Virtual Memory

CS 337 • Introduction to Operating Systems

Multiprogramming

• Modern systems keep more than one program in memory at a time
• Often, all these programs together require more memory than what is available
• What to do?
  – use a part of disk and make it look like memory
  – this is called virtual memory

Disk vs. Memory

• Memory characteristics
  – fast - typically 100 ns per access
  – small - hundreds of megabytes
  – random access of any byte
• Disk characteristics
  – very slow - several milliseconds per access
  – large - tens of gigabytes
  – random access of any block (512 bytes)
Virtual Memory

- Basic concept
  - keep frequently used process data in physical memory
  - keep the rest of a processes address space on disk
  - if a piece of infrequently used data is needed, bring it in from disk
- Before any data can be used, it must be in physical memory

Overlays

- User controls what info is on disk, and what is in memory
- To access info kept on disk
  - save some portion of current memory to disk
  - bring in desired info to memory
- Very difficult to implement
- Becomes very system dependant
  - what if more memory becomes available?
  - what if less is available?
Overlays

Paging
- This is the way it done today
- User thinks virtual memory is one large array of real memory
- Let special hardware and the OS keep up this illusion
- Basic idea
  - user enters address from virtual space
    - usually 32 or 64 bits (2^32 or 2^6 addressable bytes)
  - hardware and OS map this virtual address to physical address

Paging
- Break physical memory into frames
- Break virtual memory into pages
- Page size must be multiple of frame size
  - for simplicity, we’ll assume the same size
- When an address is accessed
  - find out which page it is
  - if not in memory, bring it in
  - now grab the data
Memory Map

- Two solutions to almost every problem in computer science
  - indirection
  - caching
- The memory map is a form of indirection
- Call this memory map the page table

Paging

![Diagram of paging]

Page Table

- Keep a record of every page in virtual memory
- Record actual location of page in this table
  - frame in memory
- Also record some other information in table
  - valid or invalid (in memory or not)
  - protection bits (read/write/execute)
Page Table Entry

- Assume 32 bit addressing
- Entry in table will be 32 bits plus a few extra
  - 32 bits are address in memory
  - extra bits are valid/invalid and protection bits
- If entry is valid, the address is the starting location of the page in main memory
- Index of entry is the page number

Page Table

- Assume the page size is 100

<table>
<thead>
<tr>
<th>Page Number</th>
<th>Location of start of page (disk or memory)</th>
<th>W</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>100</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>200</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>300</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>400</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>500</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>600</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>700</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>800</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>900</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>1000</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>2000</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Calculating Physical Address

- User supplies a virtual address
  - high order bits are the page number
  - low order bits are the offset into the page
- Go to appropriate index in page table
- Examine valid bit
  - if valid, grab starting address of page from table
  - if not, generate a page fault, bring it into memory, set page table entry, grab starting address
  - OS uses another table to find location on disk
- now combine the page table entry and offset to calculate the true physical address
Calculating Physical Address

1. [Diagram showing the process of calculating physical address]

2. **User Instruction:** `srh x` 
   - Page Number: 5
   - Offset: 33
   - Check index 5 in page table:
     - It is valid and non-leaf
     - Base = 500
   - New calculate physical address:
     - PA = base + offset = 500 + 33 = 533

3. **User Instruction:** `srh y` 
   - Page Number: 2
   - Offset: 75
   - Check index 2 in page table:
     - It is not valid
     - Assume following is now in memory
     - Assume following is now in memory
     - Assume following is now in memory
     - Assume following is now in memory
   - New calculate physical address:
     - PA = base + offset = 500 + 75 = 575

Virtual Address

- Make all pages a power of 2 in size
  - and make them a multiple of 512 (disk blocks)
- A virtual address consists of 32 bits
  - 64 bits in some systems
- Assume a page size of 4K
  - need 12 bits for the offset ($2^{12} = 4K$)
  - that leaves 20 bits for the page number
  - our system can hold 1M ($2^{20}$) of 4K pages
  - 4 GB
Virtual Address

- Given the following virtual address:

<table>
<thead>
<tr>
<th>page number</th>
<th>offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000000011</td>
<td>00000100</td>
</tr>
</tbody>
</table>

- page number = 51
- offset in page = 36 bytes

- How many pages would there be with:
  - 16 K pages
  - 1 K pages

Locality of Reference

- Important concept in computer science
- Spatial locality
  - if an address x is accessed, high probability that address x+1 will also be referenced
- Temporal locality
  - if an address x is accessed at time t, high probability it will be accessed again in t+δ where δ is small

Page Size

- Proper page size depends on the program reference behavior
- Too small a page size
  - too much overhead
  - does not consider locality of reference
- Too large a page size
  - waste memory with data that will never be used
    - holding space that another process could use
  - assumes too much locality of reference