Virtual Memory

- Last time
  - NachOS 2nd Project Overview
  - Please attend help session this Friday
  - Started Virtual Memory Management

- Today: finish virtual memory

Virtual Memory

- So far we have accepted that:
  - Memory references are translated dynamically (with hardware assistance) from logical to physical address
  - A process image (memory) may be broken up across multiple partitions (pages or segments)

- Virtual Memory is a memory system with the above two characteristics that also does not require that all the process’ memory partitions be in memory during execution

- Does this work? How? What do we gain?
Virtual Memory (Why?)

- What does having this scheme buy us?
  - More processes can be active at the same time (only a portion of each process is resident, so more processes can fit in memory) – is that good?
  - A process memory (address space) is no longer constrained to be smaller than physical memory because the restriction that the process’ memory be all resident is removed!
  - Less I/O is needed to swap a process/unswap it

Any reservations?

Revisiting Page Table Structure

- A page table per process
- With virtual memory, the pages of a process don’t all have to be in memory at the same time
  - What if the page is not resident?

Each page table entry has:

- A valid bit (is the page in memory?)
- A dirty bit (has the page been written to?)
- Other control bits (sharing information, protection information)
- Address of frame in memory where the page is (what if the frame is not resident?)
- Address in memory where the page is (what if the frame is not resident?)
- Where should the page table be kept? In hardware? In memory (real or virtual)?

Let’s get back to the problem of the size of the page table, is it helped or made worse by virtual memory?

Where should the page table be kept?

In reality, the page table only includes only a portion of the maximum entries

- Length of Page Table Register used to demarcate the end of the “used portion” of the address space

Example

- Virtual address space of 4 Gigabytes ($2^{32}$ bytes).  Page size is 1KBytes ($2^{10}$ bytes).  Page table entry is 4 bytes long.  What is the size of the page table?

- In reality, the page table only includes only a portion of the maximum entries

  - Length of Page Table Register used to demarcate the end of the “used portion” of the address space

Why does it work?

- Locality—main memory becomes a cache for the swap (hard disk)
  - What are the advantages of caching?
- Page faults are extremely costly (disk access is very slow)
- How many page faults can one instruction cause?
- Wouldn’t page faults occur all over since part of the memory is missing?
- We must minimize page faults—otherwise thrashing will occur
  - Control Load so that working sets fit
  - Be smart with page replacement policy
Hierarchical Paging

- Also known as Multi-level Page Table
- Idea – Page the page table
  - the page table is brought in as needed; as long as the needed portions of the page table are in memory, we are ok
- How do we find where the pieces of the page table are?
  - A Page Table directory is used to keep track of
    - Directory points to the part of the page table where the accessed page is
  - Directory is always resident in memory
  - Directory is always resident in memory
- Concept is a little difficult – two levels of indirection

---

Exercise: What is the maximum number of page faults on a memory access using this scheme?

Required memory to store the page table significantly reduced

Example – Two Level Page Table

<table>
<thead>
<tr>
<th>Outer Page Table</th>
<th>Inner Page Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>(always in memory)</td>
<td></td>
</tr>
</tbody>
</table>

Address Translation

- p1: first level (page directory)
- p2: page of the process image (data, program or stack)
- d: page of the page table

Example:

- Page Table
- Memory

Also Known as Multi-Level Page Table

- Idea – Page the Page Table
- Required memory to store the page table significantly reduced
- Example: take our 4 Gigabyte process example; what is the size of the page directory?
  - Hint: Each page (1 Kbyte), can store 256 page entries assuming each is 4 bytes

Alternative – Hashed Page Tables

- Idea: Use a much smaller page table
- Hash page table entries into the new table
  - Recall: hashing is a function that maps numbers randomly into a much smaller number range
  - Two or more numbers may hash into the same location
    - A technique such as chaining (shown above) can be used to address this case
    - Need to store the logical page number to find the correct entry

Inverted Page Tables

- Page table is potentially huge
- Each process has its own page table
- Can we index on the physical frames instead?
  - Each entry would point to the virtual page that is currently resident in the frame + which process it belongs to
  - A page hit occurs if there is a match on the virtual page number, as well as the process id
- What happens if page miss? Default to multi-level page table
- Problem: we need to search the full page table to detect whether there is a page hit – what to do?
- Solution, use a hashing function when selecting a free frame
- Used by PowerPC, PA-RISC/Alpha, and IBM RT

Discussion

- Recall: TLB was used to speedup translation
  - How is the TLB affected by virtual memory?
  - TLB miss – software or hardware?
  - What happens on a context switch; what happens on a page fault?
- Recall: caches used to speedup memory access
  - How are caches affected by virtual or physical address?
  - Are caches indexed based on virtual or physical address?
  - What happens on a context switch; what happens on a page fault?
General Flow of Memory Access

