Solutions for the Second Quiz

COSC 6360
Fall 2017
First question

- What characterizes a **self-tuning** cache replacement policy?

- Which feature(s) of the ARC cache replacement make that policy self-tuning?
First question

- What characterizes a self-tuning cache replacement policy?
  - *It does not require any workload-dependent adjustments.*

- Which feature(s) of the ARC cache replacement make that policy self-tuning?
  - *It has no user-tunable parameter.*
Alternate first question

- What characterizes a *scan-resistant* cache replacement policy?

- Which feature(s) of the ARC cache replacement make that policy scan-resistant?
Alternate first question

- What characterizes a *scan-resistant* cache replacement policy?
  - *Pages that are only accessed once are expelled faster than other pages.*

- Which feature(s) of the ARC cache replacement make that policy scan-resistant?
  - *ARC maintains a separate list of pages that have been accessed once.*
Second question

- What problem do Corey *kernel cores* address?

- How do they solve that problem?
Second question

- What problem do Corey kernel cores address?
  - In most OSes, system calls are executed on the core of the invoking process
  - Bad idea if the system call needs to access large shared data structure

- How do they solve that problem?
  - Kernel cores let applications dedicate cores to run specific kernel functions
    - Avoids inter-core contention over the data these functions access
Alternate second question

- What problem do Corey *address ranges* try to solve?
  -

- How do they solve that problem?
  -
Alternate second question

- What problem do Corey *address ranges* try to solve?
  - *Current solutions do not let the cores of multicore applications access both shared and private data in an efficient fashion.*
- How do they solve that problem?
  - *They let applications define both shared and private address ranges in their address spaces.*
Third question

- What *must happen* before Proof Carrying Code becomes widely used?
Third question

- What *must happen* before Proof Carrying Code becomes widely used?

  - *We must find a cost-effective way to construct safety proofs for non-trivial extensions*
Fourth question

- Consider a hypothetical 8-way associative L2 TLB with 2,048 entries

- What would be its coverage for a page size of 4 kilobytes?
Fourth question

- Consider a hypothetical 8-way associative L2 TLB with 2,048 entries

- What would be its coverage for a page size of 4 kilobytes?

  - $2K \times 4KB = 8MB$
Alternate fourth question

- Consider a hypothetical 4-way associative L2 TLB with 1,024 entries

- What would be its coverage for a page size of 4 kilobytes?
Alternate fourth question

- Consider a hypothetical 4-way associative L2 TLB with 1,024 entries

- What would be its coverage for a page size of 4 kilobytes?

- $1K \times 4 \text{KB} = 4 \text{MB}$
Fifth question

- What do Navarro et al. propose to do whenever a process attempts to modify a superpage?
  - Why?
    - Why?
Fifth question

- What do Navarro et al. propose to do whenever a process attempts to modify a superpage?
  - Whenever a process attempts to modify a superpage, that superpage is demoted and replaced by its constituent base pages.

- Why?
  - To avoid having to flush back the whole superpage when it will be expelled from main memory.
Sixth question

- How does Nooks *recovery* from an extension failure?

- What is the *main limitation* of this approach?
Sixth question

- How does Nooks *recover* from an extension failure?
  - *It restarts the extension.*

- What is the *main limitation* of this approach?
  - *It does not work for all extensions.*
Seventh question

- What is the major performance penalty occurring when Nooks crosses a lightweight protection domain boundary?
Seventh question

What is the major performance penalty occurring when Nooks crosses a *lightweight protection domain boundary*?

- Crossing protection boundaries requires switching the kernel page table, which results in a flush of the current TLB (and an avalanche of TLB misses).