Virtual Memory (VM)

CIT 595
Spring 2010

Virtual Memory

- Gives a program an illusion that it has contiguous working memory
  - Even though main/physical memory may not be as large as its address space

- Idea is to treat main memory like a cache
  - Store only a subset of program’s address space

- Allows transparent memory management between main memory & permanent storage (disk)
  - Note that VM predates cache

Motivation for Virtual Memory

- Program Address Space
  - 0 to \(2^n - 1\) where \(n\) = machine size
    - Full address space is quite large. E.g. 32-bit address (with 1 byte storage) = 4 GB
  - Making main memory as large as address space is too expensive

- Original Motivation by IBM in 1970
  - IBM S/370 machine size 31 bits vs. IBM S/360 had 24 bits
  - Compatibility of software between different machine sizes
  - Prevent explicit memory management

Big Picture: How VM works?

Virtual Address (VA)
- Is the address generated by your program
- Address range 0 to \(2^n - 1\) where \(n\) is machine width

Physical Address (PA)
- Where the virtual address is physically stored in Main Memory (DRAM)
- Address ranges from 0 to \(2^m - 1\) where \(m < n\)

Program's Logical View (VM)

- Note: Logical address order need not be maintained in physical memory
Big Picture: How VM works?

- Use the disk as an extension to main memory
- Address mapping scheme tells us where the VA is actually located in main memory
  - Mapping need not preserve continuity of data in both physical memory and disk
  - Logical address is only from program's point of view
- Address translation from is done by Operating System (OS) + hardware

Uses of VM: Multiprogramming

- Appearance to the user that different processes are being executed at the same time
- Only *active* part of the code and data of process is in main memory
- Allocate more memory to process as needed

Uses for VM: Program Isolation/Protection

- VM creates an illusion i.e. each process thinks it has 2^n address space
- Map VA of a process to a PA that is separate from another process
  - Prevent processes from reading/writing each other's memory

Data Size Transfer between Disk & Main Memory

- Due to Principle of Locality
  - Large chunks or pieces of data are transferred between Disk and Main Memory
- Unit of transfer is called a *page*
  - Analogous to *block* transfer between cache and main memory
  - However, the page is much larger than block
    - E.g., Transfer 8-32 bytes vs. 2048-8192 bytes for most modern systems
Virtual Memory Technique: Paging

- Allocate physical memory to processes in fixed size chunks called page frame
- VA space is divided into pages of equal size
  - Page size is also same as the frame size
  - Each page has same number of words (similar to cache)
- Entire address space required by a process need not be in memory at once
- Pages allocated to a process do not need to be stored contiguously either on disk or in main memory
  - In both cases O.S. keeps a data structures to track actual locations

Paging: Address Translation Overview

Virtual Address (VA) is translated into Physical Address (PA)

Address Translation using Page Table

- Location of each page, is maintained in a data structure called a page table
  - Updated by Operating System (OS)
  - Placed in memory at a known location
  - Virtual Page number is used to index the page to find which frame in Physical Memory is the data located

Example

- A system has a virtual address space of $2^8$ and a physical address space of $2^7$. Further assume that each page has 32 words
  - A virtual address has 8 bits
  - Of the 8 bits, 5 bits are used for offset ($2^5 = 32$ words)
  - Remaining 3 bits are used for Virtual Page Number (VPN)
  - Since physical memory address is 7 bits, 2 bits are used for Page Frame Number (PFN)

Note: Page Table entries are equal to number of pages to minimize search time
Example (contd..)

Virtual Address 13 is produced by the processor for a process, what is the Physical Address?

Virtual Address = 1001101 (77_{10})
(Note: frame # is appended to offset)

Paging in Multiprogramming Environment

Each program’s logical view is the same

Program’s Logical View

- Code
- Heap
- Stack

Problem: Different processes use the same virtual addresses
Solution: Each process has its own page table
OS responsible for updating page tables so that virtual address spaces of different processes do not collide

Paging in Multiprogramming Environment (contd..)

On a context switch:
- OS saves the state of existing process before switch
  - Save PC, registers and page table
  - Store only the address where the first page table entry is located
- OS loads the state of the new process
  - Page table address is loaded into an internal register
    - Known as Page Table Pointer
    - Like PC register is used to load the value of PC

Overview of Paging Technique

Address Translation in a Paging System
Valid Bit
- A bit is needed to indicate whether the page is in main memory or not
- Use a valid bit like we used in cache
- Valid bit = 1, means page is in physical memory

Page Fault
- If valid bit is 0, then page is not in physical memory
- This is known as an occurrence of page fault
- An exception is thrown and the OS intervenes
- Finds page required from disk & loads it into main memory
- May need replacement policy if all frames are occupied
- OS also maintains a data structure to record where each virtual page is stored on disk

Write Policy
- Modified data in physical memory must also be updated in the disk
- Disk access time is very slow
  - 5ms – 20ms compared to DRAM access time of 50-70ns
- Write-Back policy is employed i.e. update disk only when the page is going to be replaced
- Dirty (Modify) bit indicates if the page has been altered since it was last loaded into main memory
  - If dirty = 1, then write-back page to disk upon replacement

Page-level Protection
- Along with VPN to PPN mapping
- Add Read/Write/Executive permissions
  - Read & execute for code
  - Read for read only data
  - Read & Write for read-write data
- Example of Protection
  - Raise an exception on write to read-only data
    - OS terminates the program (process)
Unlike caches, replacement algorithm can be done in software.

**EAT Example**

- Suppose a main memory access takes 200ns, the page fault rate is 1%, and it takes 10ms to load a page from disk.

- What is Effective Access Time?

\[
EAT = 0.99(200\text{ns} + 200\text{ns}) + 0.01(10\text{ms}) = 100396\text{ns}
\]

**Accelerating Address Translation**

- Each virtual memory reference causes 2 physical memory accesses
  1. fetch the page table
  2. fetch the data

- Solution: Translation Lookaside Buffer (TLB)
  - High-speed memory
  - TLB is made of SRAM technology

- TLB stores the most frequently used mappings
  - Stores page number and corresponding frame number
    - Fully associative in terms of mapping
  - SRAM is costly in terms of storage
    - Hence use it to store the data that you will likely need
Caches are indexed by PA
- Use TLB to perform the translation
- Access time for data is faster
  - SRAM (TLB and Cache) vs. two DRAM (main memory) accesses

Disadvantage of Paging Technique
- A process may not need the entire range of addresses contained within the page
- There may be many pages containing unused fragments of memory
  - Known as *internal fragmentation*
- Unused fragments could have allow more processes to be existent in physical memory
  - Constraint due to fixed page size

Alternatives
- Have variable sized partitions (Segments)
- More modern: Combine Paging and Segmentation