Paging: Smaller Tables

CMPU 334 – Operating Systems
Jason Waterman
Simple Memory System

- 14-bit virtual addresses, 12-bit physical address, 64 byte page size
- TLB: 16 entries, 4-way associative

Page Table: first 16 out of 256 entries shown

<table>
<thead>
<tr>
<th>VPN</th>
<th>PFN</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>28</td>
<td>1</td>
</tr>
<tr>
<td>01</td>
<td>–</td>
<td>0</td>
</tr>
<tr>
<td>02</td>
<td>33</td>
<td>1</td>
</tr>
<tr>
<td>03</td>
<td>02</td>
<td>1</td>
</tr>
<tr>
<td>04</td>
<td>–</td>
<td>0</td>
</tr>
<tr>
<td>05</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>06</td>
<td>–</td>
<td>0</td>
</tr>
<tr>
<td>07</td>
<td>–</td>
<td>0</td>
</tr>
<tr>
<td>08</td>
<td>13</td>
<td>1</td>
</tr>
<tr>
<td>09</td>
<td>17</td>
<td>1</td>
</tr>
<tr>
<td>0A</td>
<td>09</td>
<td>1</td>
</tr>
<tr>
<td>0B</td>
<td>–</td>
<td>0</td>
</tr>
<tr>
<td>0C</td>
<td>–</td>
<td>0</td>
</tr>
<tr>
<td>0D</td>
<td>2D</td>
<td>1</td>
</tr>
<tr>
<td>0E</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>0F</td>
<td>0D</td>
<td>1</td>
</tr>
</tbody>
</table>
Simple Memory System

- 14-bit virtual addresses, 12-bit physical address, 64 byte page size
- TLB: 16 entries, 4-way associative

Page Table: first 16 out of 256 entries shown

<table>
<thead>
<tr>
<th>VPN</th>
<th>PFN</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>28</td>
<td>1</td>
</tr>
<tr>
<td>01</td>
<td>–</td>
<td>0</td>
</tr>
<tr>
<td>02</td>
<td>33</td>
<td>1</td>
</tr>
<tr>
<td>03</td>
<td>02</td>
<td>1</td>
</tr>
<tr>
<td>04</td>
<td>–</td>
<td>0</td>
</tr>
<tr>
<td>05</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>06</td>
<td>–</td>
<td>0</td>
</tr>
<tr>
<td>07</td>
<td>–</td>
<td>0</td>
</tr>
<tr>
<td>08</td>
<td>13</td>
<td>1</td>
</tr>
<tr>
<td>09</td>
<td>17</td>
<td>1</td>
</tr>
<tr>
<td>0A</td>
<td>09</td>
<td>1</td>
</tr>
<tr>
<td>0B</td>
<td>–</td>
<td>0</td>
</tr>
<tr>
<td>0C</td>
<td>–</td>
<td>0</td>
</tr>
<tr>
<td>0D</td>
<td>2D</td>
<td>1</td>
</tr>
<tr>
<td>0E</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>0F</td>
<td>0D</td>
<td>1</td>
</tr>
</tbody>
</table>
Address Translation Examples

Virtual Address: 0x03D4

Virtual Address: 0x0020

10/4/2022
CMPU 334 -- Operating Systems
The story so far... Linear Page Tables

- We have one page table for every process in the system
  - Assume that 32-bit address space with 4KB pages and 4-byte page-table entry

- Page Table of Process A

- Physical Memory

Page table size = \( \frac{2^{32}}{2^{12}} \times 4\text{Byte} = 4\text{MByte} \)
Where Are Page Tables Stored?

• Page tables for each process are stored in memory!

• Page tables can get awfully large
  • 32-bit address space with 4-KB pages, 20 bits for VPN
    • $4MB = 2^{20} \text{ entries} \times 4 \text{ Bytes per page table entry}$
    • Size of a 64-bit address space page table?
      • 16,000 Terabytes

• We need a more efficient way of storing page tables!
Idea: Larger Page Size

• Assume that 32-bit address space with 16KB pages and 4-byte page-table entry

\[ \frac{2^{32}}{2^{16}} \times 4 = 1MB \] per page table

Big pages lead to internal fragmentation
Tiny Page Table Example

- Most of the Page Table is unused
- Filled with invalid entries

A Page Table For 16KB Address Space
Hybrid Approach: Paging and Segments

• Use base and bounds registers to reduce the memory overhead of page tables
  • Base holds the physical address of the page table of that segment
  • Bounds register indicates the end of the page table (i.e., how many pages in segment)

• Each process has a page table associated with each segment
  • When process is running, the base register for each segment contains the physical address of a linear page table for that segment

• We still can have page table waste with a large but sparse heap

• Arbitrary sized page tables can lead to external fragmentation
Multi-Level Page Table

• Chop up the page table into page-sized units
• If an entire page of page-table entries is invalid, don’t allocate that page of the page table
• To track whether a page of the page table is valid, use a new structure, called page directory

Linear (Left) And Multi-Level (Right) Page Tables
A Detailed Multi-Level Example

A 16-KB Address Space With 64-byte Pages

<table>
<thead>
<tr>
<th>Address space</th>
<th>16 KB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page size</td>
<td>64 byte</td>
</tr>
<tr>
<td>Virtual address</td>
<td>14 bit</td>
</tr>
<tr>
<td>Offset</td>
<td>6 bit</td>
</tr>
<tr>
<td>VPN</td>
<td>8 bit</td>
</tr>
<tr>
<td>Number of Page table entries</td>
<td>$2^8(256)$</td>
</tr>
</tbody>
</table>

- Pages 0 and 1 for code
- Pages 4 and 5 for heap
- Pages 254 and 255 for the stack

