10th Planet!? ⇒ Named Sedna, it has a 10,500-year orbit and is the most distant object known to orbit our Sun. They predict more!
Review…

• Cache design choices:
  • size of cache: speed v. capacity
  • direct-mapped v. associative
  • for N-way set assoc: choice of N
  • block replacement policy
  • 2nd level cache?
  • Write through v. write back?

• Use performance model to pick between choices, depending on programs, technology, budget, ...

• Virtual Memory
  • Predates caches; each process thinks it has all the memory to itself; protection!
Virtual to Physical Addr. Translation

- Each program operates in its own virtual address space; only one program running
- Each is protected from the other
- OS can decide where each goes in memory
- Hardware (HW) provides virtual \( \Rightarrow \) physical mapping
Analogy

• Book title like virtual address

• Library of Congress call number like physical address

• Card catalogue like page table, mapping from book title to call number

• On card for book, in local library vs. in another branch like valid bit indicating in main memory vs. on disk

• On card, available for 2-hour in library use (vs. 2-week checkout) like access rights
Simple Example: Base and Bound Reg

- $base$
- $base+$
- $bound$

User A

User B

User C

OS

$\infty$

- Enough space for User D, but discontinuous
  (“fragmentation problem”)
- Want discontinuous mapping
- Process size $\gg$ mem
- Addition not enough!

$\Rightarrow$ use Indirection!
Mapping Virtual Memory to Physical Memory

- Divide into equal sized chunks (about 4 KB - 8 KB)
- Any chunk of Virtual Memory assigned to any chuck of Physical Memory ("page")

64 MB

Virtual Memory

Heap

Static

Physical Memory
## Paging Organization (assume 1 KB pages)

<table>
<thead>
<tr>
<th>Physical Address</th>
<th>Page is unit of mapping</th>
<th>Virtual Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>page 0</td>
<td>page 0</td>
</tr>
<tr>
<td>1K</td>
<td>page 1</td>
<td>page 1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>7168</td>
<td>page 7</td>
<td>page 2</td>
</tr>
</tbody>
</table>

**Physical Memory**

- Page is unit of transfer from disk to physical memory

**Virtual Memory**

- Page also unit of transfer from disk to physical memory

- Addr Trans MAP
- 0
- 1024
- 2048
- 31744
Virtual Memory Mapping Function

• Cannot have simple function to predict arbitrary mapping

• Use table lookup of mappings

<table>
<thead>
<tr>
<th>Page Number</th>
<th>Offset</th>
</tr>
</thead>
</table>

• Use table lookup ("Page Table") for mappings: Page number is index

• Virtual Memory Mapping Function

  • Physical Offset = Virtual Offset

  • Physical Page Number
    = PageTable[Virtual Page Number]

  (P.P.N. also called "Page Frame")
Address Mapping: Page Table

Virtual Address:

Virtual Address = page no. \( + \) offset

Page Table Base Reg

Page Table located in physical memory

Page Table

<table>
<thead>
<tr>
<th>V</th>
<th>Access Rights</th>
<th>Physical Page Address</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Garcia, Spring 2004 © UCB
Page Table

• A page table is an operating system structure which contains the mapping of virtual addresses to physical locations
  • There are several different ways, all up to the operating system, to keep this data around

• Each process running in the operating system has its own page table
  • “State” of process is PC, all registers, plus page table
  • OS changes page tables by changing contents of Page Table Base Register
Requirements revisited

• Remember the motivation for VM:

• Sharing memory with protection
  • Different physical pages can be allocated to different processes (sharing)
  • A process can only touch pages in its own page table (protection)

• Separate address spaces
  • Since programs work only with virtual addresses, different programs can have different data/code at the same address!

