Virtual Memory and Operating Systems
Topics

- Operating systems
- Address spaces
- Virtual memory
- Address translation
- Memory allocation
Lingering questions

• What happens when you call `malloc()`?
  – How exactly is memory “allocated”?

• What is the correspondence between addresses in machine code and physical memory cells?
  – Are Y86 operand addresses used by the hardware?

• *There’s a gap here ...*
  – In early machines, there was no gap; the machine ran one program at a time and every program had complete control of the machine – there was no need for `malloc()`
  – Modern machines support multi-tasking, so this is not sufficient
  – What we need is some systems *software* to mediate between user programs and the hardware
An operating system (OS) is systems software that provides essential / fundamental system services:

- Manages initialization (booting) and cleanup (shutdown)
- Manages hardware/software interactions (I/O)
- Manages running programs (scheduling)
- Manages memory (virtual memory)
- Manages data (file systems)
- Manages external devices (drivers & interrupts)
- Manages communication (networking)
- Manages security (permissions)
Kernel

- The OS **kernel** is the core piece of software that has complete control over the system
  - Direct access to all hardware ("kernel mode")
    - All other software runs in user mode
  - Design philosophies: **monolithic kernels** vs. **microkernels**
    - Classic debate: Tanenbaum vs. Torvalds
  - Often designed to be small but extensible
    - Plugins are called **drivers**
  - Technically, "**Linux**" is a kernel
    - Some call the operating system “**GNU/Linux**”
    - Combination of **Linux** kernel and **GNU** userspace utilities
The OS provides many useful abstractions so that programs don’t need to handle hardware details

- CS 450 covers operating systems in detail

In this class:

- **Virtual memory**: logical view of memory hierarchy
- **Process**: logical view of a program running on a CPU
- **File**: logical view of data on a disk
- **Thread**: *logical flow of execution in a program*
Virtual memory

- Kernel translates between virtual and physical addresses
- Goals:
  - Use main memory as a cache for disks
  - Provide every process with a uniform view of memory
  - Protect processes from interference
An address space is an ordered set of non-negative integer addresses
- Ex: \{ 0, 1, 2, 3, \ldots, 499, 500 \}
- Linear address spaces don’t skip any addresses
- Two address spaces: virtual and physical
- Every byte has two addresses (virtual and physical)

Example: Y86 programs have a virtual address space with addresses that range from 0x0 to 0x1000, which is large enough to store 4K bytes
Virtual memory

- Fixed-sized memory partitioning
  - Virtual address space into virtual pages
  - Physical address space into physical pages (or frames)
  - Pages are usually relatively large (4 KB to 2 MB)

- Virtual memory uses RAM as a cache for pages
  - Process uses consistent virtual / logical addresses
  - OS translates these to physical addresses as necessary
    - Use a table for fast lookups!
  - We will assume hardware handles L1, L2, & L3 SRAM caches
• **Page table**: OS data structure for page lookups (array of page table entries)

• DRAM cache misses (called page faults) are very expensive
  - Disks are MUCH slower than DRAM
  - Transferring pages back and forth is called paging or swapping

before page fault on VP 3

after page fault on VP 3
Address translation

- n-bit virtual address space => m-bit physical address space
- p-bit page offsets (page size is \(2^p\))

[Diagram showing address translation process with virtual and physical address components]

- Page table base register (PTBR)
- Virtual page number (VPN) and virtual page offset (VPO)
- Valid bit indicating page presence in memory
- Physical page number (PPN) and physical page offset (PPO)
- The VPN acts as an index into the page table
- If valid = 0, page not in memory (page fault)
Address translation

- Memory management unit (MMU)
  - On-chip CPU component for address translation
  - Goal: perform translation as quickly as possible

Page hit

Page miss
Address translation

- Translation lookaside buffer (TLB)
  - Small cache of page table entries (PTEs) in MMUs
  - Provides faster address translations (in most cases)
  - *It’s caches all the way down* ...
CPU contains ALU, L1 cache, MMU, and TLB
  - L1 cache contains data pages

