Virtual Memory Concepts

CSC 235 - Computer Organization

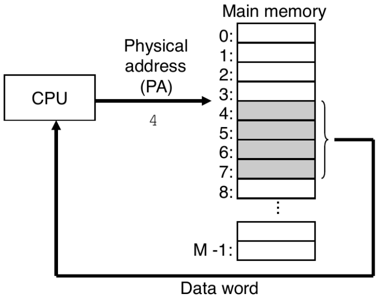
References

- Slides adapted from CMU

Overview

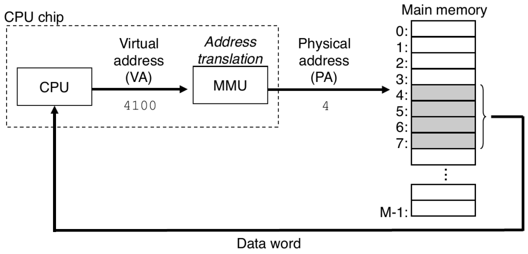
- Address Spaces
- Virtual Memory as a tool for caching
- Virtual Memory as a tool for memory management
- Virtual Memory as a tool for memory protection
- Address translation

A System Using Physical Addressing



- Used in “simple” systems like embedded microcontrollers in devices like cars, elevators, etc.

A System Using Virtual Addressing



- Used in all modern servers, laptops, and smart phones
- One of the great ideas in computer science

Address Spaces

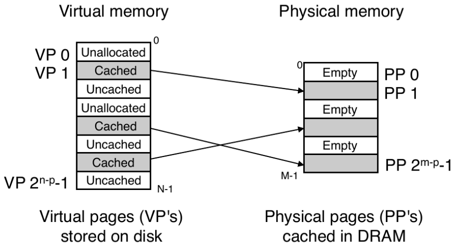
- Linear address space: ordered set of contiguous non-negative integer addresses
- Virtual address space: set of $N = 2^n$ virtual addresses
- Physical address space: Set of $M = 2^m$ physical addresses

Why Virtual Memory (VM)?

- Uses main memory more efficiently
  - Use DRAM as a cache for parts of a virtual address space
- Simplifies memory management
  - Each process gets the same uniform linear address space
- Isolates address spaces
  - One process cannot interfere with another's memory
  - User program cannot access privileged kernel information and code

VM as a Tool for Caching

- Conceptually, *virtual memory* is an array of N contiguous bytes stored on disk.
- The contents of the array on disk are cached in *physical memory* (DRAM cache)
  - These cache blocks are called *pages* (size is $P = 2^p$ bytes)

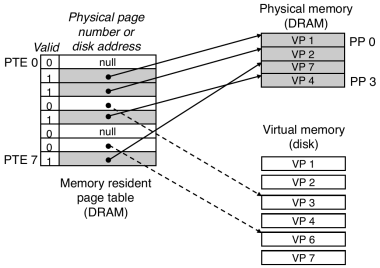


DRAM Cache Organization

- DRAM cache organization driven by the enormous miss penalty
  - DRAM is about 10x slower than SRAM
  - Disk is about 10,000x slower the DRAM
  - Time to load block from disk $>$ 1 ms
- Consequences
  - Large page (block) size: typically 4 KB
    - Linux “huge pages” are 2 MB (default) to 1 GB
  - Full associative
    - Any virtual page can be placed in any physical page
    - Requires a “large” mapping function - different from cache memories
  - Highly sophisticated, expensive replacement algorithms
    - Too complicated and open-ended to be implemented in hardware
  - Write-back rather than write-through

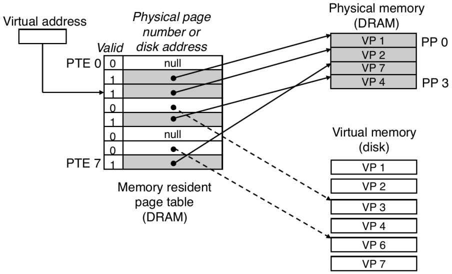
Enabling Data Structure: Page Table

- A *page table* is an array of page table entries (PTEs) that maps virtual pages to physical pages
  - Per process kernel data structure in DRAM



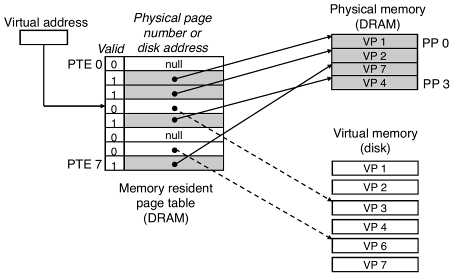
Page Hit

- Page hit: reference to VM word that is in physical memory (DRAM cache hit)



Page Fault

- Page fault: reference to VM word that is not in physical memory (DRAM cache miss)