• What about the memory hierarchy?
Page Table Entry (PTE) Format

- Contains either Physical Page Number or indication not in Main Memory

- OS maps to disk if Not Valid \((V = 0)\)

<table>
<thead>
<tr>
<th>V</th>
<th>A.R.</th>
<th>P. P. N.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Val-id</td>
<td>Access Rights</td>
<td>Physical Page Number</td>
</tr>
</tbody>
</table>

- If valid, also check if have permission to use page: **Access Rights** (A.R.) may be Read Only, Read/Write, Executable
### Comparing the 2 levels of hierarchy

<table>
<thead>
<tr>
<th>Cache Version</th>
<th>Virtual Memory vers.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block or Line</td>
<td>Page</td>
</tr>
<tr>
<td>Miss</td>
<td>Page Fault</td>
</tr>
<tr>
<td>Block Size: 32-64B</td>
<td>Page Size: 4K-8KB</td>
</tr>
<tr>
<td>Placement:</td>
<td>Fully Associative</td>
</tr>
<tr>
<td>Direct Mapped, N-way Set Associative</td>
<td></td>
</tr>
<tr>
<td>Replacement:</td>
<td>Least Recently Used (LRU)</td>
</tr>
<tr>
<td>LRU or Random</td>
<td></td>
</tr>
<tr>
<td>Write Thru or Back</td>
<td>Write Back</td>
</tr>
</tbody>
</table>
Notes on Page Table

• Solves Fragmentation problem: all chunks same size, so all holes can be used

• OS must reserve “Swap Space” on disk for each process

• To grow a process, ask Operating System
  • If unused pages, OS uses them first
  • If not, OS swaps some old pages to disk
    • (Least Recently Used to pick pages to swap)

• Each process has own Page Table

• Will add details, but Page Table is essence of Virtual Memory
Virtual Memory Problem #1

• Map every address ⇒ 1 indirection via Page Table in memory per virtual address ⇒ 1 virtual memory accesses = 2 physical memory accesses ⇒ SLOW!

• Observation: since locality in pages of data, there must be locality in virtual address translations of those pages

• Since small is fast, why not use a small cache of virtual to physical address translations to make translation fast?

• For historical reasons, cache is called a Translation Lookaside Buffer, or TLB
Translation Look-Aside Buffers (TLBs)

• TLBs usually small, typically 128 - 256 entries

• Like any other cache, the TLB can be direct mapped, set associative, or fully associative

On TLB miss, get page table entry from main memory
## Typical TLB Format

<table>
<thead>
<tr>
<th>Virtual Address</th>
<th>Physical Address</th>
<th>Dirty</th>
<th>Ref</th>
<th>Valid</th>
<th>Access Rights</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- TLB just a cache on the page table mappings
- TLB access time comparable to cache (much less than main memory access time)
- **Dirty**: since use write back, need to know whether or not to write page to disk when replaced
- **Ref**: Used to help calculate LRU on replacement
  - Cleared by OS periodically, then checked to see if page was referenced
What if not in TLB?

• Option 1: Hardware checks page table and loads new Page Table Entry into TLB

• Option 2: Hardware traps to OS, up to OS to decide what to do

• MIPS follows Option 2: Hardware knows nothing about page table
What if the data is on disk?

• We load the page off the disk into a free block of memory, using a DMA (Direct Memory Access – very fast!) transfer
  • Meantime we switch to some other process waiting to be run

• When the DMA is complete, we get an interrupt and update the process's page table
  • So when we switch back to the task, the desired data will be in memory
What if we don't have enough memory?

• We chose some other page belonging to a program and transfer it onto the disk if it is dirty
  • If clean (disk copy is up-to-date), just overwrite that data in memory
  • We chose the page to evict based on replacement policy (e.g., LRU)

• And update that program's page table to reflect the fact that its memory moved somewhere else

• If continuously swap between disk and memory, called Thrashing
Paging/Virtual Memory Review

User A:
Virtual Memory

∞
Stack

Static
Code

0
Page Table

A

Physical Memory

64 MB

User B:
Virtual Memory

∞
Stack

Static
Code

0
Page Table

B

TLB
Peer Instructions

A. Locality is important yet different for cache and virtual memory (VM): temporal locality for caches but spatial locality for VM

B. Cache management is done by hardware (HW), page table management by the operating system (OS), but TLB management is either by HW or OS

C. VM helps both with security and cost
And in conclusion…

• Manage memory to disk? Treat as cache
  • Included protection as bonus, now critical
  • Use Page Table of mappings for each user vs. tag/data in cache
  • TLB is cache of Virtual⇒Physical addr trans

• Virtual Memory allows protected sharing of memory between processes

• Spatial Locality means Working Set of Pages is all that must be in memory for process to run fairly well