Modern Virtual Memory Systems

Mengjia Yan
Computer Science and Artificial Intelligence Laboratory
M.I.T.

Based on slides from Daniel Sanchez
Recap: Virtual Memory Systems

Illusion of a large, private, uniform store

Protection & Privacy
- several users, each with their private address space and one or more shared address spaces
  - page table ≡ name space

Demand Paging
- Provides the ability to run programs larger than the primary memory
- Hides differences in machine configurations

The price is address translation on each memory reference
Recap: Hierarchical Page Table

Virtual Address

31 22 21 12 11 0

p1 p2 offset

10-bit 10-bit
L1 index L2 index

Root of the Current Page Table

(Processor Register)

p1

Level 1 Page Table

p2

Level 2 Page Tables

offset

page in primary memory

page in secondary memory

PTE of a nonexistent page

Data Pages
Recap: Translation Lookaside Buffers

Address translation is very expensive!

- In a two-level page table, each reference becomes several memory accesses

Solution: *Cache translations in TLB*

TLB hit  \(\Rightarrow\) *Single-cycle Translation*

TLB miss  \(\Rightarrow\) *Page Table Walk to refill*

\[ \begin{array}{c|c|c|c|c|c|c|c|c} V & R & W & D & \text{tag} & \text{PPN} & \text{VPN} & \text{offset} \\ \hline \end{array} \]

\( (\text{VPN} = \text{virtual page number}) \)

\( (\text{PPN} = \text{physical page number}) \)
Recap: TLB Designs

- Typically 32-128 entries, usually fully associative
  - Each entry maps a large page, hence less spatial locality across pages ➔ more likely that two entries conflict
  - Sometimes larger TLBs (256-512 entries) are 4-16 way set-associative

- Random or FIFO replacement policy

- No process information in TLB?

- TLB Reach: Size of largest virtual address space that can be simultaneously mapped by TLB

Example: 64 TLB entries, 4KB pages, one page per entry

TLB Reach = \( \text{64 entries} \times 4 \text{ KB} = 256 \text{ KB (if contiguous)} \)
Variable-Sized Page Support

Virtual Address

31 22 21 12 11 0

p1 p2 offset

10-bit 10-bit

L1 index L2 index

Root of the Current
Page Table

(Processor
Register)

Level 1
Page Table

p1

p2

offset

Level 2
Page Tables

Data Pages

page in primary memory
large page in primary memory
page in secondary memory
PTE of a nonexistent page
Variable-Size Page TLB

- Chicken-and-egg problem
- Q: how about set-associative TLBs?
- Some systems use separate TLBs for different page sizes.

![Diagram of TLB operation with virtual and physical address mapping]
Handling a TLB Miss

- **Software (MIPS, Alpha)**
  - TLB miss causes an exception and the operating system walks the page tables and reloads TLB.
  - A privileged “untranslated” addressing mode used for walk

- **Hardware (SPARC v8, x86, PowerPC)**
  - A memory management unit (MMU) walks the page tables and reloads the TLB
  - If a missing (data or PT) page is encountered during the TLB reloading, MMU gives up and signals a Page-Fault exception for the original instruction

Q: Pros and cons of software and hardware approaches?
Hierarchical Page Table Walk: SPARC v8

Virtual Address

Context Table Register

Context Register

root ptr

Context Table

L1 Table

PTP

L2 Table

PTP

L3 Table

PTP

PTE

Physical Address

PPN

Offset

MMU does this table walk in hardware on a TLB miss
Address Translation: putting it all together

Virtual Address

TLB Lookup

hit

Protection Check

permitted

Physical Address (to cache)

denied

Protection Fault

Update TLB

Page Table Walk

the page is

∈ memory

≠ memory

Page Fault (OS loads page)

Where?

