Review: Computer Organization

Virtual Memory

Contents

• Virtual memory
• Page table and TLB
• Interaction of virtual memory with cache
Virtual Memory: Motivation

- Historically, motivations for VM are
  - Allow efficient and safe sharing of memory among multiple programs
  - Remove the programming burdens of a small, limited amount of main memory
    - Programmers make overlays and load/unload manually
    - Not that important now

- A number of programs are running concurrently
  - The total memory required to run all the programs exceeds the main memory available
  - But since only a small fraction of this memory is actively being used at any point of time, we can “cache” them in main memory and put others in disk
  - I.e., use disk to simulate more memory!

- Multiple users on same computer
  - Want object code to have fixed addresses.
  - How to relocate multiple users’ programs into same memory?
  - How to protect one user program from the other?
  - Can automatically relocate code at run-time – separate address space per program!

Most common solution is virtual memory.

Can address more virtual memory than physical memory.
So, think of disk as the “usual” place to store VM.
Then, physical memory is mainly a cache for the VM on the disk.

So, VM very similar to caches.
Different motivation.
Different evolution → different terminology.

VM page = cache block.
VM page fault = cache miss.
Virtual Memory: Motivation

CPU address (memory addr.)
cache hit
cache miss -> goto Memory

CPU address (VM addr.)
page hit
page fault -> goto VM (disk)

Virtual Memory = Cache

Willing to do a lot to minimize access to the VERY SLOW disk.
Disk access: ≈ 100K-1M cycles!

Choose caching strategies which minimize disk access, & thus misses:

• Fully associative
• Write-back
  Reduce disk access on writes by grouping them.
• Approximate LRU
  Exact LRU too expensive.
• Fairly large pages (typically 4-64KB)
  Amortize high access time. Not too large to lose good spatial locality.
Virtual Memory: Mapping (= Block Placement)

- Main memory can act as a cache for the secondary storage (disk)
  
  This is based on mapping table or “page table”

- Why fully associate?
  - Huge miss penalty (100k-1M cycles): the data is not in memory, retrieve it from disk; it is called Page faults
  - Pages should be fairly large (e.g., 4KB)
  - LRU is worth the price
  - Handle the faults in software instead of hardware
  - Writeback

Page Tables for Address Translation (VA to PA)

4KB page
VM: 4GB (1M pages)
PM: 512MB (256K pages)
Page Tables for Address Translation (VA to PA)

Starting address of the page table in physical memory

1M entries x 32 bits = 4MB !!!
(page table size)

A Performance Problem

Every VM access uses multiple memory accesses:
- 1 for page table (or more if multi-level).
- 1 for data.

What’s our general method for improving memory access performance?

? ?

Caches. Let’s add a cache for the page table!
Translation Look-aside Buffer (TLB)

TLB = Hardware cache for page table access.

Only one TLB, not one per process.
Either caches for all processes or only for current process.
Will see issues...

Only cache page table entries which map to physical memory.
Speed up the common case.
Disk access is VERY SLOW – an additional page table access is negligible.

Low miss rate because of locality & large page size.
Usually write-back to minimize memory access.

Making Address Translation Fast

- A cache for address translations: translation lookaside buffer

TLB size = 32–4096 entries
Tag size = 20-bit (V.PN) if fully associative
Example: DECStation 3100 (MIPS R2000)

Page size = VM space = PM space = No. entries in TLB = TLB access method = Block size = Tag size = No. entries in cache = Cache access method =

Example

- **Virtual Memory**
  - Page size is 256 bytes ($2^8$)
  - Virtual memory is 2048 bytes ($2^{11}$, 0-0x7ff) -> $2^3 = 8$ pages
  - (Physical) Memory has 1024 bytes ($2^{10}$, 0-0x3ff) -> $2^2 = 4$ pages

- **TLB**
  - TLB has 2 entries

- **Cache**
  - Block size is 16 bytes of data ($2^4$, 0-0xf)
  - Memory has 1024 bytes ($2^{10}$, 0-0x3ff) -> $2^6 = 64$ blocks
  - Cache has 256 bytes ($2^8$, 0-0xff) -> $2^4 = 16$ blocks
  - “Direct mapped“
Virtual Memory

Cache
Where to put Page Table?

