Announcement

• Homework 2 and project 1 graded
• Homework 3 out
• Work on your project 2!
<table>
<thead>
<tr>
<th>Scheme</th>
<th>How</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple uniprogramming</td>
<td>1 segment loaded to starting address 0</td>
<td>Simple</td>
<td>1 process</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 segment</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>No protection</td>
</tr>
<tr>
<td>Simple multiprogramming</td>
<td>1 segment relocated at loading time</td>
<td>Simple, Multiple processes</td>
<td>1 segment/process</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>No protection</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>External frag.</td>
</tr>
<tr>
<td>Base &amp; Bound</td>
<td>Dynamic mem relocation at runtime</td>
<td>Simple hardware, Multiple processes Protection</td>
<td>1 segment/process, External frag.</td>
</tr>
<tr>
<td>Multiple segments</td>
<td>Dynamic mem relocation at runtime</td>
<td>Sharing, Protection, multi segs/process</td>
<td>External frag.</td>
</tr>
</tbody>
</table>
What fundamentally causes external fragmentation?

1. Segments of many different sizes

2. Each has to be allocated contiguously

- “Million-dollar” question:
  
  Physical memory is precious.
  Can we limit the waste to a single hole of X bytes?
Paging

- Paging solves the external fragmentation problem by using fixed sized units in both physical and virtual memory.
Paging

• Translating addresses
  ♦ Virtual address has two parts: virtual page number and offset
  ♦ Virtual page number (VPN) is an index into a page table
  ♦ Page table determines page frame number (PFN)
  ♦ Physical address is PFN::offset ("::" means concatenate)

• Page tables
  ♦ Map virtual page number (VPN) to page frame number (PFN)
    » VPN is the index into the table that determines PFN
  ♦ One page table entry (PTE) per page in virtual address space
    » Or, one PTE per VPN
Paging

Virtual address

VPN | offset

Page table

PFN | ...  
....... 
....... 
PFN | ...

PFN | offset

Physical address

page table size

error

Physical address
Paging Example

- **Pages are 4K**
  - 4K → offset is 12 bits → VPN is 20 bits \(2^{20}\) VPNs), assuming 32bit system

- **Virtual address is 0x7468**
  - Virtual page is 0x7, offset is 0x468 (lowest 12 bits of address)

- **Page table entry 0x7 contains 0x2**
  - Page frame number is 0x2
  - Seventh virtual page is at address 0x2000 (physical page 2)

- **Physical address = 0x2000 :: 0x468 = 0x2468**
Page Tables

- Page tables completely define the mapping between virtual pages and physical pages for an address space.
- Each process has an address space, so each process has a page table.
- Page tables are data structures maintained by the OS (and accessed by hardware).
Page Table Entries (PTEs)

- Page table entries control mapping
  - The **Modify** bit says whether or not the page has been written
    » It is set when a write to the page occurs
  - The **Reference** bit says whether the page has been accessed
    » It is set when a read or write to the page occurs
  - The **Valid** bit says whether or not the PTE can be used
    » It is checked each time the virtual address is used
  - The **Protection** bits say what operations are allowed on page
    » Read, write, execute
  - The **page frame number** (PFN) determines physical page

<table>
<thead>
<tr>
<th>M</th>
<th>R</th>
<th>V</th>
<th>Prot</th>
<th>Page Frame Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>20</td>
</tr>
</tbody>
</table>
x86 Page Table Entry

<table>
<thead>
<tr>
<th>Page frame number</th>
<th>U</th>
<th>P</th>
<th>Cw</th>
<th>Gl</th>
<th>L</th>
<th>D</th>
<th>A</th>
<th>Cd</th>
<th>Wt</th>
<th>O</th>
<th>W</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Valid (present)
- Read/write
- Owner (user/kernel)
- Write-through
- Cache disabled
- Accessed (referenced)
- Dirty
- PDE maps 4MB
- Global

Reserved: 12 bits
Paging implementation – how does it really work?

- Where to store page table?

