Virtual Memory

- Topics (Chapter 9)
  - Motivations for VM
  - Address translation
Motivations for Virtual Memory

- **Use Physical DRAM as a Cache for the Disk**
  - Address space of a process can exceed physical memory size
  - Sum of address spaces of multiple processes can exceed physical memory

- **Simplify Memory Management**
  - Multiple processes resident in main memory.
    - Each process with its own address space
  - Only “active” code and data is actually in memory
    - Allocate more memory to process as needed.

- **Provide Protection**
  - One process can’t interfere with another.
    - because they operate in different address spaces.
  - User process cannot access privileged information
    - different sections of address spaces have different permissions.
Motivation #1: DRAM a “Cache” for Disk

- Full address space is quite large:
  - 32-bit addresses: \(~4,000,000,000\) (4 billion) bytes
  - 64-bit addresses: \(~16,000,000,000,000,000,000\) (16 quintillion) bytes
- Disk storage is \(~100\)X cheaper than DRAM storage:
  - 100 GB of DRAM: \(~$1,500\)
  - 100 GB of disk: \(~$150\)
- To access large amounts of data in a cost-effective manner, the bulk of the data must be stored on disk

\[\begin{align*}
8 \text{ MB}: \sim$80 \\
1 \text{ GB}: \sim$15 \\
100 \text{ GB}: \sim$15 - $20 \\
1 \text{ TB}: \sim$100 - $150
\end{align*}\]
A System with Physical Memory Only

- Examples:
  - most Cray machines, early PCs, nearly all embedded systems, etc.

Addresses generated by the CPU correspond directly to bytes in physical memory.
Locating an Object in a “Cache”

- **SRAM Cache (L1 & L2)**
  - Tag stored with cache line
  - Maps from cache block to memory blocks
    - From cached to uncached form
    - Save a few bits by only storing tag
  - No tag for block not in cache
  - Hardware retrieves information
    - can quickly match against multiple tags

```
Object Name

X

= X?
```

```
<table>
<thead>
<tr>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0:</td>
<td>D</td>
</tr>
<tr>
<td>1:</td>
<td>X</td>
</tr>
<tr>
<td>N-1:</td>
<td>J</td>
</tr>
</tbody>
</table>

“Cache”
```
Main Memory As a cache of Disk

- Unit of memory allocation pages
- Consider each executable (a.out) as a sequence of pages (0…N)
- Parameter: Page Size
- CPU emits Virtual addresses.
- Need to translate virtual address to physical address
- Maintain a data structure called page table
Virtual address vs physical address

- **DRAM Cache (i.e. Main Memory)**
  - Each allocated page of virtual memory has entry in *page table*
  - Mapping from virtual pages to physical pages
    - From uncached form to cached form
  - Page table entry even if page not in memory
    - Specifies disk address
    - Only way to indicate where to find page
  - OS retrieves information
A System with Virtual Memory

- Examples:
  - workstations, servers, modern PCs, etc.

- Address Translation: Hardware converts virtual addresses to physical addresses via OS-managed lookup table (page table)
Page Faults (like “Cache Misses”)

- What if an object is on disk rather than in memory?
  - Page table entry indicates virtual address not in memory
  - OS exception handler invoked to move data from disk into memory (Page Fault)
    - current process suspends, others can resume
    - OS has full control over placement, etc.

Before fault

After fault
Servicing a Page Fault

- **Processor Signals Controller**
  - Read block of length $P$ starting at disk address $X$ and store starting at memory address $Y$
- **Read Occurs**
  - Direct Memory Access (DMA)
  - Under control of I/O controller
- **I/O Controller Signals Completion**
  - Interrupt processor
  - OS resumes suspended process
Motivation #2: Memory Management

- Multiple processes can reside in physical memory.
- How do we resolve address conflicts?
  - what if two processes access something at the same address?

Linux/x86 process memory image

- kernel virtual memory
- stack
- Memory mapped region for shared libraries
- runtime heap (via malloc)
- uninitialized data (.bss)
- initialized data (.data)
- program text (.text)
- forbidden

memory invisible to user code

Linux/x86 process memory image

the “brk” ptr

%esp

memory invisible to user code
Solution: Separate Virt. Addr. Spaces

- Virtual and physical address spaces divided into equal-sized blocks
  - blocks are called “pages” (both virtual and physical)
- Each process has its own virtual address space
  - operating system controls how virtual pages are assigned to physical memory

```
Virtual Address Space for Process 1:

Virtual Address Space for Process 2:
```
Motivation #3: Protection

- Page table entry contains access rights information
  - hardware enforces this protection (trap into OS if violation occurs)
VM Address Translation

- **Virtual Address Space**
  - \( V = \{0, 1, \ldots, N-1\} \)

- **Physical Address Space**
  - \( P = \{0, 1, \ldots, M-1\} \)
  - \( M < N \)

- **Address Translation**
  - **MAP**: \( V \rightarrow P \cup \{\emptyset\} \)
  - For virtual address \( a \):
    - \( \text{MAP}(a) = a' \) if data at virtual address \( a \) at physical address \( a' \) in \( P \)
    - \( \text{MAP}(a) = \emptyset \) if data at virtual address \( a \) not in physical memory
      - Either invalid or stored on disk
VM Address Translation: Hit

Processor → Hardware Addr Trans Mechanism → Main Memory

virtual address → part of the on-chip memory mgmt unit (MMU) → physical address
VM Address Translation: Miss

- Processor
- Virtual address (a)
- Hardware Addr Trans Mechanism
  - Part of the on-chip memory mgmt unit (MMU)
- Main Memory
- Physical address (a')
- Secondary Memory
- OS performs this transfer (only if miss)
- Page fault
- Fault handler

The diagram illustrates the process of virtual address translation in the context of a missing page fault. The processor generates a virtual address (a), which is processed by the hardware address translation mechanism. If the address is not found in the on-chip memory management unit (MMU), a page fault occurs, leading to a fault handler. Depending on the outcome, the memory is accessed from either main memory or secondary memory.
VM Address Translation

- **Parameters**
  - $P = 2^p = \text{page size (bytes)}$.
  - $N = 2^n = \text{Virtual address limit}$
  - $M = 2^m = \text{Physical address limit}$

Page offset bits don’t change as a result of translation
Page Tables

Virtual Page Number

Memory resident page table (physical page or disk address)

Valid

1
1
0
1
1
1
0
1
0
1

Physical Memory

Disk Storage (swap file or regular file system file)
Address Translation via Page Table

- **virtual page number (VPN)**
- **page offset**
- **virtual address**
- **physical page number (PPN)**
- **page offset**
- **physical address**

- **page table base register**
- **VPN acts as table index**
- **valid access physical page number (PPN)**
- **if valid=0 then page not in memory**

```
0
     p
     p-1
n-1
     m-1
```
Page Table Operation

- Translation
  - Separate (set of) page table(s) per process
  - VPN forms index into page table (points to a page table entry)
Page Table Operation

• Computing Physical Address
  • Page Table Entry (PTE) provides information about page
    • if (valid bit = 1) then the page is in memory.
      • Use physical page number (PPN) to construct address
    • if (valid bit = 0) then the page is on disk
      • Page fault
Page Table Operation

- Checking Protection
  - Access rights field indicate allowable access
    - e.g., read-only, read-write, execute-only
    - typically support multiple protection modes (e.g., kernel vs. user)
  - Protection violation fault if user doesn’t have necessary permission
Summary

• Cache is a faster store than main memory
• Physical memory is faster than disk
• Virtual memory allows Physical memory to act as a cache for Disk
• Need to use page table for translating Virtual address to physical address