Virtual Memory

Instructor: Justin Hsia
Review of Last Lecture

• Multiple instruction issue increases max speedup, but higher penalty for a stall
  – Superscalar because can achieve CPI < 1
  – Requires a significant amount of extra hardware

• Employ more aggressive scheduling techniques to increase performance
  – Register renaming
  – Speculation (guessing)
  – Out-of-order execution
Agenda

• Virtual Memory
• Page Tables
• Administrivia
• Translation Lookaside Buffer (TLB)
• VM Performance
Memory Hierarchy

Earlier: Caches

Next Up: Virtual Memory

Upper Level Faster

Larger Lower Level

Regs

Instr Operands

L1 Cache

Blocks

L2 Cache

Blocks

Memory

Pages

Disk

Files

Tape
Virtual Memory

• Next level in the memory hierarchy
  – Provides illusion of very large main memory
  – Working set of “pages” residing in main memory
    (subset of all pages residing on disk)

• **Main goal:** Avoid reaching all the way back to disk as much as possible

• **Additional goals:**
  – Let OS share memory among many programs and protect them from each other
  – Each process thinks it has all the memory to itself
Virtual to Physical Address Translation

- Each program operates in its own virtual address space and thinks it’s the only program running
- Each is protected from the other
- OS can decide where each goes in memory
- Hardware gives virtual \(\rightarrow\) physical mapping
Agenda

• Virtual Memory
• Page Tables
• Administrivia
• Translation Lookaside Buffer (TLB)
• VM Performance
• VM Wrap-up
• Divide into equal sized chunks (about 4 KiB - 8 KiB)
• Any chunk of Virtual Memory can be assigned to any chunk of Physical Memory ("page")

Mapping VM to PM

Virtual Memory

∞

Heap

Static

Physical Memory

64 MB
Virtual Memory Mapping Function

• How large is main memory? Disk?
  – Don’t know! Designed to be interchangeable components
  – Need a system that works regardless of sizes

• Use lookup table (page table) to deal with arbitrary mapping
  – Index lookup table by # of pages in VM (not all entries will be used/valid)
  – Size of PM will affect size of stored translation
Address Mapping: Page Table

• **Page Table functionality:**
  – Incoming request is Virtual Address (VA), want Physical Address (PA)
  – Physical Offset = Virtual Offset (page-aligned)
  – So just swap Virtual Page Number (VPN) for Physical Page Number (PPN)

**Table:**

| Physical Page # | Virtual Page # | Page Offset |

• **Implementation?**
  – Use VPN as index into PT
  – Store PPN and management bits (Valid, Access Rights)
  – Does NOT store actual data (the data sits in PM)
### Page Table Layout

**Virtual Address:**
- VPN
- Offset

1) Index into PT using VPN

<table>
<thead>
<tr>
<th>V</th>
<th>AR</th>
<th>PPN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>XX</td>
<td></td>
</tr>
</tbody>
</table>

2) Check Valid and Access Rights bits

3) Combine PPN and offset

4) Use PA to access memory

- Physical Address
Page Table Entry Format

• Contains either PPN or indication not in main memory
• **Valid** = Valid page table entry
  – 1 → virtual page is in physical memory
  – 0 → OS needs to fetch page from disk
• **Access Rights** checked on every access to see if allowed (provides protection)
  – *Read Only*: Can read, but not write page
  – *Read/Write*: Read or write data on page
  – *Executable*: Can fetch instructions from page
Page Tables (1/2)

• A page table (PT) contains the mapping of virtual addresses to physical addresses

• Page tables located in main memory – Why?
  – Too large to fit in registers ($2^{20}$ entries for 4 KiB pages)
  – Faster to access than disk and can be shared by multiple processors

• The OS maintains the PTs
  – Each process has its own page table
    • “State” of a process is PC, all registers, and PT
  – OS stores address of the PT of the current process in the Page Table Base Register
Page Tables (2/2)

• *Solves fragmentation problem*: all pages are the same size, so can utilize all available slots

• OS must reserve “*swap space*” on disk for each process
  – Running programs requires hard drive space!