- Rest of the address space is unallocated
A Detailed Multi-Level Example: Page Directory Index

- Divide the page table entries into pages
  - Assume a 4 byte page table entry
  - Page size is 64 bytes
  - Each page can hold 16 page table entries
  - Need 16 pages to hold all the entries (256/16)

- Page Directory
  - One Page Directory Entry (PDE) for each page of the page table
  - Page Directory Index is 4-bits
  - If a PDE is valid, there is a Page Table in memory
A Detailed Multi-Level Example: Page Table Index

- The PDE is valid we use the Page Table Index to get the PTE
  - The Page Table Index is the offset into the Page Table Pointed to by the PDE
- If the PDE is not valid, we have an illegal memory access

<table>
<thead>
<tr>
<th>Page Directory</th>
<th>Page of PT (@PFN:100)</th>
<th>Page of PT (@PFN:101)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PFN</td>
<td>valid</td>
<td>PFN</td>
</tr>
<tr>
<td>100</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>—</td>
<td>0</td>
<td>23</td>
</tr>
<tr>
<td>—</td>
<td>0</td>
<td>—</td>
</tr>
<tr>
<td>—</td>
<td>0</td>
<td>—</td>
</tr>
<tr>
<td>—</td>
<td>0</td>
<td>80</td>
</tr>
<tr>
<td>—</td>
<td>0</td>
<td>50</td>
</tr>
<tr>
<td>—</td>
<td>0</td>
<td>—</td>
</tr>
<tr>
<td>—</td>
<td>0</td>
<td>—</td>
</tr>
<tr>
<td>—</td>
<td>0</td>
<td>—</td>
</tr>
<tr>
<td>—</td>
<td>0</td>
<td>—</td>
</tr>
<tr>
<td>—</td>
<td>0</td>
<td>—</td>
</tr>
<tr>
<td>—</td>
<td>0</td>
<td>—</td>
</tr>
<tr>
<td>—</td>
<td>0</td>
<td>—</td>
</tr>
<tr>
<td>—</td>
<td>0</td>
<td>—</td>
</tr>
<tr>
<td>—</td>
<td>0</td>
<td>—</td>
</tr>
<tr>
<td>—</td>
<td>0</td>
<td>—</td>
</tr>
<tr>
<td>101</td>
<td>1</td>
<td>—</td>
</tr>
<tr>
<td>—</td>
<td>0</td>
<td>—</td>
</tr>
</tbody>
</table>
A more realistic example

Virtual address: 30 bit
Page size: 512 byte
Offset: 9 bit
VPN: 21 bit
More than Two Level

• In some cases, more than two levels are needed

<table>
<thead>
<tr>
<th>Virtual address</th>
<th>30 bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page size</td>
<td>512 byte</td>
</tr>
<tr>
<td>Offset</td>
<td>9 bit</td>
</tr>
<tr>
<td>VPN</td>
<td>21 bit</td>
</tr>
<tr>
<td>Page entries per page</td>
<td>128 PTEs</td>
</tr>
<tr>
<td>Page Directory Entries</td>
<td>$2^{14}$ (16K)</td>
</tr>
</tbody>
</table>
More than Two Level: Page Directory

- If our page directory has $2^{14}$ entries, it spans not one page but 128.
- To remedy this problem, we build a further level of the tree, by splitting the page directory itself into multiple pages of the page directory.
Intel Core i7 Memory System

Processor package

Core x4

- Registers
  - L1 d-cache: 32 KB, 8-way
  - L1 i-cache: 32 KB, 8-way

- Instruction fetch
  - L2 unified cache: 256 KB, 8-way

- MMU (addr translation)
  - L1 d-TLB: 64 entries, 4-way
  - L1 i-TLB: 128 entries, 4-way

- L2 unified TLB: 512 entries, 4-way

- L3 unified cache: 8 MB, 16-way (shared by all cores)

- DDR3 Memory controller
  - 3 x 64 bit @ 10.66 GB/s
  - 32 GB/s total (shared by all cores)

- Main memory

QuickPath interconnect: 4 links @ 25.6 GB/s each

To other cores
To I/O bridge
Core i7 Page Table Translation

Virtual address

CR3
Physical address of L1 PT

L1 PT
Page global directory

VPN 1
VPN 2
VPN 3
VPN 4
VPO

9
9
9
9
12

L1 PTE

L2 PT
Page upper directory

VPN 2
VPN 3

L2 PTE

L3 PT
Page middle directory

VPN 3
VPN 4

L3 PTE

L4 PT
Page table

VPN 4
VPO

L4 PTE

Physical address

Offset into physical and virtual page

Physical address of page

40
40
40
40

512 GB region per entry
1 GB region per entry
2 MB region per entry
4 KB region per entry

Physical address

VPN 1
VPN 2
VPN 3
VPN 4
VPO

PPN

PPO

40
12

10/4/2022

CMPU 334 -- Operating Systems
End-to-end Core i7 Address Translation

Virtual address (VA)

CPU

VPN

VPO

36

12

TLBT

TLBI

TLB miss

L1 TLB (16 sets, 4 entries/set)

VPN1

VPN2

VPN3

VPN4

9

9

9

9

CR3

PTE

PTE

PTE

PTE

Page tables

TLB hit

L1 hit

L2, L3, and main memory

L2, L3, and main memory

L1 miss

Physical address (PA)

Result

32/64

L1 d-cache (64 sets, 8 lines/set)
Multi-level Page Tables: Advantage & Disadvantage

• Advantages
  • Only allocates page-table space in proportion to the amount of address space you are using
  • The OS can grab the next free page when it needs to allocate or grow a page table

• Disadvantages
  • Multi-level table is a small example of a time-space trade-off
  • Complexity