SEGFAULT

miss
Topics

- Interrupts
- Speeding up the common case:
  - TLB & Cache organization
- Speeding up page table walks
- Modern Usage

Diagram:
- Virtual Address
- TLB Lookup
  - miss
  - hit
    - Page Table Walk
      - the page is not in memory
      - the page is in memory
    - Page Fault (OS loads page)
    - Update TLB
    - Protection Check
      - permitted
      - denied
      - Protection Fault
        - Physical Address (to cache)
        - SEGFAULT
        - Where?
Interrupts: altering the normal flow of control

An external or internal event that needs to be processed by another (system) program. The event is usually unexpected or rare from program’s point of view.
Causes of Interrupts

Interrupt: an event that requests the attention of the processor

• Asynchronous: an external event
  – input/output device service-request
  – timer expiration
  – power disruptions, hardware failure

• Synchronous: an internal event (a.k.a exception)
  – undefined opcode, privileged instruction
  – arithmetic overflow, FPU exception
  – misaligned memory access
  – virtual memory exceptions:
    • page faults, TLB misses, protection violations
  – traps: system calls, e.g., jumps into kernel
Asynchronous Interrupts: invoking the interrupt handler

- An I/O device requests attention by asserting one of the *prioritized interrupt request lines*
- Tricky: interrupted thread cannot anticipate/prepare for this control transfer
- When the processor decides to process the interrupt
  - **Precise interrupt:** It stops the current program at instruction $I_i$, completing all the instructions up to $I_{i-1}$
  - It saves the PC of instruction $I_i$ in a special register (EPC)
  - It disables interrupts and transfers control to a designated interrupt handler running in the kernel mode
Interrupt Handler

• Needs to read a status register that indicates the cause of the interrupt

• Uses a special indirect jump instruction RFE (return-from-exception) that
  – enables interrupts
  – restores the processor to the user mode
  – restores hardware status and control state

• Nested interrupts
  – Saves EPC before enabling interrupts
  – need an instruction to move EPC into GPRs
  – need a way to mask further interrupts at least until EPC can be saved
Synchronous Interrupts

• A synchronous interrupt (exception) is caused by a *particular instruction*

• In general, the instruction cannot be completed and needs to be *restarted* after the exception has been handled
  – With pipelining, requires undoing the effect of one or more partially executed instructions

• In case of a trap (system call), the instruction is considered to have been completed
  – A special jump instruction involving a change to privileged kernel mode
Topics

• Interrupts

• Speeding up the common case:
  – TLB & Cache organization

• Speeding up page table walks

• Modern Usage
Address Translation in CPU

- TLB miss: a *hardware* or *software* mechanism
- Page fault or protection violation: software handler
- The common case: TLB lookup before every cache access

- Need mechanisms to cope with the additional latency of TLB:
  - slow down the clock
  - pipeline the TLB and cache access
  - virtual-address caches
  - parallel TLB/cache access
Virtual-Address Caches

Pros and cons:

• one-step process in case of a hit (+)
• cache needs to be flushed on a context switch unless address space identifiers (ASIDs) included in tags (-)
• aliasing problems due to the sharing of pages (-)

Alternative: place the cache before the TLB
Aliasing in Virtual-Address Caches

Two virtual pages share one physical page

Tag | Data
---|---
VA\(_1\) | 1st Copy of Data at PA
VA\(_2\) | 2nd Copy of Data at PA

Virtual cache can have two copies of same physical data. Writes to one copy not visible to reads of other!

General Solution: *Disallow aliases to coexist in cache*

Software (i.e., OS) solution for direct-mapped cache

VAs of shared pages must agree in cache index bits; this ensures all VAs accessing same PA will conflict in direct-mapped cache (early SPARCs)
Index L is available without consulting the TLB
⇒ cache and TLB accesses can begin simultaneously
Tag comparison is made after both accesses are completed

When does this work? L + b < k ✓  L + b = k ✓  L + b > k ✗
Concurrent Access to TLB & Large L1
The problem with L1 > Page size

Can VA₁ and VA₂ both map to PA? Yes
Virtual-Index Physical-Tag Caches: Associative Organization

After the PPN is known, $2^a$ physical tags are compared

*Is this scheme realistic?*
A solution via Second-Level Cache

- CPU
- L05
- RF
- L1 Instruction Cache
- L1 Data Cache
- Unified L2 Cache
- Memory
- Memory
- Memory

Usually a common L2 cache backs up both Instruction and Data L1 caches

L2 is “inclusive” of both Instruction and Data caches
Anti-Aliasing Using L2: *MIPS R10000*

