Virtual Memory

CS740
October 13, 1998

Topics
• page tables
• TLBs
• Alpha 21X64 memory system
Levels in a Typical Memory Hierarchy

- CPU
  - disk memory reference: size: 20 GB, speed: 10 ms, $/Mbyte: $0.10/MB

- Larger, slower, cheaper
Virtual Memory

Main memory acts as a cache for the secondary storage (disk)

Increases Program-Accessible Memory
- address space of each job larger than physical memory
- sum of the memory of many jobs greater than physical memory
Address Spaces

- Virtual and physical address spaces divided into equal-sized blocks
  - “Pages” (both virtual and physical)
- Virtual address space typically larger than physical
- Each process has separate virtual address space

Virtual addresses (VA)  Physical addresses (PA)

Process 1:

Process 2:

(address translation)

(Read-only library code)
Other Motivations

Simplifies memory management

- main reason today
- Can have multiple processes resident in physical memory
- Their program addresses mapped dynamically
  - Address 0x100 for process P1 doesn’t collide with address 0x100 for process P2
- Allocate more memory to process as its needs grow

Provides Protection

- One process can’t interfere with another
  - Since operate in different address spaces
- Process cannot access privileged information
  - Different sections of address space have different access permissions
Contrast: Macintosh Memory Model

Does not Use Traditional Virtual Memory

All objects accessed through “Handles”

- Indirect reference through table
- Objects can be relocated by updating pointer in table
VM as part of the memory hierarchy

Access word w in virtual page p (hit)

\[ w \text{ cache block} \]

Disk

\[ p \]

\[ q \]

Memory

\[ p \]

\[ (\text{page frames}) \]

Access word v in virtual page q (miss or "page fault")

\[ v \text{ cache block} \]

\[ p \]

\[ q \]

\[ (\text{pages}) \]
VM address translation

\[ V = \{0, 1, \ldots, n - 1\} \quad \text{virtual address space} \]
\[ M = \{0, 1, \ldots, m - 1\} \quad \text{physical address space} \]

\[ n > m \]

\[ \text{MAP: } V \rightarrow M \cup \{\emptyset\} \quad \text{address mapping function} \]

\[ \text{MAP}(a) = a' \quad \text{if data at virtual address } a \text{ is present at physical address } a' \text{ and } a' \text{ in } M \]

\[ = \emptyset \quad \text{if data at virtual address } a \text{ is not present in } M \]

Diagram:

- Processor
- Name Space V
- Processor
- Addr Trans Mechanism
- Main Memory
- Secondary memory
- OS performs this transfer

Fault handler

Missing item fault

Physical address

Fault handler

Main Memory

Secondary memory

OS performs this transfer
VM address translation

Notice that the page offset bits don't change as a result of translation
Address translation with a page table

- **Virtual address**: 31 12 11 0
- **Virtual page number**: 12 11
- **Page offset**: 0

- **Page table base register**: 0111229
- **VPN acts as table index**: 31
- **Valid**: 31 12 11
- **Access**: 0

- **Physical page number**: 12 11
- **Page offset**: 0

- If valid=0, then page is not in memory and page fault exception.

- **Physical address**: 0111229
Page Tables

Virtual page number

Page table
Valid
Physical page or
disk address

Physical memory

Disk storage

Physical memory

Disk storage

Virtual page number

Page table
Valid
Physical page or
disk address

Physical memory

Disk storage

Virtual page number

Page table
Valid
Physical page or
disk address

Physical memory

Disk storage

Virtual page number

Page table
Valid
Physical page or
disk address

Physical memory

Disk storage
Page Table Operation

Translation
• separate (set of) page table(s) per process
• VPN forms index into page table

Computing Physical Address
• Page Table Entry (PTE) provides information about page
  – Valid bit = 1 ==> page in memory.
    » Use physical page number (PPN) to construct address
  – Valid bit = 0 ==> page in secondary memory
    » Page fault
    » Must load into main memory before continuing

Checking Protection
• Access rights field indicate allowable access
  – E.g., read-only, read-write, execute-only
  – Typically support multiple protection modes (e.g., kernel vs. user)
• Protection violation fault if don’t have necessary permission
VM design issues

Everything driven by enormous cost of misses:

- hundreds of thousands to millions of clocks.
  - vs units or tens of clocks for cache misses.
- disks are high latency
  - Typically 10 ms access time
- Moderate disk to memory bandwidth
  - 10 MBytes/sec transfer rate

Large block sizes:

- Typically 4KB–16 KB
- amortize high access time
- reduce miss rate by exploiting spatial locality

Perform Context Switch While Waiting

- Memory filled from disk by direct memory access
- Meanwhile, processor can be executing other processes
VM design issues (cont)