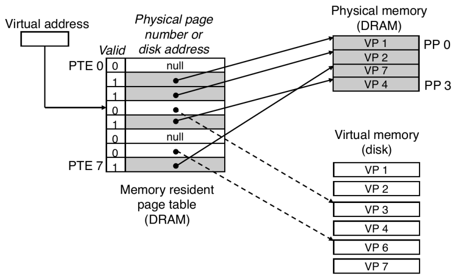


Triggering a Page Fault

- User writes to memory location
- That portion (page) of user's memory is currently on disk
- Memory management unit (MMU) triggers page fault exception
  - (More details later)
  - Raises privilege level to supervisor mode
  - Causes procedure call to software page fault handler

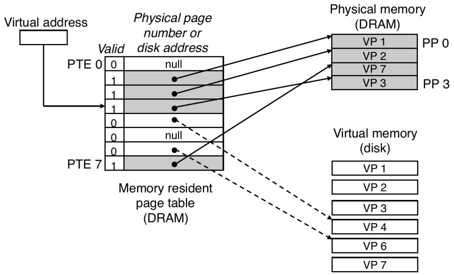
Handling a Page Fault

- Page miss causes page fault (an exception)
- Page fault handler selects a victim to be evicted (here VP 4)



Handling a Page Fault

- Page miss causes page fault (an exception)
- Page fault handler selects a victim to be evicted (here VP 4)

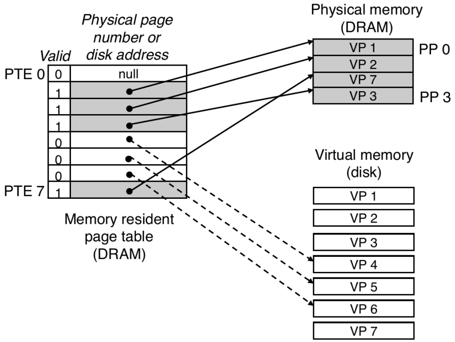


Completing Page Fault

- Page fault handler executes return from interrupt (`iret`) instruction
  - Like `ret`, but also restores privilege level
  - Return to instruction that caused page fault
  - But, this time there is no page fault

Allocating Pages

- Allocating a new page (VP 5) of virtual memory



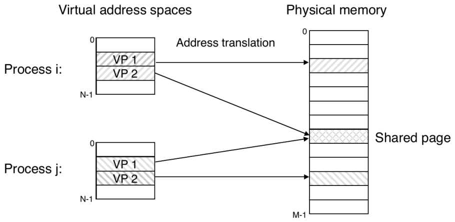
- Subsequent miss will bring it into memory

Locality to the Rescue Again

- Virtual memory seems terribly inefficient, but it works because of locality
- At any point in time, programs tend to access a set of active virtual pages called the working set
  - Programs with better temporal locality will have smaller working sets
- If the working set size is less than the main memory size:
  - good performance for one process (after cold misses)
- If the working set size is greater than main memory size:
  - Thrashing: performance meltdown where pages are swapped (copied) in and out continuously
  - If multiple processes run at the same time time, thrashing occurs if their total working set size is greater than main memory size

VM as a Tool for Memory Management

- Key idea: each process has its own virtual address space
  - It can view memory as a simple linear array
  - Mapping function scatters addresses through physical memory
    - Well-chosen mappings can improve locality

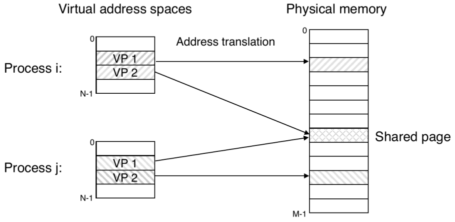


VM as a Tool for Memory Management

- Key idea: each process has its own virtual address space
  - It can view memory as a simple linear array
  - Mapping function scatters addresses through physical memory
    - Well-chosen mappings can improve locality
- Simplifying memory allocation
  - Each virtual page can be mapped to any physical page
  - A virtual page can be stored in different physical pages at different times

VM as a Tool for Memory Management

- Sharing code and data among processes
  - Map virtual pages to the same physical page

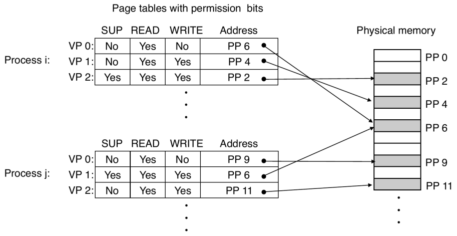


Simplified Linking and Loading

- Linking
  - Each program has similar virtual address space
  - Code, data, and heap always start at the same addresses
- Loading
  - `execve` allocated virtual pages for `.text` and `.data` sections and creates PTEs marked as invalid
  - The `.text` and `.data` sections are copied, page by page, on demand by the virtual memory system

