CS354: Machine Organization and Programming

Lecture 30
Wednesday the November 11th 2015

Section 2
Instructor: Leo Arulraj

Class Announcements

1. How was Midterm2? Easy, Hard?
2. Grades for Programming Assignment 3 are now available in your handing directory for P3 as a html file. (See Piazza post for more details)

Lecture Overview

1. Physical and Virtual Addressing
2. Address spaces
3. As a tool for caching
4. As a tool for memory management
5. As a tool for protection
6. Address Translation

Virtual Memory Intro

Virtual Memory provides three important capabilities(all with a one clean mechanism):

A. Uses Main Memory efficiently as a cache

B. Simplifies memory management

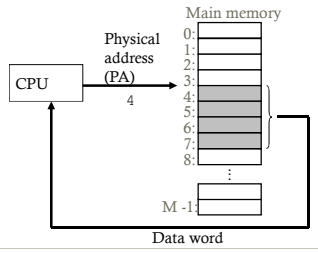
C. Protects address space of each process

Virtual Memory Intro

1. Virtual memory pervades all levels of computer systems: exceptions, assemblers, linkers, loaders, files, processes.
2. Virtual memory is powerful: sharing, protection, allocation etc. is easy
3. Virtual memory is dangerous: improper use can lead to hard to find bugs

Virtual Memory Intro

An example Physical Addressing Machine.

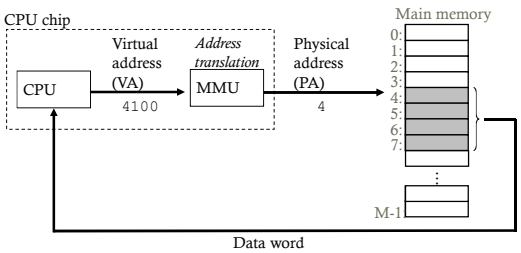


Virtual Addressing

1. Virtual Address Space: (0 to N-1) Bytes and is addressed with $\log_2(N)$ bits. E.g. 32 bit address space for x86 , 64 bits for x86_64
2. Physical Address Space: (0 to M-1) Bytes and is addressed with $\log_2(M)$ bits. Defined by Hardware. Often 40-48 bits in real architectures. Physical address space can be smaller than virtual address space.

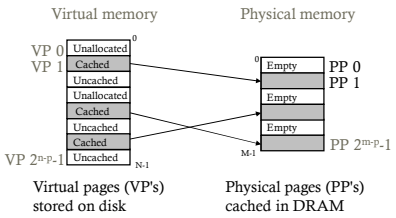
Virtual Memory Intro

System with virtual addressing:



As a tool for caching

Each virtual page is P = 2^p Bytes in size
Each physical page is also P Bytes in size



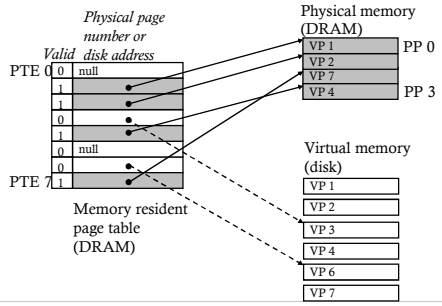
As a tool for caching

1. Unallocated Pages: Not yet created by the VM system.(No data in memory or disk)
2. Cached Pages: Allocated and are currently in Memory.
3. Uncached Pages: Allocated but are in disk and not cached in Memory.

Similarity with CPU Caching

1. DRAM is a fully set associative cache. Any virtual page can be placed into any physical page in memory. Cannot afford conflict miss penalties.
2. DRAM is approx. 10 times slower than SRAM
3. Disk is approx. 100000 times slower than DRAM
4. So, caching with Virtual Memory is very effective in hiding latencies.

Simple Page table (Real ones later)



Page Hits and Faults

1. Page Hits: Address being referenced in already in the main memory. E.g. VP2 results in a page hit.
2. Page Faults: a DRAM cache miss is known as a page fault. E.g. VP3 causes a page fault.