DRAM contains page table and data pages

HDD contains data pages
Address translation w/ L1 cache

1. Look up virtual page number in TLB using TLB index and tag
   - *TLB hit?*
     - yes
     - Get physical page number
       - *cache hit?*
         - yes
         - Load data from cache
         - *cache hit?*
           - no
           - Load line into cache
   - no
     - page fault!
     - OS loads page & assigns physical page number
     - Look up virtual page number in page table(s)
       - *page table hit?*
         - yes
         - Get physical page number
         - *cache hit?*
           - yes
           - Load data from cache
           - *cache hit?*
             - no
             - Load line into cache
   - no
     - page table hit?
       - yes
       - Get physical page number
       - *cache hit?*
         - yes
         - Load data from cache
         - *cache hit?*
           - no
           - Load line into cache
   - no
Memory management

- Operating system provides memory allocation service
  - `mmap` system call (`malloc` uses this)
  - Creates virtual memory allocation
  - **Private** regions: changes are only seen by owner
    - Private, variable-sized region called the heap
  - **Shared** regions: changes are seen by all processes
    - Usually between heap and stack
    - Multiple virtual addresses map to the same physical address
    - Changes are seen by all processes
    - Usually a read-only region for shared library code
Process address spaces

Kernel uses higher addresses

Typical Linux process address space
Process address spaces

- OSes maintain a separate page table for every process
  - Provides program **linking consistency**
    - E.g., code always begins at 0x400000
  - Simplifies **efficient loading**
    - Don’t actually load data from disk until needed (more efficient than P2!)
  - Streamlines **library sharing**
    - Keep one physical copy with multiple virtual mappings
  - Simplifies **memory allocation**
    - `malloc()` doesn’t need to find contiguous physical memory
  - Improves **security**
    - Processes can’t see/edit each others’ address spaces
Virtual memory caveats

- Virtual memory works well if a program has good locality
  - Especially temporal locality
  - This is a compelling reason to design for good locality
- Virtual memory works well if a program has a working set that fits in main memory
  - If this is not true, the system may need to continuously swap pages in and out
  - This is called thrashing, and is a significant cause of poor program performance
  - Can be detected by profilers (via counting page faults)
Our final module

- For the rest of the semester, we will continue discussing **operating systems** principles
  - Layers of abstraction that simplify development
  - Theme: *systems software is a foundation*
  - If you like this material, plan on taking **CS 450**
## Virtual address translation

### Virtual Address

<table>
<thead>
<tr>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
</table>

### Physical Address

<table>
<thead>
<tr>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
</table>

### Section of Page Tables in Main Memory

<table>
<thead>
<tr>
<th>VPN</th>
<th>PPN</th>
<th>Valid</th>
<th>VPN</th>
<th>PPN</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>28</td>
<td>1</td>
<td>08</td>
<td>13</td>
<td>1</td>
</tr>
<tr>
<td>01</td>
<td>---</td>
<td>0</td>
<td>09</td>
<td>17</td>
<td>1</td>
</tr>
<tr>
<td>02</td>
<td>33</td>
<td>1</td>
<td>0A</td>
<td>09</td>
<td>1</td>
</tr>
<tr>
<td>03</td>
<td>02</td>
<td>1</td>
<td>0B</td>
<td>---</td>
<td>0</td>
</tr>
<tr>
<td>04</td>
<td>---</td>
<td>0</td>
<td>0C</td>
<td>---</td>
<td>0</td>
</tr>
<tr>
<td>05</td>
<td>16</td>
<td>1</td>
<td>0D</td>
<td>2D</td>
<td>1</td>
</tr>
<tr>
<td>06</td>
<td>---</td>
<td>0</td>
<td>0E</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>07</td>
<td>---</td>
<td>0</td>
<td>0F</td>
<td>0D</td>
<td>1</td>
</tr>
</tbody>
</table>
# TLB cache

