Virtual Memory

- Distinguish between memory addresses used by a process and the physical address used by the hardware
- Memory organized in constant blocks called pages
- Page table maps virtual page numbers to physical frames
Advantages of Virtual Memory

- **Extension:**
  - Physical Memory can act as a cache for disk space
  - Process(es) can use more memory than actually available
  - Only a subset of program (“Working Set”) stored in physical memory

- **Translation:**
  - Program has consistent view of memory, physical memory could be fragmented
  - Contiguous structures (e.g. stacks) use only as much physical memory as necessary yet can still grow later

- **Protection:**
  - Different processes protected from each other
  - Kernel data protected from user programs
  - Different pages can be given special behavior (e.g. read only)

- **Sharing:**
  - Can map same physical page to multiple users (“Shared memory”)

Address translation (I)

- **Translation via page tables:**
  - Page tables provide translation between page frame addresses in virtual memory and physical memory
  - Separate page table is maintained for every process
  - Process can read page table, modifications to page table performed by OS only

- **Problems with page tables**
  - Page tables can be huge:
    - Consider a process using 1 GB of memory
      - Memory organized in 4k pages => 262,144 entries in page table
    - Translation step can be slow
    - Requires multiple memory pages to store the page table
  - Page table is stored in virtual memory
    - Some memory pages storing the page table might be swapped onto disk
  - If using large number of processes, lots of memory consumed for storing page tables (1 per process)
Address translation (II)

- Translation Look-Aside Buffer (TLB)
  - Cache for address translation containing the most frequently/recently used page frame addresses
  - TLB is hardware
  - Multi-level TLBs can be used similar to multi-level caches
  - TLBs typically much smaller than data/instruction caches
- If a virtual address is not found in TLB, use page table

Address translation (III)

- On a context switch (i.e. executing another process), entries in the TLB might be invalid
  - Same virtual address could be used by multiple processes
- Older CPU flushed the TLB on a context switch
  - Initial access to any memory address will be a TLB miss
- Newer CPUs add process context identifier (PCID) to each TLB entry
  - Entries in the TLB will not necessarily lost over a context switch
  - Example: Intel Westmere architecture used a 12 bit PCID value
Virtualization

- **Goal:**
  - Run multiple instances of different OS on the same hardware
  - Present a *transparent* view of one or more environments (M-to-N mapping of M “real” resources, N “virtual” resources)

- **Challenges:**
  - Have to split all resources (processor, memory, hard drive, graphics card, networking card etc.) among the different OS -> virtualize the resources
  - The OS can not be aware that it is using virtual resources instead of real resources

---

- **Server consolidation**
  - What are you doing with hundreds of cores in a single server?

- **Sandboxing**
  - Have the web server and the database server on the same physical machine but run in different OS
  - If the web server is hacked, the data base server is not necessarily affected by that

- **Multiple execution environments**
  - Service a runs best on Windows, Service b runs best on Linux
Virtualization - concepts

- Two components when using virtualization:
  - Virtual Machine Monitor (VMM)
  - Virtual Machine(s) (VM)
- Para-virtualization:
  - Operating System has been modified in order to run as a VM
- ‘Fully’ Virtualized:
  - No modification required of an OS to run as a VM

Virtual Machine Monitor

- Also called ‘hypervisor’
- Isolates the state of each guest OS from each other
- Protects itself from guest software
- Determines how to map virtual resources to physical resources
  - Access to privileged state
  - Address translation
  - I/O
  - Exceptions and interrupts
- Relatively small code (compared to an OS)
- VMM must run in a higher privilege mode than guest OS
Managing Virtual Memory (I)

- Virtual memory offers many of the features required for hardware virtualization
  - Separates the physical memory for multiple processes
  - Page table restricts processes from
    - Reading a page of on another process or
    - Allow reading but not modifying a memory page or
    - Do not allow to interpret data in the memory page as instructions and do not try to execute them

Managing Virtual Memory (II)

- Virtual Memory management requires
  - Mechanisms to limit memory access to protected memory
  - At least two modes of execution for instructions
    - Privileged mode: an instruction is allowed to do whatever it wants -> kernel mode for OS
    - Non-privileged mode: user-level processes
    - Intel x86 Architecture: processor supports four levels
      - Level 0 used by OS
      - Level 3 used by regular applications
  - Provide mechanisms to go from non-privileged mode to privileged mode -> system call
Virtual Memory in Virtualized Environments

- Physical memory used by different guest OSes could collide
  - Each guest OS maintains its page tables to do the mapping from virtual address to physical address
- Solution 1: Introduce another translation step for every memory access
  - VMM holds additional table to map the physical address of a guest OS onto the ‘machine address’
- Solution 2: VMM maintains a shadow page table for each guest OS
  - Copy of the page table of the OS
  - Page tables of guest OS still works with physical addresses
  - VMM intercepts page table modifications by guest OS

Virtual Memory in Virtualized Environments

- TLBs using Process Context Identifiers (PCID) can in theory manage multiple entries from different guest OSes
  - Since PCID are created however by the OS and number of bits used for a PCID is limited, there might by PCID collisions between different guest OSes
  - VMM has to intercept creation of PCIDs or provide a mapping between guest OS PCIDs and ‘real’ PCIDs
Privilege Level on Intel Processors

- A protection ring is one of two or more hierarchical levels or layers of privilege within the architecture of a computer system.
- Ring 0 is the level with the most privileges and interacts most directly with the physical hardware such as the CPU and memory.
- Windows and Linux historically both used only levels 0 and level 3.


