Recap of the Last Class

- Running a user program
  - compile? link? load? execute

- Address binding
  - compile-time, load-time, execution-time

- Logical vs. physical address
  - address translation: memory mapping unit

- Swapping system vs. virtual memory system
  - contiguous allocation for swapping systems

Paging (non-contiguous allocation)

- Physical address space of a process can be noncontiguous: process is allocated physical memory whenever the latter is available.
- Divide physical memory into fixed-sized blocks called frames (typically between 512 bytes and 8192 bytes).
- Divide logical memory into blocks of same size called pages.
- To run a program of size \( n \) pages, need to find \( n \) free frames and load program.
- Internal fragmentation.

Paging: Address Translation Scheme

A logical address is divided into:

- Page number (\( p \)) - used as an index into a page table which contains base address of each page in physical memory.
- Page offset (\( d \)) - combined with base address to define the physical memory address that is sent to the memory.
**Load A User Program: An Example**

Before loading  

After loading  

---

**Implementation of Page Table**

- Page table is kept in main memory (kernel or user space?).
- Hardware MMU:
  - Page-table base register (PTBR) points to the page table.
  - Page-table length register (PRLR) indicates size of the page table.
- In this scheme every data/instruction access requires two memory accesses. One for the page table and one for the data/instruction.
- Solution:
  - A special fast-lookup hardware cache called associative memory or translation look-aside buffers (TLBs)

---

**Paging MMU With TLB**

---

**Effective Access Time**

- TLB Lookup = 1 ns
- Assume memory cycle time is 100 ns
- Hit ratio ($\alpha$) - percentage of times that a page number is found in the TLB.
- Effective Access Time (EAT)
  
  \[ EAT = 101 \times \alpha + 201 \times (1 - \alpha) \]
Memory Protection

- How is memory protection achieved?
  - protecting the page tables in the kernel memory space.

- Parts of the logical address space may not be mapped
  - Valid-invalid bit attached to each entry in the page table:
    - "valid" indicates that the associated page is in the process' logical address space, and is thus a legal page.
    - "invalid" indicates that the page is not in the process' logical address space.
  - Software trap if attempting to access an invalid page.

Page Table Structure

- Problem with a flat linear page table
  - assume a page table entry is 4-byte; page size is 4KB; the 32-bit address space is 4GB large
  - how big is the flat linear page table?

- Solutions:
  - Hierarchical Page Tables
    - break the logical page number into multiple levels
  - Hashed Page Tables
  - Inverted Page Tables

- Metrics:
  - Space consumption and lookup speed

Two-Level Page Table

- A logical address (on 32-bit machine with 4K page size) is divided into:
  - a page offset consisting of 12 bits.
  - a page number consisting of 20 bits; further divided into:
    - a 10-bit level-2 page number.
    - a 10-bit level-1 page number.

  Thus, a logical address looks like:

<table>
<thead>
<tr>
<th>page number</th>
<th>page offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p_1 )</td>
<td>( p_2 )</td>
</tr>
</tbody>
</table>

- Address translation scheme:

  ![Two-Level Page Table Diagram](image)

Two-Level Page Table: An Example

- A logical address (on 32-bit machine with 4K page size) is divided into:
  - a page offset consisting of 12 bits.
  - a page number consisting of 20 bits; further divided into:
    - a 10-bit level-2 page number.
    - a 10-bit level-1 page number.

  Thus, a logical address looks like:

<table>
<thead>
<tr>
<th>page number</th>
<th>page offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p_1 )</td>
<td>( p_2 )</td>
</tr>
</tbody>
</table>

- Address translation scheme:

  ![Two-Level Page Table: An Example Diagram](image)
Deal With 64-bit Address Space

- Two-level page tables for 64-bit address space
  - more levels are needed
- Inverted page tables
  - One entry for each real page of memory.
  - Entry consists of the process ID and virtual address of the page stored in that real memory location.

Hashed Page Tables

- The virtual page number is hashed into a page table. This page table contains a chain of elements hashing to the same location.
- Virtual page numbers are compared in this chain searching for a match. If a match is found, the corresponding physical frame is extracted.

Segmentation

- One-dimensional address space with growing pieces
- At compile time, one table may bump into another
- Segmentation:
  - generate segmented logical address at compile time
  - segmented logical address is translated into physical address at execution time
  - by software or hardware?

Example of Segmentation
Sharing of Segments

Segmentation & Paging - Intel 386

- Segmentation and paging with a two-level paging scheme.

Pages vs. Segmentation

<table>
<thead>
<tr>
<th>Consideration</th>
<th>Paging</th>
<th>Segmentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Need the programmer be aware that this technique is being used?</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>How many linear address spaces are there?</td>
<td>1</td>
<td>Many</td>
</tr>
<tr>
<td>Can the total address space exceed the size of physical memory?</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Can procedures and data be distinguished and separately protected?</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Can tables whose size fluctuates be accommodated easily?</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Is sharing of procedures between users facilitated?</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Why was this technique invented?</td>
<td>To get a large linear address space without having to buy more physical memory</td>
<td>To allow programs and data to be broken up into logically independent address spaces and to aid sharing and protection</td>
</tr>
</tbody>
</table>

Disclaimer

- Parts of the lecture slides contain original work of Abraham Silberschatz, Peter B. Galvin, Greg Gagne, Andrew S. Tanenbaum, and Gary Nutt. The slides are intended for the sole purpose of instruction of operating systems at the University of Rochester. All copyrighted materials belong to their original owner(s).