**Virtual Address**

```
| 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
```

---

**VPN**

---

**VPO**

---

**TLB (Translation Lookaside Buffer)**

<table>
<thead>
<tr>
<th>Set</th>
<th>Tag</th>
<th>PPN</th>
<th>Valid</th>
<th>Tag</th>
<th>PPN</th>
<th>Valid</th>
<th>Tag</th>
<th>PPN</th>
<th>Valid</th>
<th>Tag</th>
<th>PPN</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>03</td>
<td>---</td>
<td>0</td>
<td>09</td>
<td>0D</td>
<td>1</td>
<td>00</td>
<td>---</td>
<td>0</td>
<td>07</td>
<td>02</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>03</td>
<td>2D</td>
<td>1</td>
<td>02</td>
<td>---</td>
<td>0</td>
<td>04</td>
<td>---</td>
<td>0</td>
<td>0A</td>
<td>---</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>02</td>
<td>---</td>
<td>0</td>
<td>08</td>
<td>---</td>
<td>0</td>
<td>06</td>
<td>---</td>
<td>0</td>
<td>03</td>
<td>---</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>07</td>
<td>---</td>
<td>0</td>
<td>03</td>
<td>0D</td>
<td>1</td>
<td>0A</td>
<td>34</td>
<td>1</td>
<td>02</td>
<td>---</td>
<td>0</td>
</tr>
</tbody>
</table>
## L1 data cache

### Physical Address

```
11 10  9  8  7  6  5  4  3  2  1  0
```

### Direct-mapped L1 Data Cache

<table>
<thead>
<tr>
<th>Index</th>
<th>Tag</th>
<th>Valid</th>
<th>BLK 0</th>
<th>BLK 1</th>
<th>BLK 2</th>
<th>BLK 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>19</td>
<td>1</td>
<td>99</td>
<td>11</td>
<td>23</td>
<td>11</td>
</tr>
<tr>
<td>1</td>
<td>15</td>
<td>0</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>2</td>
<td>1B</td>
<td>1</td>
<td>00</td>
<td>02</td>
<td>04</td>
<td>08</td>
</tr>
<tr>
<td>3</td>
<td>36</td>
<td>0</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>4</td>
<td>32</td>
<td>1</td>
<td>43</td>
<td>6D</td>
<td>8F</td>
<td>09</td>
</tr>
<tr>
<td>5</td>
<td>0D</td>
<td>1</td>
<td>38</td>
<td>72</td>
<td>F0</td>
<td>1D</td>
</tr>
<tr>
<td>6</td>
<td>31</td>
<td>0</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>7</td>
<td>16</td>
<td>1</td>
<td>11</td>
<td>C2</td>
<td>DF</td>
<td>03</td>
</tr>
<tr>
<td>8</td>
<td>24</td>
<td>1</td>
<td>3A</td>
<td>00</td>
<td>51</td>
<td>89</td>
</tr>
<tr>
<td>9</td>
<td>2D</td>
<td>0</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>A</td>
<td>2D</td>
<td>1</td>
<td>93</td>
<td>15</td>
<td>DA</td>
<td>3B</td>
</tr>
<tr>
<td>B</td>
<td>0B</td>
<td>0</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>C</td>
<td>12</td>
<td>0</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>D</td>
<td>16</td>
<td>1</td>
<td>04</td>
<td>96</td>
<td>34</td>
<td>15</td>
</tr>
<tr>
<td>E</td>
<td>13</td>
<td>1</td>
<td>83</td>
<td>77</td>
<td>1B</td>
<td>D3</td>
</tr>
<tr>
<td>F</td>
<td>14</td>
<td>0</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>
Address translation w/ L1 cache

- Look up virtual page number in TLB using TLB index and tag
- TLB hit?
  - yes
  - Get physical page number
  - page table hit?
    - yes
    - Load data from cache
    - no
    - page fault!
    - OS loads page & assigns physical page number
  - no
  - page table hit?
    - no
    - cache hit?
      - yes
      - Load data from cache
      - no
      - Load line into cache

Diagram:
- CPU
  - ALU
  - MMU
  - L1 Cache
- TLB
- DRAM
  - Page Table
- HDD
- MMU
  - Page Table
- L1 Cache
- CPU
  - ALU
Address translation w/ L1 cache

1) Convert hex virtual address to binary representation
   - Fill in virtual address bits from RIGHT TO LEFT (extra is zeros on left)
2) Extract page number (VPN) and page offset (VPO) from virtual address
3) Extract TLB index and TLB tag from virtual address
4) In TLB, look up TLB index and tag to find PPN
   - If not valid: TLB miss!
5) If not in TLB look up VPN in page table to find PPN
   - If not in page table: page fault!
6) Assemble physical address from page number (PPN) and page offset (PPO)
   - Physical page offset (PPO) is the same as the virtual page offset (VPO)
7) Extract cache index and cache tag from physical address
8) In cache, look up cache index and tag
   - If not found, cache miss!
   - If found, return data