10/13 CS240 - Superpages

Announcements

For next class (Thursday 10/15)

1. Read: A Comparison of Software and Hardware Techniques for x86 Virtualization
2. No reading questions

Looking ahead:

In-class **Midterm Exam next week** (Tuesday, 10/20)

Open notes (i.e. printed material). No electronics.
Material through Thursday (10/15) lecture
Similar to sample exams on course website

Lab 1 is due end of day November 1st.

Don't use AI for reading questions.

Paper background

- OSDI 2002 - 5th Symposium on Operating Systems Design and Implementation
  - FreeBSD - open source OS helps with OS research
  - Alpha 21264 CPU - Supports multiple page sizes: 8K and 8x multiples
    - 8KB, 64KB 512KB, 4MB, 32MB, 256MB, 2GB, 16GB
      - Page alignment restrictions, coloring.
    - Software-reloaded TLB - PALcode firmware
    - 40 bit Virtual Address (43 bit in arch spec) -> 44 bit Physical Address (48 in arch spec)
      - Half of virtual address given to kernel
- Review:
  - Address spaces - Memory objects
    - Kernel, code, heap, stack, shared libraries,

What are the advantages and disadvantages of page sizes?

- Minimum (base) page sizes:
  - DEC Vax (1978): 512 bytes, typical memory size: 512 KB - 1MB
  - Intel x86 (1985): 4 KB, typical memory size 1 - 2 MB
  - ARM (1990): 4 KB, typical memory size 512KB - 1 MB
  - DEC Alpha (1992): 8 KB, typical memory size 16 – 32 MB
  - Apple ARM (2025): 16 KB, typical memory size 16 - 32 GB
- What's hard about multiple page sizes?

What does Transparency mean in this paper?

- Transparent to whom?
  - Application programmer?
  - Kernel programmer?
  - Firmware programmer?
- Let's sketch out a non-transparent solution and map on to paper's issues:
  - Allocation
  - Fragmentation control
  - Promotion
  - Demotion
  - Eviction

Allocation

Challenge: Got a page fault but can't know the future

Trade Offs:

- Relocation
- Reservation

What is a buddy allocator? What did it do before?

Preferred superpage size? What about stacks and heaps?

Multi-list reservation scheme? Sort order?

Population map function?

Fragmentation control

What? How?

Promotion

Only when fully populated at the small page size. Why?

Demotion

When?

Eviction

- Why not page out superpages?
- How did the suggest hash digests could be used here?

Other VM subsystem changes

- Made page out daemon do contiguity enhancement
- Recall that `get_free_pages()` and `MmProbeAndLockPages()` from ESX paper?

```
mmap(size_t length; void addr, size_t length, int prot, int flags, int fd, off_t offset)
      addr == NULL, kernel chooses
      MAP_FIXED in flags,
```

- File closes move pages into inactive list?

Detailed experimental results

- Plentiful memory: minimal overheads from superpages
  - SPEC CINT2000 - Speed up 11.2% (0% eon - 68% mcf)
  - SPEC CFP200 - Speed up 11% (-1.5% mesa - 82.7% apsi)
  - Disappointments:
    - Web, 315MB size: 1.9%
- Challenges for running for a long time

Nice: Adversary applications

- Incremental promotion overhead
- Sequential access overhead
- Preemption overhead

Modern systems

- Except for a few embedded CPUs, hardware reloaded TLBs are used
  - Page table format dictated by hardware
- Multiple page tables supported by truncating page table walk
  - X86_64: 4 KB, 2 MB, 1 GB
  - ARM64: (supports base page sizes of 4K, 16K, or 64K) 3 levels of PT
    - Example: Apple 16K, 512 KB, 32MB
  - RISC-V: Same as X86_64
  - CPUs can count TLB misses but not point OS at the pages
- OSes
  - Linux: Transparent and explicit large pages (4K, 2 MB, 1 GB)
  - Windows: Explicit only (4K, 2 MB typical, 1 GB optional)
  - MacOS: No explicit. Some internal use of large pages

Random issues

Address Space Layout Randomization (ASLR)

Guard pages

Reading questions

1. Table 2: What is weird about FFTW? What type of access pattern must it be doing?
2. An implicit but overriding principle of the superpages paper is *primum non nocere*[1] in that they try to never be worse than the base system. Give two examples of choices they made that satisfy this principle and one example of a choice that does not.