Memory Management

From Chapter 3, Modern Operating Systems, Andrew S. Tanenbaum

Kartik Gopalan
Memory Management

• Ideally programmers want memory that is
  • large
  • fast
  • non volatile

• Memory hierarchy
  • small amount of fast, expensive memory – cache
  • some medium-speed, medium price main memory
  • Lots of slow, cheap disk storage

• Memory manager handles the memory hierarchy
Typical Memory Hierarchy

Typical access time

1 nsec
2 nsec
10 nsec
10 msec
100 sec

Typical capacity

<1 KB
1 MB
64-512 MB
5-50 GB
20-100 GB

SLOWER
FASTER

Registers
Cache
Main memory
Magnetic disk
Magnetic tape

BIGGER
SMALLER
Basic Memory Management

"Mono-programming" without Swapping or Paging

Three simple ways of organizing memory
- an operating system with one user process
Multiprogramming with Fixed Partitions

- Fixed memory partitions
  (a) separate input queues of processes for each partition
  (b) single input queue
Relocation and Protection

- **Problem:** A programmer doesn’t know where a program will be loaded in memory
  - address locations of variables and code routines cannot be absolute
  - must keep a program out of other processes’ partitions
- **Solution:** Use base and limit values
- **Relocation**
  - Address locations in a program are relative.
  - They are added to a **base value** to map to physical addresses.
- **Protection**
  - Access to address locations larger than **limit value** results in an error
What if physical memory is not enough to hold all processes?  
— Swapping

- Physical memory may not be enough to accommodate the needs of all processes
- Memory allocation changes as
  - processes come into memory
  - leave memory and are *swapped out* to disk
  - Re-enter memory by getting *swapped-in* from disk
- Shaded regions are unused memory
What if physical memory is not enough for a single process? — Virtual Memory

- Swapping is useful when the sum total of memory requirements of all processes is greater than DRAM available in the system.

- But sometimes, a single process might require more memory than the available DRAM in the system.

- In such cases swapping is not enough. Rather, we need to break up the memory space of a process into smaller equal-sized pieces (called pages).

- Operating system then decides which pages stay in memory and which get moved to disk.

- **Virtual memory**: means that each process gets an illusion that it has more memory than the physical DRAM in the system.
Virtual Memory and MMU

- MMU = Memory Management Unit
- Part of Hardware that accompanies the CPU
- Converts Virtual Addresses to Physical Addresses
Page Tables

- Stores the mapping between virtual addresses and physical addresses
Virtual Address Translation
For Small Address Space

Internal operation of MMU with 16 4 KB pages
Virtual Address Translation
For Large Address Space

• 32 bit address with 2 page table fields

• Two-level page tables

• PT too Big for MMU
  • Keep it in main memory

• But how does MMU know where to find PT?
  • Registers (CR2 on Intel)
Typical Page Table Entry (PTE)

- Page Frame number = physical page number for the virtual page represented by the PTE.
- Referenced bit: Whether the page was accessed since last time the bit was reset.
- Modified bit: Also called “Dirty” bit. Whether the page was written to, since the last time the bit was reset.
- Protection bits: Whether the page is readable? writeable? executable? contains higher privilege code/data?
- Present/Absent bit: Whether the PTE contains a valid page frame #. Used for marking swapped/unallocated pages.
TLBs – Translation Lookaside Buffers

<table>
<thead>
<tr>
<th>Valid</th>
<th>Virtual page</th>
<th>Modified</th>
<th>Protection</th>
<th>Page frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>140</td>
<td>1</td>
<td>RW</td>
<td>31</td>
</tr>
<tr>
<td>1</td>
<td>20</td>
<td>0</td>
<td>R X</td>
<td>38</td>
</tr>
<tr>
<td>1</td>
<td>130</td>
<td>1</td>
<td>RW</td>
<td>29</td>
</tr>
<tr>
<td>1</td>
<td>129</td>
<td>1</td>
<td>RW</td>
<td>62</td>
</tr>
<tr>
<td>1</td>
<td>19</td>
<td>0</td>
<td>R X</td>
<td>50</td>
</tr>
<tr>
<td>1</td>
<td>21</td>
<td>0</td>
<td>R X</td>
<td>45</td>
</tr>
<tr>
<td>1</td>
<td>860</td>
<td>1</td>
<td>RW</td>
<td>14</td>
</tr>
<tr>
<td>1</td>
<td>861</td>
<td>1</td>
<td>RW</td>
<td>75</td>
</tr>
</tbody>
</table>

- TLB is a small cache that speeds up the translation of virtual addresses to physical addresses.
- TLB is part of the MMU hardware (comes with CPU)
- It is not a Data Cache or Instruction Cache. Those are separate.
- TLB simply caches translations from virtual page number to physical page number so that the MMU don’t have to access page-table in memory too often.
- On x86 architecture, TLB has to be “flushed” upon every context switch because there is no field in TLB to identify the process context.
Cold Start Penalty

• Cost of repopulating the TLB (and other caches) upon a context switch.

