3. **Inverted Page Table:**
   * Instead of one entry per virtual page, keeps one entry per **physical frame**.
   * Saves memory for large address spaces.
   * Each entry stores which process and virtual page currently occupies the frame.

**4. Translation Lookaside Buffer (TLB)**

* A **TLB is a small, fast associative cache** inside the MMU.
* Stores recent **page number → frame number** translations.

**Working of TLB**

1. CPU generates virtual address.
2. MMU checks TLB.
   * **TLB hit**: Frame number found, physical address generated instantly.
   * **TLB miss**: Page table consulted in RAM → translation fetched and placed into TLB..

**Why TLB is essential?**

* Without TLB: Each memory access requires **two accesses** (page table + actual memory).
* With TLB: Most accesses are resolved in one cycle (hit ratio > 95%).

**Real-world example:**

* Intel i7 processors have separate **Instruction TLB** and **Data TLB**, often with multiple levels (L1, L2).

**5. Page Faults**

* **Definition:** Occurs when a program accesses a page that is not currently in RAM.

**Steps in Handling a Page Fault**

1. CPU generates address.
2. MMU checks TLB → Miss → Page Table consulted.
3. Page Table entry → Marked invalid → Page fault trap to OS.
4. OS selects a free frame (or uses replacement algorithm).
5. Required page is read from disk into RAM.
6. Page Table and TLB updated.
7. Faulting instruction restarted.

**Performance Impact**

* Disk access is **1,000,000 times slower** than RAM.
* Excessive page faults lead to **thrashing** (system spends more time swapping than executing).

**6. Segmentation with Paging (Hybrid Model)**

* Pure **segmentation**: Divides program into variable-sized segments (Code, Data, Stack).
* Pure **paging**: Divides memory into fixed-size pages.
* **Hybrid system (used in modern CPUs):**
  + Logical address = **[Segment Number | Page Number | Offset]**
  + Each segment is divided into pages.

**Advantages:**

* Segmentation = better program organization.
* Paging = efficient use of memory (avoids fragmentation).

**Real-world usage:** Intel x86 uses segmentation + paging for memory protection and sharing.

**7. Protection and Sharing**

Memory hardware enforces **access control** for each page/segment:

* **Read-only**: Useful for shared libraries (multiple processes can read the same code).
* **Read-Write**: For data segments.
* **Execute-only**: For security (prevents code injection).

This ensures:

* No process can overwrite another’s data.
* Memory isolation between processes.
* Controlled inter-process communication.

**8. Modern Architectures**

* **Intel x86 / x86-64**:
  + Uses 4-level page tables (supports huge address spaces).
  + Separate Instruction & Data TLBs.
* **ARM Processors (mobile devices):**
  + Multi-level paging.
  + Strong reliance on TLB for performance.
* **RISC-V**:
  + Uses Sv39 (39-bit virtual address space) with multi-level paging.

**9. Advantages of Advanced Memory Management**

1. Efficient RAM utilization.
2. Faster address translation with TLB.
3. Virtual memory allows execution of large programs.
4. Process protection & isolation.
5. Support for multiprogramming and multitasking.
6. Enables modern features like memory-mapped files, copy-on-write, and shared memory.

**10. Summary**

* Modern memory management hardware combines **paging, segmentation, TLB, and virtual memory**.
* **Page tables** map virtual pages to physical frames.
* **TLB** accelerates translation by caching mappings.
* **Page faults** are handled by OS with hardware support.
* Hybrid approaches ensure both efficiency and logical program structure.
* Real-world CPUs (Intel, ARM) rely heavily on these mechanisms for performance and security.

**11. Practice Questions**

**Short Answer (2 Marks):**

1. What is the role of a TLB?
2. Define page fault.
3. What is the difference between valid and invalid bit in page table entries?

**Medium Answer (5 Marks):**

1. Explain how a TLB improves the performance of virtual memory.
2. Differentiate between multi-level page tables and inverted page tables.
3. Describe the steps involved in handling a page fault.

**Long Answer (10 Marks):**

1. Discuss the working of Virtual Memory with diagrams.
2. Explain segmentation with paging in modern systems.
3. Write a detailed note on Translation Lookaside Buffers (TLB) and their importance in memory management.