CSE321 Theory Assignment 02

Research and **Explore** the fundamental differences between hashed page tables, and inverted page tables by analyzing the following key areas:

- **Hashed Page Table Implementation**
- **Inverted Page Table Mechanisms**

ID: 21301289,
Name: Ishtiaq Ahmed,
Sec:12, [NAN]

==========

In today's computer systems, efficient memory management is responsible for achieving performance, scalability, and reliability. Virtual memory provides processes the appearance of an enormous, linear memory space, even though the true physical memory could be tiny and disjointed. To achieve this, the kernel relies on page tables for translating virtual addresses into physical ones. However, the classic page table grows linearly with the virtual address space, leading to enormous overhead for systems that support 64-bit addressing. In order to overcome the limitations, other data structures such as hashed page tables and inverted page tables were created. Though both endeavor to cut down memory overhead and support fast lookup, they differ significantly from their design, functioning, and performance characteristics. This report compares the two techniques, focusing on the memory organization and layout, and their performance characteristics.

Memory Organization and Structure

Hashed Page Tables

A hashed page table employs the use of a hash function for mapping virtual page numbers into related entries within a hash table. Each entry holds a chain of records for the virtual pages that collide at the same address. In most cases, the table holds:

- **Hash Table:** An array whose slot maps to a hash value.
- **Entry chains:** Entry chains are linked lists or arrays designed to resolve hash collisions, instances in which multiple virtual pages correspond to the same slot.
- **Mapping information:** There is one record that contains the virtual page number, the associated physical frame number, and a pointer for the next record for cases of collision.

Translation of addressing via hashed page tables goes like this:

1. The virtual page number (VPN) is hashed to derive the index.

2. The system finds the chain at that index for the matching virtual page entry.

3. When it is found, the physical frame number is taken and the page offset is used to form the physical address.

It is especially efficient for big address spaces (such as 64-bit address spaces) since it doesn't preserve large sparse ordinary page tables.

Inverted Page Tables

Unlike hashed page tables, the inverted page table (IPT) holds one entry for each individual physical frame and not for each individual virtual page. Its format consists of:

- **Fixed Table Size**: Same as the number of physical frames in the system.

- **Table Entries**: Each entry contains the virtual page number that is resident in the frame, the process ID (for the purpose of process differentiation), and status/control bits.

Address translation for inverted page tables occurs otherwise:

1. Given a virtual address, the system must search the IPT to determine if the virtual page is currently mapped.

2. Typically, an associative search or a hashing function (potentially with the help of a TLB) is used to locate the entry.

3. When a record is found, the corresponding physical frame is identified and the physical address is computed.

Since IPTs scale with physical memory size, they can be more space efficient than traditional or hashed tables for very large address spaces.

Comparison of Structure and Overhead

- **Hashed Page Table:** Linear growth with the number of virtual pages active. Memory overhead depends on size of the hash table and collision chains.

- **Inverted Page Table:** Grows linearly with the number of frames alone and not the virtual address space size. This makes the IPT more memory-efficient for very large 64-bit applications.

In short:

- **Hashed tables** → simpler translation, but increased usage of memory in busy virtual spaces.

- **Inverted tables** → concise presentation, but slower searches since we must search.

Performance Characteristics

Hash Collisions in Hashed Page Tables

Hash collisions are performance bottlenecks for hashed page tables. Worst-case, where numerous virtual pages hash into a single slot, the system must iterate through linked chains and therefore induce higher translation time. Even though quality hash function minimizes the collisions, worst-case lookup time is linearly proportional to the length of the collision chain.

Mitigation measures are:

- Choosing big hash tables over the number of webpages.

- Use of secondary hash functions or higher-level collision-resolution schemes.

- Caching of most up-to-date translations within a Translation Lookaside Buffer (TLB).

Search Overhead in Inverted Page Tables

Inverted page tables sidestep the table size explosion but pay in search overhead. Because there is only one entry per physical frame, we want to find the right frame for a specific virtual page through hashing or associative search. In the absence of good hashing, it would be too expensive for a linear search of the table.
Thus, IPTs nearly always use hash functions for table indexing. But it brings the collision possibility back again, which needs further probing.

TLB Miss Handling

Both the prototypes largely depend on the Translation Lookaside Buffer (TLB) for minimizing the lookup time.

- **Hashed Page Tables:** During a TLB miss, the system does a hash lookup, and can subsequently follow through a chain of entries. This introduces latency if there are collisions.

- **Inverted Page Tables:** Upon TLB miss, the system will hash the virtual address and possibly probe a number of locations, which can be slower than the direct hashed table lookups. Yet the small IPT size can assist for improved locality and offset some costs.

Page Fault Processing

- **Hashed Page Tables:** In the case of a page fault, a new entry is added into the hash table. In the case of collisions, it can expand the chain further, increasingly hampering performance under the pressure of intense memory usage.

- **Inverted Page Table:** When there is a page fault, the system must replace an active frame. Since IPT entries are tied to frames, maintaining these and resolving conflicts would make replacement policies more complex but avoid table bloat.

Comparative Analysis

Factor	Hashed Page Table	Inverted Page Table
Size Dependency	Proportional to number of virtual pages.	Proportional to number of physical frames
Memory Overhead	Higher for large address spaces.	Lower and more space-efficient
Lookup Speed	Fast with good hash, slower with collisions.	Slower due to associative search or collisions
TLB Miss Handling	Traverses chains, overhead from collisions.	Hash/probe overhead, but compact representation
Page Fault Processing	Simple insertion into hash chain.	Requires frame replacement updates
Best Use Case	Moderate-to-large address spaces with fewer collisions.	Extremely large 64-bit systems with limited physical memory

Both the hashed page tables and the inverted page tables are the solutions of the scalability issue raised by the classic page tables. Hashed tables provide quicker lookups under the assumption of the low collision rates, but can consume more memory under the very large virtual spaces. Inverted page tables, however, lower the memory overhead dramatically by matching up with the physical memory instead of the virtual space, although with the penalty of slower address translation due to the searches and the hashing.
In practice, contemporary systems frequently depend on an interplay of TLBs, hashed indexing, and refined caching mechanisms to strike a balance amid various trade-offs. The decision between hashed and inverted page tables hinges on the architecture of the system: hashed page tables cater to those systems that emphasize speed while managing moderate memory constraints, whereas inverted page tables prove to be optimal in environments characterized by vast address spaces, where space efficiency takes precedence.