Michael Kester

Left off at incurring pagefault and jumping instruction

Next attack: memset welcome's code to 0s

- Both processes pagefault, then general protection fault (likely happens once you step off page of 0 instructions)
- Potential solution: memory map welcome's code to make it inaccessible by user
- Now welcome can't read its own code either (problem arises because of global pagetable)
- Create static separate 2-level pagetable for hello
    - Memory is small enough for first level pagetable to use only first entry (must have all permissions)
    - Rest of entries are set to 0
    - Second level page table is restricted to be inaccessible by user processes except for portion that hello should be able to access (e.g., its own code)

We have only been using pagetables to manage permissions so far

- Can also give each process point of view that it has all of memory accessible to itself
    - Map position x in process 1 to some physical place and position x in process 2 to different physical place

Example:

- VA: 0x00408050
- PA: 0x11111050
- %cr3 is a control register: holds pointer to current process's first-level pagetable (0x10000000)
    - Array looks like [0, 0x03000000|PTE_P|PTE_W|PTE_U, 0]
- PTL1: 004 (highest 10 bits of VA, including highest 2 bits of 4)
- PTL2: 408 (next 10 bits of VA, including lowest 2 bits of 4)
- Off: 050
- Store 0x11111000|PTE_P|PTE_W|PTE_U in address 0x03000020
- Can't map VA: 0x00408050 to PA: 0x00408052
    - Page table only works on higher order bits, never messes with offsets (…050 only maps to …050)

Process can also think it has full control over the CPU

- Kernel keeps track of process descriptor holds all running state information of a running process
- Saves program counter, stack pointer, and registers when control shifts to different process and reloads data when control comes back to process

os02

- 3 processes: allocator, recursive process that does recursive calculation, and simple fork (later lecture)
- System calls: almost directly analogous to system calls
    - Put arguments in registers, expect return value in %eax, but cause interrupt which switches control to kernel to run call for user
- Allocator: allocates page that is writeable and readable (always runs when operating system boots)
    - Problem: data misaligned at allocated address (get assertion panic)
    - Solution: kernel can check for misaligned address arguments (system call returns -1)
    - Problem: pass addresses in the kernel itself (operating system continuously reboots) – triple fault
        - Trap into kernel and interrupt is single fault
        - Caught by handler, if handler has problem, then double fault
        - If double fault handler faults then computer gets reset (triple fault)
    - Solution: check in kernel that address is in user memory
- Recursive function
    - Computes sum from 1 to n (n(n+1)/2)
    - Called to print sums for n up to 999
    - Contains helper function so compiler won't inline stuff
    - Stack overflow
        - Pagefault in this case (ran out of allocated memory and tried to step over)
    - Solution: when pagefault is taken, allocate new page and continue running (trapping mechanism in handling allows kernel to experience error, fix it in kernel, and restart process)
    - Potential problem: might not always want to allocate page when taking a pagefault

Brian Kate

Grad student of Eddie's

- Worked on project over the summer
- Systems research is alive and well!

Browser

- Makes data connection to server where content is being hosted
- Web server will run code that constructs page, perhaps by contacting data storage that will give you back results
- Web server is almost like processor, data storage is almost like disk

Persistent data storage is slow

- Might have to read from disk
- Can employ caching to speed up this process
    - Store bunch of data in RAM and give quick access to it
    - Reduce load on datastore
    - Improve read performance
        - Higher throughput (handle more users at once)
        - Lower latency (faster response to user)
- What/when to cache?
    - When reads are repeated
        - Whole pages (IRS, Wikipedia)
        - Fragments (Reddit)
    - When reads are predictable
        - Computed results (Facebook)
        - Aggregates, summaries (Google Analytics)

How can we use caching to help construct Twitter timelines faster?

- Follow someone
- Post new tweet
- Check your timeline (read tweets from all people that you follow)
    - Grab all recent tweets from people you follow, sort, then put into timeline

Twitter workload

- 150M active users
- 3-6K new tweets per second (20K+ peak)
- 100+ timeline checks per new tweet

Caching idea

- Cache as many recent posts as possible
    - Also need to cache follows list
- Client collects posts and assembles timeline
- Too slow to do on every timeline check
    - Each user follows 100 others
    - What if there are no updates to read?
    - Work scales up linearly with users
- How does timeline work?
    - Anything repeatable? (no)
    - Is it predictable? (yes)
    - Is anything computed? (yes)
    - Is it expensive to compute? (yes)

Good idea

- Optimize for reads by doing more work on writes
- Keep an updated, cached timeline per user
    - On write, append tweet to each follower's timeline
- For timeline check, just read from cache
    - In case of cache miss, construct timeline as before
- Only 4x slower for 100x more active users
- Doing more work up front (pushing tweets even when few people are reading timelines), but faster with more active users

Research idea

- Generalization of Twitter's cache
- Allow cache to generate new data from existing
    - On first access, generate anew
    - Maintain derived data incrementally
- Optimize for predictable reads
    - Social media "streams"
    - Aggregates of high-rate data
- Goal: better performance and simpler code

Pequod

- Key-value cache with additional features

    - Range scan primitive
    - Cache joins
        - Relate input data (sources) with generated data (sinks)
        - Specify operation for generating data (copy, filter, aggregate)
    - Example: Twitter
        - Compute timeline from follows and posts
        - To tweet, write new post to cache
        - To check timeline, scan the timeline range
- Hybrid between the two aforementioned types of caches (pull and push)