- Assume TLB; inverted page table; physically indexed cache; software TLB

1. Physically indexed cache: must translate address before we can check cache
2. Translation
   - Check TLB – TLB hit means translation ready
   - TLB Miss – trap to OS
     - Check Page table (depending on page table organization); is page in memory?
       - Yes – place entry in TLB (possibly replacing another)
       - No – page fault, bring page from disk, update page table and TLB
     - Redo Translation
3. Check cache
   - Cache hit, get data
   - Cache miss, access memory (address ready)

- What if cache is indexed on virtual address?

The Role of the OS

- So far, we have discussed the mechanics of virtual memory; what support does the operating system need to provide?
  - Fetch Policy: how to bring the pages of a process into memory (on-demand or pre-page?)
  - Placement/Replacement Policy: if all the frames are full, we need to swap a victim page out; how do we decide which one to replace?
  - Cleaning policy: when should a “dirty” page be saved (on-demand or pre-clean?)
  - Other?
    - Load control; remember the medium term scheduler?
    - How many resident pages should a process be allowed to have?

Fetch Policy

- Pre-Paging: bring pages in as they are needed
- Demand Paging: bring pages in as they are needed

Placement/Replacement Policy

- Does it matter where a page gets placed?
  - One exception is Shared memory machines
- Placement an issue with segments; need to find a “hole” big enough
- Why is replacement needed?
- Replacement: how to pick a victim! Are all pages fair game?
  - What if the OS gets replaced?
  - What if the “page directory” (or “Frame Table”) gets replaced?
  - What if an I/O buffer page gets replaced while DMA controller is writing to it?
- Frame-locking (aka Page pinning): OS is able to lock some pages so they are not replaced (using a lock bit)
Page Replacement

If:

1. Victim

(1) Swap out victim

(2) Change victim’s table entry to invalid

3. Swap in new page

4. Update table entry for new page

With virtual memory, can over-allocate the physical memory—why?

- Allow a bigger effective memory for each process
- Allow more processes to run concurrently
- Reduce loading/swapping times

Allow a bigger effective memory for each process—why?

- Reduce loading/swapping times

An Analogy to Help Think About Page Replacement

You are the owner of a hardware store

- Limited number of shelves in the store
- Must pick what types of goods to display on the shelves to maximize profit
- When a customer comes in and wants something that is not on display, you will lose money
- When you want to place a new item on the shelves, you have to remove an existing item

Replacement Policy: picking what item to remove to make room for a new item so that you wont hurt your future sales too badly

Optimal Policy

- Optimal Policy: pick the page with the maximum page reference time among all the resident pages (it will be referenced furthest in the future)

Retry our example with optimal policy: 7, 0, 1, 2, 0, 3, 4, 2, 3, 2, 1, 2, 0, 1, 7, 0, 1

Can this be determined realistically?

Need policies that will approximate this behavior

First In First Out

- Replace the page that has been in Memory longest

Implementation easy:

- Keep a linked-list (queue) of the pages
- When it is time to replace a page, remove the head of the queue (the oldest page is removed)
- When you want to place a new item on the shelves, you have to remove an existing item

Example: 3 frames, Reference string is 7, 0, 1, 2, 0, 3, 4, 2, 3, 2, 1, 2, 0, 1, 7, 0, 1

Belady’s Anomaly: sometimes, we can get more page faults with a higher number of available frames!!

Belady’s Anomaly: sometimes, we can get more page faults with a higher number of available frames!!

- Consider the reference string 1, 2, 3, 4, 1, 5, 2, 4, 5

- Is this a good idea? Do you think it will perform better than random?

- Example: 3 frames, Reference string is 7, 0, 1, 2, 0, 3, 4, 2, 3, 2, 1, 2, 0, 1, 7, 0, 1

- Belady’s Anomaly: sometimes, we can get more page faults with a higher number of available frames!!

- Consider the reference string 1, 2, 3, 4, 1, 5, 2, 4, 5

- Is this a good idea? Do you think it will perform better than random?
Aside/Hardware or Software support?

- What to implement in software, and what in hardware?
- Expensive to implement everything in hardware; might slow everything down (CS325 would explain that if you are interested in hardware)
- Only operations on the critical path (with every memory reference) must be in hardware
  - Otherwise, we need to waste several instructions with every memory reference (remember – 1.3 mem. references per instructions on RISC processors)
- Operations that happen infrequently (e.g., when a page fault occurs) can be supported in software