CSMC 412
Operating Systems
Prof. Ashok K Agrawala

Memory Management - III
Online Set3
Memory Management Schemes from II

- Segmentation
- Paging
- Address Translation Mechanisms

Desirable Features:
- Very large address space
- Ability to execute partially loaded programs
- Dynamic Relocatability
- Sharing
- Protection
Shared Pages

- **Shared code**
  - One copy of read-only (reentrant) code shared among processes (i.e., text editors, compilers, window systems)
  - Similar to multiple threads sharing the same process space
  - Also useful for interprocess communication if sharing of read-write pages is allowed

- **Private code and data**
  - Each process keeps a separate copy of the code and data
  - The pages for the private code and data can appear anywhere in the logical address space
Shared Code
Shared Pages Example

process $P_1$

| ed 1 | 3 |
| ed 2 | 4 |
| ed 3 | 6 |
| data 1 | 7 |

page table for $P_1$

process $P_2$

| ed 1 | 3 |
| ed 2 | 4 |
| ed 3 | 6 |
| data 2 | 7 |

page table for $P_2$

process $P_3$

| ed 1 | 3 |
| ed 2 | 4 |
| ed 3 | 6 |
| data 3 | 7 |

page table for $P_3$
Structure of the Page Table

• Memory structures for paging can get huge using straight-forward methods
  • Consider a 32-bit logical address space as on modern computers
  • Page size of 4 KB ($2^{12}$)
  • Page table would have 1 million entries ($2^{32} / 2^{12}$)
  • If each entry is 4 bytes -> 4 MB of physical address space / memory for page table alone
    • That amount of memory used to cost a lot
    • Don’t want to allocate that contiguously in main memory

• Hierarchical Paging
• Hashed Page Tables
• Inverted Page Tables
Hierarchical Page Tables

• Break up the logical address space into multiple page tables
• A simple technique is a two-level page table
• We then page the page table
Two-Level Page-Table Scheme
Two-Level Paging Example

- A logical address (on 32-bit machine with 1K page size) is divided into:
  - a page number consisting of 22 bits
  - a page offset consisting of 10 bits

- Since the page table is paged, the page number is further divided into:
  - a 12-bit page number
  - a 10-bit page offset

- Thus, a logical address is:

<table>
<thead>
<tr>
<th>$p_1$</th>
<th>$p_2$</th>
<th>$d$</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

- where $p_1$ is an index into the outer page table, and $p_2$ is the displacement within the page of the inner page table
- Known as forward-mapped page table
Address-Translation Scheme
64-bit Logical Address Space

- Even two-level paging scheme not sufficient
- If page size is 4 KB ($2^{12}$)
  - Then page table has $2^{52}$ entries
  - If two level scheme, inner page tables could be $2^{10}$ 4-byte entries
  - Address would look like

<table>
<thead>
<tr>
<th>outer page</th>
<th>inner page</th>
<th>page offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$</td>
<td>$p_2$</td>
<td>$d$</td>
</tr>
<tr>
<td>42</td>
<td>10</td>
<td>12</td>
</tr>
</tbody>
</table>

- Outer page table has $2^{42}$ entries or $2^{44}$ bytes
- One solution is to add a $2^{nd}$ outer page table
- But in the following example the $2^{nd}$ outer page table is still $2^{34}$ bytes in size
  - And possibly 4 memory access to get to one physical memory location
# Three-level Paging Scheme

<table>
<thead>
<tr>
<th>outer page</th>
<th>inner page</th>
<th>offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$</td>
<td>$p_2$</td>
<td>$d$</td>
</tr>
<tr>
<td>42</td>
<td>10</td>
<td>12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2nd outer page</th>
<th>outer page</th>
<th>inner page</th>
<th>offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$</td>
<td>$p_2$</td>
<td>$p_3$</td>
<td>$d$</td>
</tr>
<tr>
<td>32</td>
<td>10</td>
<td>10</td>
<td>12</td>
</tr>
</tbody>
</table>
Hashed Page Tables

• Common in address spaces > 32 bits
• The virtual page number is hashed into a page table
  • This page table contains a chain of elements hashing to the same location
• Each element contains (1) the virtual page number (2) the value of the mapped page frame (3) a pointer to the next element
• Virtual page numbers are compared in this chain searching for a match
  • If a match is found, the corresponding physical frame is extracted
• Variation for 64-bit addresses is clustered page tables
  • Similar to hashed but each entry refers to several pages (such as 16) rather than 1
  • Especially useful for sparse address spaces (where memory references are non-contiguous and scattered)
Hashed Page Table
Inverted Page Table

• Rather than each process having a page table and keeping track of all possible logical pages, track all physical pages
• One entry for each real page of memory
• Entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page
• Decreases memory needed to store each page table, but increases time needed to search the table when a page reference occurs
• Use hash table to limit the search to one — or at most a few — page-table entries
  • TLB can accelerate access
• But how to implement shared memory?
  • One mapping of a virtual address to the shared physical address
Inverted Page Table Architecture
Example: The Intel 32 and 64-bit Architectures

• Dominant industry chips

• Pentium CPUs are 32-bit and called IA-32 architecture

• Current Intel CPUs are 64-bit and called IA-64 architecture

• Many variations in the chips, cover the main ideas here
Example: The Intel IA-32 Architecture

- Supports both segmentation and segmentation with paging
  - Each segment can be 4 GB
  - Up to 16 K segments per process
  - Divided into two partitions
    - First partition of up to 8 K segments are private to process (kept in local descriptor table (LDT))
    - Second partition of up to 8K segments shared among all processes (kept in global descriptor table (GDT))
Example: The Intel IA-32 Architecture (Cont.)

- CPU generates logical address
  - Selector given to segmentation unit
    - Which produces linear addresses

- Linear address given to paging unit
  - Which generates physical address in main memory
  - Paging units form equivalent of MMU
  - Pages sizes can be 4 KB or 4 MB
Logical to Physical Address Translation in IA-32

CPU → segmentation unit → paging unit → physical memory

<table>
<thead>
<tr>
<th>page number</th>
<th>page offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$</td>
<td>$p_2$</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>$d$</td>
</tr>
<tr>
<td></td>
<td>12</td>
</tr>
</tbody>
</table>
Intel IA-32 Segmentation

logical address

selector

offset

descriptor table

segment descriptor

32-bit linear address
Intel IA-32 Paging Architecture
32-bit address limits led Intel to create page address extension (PAE), allowing 32-bit apps access to more than 4GB of memory space.

- Paging went to a 3-level scheme
- Top two bits refer to a page directory pointer table
- Page-directory and page-table entries moved to 64-bits in size
- Net effect is increasing address space to 36 bits – 64GB of physical memory
Intel x86-64

- Current generation Intel x86 architecture
- 64 bits is ginormous (> 16 exabytes)
- In practice only implement 48 bit addressing
  - Page sizes of 4 KB, 2 MB, 1 GB
  - Four levels of paging hierarchy
- Can also use PAE so virtual addresses are 48 bits and physical addresses are 52 bits

```
<table>
<thead>
<tr>
<th>unused</th>
<th>page map level 4</th>
<th>page directory pointer table</th>
<th>page directory</th>
<th>page table</th>
<th>offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>48 47</td>
<td>39 38</td>
<td>30 29</td>
<td>21 20</td>
<td>12 11</td>
</tr>
</tbody>
</table>
```