Group Activity - Paging

Hashed Page Tables:

Hashed page tables are particularly useful for systems with large address spaces (higher than 32 bits). In these systems, traditional page tables would become too large, and a more efficient solution is needed to map virtual addresses to physical frames.

In a **hashed page table**, the virtual page number is hashed to a page table. Each entry in the table contains:

1. **The virtual page number.**
2. **The corresponding physical frame.**
3. **A pointer to the next element in the chain** (if there is a hash collision, i.e., multiple virtual page numbers hash to the same index).

When a virtual address is accessed, the system hashes the virtual page number to locate an entry in the hashed page table. The system then searches the chain of entries for a match. Once found, the corresponding physical frame is used to access the data.

Example:

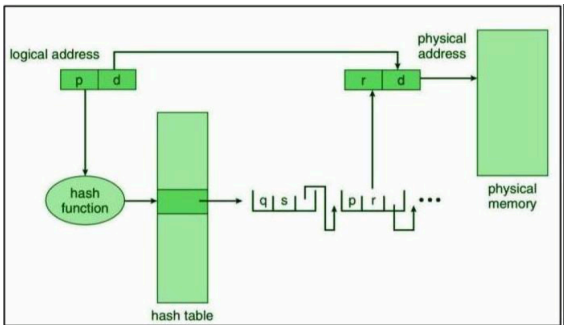
Imagine a system where the virtual page number 1234 hashes to index 3 in a hashed page table. At index 3, there are two entries:

- The first entry contains the virtual page 4321.
- The second entry contains the virtual page 1234.

To translate virtual address 1234, the system hashes it to index 3, then walks through the chain to find the matching virtual page and retrieve its corresponding physical frame.

Benefits:

- Efficient in handling large address spaces.
- Suitable for systems with sparse address spaces, where virtual pages are scattered non-contiguously.



(Image source: GeeksForGeeks)

Inverted Page Tables:

In an **inverted page table**, instead of having a page table for each process, the system maintains a single table that tracks physical memory. Each entry in this table represents a physical frame in memory and stores:

1. **The virtual page number** of the page that is currently stored in that physical frame.
2. **Information about the process** that owns the page.

This reduces the amount of memory required to store page tables, as there is only one entry per physical page, rather than an entry for every possible virtual page.

When a virtual address is accessed, the system must search the inverted page table to find the entry corresponding to the physical frame. A hash function is often used to speed up this search.

Example:

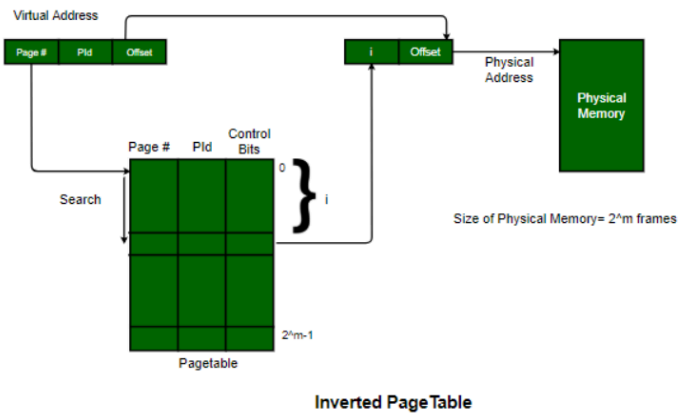
If you have a system with 100 physical frames, the inverted page table will have 100 entries. If process A tries to access a virtual address, the system checks the inverted page table to find which physical frame holds the data for that virtual page and process.

Benefits:

- Reduces memory overhead since there's only one page table, regardless of the number of processes.
- Suitable for systems with a large number of processes.

Challenges:

- Slower lookups due to the need to search the inverted table.
- Complex handling of shared memory between processes.



Inverted PageTable

(Image source: GeeksForGeeks)

Group 7- Team members

- **Waduge S.S. - 220673K**
- **Vidyananda H.K.H - 220666R**
- **Kavinda L.G.L - 220316V**
- **Chathurangi K.L. - 220087R**