Questions from Lecture?

Questions from the Project?

Questions from the Exam?

Questions from Homework?

- 9.11: what is the effect of have two entries in a page table point at the same physical page
  - not two entries in two page tables
- Aliasing
  - Same memory shows up twice
  - Changing a byte in one view changes the other
- Copying
  - Can implement copy by just remapping
  - Need to make both copies read-only, do copy when one changes to get copy semantics

Memory Protection

- 9.9: How do operating systems prevent processes from seeing other memory?
  - Common answer: valid/invalid bit
  - Correct answer:
    - Every address a process uses is translated by the page table
    - A process has no language to talk about memory other than its own
  - How can we implement this?
    - Share a physical page between two PTEs
    - Provide a system call to read/write memory in other processes
Project 3 - Virtual Memory
• Experiments
  – Have a hypothesis
    • “Big pages are better”
    • “Algorithm y is better”
    • “Prefetching will reduce the number of page faults”
    • “If we understand why x happens, we can fix it”

Running experiments
• Two steps
  – Control: what is the baseline?
    • What happens with existing page sizes / page replacement algorithms / no prefetching
  – New test: what happens with the new system
    • Try to change just one aspect of the system to isolate the difference

Some ideas
• What is the ideal page size for this trace under different amount of main memory?
• How much better is page replacement algorithm x than LRU
  – e.g. 2Q, ARC (currently best known techniques)
• How close can we come to LRU without doing any work between page faults?
  – e.g. no scanning, constant work per page fault
• How important is recency vs. frequency in predicting page re-use?

No so good ideas
• What kind of music is made when I set the convert the address trace to notes?
• Can I make a fractal out of this data?

Today’s Topic: Optional
• Memory Management in Linux
• High level:
  – 3 level page tables
  – common kernel heap allocator (“kmalloc”, “slab”)
  – Virtual memory areas
  – Physical memory regions

Page Tables
• PGD = top level pointer
  – Points to a page of PMDs (page middle directory)
• PMD = middle level
  – Can have just one entry for two-level hardware
  – Points to a page of PTEs
• PTE = page table
  – Translates a single virtual page into a physical page
Page Frame Database

- struct page {
  address_space * mapping;
  unsigned long index; // offset in mapping
  struct page * next_hash;
  struct page * prev_hash;
  unsigned long flags;
  atomic_t count; // usage count
  struct list_head lru; // pageout list
  void * virtual; // virtual address of this page in kernel memory map
  struct buffer_head * buffers;
}

Page Flags

- PG_locked
- PC_error
- PG_referenced
- PG_uptodate
- PG_dirty
- PG_unused
- PG_lru
- PG_active
- PG_slab
- PG_highmem
- PG_checked
- ...

Physical Memory Zones

- ZONE_DMA - used for DMA to antique devices, < 16 MB
- ZONE_NORMAL: 16-896 MB
- ZONE_HIGHMEM: > 896 MB

Linux Memory Picture

Page Replacement

- Triggered by low memory during allocation
- kswapd daemon also triggers swapping
  - wakes up every second when few pages available
- try_to_free_pages() finds pages to release (many)
  - walks through each memory type and asks it to release some pages

Page replacement (2)

- Two lists of memory
  - active pages: currently actively in use
  - inactive pages: candidates for swapping
- Pages are moved from active to inactive using clock algorithm
  - active list scanned for non-referenced pages
  - unused pages put on inactive list
Page replacement (3)

- Inactive list scanned for pages to swap out
  - Put pages that have been referenced back on active list
  - Leave behind pages locked for I/O
  - Try to free buffers in buffer cache using the page
- Keep swapping out pages until enough have been freed

Issues

- How do you check reference bits on shared pages?
- What happens if the swapper needs to allocate memory?
- What happens if part of the swapper code is swapped out?

Slab Allocator

- Goal:
  - Locality
  - Fast
  - Keep similar types together
  - Can create a heap for a specific type
  - Non-blocking synchronization
  - Uses atomic instructions instead of locks

Slab Allocator Design

- struct page knows what cache holds memory
  - Makes free easy - no need to specify/locate cache
- Heaps allocate from chunks called “slabs”
  - 3 types:
    - fully used - not used for allocation
    - partially used - used first
    - empty - reserved until no partially full
  - Can span multiple page sizes
    - optimized to reduce wastage

Page allocator

- Used for requesting contiguous chunks of physical memory
  - Can specify zone (DMA, normal, highmem)
  - Only allocates powers of 2 pages
  - Used for I/O - need contiguous physical memory or for single pages
- Uses buddy allocator
  - coalesces adjacent pages into powers of 2
- Triggers swapper when memory runs low
  - Can wait until memory is available

Virtual Memory Areas

- Used when contiguous virtual address is needed
  - e.g. allocating memory for dynamically loaded kernel code
- Kernel maintains sorted linked-list of areas in use
  - traverses list to find free space
  - allocate PMT/PTE/pages for the space
Page Fault Handler

- in arch/i386/mm/fault.c: do_page_fault()
  - Looks up faulting address in virtual memory area list to see what kind of memory it is
- Checks common fault cases:
  - usermode/kernel mode
  - write protect/invalid
  - stack growth
  - lazy synchronization of page table
- Resolutions:
  - call handle_mm_fault
  - send SIGSEGV
  - panic()