• To grow a process, ask Operating System
  – If unused pages in PM, OS uses them first
  – If not, OS swaps some old pages (LRU) to disk
Paging/Virtual Memory Multiple Processes

User A:
Virtual Memory

User B:
Virtual Memory

Physical Memory

64 MB

Stack

Static

Code

Page Table A

Page Table B

0

0

0

0

8/05/2013
Summer 2013 -- Lecture #24
Review: Paging Terminology

• Programs use *virtual addresses (VAs)*
  – Space of all virtual addresses called *virtual memory (VM)*
  – Divided into pages indexed by *virtual page number (VPN)*

• Main memory indexed by *physical addresses (PAs)*
  – Space of all physical addresses called *physical memory (PM)*
  – Divided into pages indexed by *physical page number (PPN)*
**Question:** How many bits wide are the following fields?

- 16 KiB pages
- 40-bit virtual addresses
- 64 GiB physical memory

<table>
<thead>
<tr>
<th></th>
<th>VPN</th>
<th>PPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A)</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>(B)</td>
<td>24</td>
<td>20</td>
</tr>
<tr>
<td>(C)</td>
<td>22</td>
<td>22</td>
</tr>
<tr>
<td>(D)</td>
<td>26</td>
<td>22</td>
</tr>
</tbody>
</table>
Agenda

• Virtual Memory
• Page Tables
• Administrivia
• Translation Lookaside Buffer (TLB)
• VM Performance
• VM Wrap-up
Administrivia (1/2)

• Project 3 (individual) due Sunday 8/11
• Final Review – Tue 8/13, 7-10pm in 10 Evans
• Final – Fri 8/16, 9am-12pm, 155 Dwinelle
  – 2\textsuperscript{nd} half material + self-modifying MIPS
  – MIPS Green Sheet provided again
  – Two two-sided handwritten cheat sheets
    • Can re-use your midterm cheat sheet!
Administrivia (2/2)

"Only the bolded text in the notes will be on the test"

Everything is bolded

http://www.geekosystem.com/engineering-professor-meme/2/
Agenda

- Virtual Memory
- Page Tables
- Administrivia
- Translation Lookaside Buffer (TLB)
- VM Performance
- VM Wrap-up
Retrieving Data from Memory

1) Access page table for address translation

2) Access correct physical address

Requires two accesses of physical memory!
Virtual Memory Problem

• 2 physical memory accesses per data access = SLOW!

• Since locality in pages of data, there must be locality in the translations of those pages

• Build a separate cache for the Page Table
  – For historical reasons, cache is called a Translation Lookaside Buffer (TLB)
  – Notice that what is stored in the TLB is NOT data, but the VPN → PPN mapping translations
TLBs vs. Caches

- TLBs usually small, typically 16 – 512 entries
- TLB access time comparable to cache (« main memory)
- TLBs usually are fully/highly associativity
Where Are TLBs Located?

- Which should we check first: Cache or TLB?
  - Can cache hold requested data if corresponding page is not in physical memory? **No**
  - With TLB first, does cache receive VA or PA?

Notice that it is now the TLB that does translation, not the Page Table!
Address Translation Using TLB

TLB:
- TLB Tag
- TLB Index
- Page Offset

Virtual Address:
 VPN

Physical Address:
- PPN
- Page Offset

Data Cache:
- Tag
- Block Data

Note: TIO for VA & PA unrelated
Typical TLB Entry Format

<table>
<thead>
<tr>
<th>Valid</th>
<th>Dirty</th>
<th>Ref</th>
<th>Access Rights</th>
<th>TLB Tag</th>
<th>PPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>XX</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Valid** and **Access Rights**: Same usage as previously discussed for page tables
- **Dirty**: Basically always use write-back, so indicates whether or not to write page to disk when replaced
- **Ref**: Used to implement LRU
  - Set when page is accessed, cleared periodically by OS
  - If Ref = 1, then page was referenced recently
- **TLB Index**: VPN mod (# TLB sets)
- **TLB Tag**: VPN minus TLB Index (upper bits)
**Question:** How many bits wide are the following?