ISA related problems when running a guest OS

- **Ring Aliasing**: arises when software runs at a privilege level other than which it has been written for.
  - E.g. OS wants to modify page table
- **Address Space Compression**: OS expects to have access to the processors full virtual address.
  - VMM needs a portion of the virtual address space
- **Non-faulting access to privileged State**
  - E.g. read-access to certain state registers
- **Adverse Impact on Guest System Calls**
  - A system call on a guest OS should not change to the privilege level 0, but to the level of the guest OS.
ISA related problems when running a guest OS (II)

- **Interrupt Virtualization**
  - IA32 architecture provides the ability to mask external interrupts if the OS is not ready to handle it
  - OS frequently uses this feature
  - VMM will have to handle external interrupts (e.g. hard drive, network), since it has to determine which guest OS needs to be ‘informed’ about that

- **Access to Hidden State**
  - E.g. IA32 has hidden descriptor caches for segment registers
  - No mechanism available saving and restoring hidden components of a guest context when changing VM

Example 1: Intel Core i7 memory hierarchy

- 48 bit virtual address
- 36 bit physical address
- 2-level TLB caches
- 4 KB page size = 2^{12} bytes

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Instruction TLB</th>
<th>Data TLB</th>
<th>Second level TLB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>128 entries</td>
<td>64 entries</td>
<td>512 entries</td>
</tr>
<tr>
<td>Associativity</td>
<td>4-way</td>
<td>4-way</td>
<td>4-way</td>
</tr>
<tr>
<td>Access latency</td>
<td>1 cycle</td>
<td>1 cycle</td>
<td>6 cycles</td>
</tr>
<tr>
<td>Miss</td>
<td>7 cycles</td>
<td>7 cycles</td>
<td>100s of cycles</td>
</tr>
</tbody>
</table>
### Intel Core i7 memory hierarchy (II)

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>L1 Instruction</th>
<th>L1 Data</th>
<th>L2</th>
<th>L3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>32 KB</td>
<td>32 KB</td>
<td>256 KB</td>
<td>2 MB per core</td>
</tr>
<tr>
<td>Associativity</td>
<td>4-way</td>
<td>8-way</td>
<td>8-way</td>
<td>16-way</td>
</tr>
<tr>
<td>Block size</td>
<td>64 bytes</td>
<td>64 bytes</td>
<td>64 bytes</td>
<td>64 bytes</td>
</tr>
<tr>
<td>Access latency</td>
<td>4 cycles</td>
<td>4 cycles</td>
<td>10 cycles</td>
<td>35 cycles</td>
</tr>
<tr>
<td>No. of index bits</td>
<td>7</td>
<td>6</td>
<td>9</td>
<td>13 bits*</td>
</tr>
</tbody>
</table>

*Assuming a 4-core processors

- L1 and L2 separate per core, L3 shared among all cores
- L1 cache is virtually indexed but physically tagged
- L2 and L3 are physically indexed and tagged
- L3 inclusive of L1 and L2 cache

---

### Accessing the Data TLB

- **Given a PC ( = virtual address)**
  - Send page frame of virtual address to L1 data TLB to retrieve physical address
  - TLB provides physical address if found and checks for access violation
  - If not found in L1 Data TLB, 2\(^{nd}\) level TLB is checked
  - If not found in 2\(^{nd}\) level TLB operating system has to perform the translation
Accessing the Cache

• To identify address in L1 Data cache:
  - Index field of virtual address: 6 bits
  - Cache tag of physical address: 24 bits =
    36 bits physical address - 6 bits index - 6 bits block offset

• 2nd level cache is physically indexed and physically tagged
  - 36 bits physical address decomposed into:
    6 bits offset, 9 bits index and 21 bits tag
Example 2: Intel Xeon E5-2680 v4

<table>
<thead>
<tr>
<th>Cache Type</th>
<th>L1 Data Cache</th>
<th>L2 Unified Cache</th>
<th>L3 Unified Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total size</td>
<td>32 KB</td>
<td>Total size:</td>
<td>256 KB</td>
</tr>
<tr>
<td>Line size</td>
<td>64 B</td>
<td>Line size:</td>
<td>64 B</td>
</tr>
<tr>
<td>Number of Lines</td>
<td>512</td>
<td>Number of Lines:</td>
<td>4096</td>
</tr>
<tr>
<td>Associativity</td>
<td>8</td>
<td>Associativity:</td>
<td>8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cache Type</th>
<th>L1 Instruction Cache</th>
<th>L3 Unified Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total size</td>
<td>32 KB</td>
<td>Total size:</td>
</tr>
<tr>
<td>Line size</td>
<td>64 B</td>
<td>Line size:</td>
</tr>
<tr>
<td>Number of Lines</td>
<td>512</td>
<td>Number of Lines:</td>
</tr>
<tr>
<td>Associativity</td>
<td>8</td>
<td>Associativity:</td>
</tr>
</tbody>
</table>