Virtual Memory

CSE 410, Spring 2006
Computer Systems

http://www.cs.washington.edu/education/courses/410/06sp/

Reading and References

- Reading
  - *Computer Organization and Design, Patterson and Hennessy*
    - Section 7.4 Virtual Memory
    - Section 7.5 A Common Framework for Memory Hierarchies

- Reference
  - Chapter 4, Caches for MIPS, *See MIPS Run*, D. Sweetman

Layout of program memory

```
  7FFF FFFF  RESERVED (4KB)
  7FFF EFFF  stack (grows down)
           ↘
~1792 MB  ↙
  1001 0000  heap (grows up)
  1000 FFFF  global data (64 KB)
  1000 0000  program (252 MB)
  00FF FFFF  reserved (4 MB)
  0040 0000
  003F FFFF
  0000 0000
```

Program Memory Addresses

- Program addresses are fixed at the time the source file is compiled and linked
- Small, simple systems can use program addresses as the physical address in memory
- Modern systems usually much more complex
  - program address space very large
  - other programs running at the same time
  - operating system is in memory too
Direct Physical Addressing

Physical Addressing

- Address generated by the program is the same as the address of the actual memory location
- Simple approach, but lots of problems
  - Only one process can easily be in memory at a time
  - There is no way to protect the memory that the process isn’t supposed to change (i.e., the OS or other processes)
  - A process can only use as much memory as is physically in the computer
  - A process occupies all the memory in its address space, even if most of that space is never used
    - 2 GB for the program and 2 GB for the system kernel

Memory Mapping

Virtual Addresses

- The program addresses are now considered to be “virtual addresses”
- The memory management unit (MMU) translates the program addresses to the real physical addresses of locations in memory
- This is another of the many interface layers that let us work with abstractions, instead of all details at all levels
Paging

- Divide a process's virtual address space into fixed-size chunks (called **pages**)
- Divide physical memory into pages of the same size
- Any virtual page can be located at any physical page
- Translation box converts from virtual pages to physical pages

Multiple Processes Share Memory

- Each process thinks it starts at address 0x0000 and has all of memory
- A process doesn't know anything about physical addresses and doesn't care

Protection

- A process can only use virtual addresses
- A process can't corrupt another process's memory
  » It has no address to refer to it
- How can Blue write to Green's page 2?
  » needs an address to refer to physical page 7, but it doesn't have one

Store Memory on Disk

- Memory that isn't being used can be saved on disk
  » swapped back in when it is referenced via page fault
- Programs can address more memory than is physically available
- This is an important reason for virtual memory
  » too hard for programs to do this on their own (using overlays, for example)
Sparse Address Spaces

- Memory addresses that aren't being used at all don't have to be in memory or on disk
  - Code can start at a very low logical address
  - Stack can start at a very high logical address
  - No physical pages allocated for unused addresses in between

Sharing Memory

- Two processes can share memory by mapping two virtual pages to the same physical page
  - The code for Word can be shared for two Word processes
    - code pages are read only
  - Each process has its own data pages
    - possible to share data pages too, but less common
Page Tables

Offset field is 12 bits
  » so each page is \(2^{12}\) bytes = 4096 bytes = 4KB

Virtual Page Number field is 20 bits
  » so \(2^{20} = 1\) million virtual pages

Page table is an array with one entry for each virtual page
  » 1 million entries
  » entry includes physical page number and flags

Gack!

Each process has a page table with 1 Million entries - *big*
  » no memory left to store the actual programs

Each page table must be referenced for every address reference in a program - *slow*
  » no time left to do any useful work

But wait, system designers are clever kids

Page tables - size problem

The page tables are addressed using virtual addresses in the kernel
Therefore they don’t need physical memory except for the parts that are actually used
  » see “Sparse Address Spaces” diagram

Operating System manages these tables in its own address space
  » kernel address space

Page Tables - speed problem

Use special memory cache for page table entries - Translation Lookaside Buffer
Each TLB entry contains
  » address space ID number (part of the tag)
  » virtual page number (rest of the tag)
  » flags (read only, dirty, etc)
  » associated physical page number (the data)

TLB is a fully associative cache
Using the TLB