VM as a Tool for Memory Protection

- Extend PTEs with permission bits
- Memory management unit (MMU) checks these bits on each access



Process i:

	SUP	READ	WRITE	Address
VP 0:	No	Yes	No	PP 6
VP 1:	No	Yes	Yes	PP 4
VP 2:	Yes	Yes	Yes	PP 2

Process j:

	SUP	READ	WRITE	Address
VP 0:	No	Yes	No	PP 9
VP 1:	Yes	Yes	Yes	PP 6
VP 2:	No	Yes	Yes	PP 11

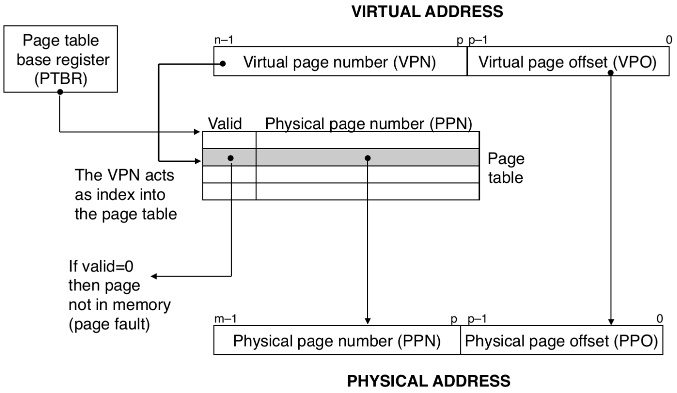
VM Address Translation

- Virtual address space
  - $V = \{0, 1, \ldots, N-1\}$
- Physical address space
  - $P = \{0, 1, \ldots, M-1\}$
- Address Translation
  - $MAP : V \rightarrow P \cup \emptyset$
  - For virtual address a:
    - $MAP(a) = a'$ if data at virtual address a is at physical address a' in P
    - $MAP(a) = \emptyset$ if data at virtual address a is not in physical memory; either invalid or stored on disk

Summary of Address Translation Symbols

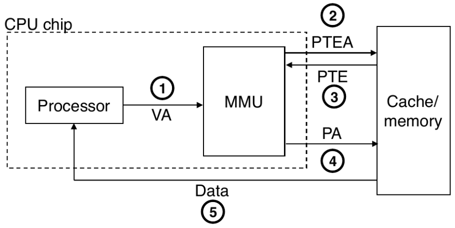
- Basic Parameters
    - $N = 2^n$: number of addresses in virtual address space
    - $M = 2^m$: number of addresses in physical address space
    - $P = 2^p$: page size (bytes)
- Components of the virtual address (VA)
    - VPO: virtual page offset
    - VPN: virtual page number
- Components of the physical address (PA)
    - PPO: physical page offset (same as VPO)
    - PPN: physical page number

Address Translation with a Page Table



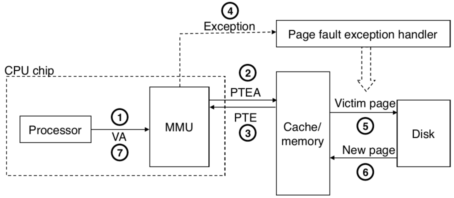
VIRTUAL ADDRESS
n–1
p
p–1
0
Virtual page number (VPN)
Virtual page offset (VPO)
Page table base register (PTBR)
Valid
Physical page number (PPN)
Page table
The VPN acts as index into the page table
If valid=0 then page not in memory (page fault)
m–1
p
p–1
0
Physical page number (PPN)
Physical page offset (PPO)
PHYSICAL ADDRESS

Address Translation: Page Hit



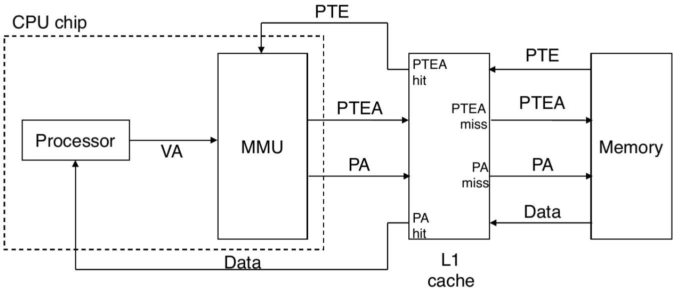
1. Processor sends virtual address to MMU
2. MMU requests PTE from page table in memory
3. MMU fetches PTE from page table in memory
4. MMU sends physical address to cache/memory
5. Cache/memory sends data word to processor

Address Translation: Page Fault



1. Processor sends virtual address to MMU
2. MMU requests PTE from page table in memory
3. MMU fetches PTE from page table in memory
4. Valid bit is zero, so MMU triggers page fault exception
5. Handler identifies victim (and, if dirty, pages it out to disk)
6. Handler pages in new page and updates PTE in memory
7. Handler returns to original process, restarting faulting instruction