- Suppose VA1 and VA2 both map to PA and VA1 is already in L1, L2 (VA1 ≠ VA2)
- After VA2 is resolved to PA, collision is detected in L2. Collision → **Field a is different.**
- VA1 will be purged from L1, and VA2 will be loaded ⇒ *no aliasing!*

February 20, 2020
Physically addressed L2 can also be used to avoid aliases in virtually addressed L1.
Topics

• Interrupts

• Speeding up the common case:
  – TLB & Cache organization

• Speeding up page table walks

• Modern Usage
Page Fault Handler

• When the referenced page is not in DRAM:
  – The missing page is located (or created)
  – It is brought in from disk, and page table is updated
    Another job may be run on the CPU while the first job waits for the requested page to be read from disk
  – If no free pages are left, a page is swapped out
    Pseudo-LRU replacement policy

• Since it takes a long time to transfer a page (msecs), page faults are handled completely in software by the OS
  – Untranslated addressing mode is essential to allow kernel to access page tables
Translation for Page Tables

- Can references to page tables cause TLB misses?
- Can this go on forever?

A program that traverses the page table needs a “no translation” addressing mode.
A PTE in primary memory contains primary or secondary memory addresses

A PTE in secondary memory contains only secondary memory addresses

⇒ a page of a PT can be swapped out only if none of its PTE’s point to pages in the primary memory

Why? Pointed-to pages become inaccessible (page fault due to swapped-out PT page)

May cause deadlock!
Atlas Revisited

• One PAR for each physical page

• PAR’s contain the VPN’s of the pages resident in primary memory

• Advantage:
  - The size is proportional to the size of the primary memory

• Disadvantage?

Must check all PARs!
Hashed Page Table: Approximating Associative Addressing

- Hashed Page Table is typically 2 to 3 times larger than the number of PPNs to reduce collision probability.
- It can also contain DPNs for some non-resident pages (*not common*).
- If a translation cannot be resolved in this table then the software consults a data structure that has an entry for every existing page.
Virtual Memory Use Today - 1

• Desktop/server/cellphone processors have full demand-paged virtual memory
  – Portability between machines with different memory sizes
  – Protection between multiple users or multiple tasks
  – Share small physical memory among active tasks
  – Simplifies implementation of some OS features

• Vector supercomputers and GPUs have translation and protection but not demand paging
  (Older Crays: base&bound, Japanese & Cray X1: pages)
  – Don’t waste expensive processor time thrashing to disk (make jobs fit in memory)
  – Mostly run in batch mode (run set of jobs that fits in memory)
  – Difficult to implement restartable vector instructions
Most embedded processors and DSPs provide physical addressing only:
- Can’t afford area/speed/power budget for virtual memory support
- Often there is no secondary storage to swap to!
- Programs custom-written for particular memory configuration in product
- Difficult to implement restartable instructions for exposed architectures
Next lecture: Pipelining!
Global System Address Space

- Level A maps users’ address spaces into the global space providing privacy, protection, sharing etc.
- Level B provides demand paging for the large global system address space
- Level A and Level B translations may be kept in separate TLB’s
Hashed Page Table Walk:
PowerPC Two-level, Segmented Addressing

Hashed Segment Table

Hashed Page Table

PA of Seg Table

per process

PA of Page Table

system-wide

[ IBM numbers bits with MSB=0 ]

64-bit user VA

80-bit System VA

40-bit PA

Global Seg ID

Hashed Page Table

PPN

Offset

Page

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Segment

Offset

Seg ID Page Offset

0 35 51 63

Hashed Segment Table

PA of Seg Table

per process

PA of Page Table

system-wide

40-bit PA

PPN

Offset

[ IBM numbers bits with MSB=0 ]
Power PC: Hashed Page Table

- Each hash table slot has 8 PTEs <VPN,PPN> that are searched sequentially.
- If the first hash slot fails, an alternate hash function is used to look in another slot. 
  
  *All these steps are done in hardware!*

- Hashed Table is typically 2 to 3 times larger than the number of physical pages.
- The full backup Page Table is a software data structure.