• Immediately after a context switch, all (or many) of TLB entries are invalidated.
  • On some x86 processors, TLB has to be “flushed” upon every context switch because there is no field in TLB to identify the process context.

• Every memory access by the newly scheduled process may result in a TLB miss.

• MMU must then walk the page-table in main memory to repopulate the missing TLB entry, which takes longer than a cache hit.
Tagged TLB

• A “tag” in each TLB entry identifies the process/thread context to which the TLB entry belongs

• Thus TLB entries for more than one execution context can be stored simultaneously in the TLB.
  • TLB lookup hardware matches the tag in addition to the virtual page number.

• With tags, context switch no longer requires a complete TLB flush.
  • Reduces cold-start penalty.
Two types of memory translation architectures

- **Architected Page Tables**
  - Page table interface defined by ISA and understood by memory translation hardware
  - E.g. x86 architecture
  - MMU handles TLB miss (in hardware)
  - OS handles page faults (in software)
  - ISA specifies page table format

- **Architected TLBs**
  - TLB interface defined by ISA and understood by MMU
  - E.g. alpha architecture
  - TLB miss handled by OS (in software)
  - ISA does not specify page table format
Impact of Page Size on Page tables

Small page size

• Advantages
  • less internal fragmentation
  • page-in/page-out less expensive

• Disadvantages
  • process that needs more pages has larger page table
TLB Coverage

- Max amount of memory mapped by TLB
  - Max amount of memory that can be accessed without TLB misses

- TLB Coverage = N x P bytes
  - N = Number of entries in TLB
  - P = Page size in bytes
  - N is fixed by hardware constraints
  - So, to increase TLB Coverage, we must increase P.

- Consider these extreme examples
  - Suppose P = 1 byte
    - TLB Coverage = N bytes only
  - Suppose P = 2^64 bytes (on a 64-bit ISA)
    - TLB Coverage = N x2^64 bytes
    - TLB can perform translations for N processes without any TLB misses!

- Of course, both examples above are impractical and meant to illustrate the tradeoffs.

- But what if P is something reasonable, but greater than the standard 4KB?

- This brings us next to superpages.
Superpages

- Memory pages of larger sizes than standard pages
  - supported by most modern CPUs

- Superpage size = power of 2 x the base page size

- Only one TLB entry per superpage
  - But multiple (identical) page-table entries, one per base page

- Constraints:
  - contiguous (physically and virtually)
  - aligned (physically and virtually)
  - uniform protection attributes
  - one reference bit, one dirty bit
A superpage TLB

**Virtual Address Space**

- **Base Page Entry (size=1)**
- **Superpage Entry (size=4)**

**Physical Memory**

**Alpha:**
- 8, 64, 512KB; 4MB

**Itanium:**
- 4, 8, 16, 64, 256KB; 1, 4, 16, 64, 256MB
Quiz

• Consider a machine that has a 32-bit virtual address space and 8KByte page size.

1. What is the total size (in bytes) of the virtual address space for each process?

2. How many bits in a 32-bit address are needed to determine the page number of the address?

3. How many bits in a 32-bit address represent the byte offset into a page?

4. How many page-table entries are present in the page table?
Quiz Answers

• Consider a machine that has a 32-bit virtual address space and 4KByte page size.

1. Total size (in bytes) of the virtual address space for each process = \(2^{32} = 4 \times 1024 \times 1024 \times 1024\) bytes = 4 GB

2. Number of pages in virtual address space = \(4\text{GB}/8\text{KB} = 512 \times 1024 = 2^9 \times 2^{10} = 2^{19}\)
   • So the number of bits in a 32-bit address are needed to determine the page number of the address = \(\log_2(4\text{GB}/8\text{KB}) = \log_2(2^{19}) = 19\) bits

3. How many bits in a 32-bit address represent the byte offset into a page?
   • \(\log_2(8\text{KB}) = \log_2(2^{13}) = 13\)
   • Also, \(32 - 19 = 13\) bits

4. How many page-table entries are present in the page table?
   • Number of PTEs = Number of pages in virtual address = \(4\text{GB}/8\text{KB} = 2^{19}\) pages
References

• Chapter 3: Modern Operating Systems, Andrew S. Tanenbaum

• X86 architecture

• Memory segment
  http://en.wikipedia.org/wiki/Memory_segment

• Memory model
  http://en.wikipedia.org/wiki/Memory_model

• IA-32 Intel Architecture Software Developer’s Manual, Volume 1: Basic Architecture