Integrating VM and Cache



Speeding up Translation with a TLB

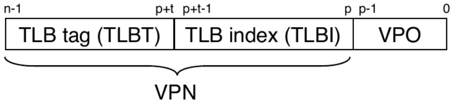
- Page table entries (PTEs) are cached in L1 like any other memory word
  - PTEs may be evicted by other data references
  - PTE hit still requires small L1 delay
- Solution: Translation Lookaside Buffer (TLB)
  - Small set-associative hardware cache in MMU
  - Maps virtual page numbers to physical page numbers
  - Contains complete page table entries for small number of pages

Summary of Address Translation Symbols

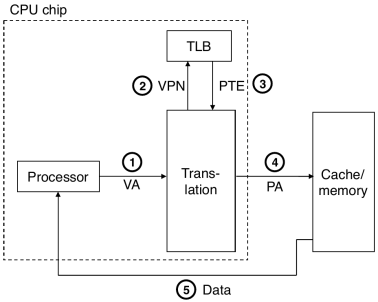
- Basic Parameters
  - $N = 2^n$: number of addresses in virtual address space
  - $M = 2^m$: number of addresses in physical address space
  - $P = 2^p$: page size (bytes)
- Components of the virtual address (VA)
  - *TLBI: translation lookaside buffer index*
  - *TLBT: translation lookaside buffer tag*
  - VPO: virtual page offset
  - VPN: virtual page number
- Components of the physical address (PA)
  - PPO: physical page offset (same as VPO)
  - PPN: physical page number

Accessing the TLB

- The MMU uses the VPN portion of the virtual address to access the TLB:

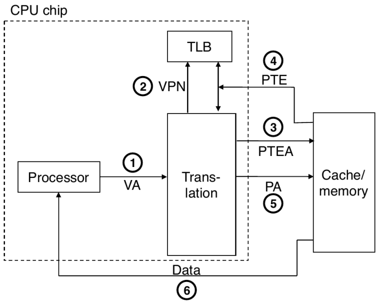


TLB Hit



- A TLB hit eliminates a cache/memory access

TLB Miss

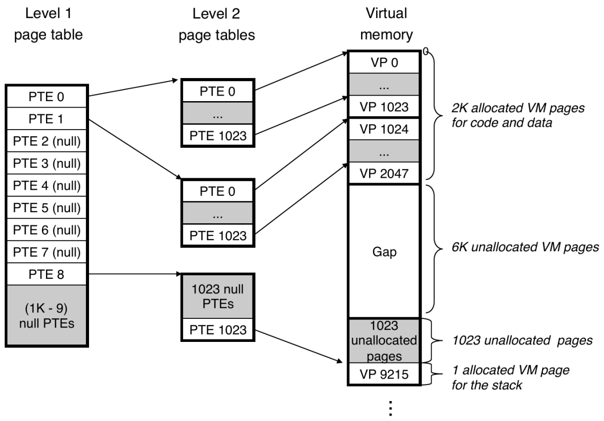


- A TLB miss incurs an additional cache/memory access (the PTE)

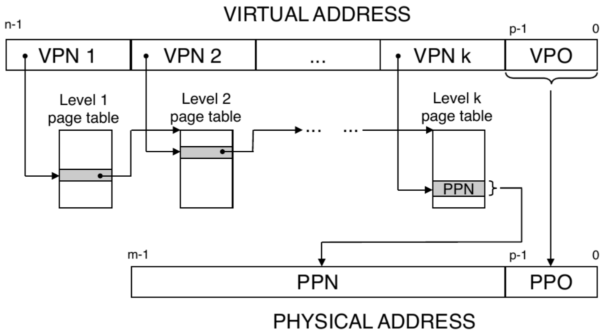
Multi-Level Page Tables

- Suppose:
  - 4 KB (2^{12}) page size, 48-bit addressable space, 8-byte PTE
- Problem:
  - Would need a 512 GB page table
- Common solution: multi-level page table
- Example: 2-level page table
  - Level 1 table: each PTE points to a page table (always memory resident)
  - Level 2 table: each PTE points to a page (paged in and out like any other data)

A Two-Level Page Table Hierarchy



Translating with a k-level Page Table



Summary

- Programmer's view of virtual memory
  - Each process has its own private linear address space
  - Cannot be corrupted by other processes
- System view of virtual memory
  - Uses memory efficiently by caching virtual memory pages
    - Efficient because of locality
  - Simplifies memory management and programming
  - Simplifies protection by providing a convenient interpositioning point to check permissions
- Implemented via a combination of hardware and software
  - MMU, TLB, exception handling mechanisms part of hardware
  - Page fault handlers, TLB management performed in software