Fully associative page placement:
- eliminates conflict misses
- every miss is a killer, so worth the lower hit time

Use smart replacement algorithms
- handle misses in software
- miss penalty is so high anyway, no reason to handle in hardware
- small improvements pay big dividends

Write back only:
- disk access too slow to afford write through + write buffer
Integrating VM and cache

Most Caches “Physically Addressed”
- Accessed by physical addresses
- Allows multiple processes to have blocks in cache at same time
- Allows multiple processes to share pages
- Cache doesn’t need to be concerned with protection issues
  - Access rights checked as part of address translation

Perform Address Translation Before Cache Lookup
- But this could involve a memory access itself
- Of course, page table entries can also become cached
Speeding up Translation with a TLB

Translation lookaside buffer (TLB)

- small, usually fully associative cache
- maps virtual page numbers to physical page numbers
- Contains complete page table entries for small number of pages
Address translation with a TLB

Virtual address

process ID | virtual page number | page offset

valid | dirty | tag | physical page number

physical address

tag

index

byte offset

data

cache hit

TLB hit

virtual address

valid tag data
Alpha AXP 21064 TLB

- Page size: 8KB
- Hit time: 1 clock
- Miss penalty: 20 clocks
- TLB size: ITLB 8 PTEs, DTLB 32 PTEs
- Replacement: random (but not last used)
- Placement: Fully assoc
TLB-Process Interactions

TLB Translates Virtual Addresses
  • But virtual address space changes each time have context switch

Could flush TLB
  • Every time perform context switch
  • Refill for new process by series of TLB misses
  • ~100 clock cycles each

Could Include Process ID Tag with TLB Entry
  • Identifies which address space being accessed
  • OK even when sharing physical pages
Virtually-Indexed Cache

Cache Index Determined from Virtual Address
- Can begin cache and TLB index at same time

Cache Physically Addressed
- Cache tag indicates physical address
- Compare with TLB result to see if match
  - Only then is it considered a hit
Generating Index from Virtual Address

Size cache so that index is determined by page offset

- Can increase associativity to allow larger cache
- E.g., early PowerPC’s had 32KB cache
  - 8-way associative, 4KB page size

Page Coloring

- Make sure lower k bits of VPN match those of PPN
- Page replacement becomes set associative
- Number of sets = \(2^k\)
Example: Alpha Addressing

Page Size
- Currently 8KB

Page Tables
- Each table fits in single page
- Page Table Entry 8 bytes
  - 32 bit physical page number
  - Other bits for valid bit, access information, etc.
- 8K page can have 1024 PTEs

Alpha Virtual Address
- Based on 3-level paging structure

<table>
<thead>
<tr>
<th>level 1</th>
<th>level 2</th>
<th>level 3</th>
<th>page offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
<td>13</td>
</tr>
</tbody>
</table>

- Each level indexes into page table
- Allows 43-bit virtual address when have 8KB page size
Alpha Page Table Structure

Tree Structure
- Node degree \( \leq 1024 \)
- Depth 3

Nice Features
- No need to enforce contiguous page layout
- Dynamically grow tree as memory needs increase
Mapping an Alpha 21064 virtual address

PT size: 1K PTEs (8 KBytes)

PTE size: 8 Bytes

Virtual address

10 bits

13 bits

seg0/seg1 Selector 000 … 0 or 111 … 1 Level1 Level2 Level3 Page offset

Page table base register

L1 page table

L2 page table

L3 page table

Page table entry

Page table entry

Page table entry

Physical page-frame number Page offset

Main memory

21 bits

13 bits

Physical address
Alpha Virtual Addresses

Binary Address  Segment  Purpose
1...1 11  xxxx...xxx  seg1  Kernel accessible virtual addresses
   – E.g., page tables for this process
1...1 10  xxxx...xxx  kseg  Kernel accessible physical addresses
   – No address translation performed
   – Used by OS to indicate physical addresses
0...0 0x  xxxx...xxx  seg0  User accessible virtual addresses
   – Only part accessible by user program

Address Patterns

• Must have high order bits all 0’s or all 1’s
   – Currently 64–43 = 21 wasted bits in each virtual address
• Prevents programmers from sticking in extra information
   – Could lead to problems when want to expand virtual address space in future
Alpha Seg0 Memory Layout

Regions

- **Data**
  - Static space for global variables
  - Allocation determined at compile time
  - Access via $gp$
  - Dynamic space for runtime allocation
  - E.g., using `malloc`

- **Text**
  - Stores machine code for program

- **Stack**
  - Implements runtime stack
  - Access via $sp$

- **Reserved**
  - Used by operating system
  - shared libraries, process info, etc.
Alpha Seg0 Memory Allocation

Address Range

- User code can access memory locations in range
  \[0x00000000000010000\text{ to }0x000003FF80000000\]
- Nearly \(2^{42} \approx 4.3980465 \times 10^{12}\) byte range
- In practice, programs access far fewer

Dynamic Memory Allocation

- Virtual memory system only allocates blocks of memory ("pages") as needed
- As stack reaches lower addresses, add to lower allocation
- As break moves toward higher addresses, add to upper allocation
  - Due to calls to \texttt{malloc}, \texttt{calloc}, etc.
Page Table Configurations

Minimal: 8MB

Maximal: 4TB (All of Seg0)
Where Are the Page Tables?

