Paging: Faster Translations (TLBs)

CS 537: Introduction to Operating Systems

Louis Oliphant & Tej Chajed

University of Wisconsin - Madison

Spring 2024

Administrivia

- P3 out, due Feb 20th at 11:59pm
- Form to request alternate exam time (see Piazza post)
- Tej's office hours: MW 10-11am, CS 7361

Review: Paging

- Paging is dividing up a process's address space into equally sized sections (called **pages**) and dividing memory into the same sized sections (called **page frames**)
- A process's **page table** keeps track of the mappings from **virtual page number (VPN)** to **physical frame number (PFN)**
- Virtual Address Translation Process:
  1. Extract **VPN** from virtual address
  2. Calculate address of **PTE**
  3. Read PTE **from memory**
  4. Extract **PFN**
  5. Build **Physical Address**
  6. Read contents of **PA** from memory into register

Paging Example

./paging-linear-translate.py

```
address space size 16k
phys mem size 64k
page size 4k

The format of the page table is simple:
The high-order (left-most) bit is the VALID bit.
  If the bit is 1, the rest of the entry is the PFN.
  If the bit is 0, the page is not valid.

Page Table (from entry 0 down to the max size)
  0x8000000c
  0x00000000
  0x00000000
  0x80000006

Virtual Address Trace
  VA 0x00003229 (decimal:    12841) --> PA or invalid address?
  VA 0x00001369 (decimal:     4969) --> PA or invalid address?
  VA 0x00001e80 (decimal:     7808) --> PA or invalid address?
  VA 0x00002556 (decimal:     9558) --> PA or invalid address?
  VA 0x00003a1e (decimal:    14878) --> PA or invalid address?

For each virtual address, write down the physical address it translates to
OR write down that it is an out-of-bounds address (e.g., segfault).
```

Paging Example

./paging-linear-translate.py -c

```
address space size 16k
phys mem size 64k
page size 4k

The format of the page table is simple:
The high-order (left-most) bit is the VALID bit.
  If the bit is 1, the rest of the entry is the PFN.
  If the bit is 0, the page is not valid.

Page Table (from entry 0 down to the max size)
  0x8000000c
  0x00000000
  0x00000000
  0x80000006

Virtual Address Trace
  VA 0x00003229 (decimal:    12841) --> 00006229 (decimal    25129) [VPN 3]
  VA 0x00001369 (decimal:     4969) -->  Invalid (VPN 1 not valid)
  VA 0x00001e80 (decimal:     7808) -->  Invalid (VPN 1 not valid)
  VA 0x00002556 (decimal:     9558) -->  Invalid (VPN 2 not valid)
  VA 0x00003a1e (decimal:    14878) --> 00006a1e (decimal    27166) [VPN 3]
```

Memory access flow

14-bit addresses
`0x0010: movl 0x1100, %edi`

Assume PT is at phys addr `0x5000`
Assume PTEs are 4 bytes
Assume 4KB pages
How many bits for offset? 12

Page table:

2
0
80
99

Fetch instruction at logical addr `0x0010`

- Access page table to get ppn for vpn 0
- Mem ref 1:
- Learn vpn 0 is at ppn:
- Fetch instruction at: ____ (Mem ref 2)

Exec, load from logical addr 0x1100

- Access page table to get ppn for vpn 1
- Mem ref 3:
- Learn vpn 1 is at ppn:
- `movl` from ____ (Mem ref 4)

Memory access flow

14-bit addresses
`0x0010: movl 0x1100, %edi`

Assume PT is at phys addr `0x5000`
Assume PTEs are 4 bytes
Assume 4KB pages
How many bits for offset? 12

Page table:

2
0
80
99

Fetch instruction at logical addr `0x0010`

- Access page table to get ppn for vpn 0
- Mem ref 1: **`0x5000`**
- Learn vpn 0 is at ppn: **2**
- Fetch instruction at: **`0x2010`** (Mem ref 2)

Exec, load from logical addr 0x1100

- Access page table to get ppn for vpn 1
- Mem ref 3: **`0x5004`**
- Learn vpn 1 is at ppn: **0**
- `movl` from **`0x0100`** (Mem ref 4)

