Virtual Memory
(Chapter 9)
x86-64 Linux Memory Layout

Stack
- Runtime stack (8MB limit)
- E. g., local variables

Heap
- Dynamically allocated as needed
- When call malloc(), calloc(), new() at run time

Data
- Statically allocated data
- Data declared in code
- E. g., global variables, string constants

Text
- Executable machine instructions
- Read-only
Virtual Memory

- Main memory can act as a cache for the secondary storage (disk)
Paging Model of Logical and Physical Memory
Advantages of Virtual Memory

- illusion of having more physical memory
- program relocation
- sharing
- protection
Address Translation

Virtual address

Virtual page number  |  Page offset

Translation

Physical page number  |  Page offset

Physical address
Paging Hardware

Diagram showing the process of address translation in paging hardware, with a page table connecting logical and physical addresses.
Pages: virtual memory blocks

- **Page faults**: the data is not in memory, retrieve it from disk
  - significant miss penalty, thus pages should be fairly large (e.g., 4KB)
  - reducing page faults is important, so need to keep in memory those pages that are needed
    - possibly prefetch important pages before use
  - can handle the faults in software instead of hardware
  - using write-through is too expensive so we use writeback for dirty (i.e., modified) pages
Page Tables

Virtual page number

Page table
Valid
Physical page or disk address

Physical memory

Disk storage
Example

- Consider a virtual memory system with the following properties:
  - 40-bit virtual byte address
  - 16-KB pages
  - 36-bit physical byte address
  - Pages tables have both VALID and DIRTY bits (as with a hardware cache, the DIRTY bits are used to track modified pages in memory that must be written back to disk)

- What is the total size of the page table for each program on this machine, assuming that the valid and dirty bits take a total of 2 bits?

  **Answer**: $2^{26} (2+22)$ bits
Steps in Handling a Page Fault

1. Load M
2. Trap
3. Page is on backing store
4. Bring in missing page
5. Reset page table
6. Restart instruction

Diagram:
- Load M
- Trap
- Page is on backing store
- Bring in missing page
- Reset page table
- Restart instruction
- Load M
- Trap
- Page is on backing store
Page Replacement

- What if there is no free frame in physical memory for page fetched from disk?
- Must select a victim page/frame and write back to disk (if modified) before replacing with new page
- Which page/frame do we replace?
- Approximation of Least Recently Used (LRU) is common approach
LRU Page Replacement

Example 1

- Let there be 4 frames to use
- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th></th>
<th>1</th>
<th></th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
LRU Page Replacement
Example 2

- Here, there are 3 frames to use

<table>
<thead>
<tr>
<th>reference string</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>page frames</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 7 7 2 2 4 4 4 0 1 1 1 3 0 0 3 3 3 0 0</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 3 3 3 3 3 2 2 2 2 2 2 7</td>
</tr>
</tbody>
</table>
Example: VM vs. Cache/memory

- Consider OS with 24-bit virtual addresses for byte-accessed memory
- 18-bit physical addresses, page size = 2KB
- 1 KB direct-mapped cache, 16B / block
  - Assume PIPT (Physically Indexed and Physically Tagged) cache
- Show VA-to-PA mapping and breakdown of cache lookup

**Answer:** VA = 13-bit page number, 11-bit page offset
PA = 7-bit frame number, 11-bit page offset
PA = 8-bit tag, 6-bit cache index, 4-bit byte offset