CSE321 Theory Assignment 02
Name: Abrar Ahmed
Id:23101354, Sec:33

Memory layout and organization for hashed and inverted page tables:

Hashed Page Table (HPT)

The HPT is an array of buckets.Each bucket contains a linked list of page table entries which is hashed to that bucket.A page table entry typically contains:

1.Virtual Page Number (VPN)

2.Physical Frame Number (PFN)

3.Process identifier (PID) or address-space ID.

4.Protection bits, valid bits and other metadata.

Inverted Page Table (IPT)

The IPT is a single table with one entry for each physical frame.Here, table size=no. of physical frames. Each entry contains:

1.The VPN currently stored in that frame
2.The PID owning that mapping
3.Protection bits, valid, dirty, referenced etc.

Virtual to physical translation in each system:

HPT:

1.Extract VPN (from virtual address) and PID if used.
2.Compute h = hash(PID, VPN) % table_size.
3.Look at bucket h. Traverse its chain and compare stored (PID, VPN) to find a match.

4.If a match is found, use PFN to build a physical address.
5.If not found, page not present (page fault).

IPT:

1.Extract VPN and PID from virtual address.
2.Search the IPT for an entry whose (PID, VPN) matches. That entry gives PFN (index into IPT).Because IPT is indexed by physical frame, finding a matching (PID, VPN) requires search.In practice, a hash is usually used (a hashed inverted page table) or additional structures (per-process small page tables) are combined to avoid linear scan.
3.If found, produce PFN and form a physical address.
4.If not, page fault.

Memory overhead and space efficiency between the two approaches:

HPT:

1.Memory cost: O(number_of_mapped_pages + overhead for hash table).
2.HPT size depends on the number of active mappings (sparse virtual spaces are efficient).
3.Extra storage for bucket heads and pointers for chaining.
4.Good when each process maps only a small subset of a huge virtual address space.

IPT:

1.Memory cost proportional to physical memory size: O(number_of_physical_frames).
2.Very space-efficient when virtual address space is huge but physical memory is limited (only one entry per frame).
3.Storing VPN and PID in each physical frame entry increases per-entry size compared to a conventional per-VPN page table entry but total entries are only as many as frames.

The impact of hash collisions in hashed page tables and search overhead in inverted page tables:

Hash collisions in Hashed Page Tables

What happens: Multiple (PID, VPN) values can map to the same bucket as a result bucket chain length grows.

Cost of collision: Lookup time increases linearly with chain length (worst-case O(n) in that bucket).

Mitigations:
1.Choose a good hash function and large table size to keep load factor low.
2.Use open addressing or balanced collision-resolution schemes.
3.Use additional caching (TLB) to avoid repeated HPT lookups.

Practical effect: With a well-sized hash table, average lookup ≈ O(1). Collisions are acceptable if infrequent.

Search overhead in Inverted Page Tables

What happens: Translation requires finding an IPT entry matching (PID, VPN). Naively this is O(#frames) which is unacceptable.

Mitigations:
1.Most IPT implementations combine hashing with inverted tables so they are actually a hybrid: hash (PID, VPN) to index into IPT buckets (so IPT becomes searchable).
2.Use additional small per-process page directories or caches.

Practical effect: With hashing on top of IPT, lookups approach O(1) on average but still require extra fields (VPN, PID) per entry and bucket chains.

TLB miss handling

The Translation Lookaside Buffer (TLB) caches recent VPN to PFN translations. Both systems rely heavily on TLB to reduce expensive table searches.

On TLB miss with HPT:
1.Walk HPT: compute hash, traverse bucket, find PFN.
2.Fill TLB with (VPN to PFN) entry.
3.Resume instruction.

Here, cost depends on hash chain length and memory accesses required.

On TLB miss with IPT:
1.Search IPT (or hashed IPT) for (PID,VPN).
2.Fill TLB with found mapping.

Here, cost depends on the IPT search method.

Comparison: If both use hashing, average TLB miss handling time is similar. The difference is structural: HPT stores an entry per mapping, IPT stores per-frame entry and needs VPN+PID to be stored per entry.

Page-fault processing

OS recognizes missing page or invalid mapping.Locate page on disk, allocate a physical frame, bring page into memory, update page-table structure, update TLB.

HPT:
OS adds an entry for (PID,VPN) to PFN in the hash table.Slight extra work to link into the correct bucket chain.

IPT:
OS picks a physical frame then updates that frame's IPT entry with (PID,VPN) and flags.If the frame was occupied, the OS must invalidate old mapping that requires finding which VPN got removed so that IPT directly identifies it.

Eviction behavior:
IPT naturally aligns with physical frame eviction (each IPT entry is a frame). When replacing a frame, IPT entry is overwritten.Whereas, HPT must find and remove any old mapping pointing to the replaced PFN.

Comparison: IPT can simplify eviction bookkeeping (since one IPT entry per frame), while HPT may require additional steps to remove stale entries referring to the evicted PFN.

References:

Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating system concepts* (10th ed.). Hoboken, NJ: Wiley.

Stallings, W. (2018). *Operating systems: Internals and design principles* (9th ed.). Pearson.