- How to use MMU?
  - Even small page tables are too large to load into MMU
  - Page tables kept in mem and MMU only has their base addresses

- What happens at context switches?
  - Switch page table by changing the value of the CR3 register to the base physical address of the page table
Paging Advantages

- Easy to allocate (physical) memory
  - Memory comes from a free list of fix-sized chunks
  - Allocating a page is just removing it from the list
  - External fragmentation not a problem
- Easy to swap out chunks of a program
  - All chunks are the same size
  - Use valid bit to detect references to swapped pages
  - Pages are a convenient multiple of the disk block size
  - More on swapping later
Paging Limitations

- Can have **internal fragmentation**
  - Process may not use memory in multiples of a page
- Memory reference overhead
  - 2 references per address lookup (page table, then memory)
  - Solution – use a hardware cache of lookups (next lec)
- Memory required to hold page table can be significant
Deep thinking

• Why does the page table we talked about so far have to be contiguous in the physical memory?
  ♦ Why did a segment have to be contiguous in memory?

• For a 4GB virtual address space, we just need 1M PTE (~4MB), what is the big deal?

• My PC has 2GB, why do we need PTEs for the entire 4GB address space?
How many PTEs do we need?
(assume page size is 4096 bytes)

• Worst case for 32-bit address machine?

• What about 64-bit address machine?

• Page size?
  ♦ Small page -> big table
  ♦ Large page -> small table but large internal fragmentation
  ♦ (Linux huge page)
Managing Page Tables

• How can we reduce page table space overhead?
  ♦ Observation: Only need to map the portion of the address space actually being used (tiny fraction of entire addr space)

• How can we be flexible?
All problems in computer science can be solved by another level of indirection.
All problems in computer science can be solved by another level of indirection

Butler Lampson

David Wheeler

but that usually will create another problem

– David Wheeler
Page Table Evolution

Virtual Address Space

Linear (Flat) Page Table

Physical Memory

Page 0
Page 1
Page 2
Page N-1
Page Table Evolution

Hierarchical Page Table

Virtual Address Space

Physical Memory

- Page 0
- Page 1
- Page 2
- Page N-1

Master

Secondary
Page Table Evolution

Hierarchical Page Table

Virtual Address Space

Physical Memory
Two-Level Page Tables

• Two-level page tables
  ♦ Virtual addresses (VAs) have three parts:
    » Directory (master page number), secondary page number, and offset
  ♦ Directory table maps VAs to secondary tables
  ♦ Secondary table maps VPNs to PFNs
  ♦ Offset indicates where in physical page address is located
Two-Level Page Tables

Virtual address

Dir | Secondary | offset

Directory table addr

secondary table

PFN | ...

PFN | ...

PFN | offset

Physical address
Two-Level Page Tables

- Example
  - 4KB pages, 4 bytes/PTE, 32-bit address space
  - How many bits in offset?
    - 4KB = 12 bits
  - Want directory table in one page, how many entries can we have in the directory page table?
    - 4KB/4 bytes = 1KB entries (each entry is a 32-bit address)
  - Hence, 1KB secondary tables. How many bits?
  - Directory (1KB) = 10, offset = 12, 32 – 10 – 12 = 10 bits left
    - One secondary table can host 4K/4bytes=1KB PTEs
    - 10 bits (inner) => exactly 1KB PTEs
Multiple-level page tables

```
<table>
<thead>
<tr>
<th>63</th>
<th>48</th>
<th>47</th>
<th>39</th>
<th>38</th>
<th>30</th>
<th>29</th>
<th>21</th>
<th>20</th>
<th>12</th>
<th>11</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>000...0 or 111...1</td>
<td>Page-map L4</td>
<td>Page-dir-ptr</td>
<td>Page-directory</td>
<td>Page-table</td>
<td>Page offset</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

- Page-map L4 base addr (CR3)
- Page-map L4 table
  - Page-mp entry
  - Page-map L4 pointer table
- Page-directory table
  - Page-dir-ptr entry
- Page-directory table
  - Page-dir entry
- Page table
  - Page-table entry
  - Physical address
    - Physical page frame number
    - Page offset
    - Main memory
Multi-level page tables

• 3 Advantages?
  ♦ L1, L2, L3 tables do not have to be consecutive
  ♦ They do not have to be allocated before use!
  ♦ They can be swapped out to disk!

The power of an extra level of indirection!

• Problems?
Efficient Translations

• Our original page table scheme already increased the cost of doing memory lookups
  ♦ Two lookups into the page table, another to fetch the data
  ♦ One lookup and one data access for original flat page table
• Now 4-level page tables require five DRAM accesses for one memory operation!
  ♦ Four lookups into the page tables, a fifth to fetch the data
• Solution: *reference locality!*
  ♦ In a short period of time, a process is likely accessing only a few pages
  ♦ Store part of the page table that is “hot” in a fast hardware unit
Next time

• TLB, swapping