Virtual Memory

Tom Kelliher, CS 240

May. 1, 2002

1 Administrivia

Announcements

Homework, toolboxes due Friday.

Assignment

From Last Time

Introduction to caches.

Outline

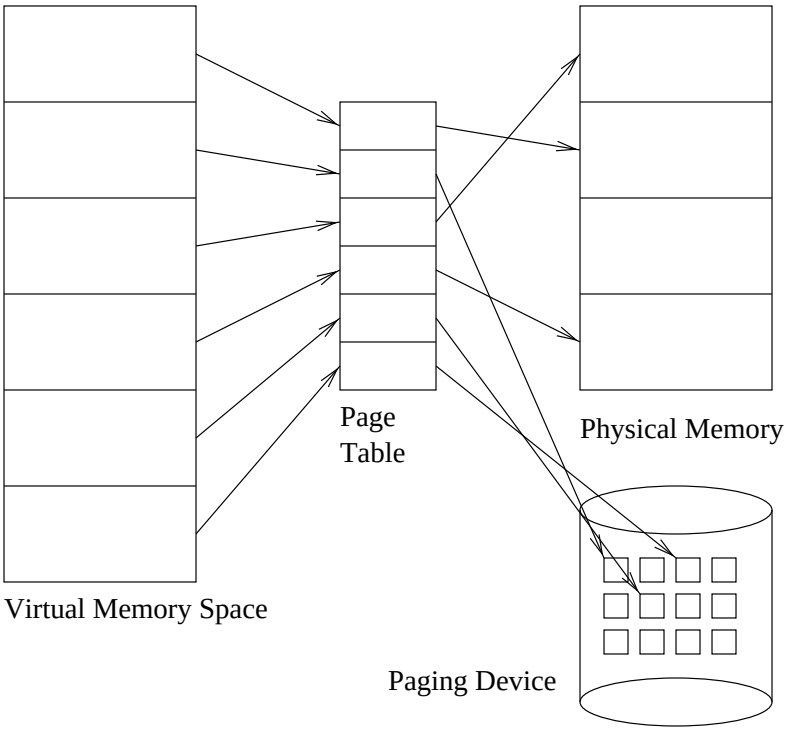
1. Virtual memory.
2. System support: kernel, MMU, CPU.
3. Page fault sequence.
4. VM performance.
5. Replacement policies.

Coming Up

Course feedback, review for final.

2 Virtual Memory

1. Virtual memory — what is it?



2. What are the advantages?

   (a) A program's logical address space can be larger than physical memory.

   (b) Degree of multiprogramming can be increased (40 pages of memory; allocate only 5 pages to processes with spaces of 10 pages).

   (c) Less I/O needed to load/swap a process.

3. Demand paging.

4. Why does it work?
    (a) A lot of code is rarely run (error-handling routines).
    (b) Oversizing of data structures.
    (c) Locality of reference:
        i. Spatial.
        ii. Temporal.

What we'll consider:

1. System support.
2. Page fault sequence.
3. Replacement policies.
4. Placement (allocation) policies.

3 System Support for Virtual Memory

1. Kernel support.
2. MMU support.
3. CPU support.

3.1 Kernel Support

1. Page fault handler.
2. Page placement policies.
3. Page replacement policies.

3.2 MMU Support

Page table changes:

1. Valid/invalid bit takes on greater role:
   (a) Valid: frame field contains frame number.
   (b) Invalid: frame field contains block number on paging device *or* invalid reference.
2. Reference bit.
3. Dirty bit: page has been modified.
4. Read/write bit.
5. Access counter: (sometimes, in place of reference bit) for implementing LRU.

Traps generated:

1. Memory fault.
2. Page fault.
3. Write on read-only fault.

3.3 CPU Support

Instructions must be restartable:

1. May page fault on instruction fetch.
2. May page fault on operand fetch/store. State may have been modified. Design approaches:
   (a) Checkpoint state.
   (b) Ensure that all required pages are in memory before proceeding.

4 Page Fault Sequence

1. Memory reference generated.
2. MMU looks up Page table entry (TLB then memory).
3. Valid bit examined. If set, get frame number and finish.
4. Otherwise, generate page fault trap.
5. Kernel page fault handler called as result of trap.
6. Kernel examines page table entry.
7. If non-mapped, generate page violation trap.
8. Otherwise, locate a frame for the incoming page (possibly designate a victim frame and page it out *first*).
9. Schedule disk I/O.
10. Re-schedule CPU.
11. Disk I/O completes, interrupt generated.
12. Kernel interrupt handler called and determines source of interrupt.
13. Update page table and put process back on ready queue.
14. Faulting instruction re-started.

4.1 Demand Paging Performance

$$\text{Effective access time} = (1 - p) \times ma + p \times \text{page fault time},$$

where:

1. p is the page fault rate.

2. *ma* is main memory access time. 100 ns or less.

3. Page fault time: 25 ms. or more.

1. If page fault rate is 1/1,000, effective access time is 25 microseconds!!!

2. If we want only a 10% penalty (110 ns), page fault rate must be less than 1/2,500,000.

4.2 Swap Space Policies

1. Copy image into swap at process start-up. Demand paging done from swap device. Wastes swap space, extra I/O, but swap device is faster than filesystem.

2. Demand page from filesystem. Read-only pages are never swapped out, just overwritten and re-read from filesystem. Conserves swap space, uses slower filesystem.

3. Demand page from filesystem, swap out to swap device. Only demanded pages are read from filesystem, only necessary pages are replaced to swap device.

5 Replacement Policies

1. What happens if a page fault occurs and all frames are in use?

2. Must select a victim frame:

   (a) Page-out victim frame.

   (b) Update victim process' page table.

   (c) Page-in faulted page.

3. How do we select the victim frame?

4. Comparison criteria for replacement algorithms.

5.1 Reference Strings

1. What is it?

2. Where do I get one?

3. What about redundancy?

5.2 FIFO Replacement

1. In the set of candidate victim pages, select the "oldest" page.

2. Example reference string: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1. Three frames allocated.

3. Belady's anomaly:

   (a) Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.

   (b) Three frames allocated.

   (c) Four frames allocated.

4. Stack property: Set of pages in memory with n frames allocated is a subset of set of pages in memory with $n + 1$ frames allocated.

5.3 Optimal Replacement

1. Replace the page which won't be used for the longest time.

2. Example reference string: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1. Three frames allocated.

3. Implementation?

5.4 LRU Replacement

1. Approximation to optimal: replace page which hasn't been used for the longest time.

2. "Reversal" of optimal.

3. Example reference string: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1. Three frames allocated.

4. Implementation:

   (a) Counters. Hardware support *required*

   (b) Stack. Expensive.

   (c) Reference bits concatenation as approximation to counter.

   (d) Second chance (clock) algorithm: FIFO, but skip over page if reference bit set (reset reference bit).

5.5 Page Buffering Optimizations

1. Keep a small pool of empty frames so paging-in can occur without waiting for victim page-out.

2. When idle, write dirty pages out and clear dirty bit.

3. Keep track of what's in free frames, so page-ins can possibly use an old, free frame.