Quiz 6: Segmentation and Paging

https://tinyurl.com/cs537-sp24-q6

Paging Disadvantages

- What was one memory access becomes two
  - first to look up the VPN→PFN translation (in the page table)
  - second to access the memory location
- Additional memory must be used to store the page tables
  - 4KB pages with 32-bit virtual addresses requires storing 1M page table entries per process

MMU's Cache – Reducing Memory Accesses for PTE lookups

- The **translation-lookaside buffer (TLB)** is part of the CPU's memory management unit
- It is a hardware cache of virtual-to-physical address translations
- The MMU first checks the TLB to see if translation mapping is there

TLBs are important for performance

- Typical TLBs might have 32 or 64 entries
- Extremely fast hit time
- Having high **hit rate** (# hits / # lookups) in TLB is extremely important for runtime performance

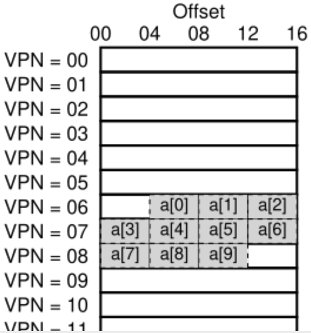
TLB Control Flow Algorithm

```
VPN = (VirtualAddress & VPN_MASK) >> SHIFT
(Success, TlbEntry) = TLB_Lookup(VPN)
if (Success == True)                            // TLB Hit
   if (CanAccess(TLBEntry.ProtectBits) == True)
      Offset = VirtualAddress & OFFSET_MASK
      PhysAddr = (TlbEntry.PFN << SHIFT) | Offset
      Register = AccessMemory(PhysAddr)
   else
      RaiseException(PROTECTION_FAULT)
else                                          // TLB Miss (OS or MMU handles)
   PTEAddr = PTBR + (VPN*sizeof(PTE))
   PTE = AccessMemory(PTEAddr)
   if (PTE.Valid == False)
      RaiseException(SEGMENTATION_FAULT)
   else if (CanAccess(PTE.ProtectBits) == False)
      RaiseException(PROTECTION_FAULT)
   else
      TLB_Insert(VPN, PTE.PFN, PTE.ProtectBits)
      RetryInstruction()
```

Example: Accessing An Array

- Sequential access is fast – only the first access to an element on the page yields a TLB miss.
- Takes advantage of **spatial locality** (referencing items close in address space)
- TLB also takes advantage of **temporal locality** (re-referencing of same address close in time).
- How would hit rate of sequential access compare to hit rate of random access?

```
int sum = 0;
for (i=0; i<10; i++)
{
   sum += a[i];
}
```

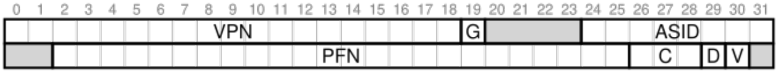


Context Switches

- Recall that a page table is unique to a specific process
- On a context switch the TLB will be full of translations for old process
- Need to flush the TLB (but causes TLB misses after switch)
- Can use Address Space Identifiers (ASID) to divide TLB per process

TLB Contents

Example MIPS TLB Entry



- VPN – used for lookup
- PFN – change the Virtual address VPN to PFN
- G – global bit (shared by all processes, don't check ASID)
- ASID – Address Space Identifier (which process's Page Table)
- D – dirty bit (changed when page has been written to)
- V – valid bit (valid translation present in entry)

TLB Replacement Policy

- When installing a new entry in the TLB, need to **replace** an old one – **which one?**
- One common approach is evict the **Least Recently Used (LRU)** entry
- Another typical approach is to evict a **random** entry
  - random avoids corner-case behaviors; for example, when a program loops over n+1 pages with a TLB of size n
  – in this case the LRU misses upon every access.

Summary

- TLB solves (or at least significantly reduces) the number of memory lookups for pagetable entries
- TLB misses can be handled by hardware (the MMU) or software (the OS)
- Different Strategies for Context Switches (flush or ASID portion)
- Different Replacement policies for TLB entries

- Next Time: talk about how to shrink the page table