Expensive memory access to access PT => TLB
Now, the questions...

- Virtual address (VA) [776]
  - TLB hit, translated to [176]
  - Cache hit (block index=7, tag=1)

- VA [133]
  - TLB hit, translated [233]
  - Cache miss (block index=3, tag=2)

- VA [309]
  - TLB miss, PT (page table) lookup, translated to [009]
  - Cache hit (block index=0, tag=0)

- VA [452]
  - TLB miss, PT lookup, Page fault

---

Cache & VM Interaction

Virtual address = VPN + Page offset

TLB miss

VPN in TLB

TLB hit

Access memory to get PPN from PT

Page fault

TLB hit

Obtain PPN from PT

Update TLB

TLB Page Cache

<table>
<thead>
<tr>
<th>TLB</th>
<th>Page</th>
<th>Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hit</td>
<td>Hit</td>
<td>Hit</td>
</tr>
<tr>
<td>Hit</td>
<td>Hit</td>
<td>Miss</td>
</tr>
<tr>
<td>Hit</td>
<td>Miss</td>
<td>Hit</td>
</tr>
<tr>
<td>Miss</td>
<td>Hit</td>
<td>Hit</td>
</tr>
<tr>
<td>Miss</td>
<td>Hit</td>
<td>Miss</td>
</tr>
<tr>
<td>Miss</td>
<td>Miss</td>
<td>Hit</td>
</tr>
<tr>
<td>Miss</td>
<td>Miss</td>
<td>Miss</td>
</tr>
</tbody>
</table>

Obtain PPN from PT

Update PT

Update TLB

Physical address = PPN + Page offset

= Tag + Index + Block offset

Block in cache

Cache miss

Access memory

To get the block

Cache hit

Update cache

Obtain the block

Obtain the desired data

Access memory

To get PPN from PT

Obtain PPN from PT

Update PT

Update TLB

Access disk to get PPN

Update PT

Update TLB

Physical address = PPN + Page offset

= Tag + Index + Block offset
Cache & VM Interaction: Details of a Single Access

Access TLB.
Hit? →
   TLB gives physical address of data.
   No need to look at page table.

Access physical memory cache. (Assume one level cache for brevity.)
Hit? →
   Cache gives data value.
   Best case. No memory or disk access at all!

Miss? →
   Access memory for data value.
   Update cache.

Cache & VM Interaction: Details of a Single Access

... TLB Miss? →
   Access page table in physical memory.
      (Assume single page table for brevity.)
      (Page table entries could be in cache. This possibility omitted for brevity.)
   No page fault? →
      Page table gives physical memory address for data value.
      Update TLB.
      Access physical memory cache.
      Hit? →
         Cache gives data value.
      Miss? →
         Access memory for data value.
         Update cache.
Cache & VM Interaction: Details of a Single Access

... TLB Miss? → ...

Page fault? →

Page table gives disk location for data value.
Access disk for data value.

(Disks usually have caches too! This possibility omitted for brevity.)

Update memory with this page.
Update page table & TLB.

Worst case. Both memory & disk access!

Memory Performance

- Most memory accesses require a sequence of two cache accesses
  - TLB cache access
  - Cache access

- How can it be faster?
  - Overlapping TLB & Cache access
Virtual address = VPN + Page offset

TLB hit

Access memory to get PPN
Update PT
Update TLB

Access disk to get PPN
Update PT
Update TLB

Physical address = PPN + Page offset
= Tag + Index + Block offset

TLB hit

Access memory to get the block
Update cache

Virtual address = VPN + Page offset

TLB miss

Update TLB

Access memory to get the block

Physical address = PPN + Page offset
= Tag + Index + Block offset

TLB hit

Access memory to get the block
Update cache

Physical address = PPN + Page offset
= Tag + Index + Block offset

TLB miss

Update TLB