Memory Management

Disclaimer: some slides are adopted from book authors’ slides with permission
Recap: Performance of Demand Paging

- Memory access time = 200 nanoseconds
- Average page-fault service time = 8 milliseconds
- EAT = \((1 - p) \times 200 + p \times 8,000,000\)
  \[= 200 + p \times 7,999,800\]
- If one access out of 1,000 causes a page fault, then
  \[EAT = 8.2 \text{ microseconds}. \rightarrow \text{This is a slowdown by a factor of } 40!\]
- If want performance degradation < 10 percent
  - \[220 > 200 + 7,999,800 \times p\]
  - \[20 > 7,999,800 \times p\]
  - \[p < .0000025\]
  - < one page fault in every 400,000 memory accesses
Thrashing

• A process is busy swapping pages in and out
  – Don’t make much progress
  – Happens when a process do not have “enough” pages in memory
  – Very high page fault rate
  – Low CPU utilization (why?)
  – CPU utilization based admission control may bring more programs to increase the utilization → more page faults
Thrashing

![Diagram showing the relationship between CPU utilization and degree of multiprogramming](image)
Concepts to Learn

• Memory-mapped I/O
• Copy-on-Write (COW)
• Memory allocator
Recap: Program Binary Sharing

- Multiple instances of the same program
  - E.g., 10 bash shells
Memory Mapped I/O

• Idea: map a file on disk onto the memory space
Memory Mapped I/O

• Benefits: you don’t need to use read()/write() system calls, just directly access data in the file via memory instructions

• How it works?
  – Just like demand paging of an executable file
  – What about writes?
    • Mark the modified (M) bit in the PTE
    • Write back the modified pages back to the original file
Copy-on-Write (COW)

• Fork() creates a copy of a parent process
  – Copy the entire pages on new page frames?
    • If the parent uses 1GB memory, then a fork() call would take a while
    • Then, suppose you immediately call exec(). Was it of any use to copy the 1GB of parent process’s memory?
Copy-on-Write

• Better way: copy the page table of the parent
  – Page table is much smaller (so copy is faster)
  – Both parent and child point to the exactly same physical page frames
Copy-on-Write

• What happens when the parent/child reads?
• What happens when the parent/child writes?
  – Trouble!!!
Page Table Entry (PTE)

- PTE format (architecture specific)
  - Valid bit (V): whether the page is in memory
  - Modify bit (M): whether the page is modified
  - Reference bit (R): whether the page is accessed
  - Protection bits (P): readable, writable, executable
Copy-on-Write

• All pages are marked as read-only

![Diagram showing Copy-on-Write concept]
Copy-on-Write

• Up on a write, a page fault occurs and the OS copies the page on a new frame and maps to it with R/W protection setting
User-level Memory Allocation

• When a process actually allocate a memory from the kernel?
  – On a page fault
  – Allocate a page (e.g., 4KB)

• What does `malloc()` do?
  – Manage a process’s heap
  – Variable size objects in heap
Kernel-level Memory Allocation

• Page-level allocator
  – Page frame allocation/free (fixed size)
  – Users: page fault handler, kernel-memory allocator

• Kernel-memory allocator (KMA)
  – Typical kernel object size $<\text{page size}$
    • File descriptor, inode, task_struct, ...
  – KMA $\leftarrow$ kernel-level malloc
  – In Linux: buddy allocator, SLAB
Buddy Allocator

• Allocate physically contiguous pages
  – Satisfies requests in units sized as power of 2
  – Request rounded up to next highest power of 2
  – When smaller allocation needed than is available, current chunk split into two buddies of next-lower power of 2
  – Quickly expand/shrink across the lists

- 32KB
- 16KB
- 8KB
- 4KB
**Buddy Allocator**

- **Example**
  - Assume 256KB chunk available, kernel requests 21KB

<table>
<thead>
<tr>
<th></th>
<th>256-Free</th>
<th>128-Free</th>
<th>128-Free</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>64-Free</td>
<td>128-Free</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>32-Free</td>
<td>64-Free</td>
<td>128-Free</td>
</tr>
<tr>
<td>32-A</td>
<td>32-Free</td>
<td>64-Free</td>
<td>128-Free</td>
</tr>
</tbody>
</table>
## Buddy Allocator

### Example

- Free A

<table>
<thead>
<tr>
<th></th>
<th>32 Free</th>
<th>64 Free</th>
<th>128 Free</th>
</tr>
</thead>
<tbody>
<tr>
<td>32 A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32 Free</td>
<td>32 Free</td>
<td>64 Free</td>
<td>128 Free</td>
</tr>
<tr>
<td>64 Free</td>
<td>64 Free</td>
<td>128 Free</td>
<td></td>
</tr>
<tr>
<td>128 Free</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>256 Free</td>
</tr>
</tbody>
</table>

KU THE UNIVERSITY OF KANSAS
Virtual Memory Summary

- MMU and address translation
- Paging
- Demand paging
- Copy-on-write
- Page replacement
Quiz: Address Translation

Virtual address format (24bits)

Page table entry (8bit)

Vaddr: 0x0703FE  Vaddr: 0x072370  Vaddr: 0x082370
Paddr: 0x3FE     Paddr: ??       Paddr: ??

Page-table base address = 0x100

<table>
<thead>
<tr>
<th>Addr</th>
<th>+0</th>
<th>+1</th>
<th>+2</th>
<th>+3</th>
<th>+4</th>
<th>+5</th>
<th>+6</th>
<th>+7</th>
<th>+8</th>
<th>+A</th>
<th>+B</th>
<th>+C</th>
<th>+D</th>
<th>+E</th>
<th>+F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x010</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x020</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x100</td>
<td>00</td>
<td>01</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x200</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Quiz: Address Translation

virtual address format (24bits):

<table>
<thead>
<tr>
<th>Frame #</th>
<th>Unused</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Page table entry (8bit):

<table>
<thead>
<tr>
<th>Vaddr</th>
<th>Paddr</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0703FE</td>
<td>0x3FE</td>
</tr>
<tr>
<td>0x072370</td>
<td>0x470</td>
</tr>
<tr>
<td>0x082370</td>
<td>invalid</td>
</tr>
</tbody>
</table>

page-table base address = 0x100

<table>
<thead>
<tr>
<th>Addr</th>
<th>+0</th>
<th>+1</th>
<th>+2</th>
<th>+3</th>
<th>+4</th>
<th>+5</th>
<th>+6</th>
<th>+7</th>
<th>+8</th>
<th>+A</th>
<th>+B</th>
<th>+C</th>
<th>+D</th>
<th>+E</th>
<th>+F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x010</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x020</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x100</td>
<td>00</td>
<td>01</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>01</td>
<td>00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x200</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>