All in Physical Memory?
- Uses up large fraction of physical address space
  - ~8GB for maximal configuration
- Hard to move around
  - E.g., whenever context switch

Some in Virtual Memory?
- E.g., level 3 page tables put in seg1
- Level 2 PTE give VPN for level 3 page
- Make sure seg1 page tables in physical memory
  - Full configuration would require 4GB of page tables
  - 1026 must be in physical memory
    » 1 Level 1
    » 512 (map seg0) + 1 (maps seg1) Level 2’s
    » 512 (maps seg1) Level 3’s
- May have two page faults to get single word into memory
Expanding Alpha Address Space

Increase Page Size

- Increasing page size 2X increases virtual address space 16X
  - 1 bit page offset, 1 bit for each level index

<table>
<thead>
<tr>
<th>level 1</th>
<th>level 2</th>
<th>level 3</th>
<th>page offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>10+k</td>
<td>10+k</td>
<td>10+k</td>
<td>13+k</td>
</tr>
</tbody>
</table>

Physical Memory Limits

- Cannot be larger than kseg
  VA bits \(-2 \geq PA\) bits
- Cannot be larger than 32 + page offset bits
  - Since PTE only has 32 bits for PPN

Configurations

- Page Size: 8K, 16K, 32K, 64K
- VA Size: 43, 47, 51, 55
- PA Size: 41, 45, 47, 48
Alpha AXP 21064 memory hierarchy

8 entries

256 32-byte blocks
8 KBytes
direct mapped

64K 32-byte blocks
2 MBytes
direct mapped
write back
write allocate

cache block size: 32 bytes
page size: 8 KBytes
virtual address size: 43 bits
physical address size: 34 bits
• Microprocessor Report, Sept. ‘94
• L1 caches small enough to allow virtual indexing
• L2 cache access not required until after TLB completes
Alpha 21164 Hierarchy

- Improving memory performance was main design goal
- Earlier Alpha’s CPUs starved for data
## Other System Examples

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Intel Pentium Pro</th>
<th>PowerPC 604</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual address</td>
<td>32 bits</td>
<td>52 bits</td>
</tr>
<tr>
<td>Physical address</td>
<td>32 bits</td>
<td>32 bits</td>
</tr>
<tr>
<td>Page size</td>
<td>4 KB, 4 MB</td>
<td>4 KB, selectable, and 256 MB</td>
</tr>
<tr>
<td>TLB organization</td>
<td>A TLB for instructions and a TLB for data</td>
<td>A TLB for instructions and a TLB for data</td>
</tr>
<tr>
<td></td>
<td>Both four-way set associative</td>
<td>Both two-way set associative</td>
</tr>
<tr>
<td></td>
<td>Pseudo-LRU replacement</td>
<td>LRU replacement</td>
</tr>
<tr>
<td>Instruction TLB: 32 entries</td>
<td></td>
<td>Instruction TLB: 128 entries</td>
</tr>
<tr>
<td>Data TLB: 64 entries</td>
<td></td>
<td>Data TLB: 128 entries</td>
</tr>
<tr>
<td>TLB misses handled in hardware</td>
<td></td>
<td>TLB misses handled in hardware</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Intel Pentium Pro</th>
<th>PowerPC 604</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache organization</td>
<td>Split instruction and data caches</td>
<td>Split instruction and data caches</td>
</tr>
<tr>
<td>Cache size</td>
<td>8 KB each for instructions/data</td>
<td>16 KB each for instructions/data</td>
</tr>
<tr>
<td>Cache associativity</td>
<td>Four-way set associative</td>
<td>Four-way set associative</td>
</tr>
<tr>
<td>Replacement</td>
<td>Approximated LRU replacement</td>
<td>LRU replacement</td>
</tr>
<tr>
<td>Block size</td>
<td>32 bytes</td>
<td>32 bytes</td>
</tr>
<tr>
<td>Write policy</td>
<td>Write-back</td>
<td>Write-back or write-through</td>
</tr>
</tbody>
</table>