- 16 KiB pages
- 40-bit virtual addresses
- 64 GiB physical memory
- 2-way set associative TLB with 512 entries

<table>
<thead>
<tr>
<th>Valid</th>
<th>Dirty</th>
<th>Ref</th>
<th>Access Rights</th>
<th>TLB Tag</th>
<th>PPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>XX</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TLB Tag</th>
<th>TLB Index</th>
<th>TLB Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A) 12</td>
<td>14</td>
<td>38</td>
</tr>
<tr>
<td>(B) 18</td>
<td>8</td>
<td>45</td>
</tr>
<tr>
<td>(C) 14</td>
<td>12</td>
<td>40</td>
</tr>
<tr>
<td>(D) 17</td>
<td>9</td>
<td>43</td>
</tr>
</tbody>
</table>
Fetching Data on a Memory Read

1) Check TLB (input: VPN, output: PPN)
   - **TLB Hit**: Fetch translation, return PPN
   - **TLB Miss**: Check page table (in memory)
     • **Page Table Hit**: Load page table entry into TLB
     • **Page Table Miss (Page Fault)**: Fetch page from disk to memory, update corresponding page table entry, then load entry into TLB

2) Check cache (input: PPN, output: data)
   - **Cache Hit**: Return data value to processor
   - **Cache Miss**: Fetch data value from memory, store it in cache, return it to processor
Page Faults

• Load the page off the disk into a free page of memory
  – Switch to some other process while we wait
• Interrupt thrown when page loaded and the process' page table is updated
  – When we switch back to the task, the desired data will be in memory
• If memory full, replace page (LRU), writing back if necessary, and update both page table entries
  – Continuous swapping between disk and memory called “thrashing”
Performance Metrics

• VM performance also uses Hit/Miss Rates and Miss Penalties
  – *TLB Miss Rate:* Fraction of TLB accesses that result in a TLB Miss
  – *Page Table Miss Rate:* Fraction of PT accesses that result in a page fault

• Caching performance definitions remain the same
  – Somewhat independent, as TLB will always pass PA to cache regardless of TLB hit or miss
Data Fetch Scenarios

Are the following scenarios for a single data access possible?

- TLB Miss, Page Fault  Yes
- TLB Hit, Page Table Hit  No
- TLB Miss, Cache Hit  Yes
- Page Table Hit, Cache Miss  Yes
- Page Fault, Cache Hit  No
**Question:** A program tries to load a word at X that causes a TLB miss but not a page fault. Are the following statements TRUE or FALSE?

1) The page table does not contain a valid mapping for the virtual page corresponding to the address X

2) The word that the program is trying to load is present in physical memory

(A) F F
(B) F T
(C) T F
(D) T T
Updating Scenarios

• Using $V = \text{valid}$, $D = \text{dirty}$, $R = \text{ref}$ to mean that field is set to the shown value for any entry in either PT or TLB.

• Which of the following scenarios for a single data access are possible?
  - Read, $D = 1$: No
  - Write, $R = 1$: Yes
  - Read, $V = 0$: Yes
  - Write, $D = 0$: No
Get To Know Your Staff

- Category: Wishlist
Agenda

• Virtual Memory
• Page Tables
• Administrivia
• Translation Lookaside Buffer (TLB)
• VM Performance
• VM Wrap-up
VM Performance

• Virtual Memory is the level of the memory hierarchy that sits *below* main memory
  – TLB comes *before* cache, but affects transfer of data from disk to main memory
  – Previously we assumed main memory was lowest level, now we just have to account for disk accesses

• Same CPI, AMAT equations apply, but now treat main memory like a mid-level cache
Typical Performance Stats

**Caching**
- cache entry
- cache block (∼32 bytes)
- cache miss rate (1% to 20%)
- cache hit (∼1 cycle)
- cache miss (∼100 cycles)

**Demand paging**
- page frame
- page (∼4Ki bytes)
- page miss rate (<0.001%)
- page hit (∼100 cycles)
- page fault (∼5M cycles)
Impact of Paging on AMAT (1/2)