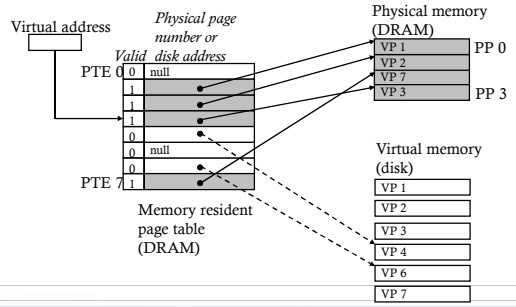
Page Faults

Invokes the page fault exception handler in OS

1. A victim page in main memory is selected and if the victim is dirty then it is written to disk.
2. Page table entry for victim page is modified to reflect that it is no longer in memory.
3. The page faulted page is brought to memory and its Page table entry is updated to reflect this.

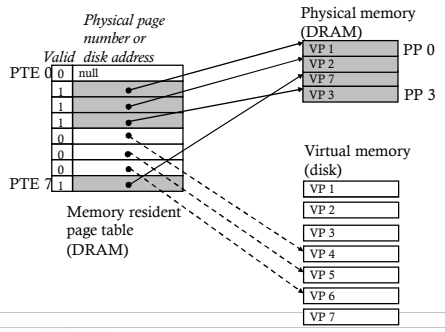
After Handling Page Fault to VP3

Victim chosen is VP4



A new page of virtual memory can be allocated using malloc() or other techniques.

Consider VP5 is being allocated



Why Is this not slow?

Spatial Locality again:

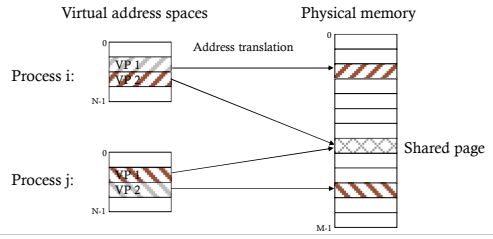
Though the total number of pages that are accessed in the entire run of a program might exceed the total size of physical memory, at any point in time, only a smaller set of active pages called the resident set or working set is accessed.

As a tool for Memory Management

1. Simplifies Linking: E.g. because code, data, stack, heap etc. always starts at same virtual address.
2. Simplifies Loading: On demand loading of necessary memory pages, when needed. Initially marked as "uncached" and maps to the executable file.
3. Simplifies Sharing: E.g. Kernel code and C Library routines can be shared across all processes instead of each with its own copy.
4. Simplifies Memory Allocation: Can be placed in arbitrary physical pages, but will look contiguous in virtual address space.

As a tool for Memory Management

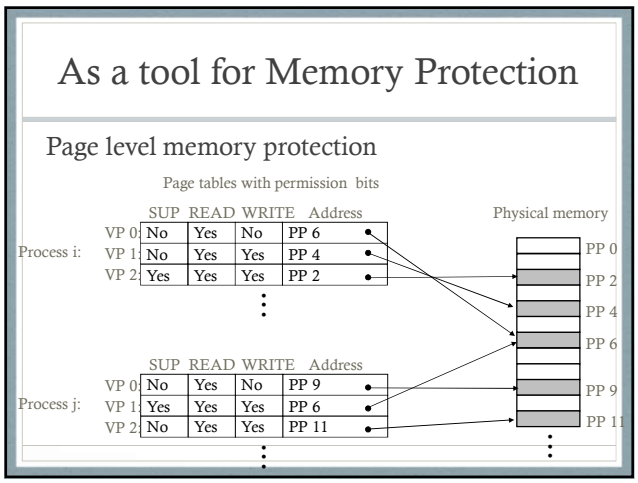
Example of Sharing of Memory between two processes.



As a tool for Memory Protection

Enforced by the Hardware. On a violation, the CPU triggers a general protection fault (often reported as segmentation fault in Unix).

- Read and Write Protection: Can a page be read from or written to?
- Execute Protection: Can a page be executed from?
- Privileged Access Protection: Can a page be accessed from normal user mode or does it need supervisor privileges?


As a tool for Memory Protection
Page level memory protection
Page tables with permission bits
SUP READ WRITE Address
Process i:
VP 0: No Yes No PP 6
VP 1: No Yes Yes PP 4
VP 2: Yes Yes Yes PP 2
SUP READ WRITE Address
Process j:
VP 0: No Yes No PP 9
VP 1: Yes Yes Yes PP 6
VP 2: No Yes Yes PP 11
Physical memory
PP 0
PP 2
PP 4
PP 6
PP 9
PP 11