• Memory Parameters:
  – L1 cache hit = 1 clock cycles, hit 95% of accesses
  – L2 cache hit = 10 clock cycles, hit 60% of L1 misses
  – DRAM = 200 clock cycles ($\approx$100 nanoseconds)
  – Disk = 20,000,000 clock cycles ($\approx$10 milliseconds)

• Average Memory Access Time (no paging):
  – $1 + 5\% \times 10 + 5\% \times 40\% \times 200 = 5.5$ clock cycles

• Average Memory Access Time (with paging):
  – $5.5$ (AMAT with no paging) + ?
Impact of Paging on AMAT (2/2)

• Average Memory Access Time (with paging) =
  • \[5.5 + 5\% \times 40\% \times (1 - HR_{Mem}) \times 20,000,000\]

• AMAT if \(HR_{Mem} = 99\%\)?
  • \[5.5 + 0.02 \times 0.01 \times 20,000,000 = 4005.5 \approx 728x\] slower
  • 1 in 20,000 memory accesses goes to disk: 10 sec program takes 2 hours!

• AMAT if \(HR_{Mem} = 99.9\%\)?
  • \[5.5 + 0.02 \times 0.001 \times 20,000,000 = 405.5\]

• AMAT if \(HR_{Mem} = 99.9999\%\)
  • \[5.5 + 0.02 \times 0.000001 \times 20,000,000 = 5.9\]
Impact of TLBs on Performance

• Each TLB miss to Page Table ~ L1 Cache miss
• TLB Reach: Amount of virtual address space that can be simultaneously mapped by TLB:
  – TLB typically has 128 entries of page size 4-8 KiB
  – \(128 \times 4 \text{ KiB} = 512 \text{ KiB} = \text{just 0.5 MiB}\)
• What can you do to have better performance?
  – Multi-level TLBs Conceptually same as multi-level caches
  – Variable page size (segments)
  – Special situationally-used “superpages” Not covered here
Agenda

• Virtual Memory
• Page Tables
• Administrivia
• Translation Lookaside Buffer (TLB)
• VM Performance
• VM Wrap-up
Virtual Memory Motivation

• Memory as cache for disk (reduce disk accesses)
  – Disk is so slow it significantly affects performance
  – Paging maximizes memory usage with large, evenly-sized pages that can go anywhere

• Allows processor to run multiple processes simultaneously
  – Gives each process illusion of its own (large) VM
  – Each process uses standard set of VAs
  – Access rights provide *protection*
Paging Summary

• Paging requires address *translation*
  – Can run programs larger than main memory
  – Hides variable machine configurations (RAM/HD)
  – Solves fragmentation problem

• Address mappings stored in page tables in memory
  – Additional memory access mitigated with TLB
  – Check TLB, then Page Table (if necessary), then Cache
Hardware/Software Support for Memory Protection

• Different tasks can share parts of their virtual address spaces
  – But need to protect against errant access
  – Requires OS assistance

• Hardware support for OS protection
  – Privileged supervisor mode (a.k.a. *kernel mode*)
  – Privileged instructions
  – Page tables and other state information only accessible in supervisor mode
  – System call exception (e.g. *syscall* in MIPS)
Context Switching

• How does a single processor run many programs at once?

• *Context switch*: Changing of internal state of processor (switching between processes)
  – Save register values (and PC) and change value in Page Table Base register

• What happens to the TLB?
  – Current entries are for a different process (similar VAs, though!)
  – Set all entries to invalid on context switch
Summary

• User program view:
  – Contiguous memory
  – Start from some set VA
  – “Infinitely” large
  – Is the only running program

• Reality:
  – Non-contiguous memory
  – Start wherever available memory is
  – Finite size
  – Many programs running simultaneously

• Virtual memory provides:
  – Illusion of contiguous memory
  – All programs starting at same set address
  – Illusion of ~ infinite memory (2^{32} or 2^{64} bytes)
  – Protection, Sharing

• Implementation:
  – Divide memory into chunks (pages)
  – OS controls page table that maps virtual into physical addresses
  – memory as a cache for disk
  – TLB is a cache for the page table