RadixVM

Scalable address spaces for multithreaded applications

Austin T. Clements,
M. Frans Kaashoek,
Nickolai Zeldovich

Presented by Simon Pratt

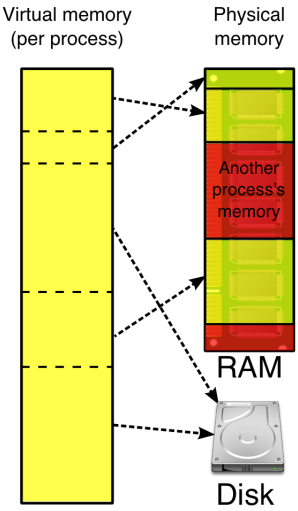
February 12, 2016

Abstract

RadixVM is a virtual memory (VM) design that attempts to increase multithreaded scalability by:

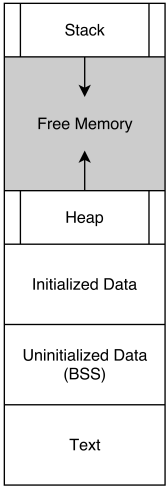
- Storing VM information in a radix tree
- Counting references to memory addresses
- Reducing inter-core virtual address invalidation (remote TLB shootdown)

Background: Virtual Memory



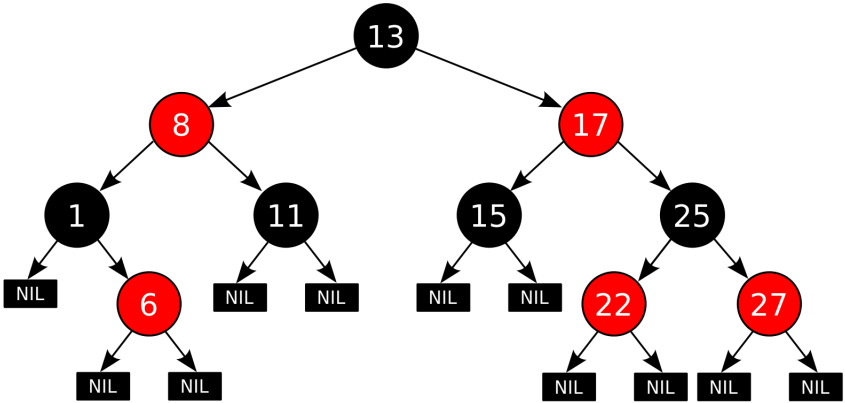
- Maps a contiguous virtual address space to:
  - physical memory (frames)
  - disk (swap)

Background: `malloc` and `mmap`



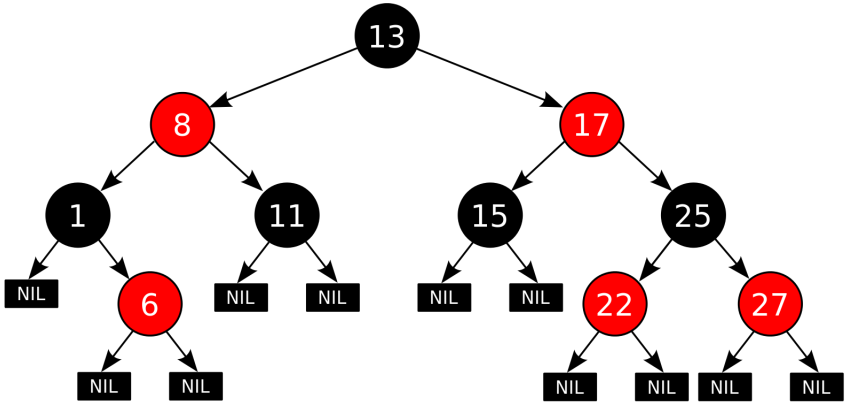
- `malloc` and `free`
  - User-level library function
  - Allocates/frees space in virtual memory
  - Often implemented using `mmap` and `munmap`
- `mmap` and `munmap`
  - System calls
  - Actually allocates/frees space in virtual memory

Background: Linux Virtual Memory



- Red-black tree
- Allows the kernel to search for memory area covering a virtual address

Background: Linux Virtual Memory



- Red-black tree
- Allows the kernel to search for memory area covering a virtual address
- Problem: A single lock per address space!

Design: High-level

RadixVM has 3 parts:

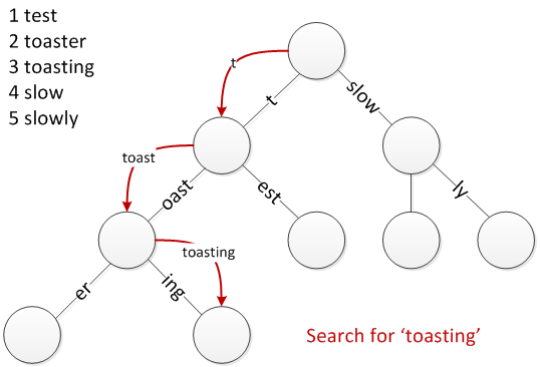
- Refcache
- Radix-tree-like data structure
- Targeted TLB shootdowns

Design: Refcache

TODO

- Counts references to memory locations
- Divides time into epochs
- Frees memory when a memory location is unreferenced for an entire epoch

Background: Radix Tree



- A.K.A. prefix tree
- Edges labeled
- Concatenation of edge labels along root→node path gives a string
- In OSes, usually strings of bits

Design: RadixVM Data Structure

TODO

- Similar to a radix-tree
- Fixed-height
- Each level indexed by up to 9 bits

Background: Remote TLB Shootdowns

TODO

When a shared memory location is unmapped:

- The TLB for every core is flushed
- This is expensive!

Design: Targeted TLB Shootdowns

TODO

- Store metadata on which cores may have address in TLB
- Only flush TLBs on cores which may share that memory

Design: Do we need all 3 pieces?

TODO

- TODO

Implementation

- Implemented on xv6
  - Academic OS
  - Based on v6 Unix
  - Rewritten in ANSI C for x86
  - `https://pdos.csail.mit.edu/6.828/2014/xv6.html`

Application: Metis

TODO

- MapReduce Library
- Single-server
- Multithreaded
- Stresses concurrent `mmap`s and `pagefault`s, but not concurrent `munmap`s
- Compiles on xv6 and linux

Microbenchmark: Local

TODO

- `mmap` a private 4KB region in shared address space
- Write to every page in region
- `munmap` region

Microbenchmark: Pipeline

TODO

- Each thread `mmap` a region
- Write to every page in region
- Pass region to next thread
- Write to every page in passed region
- `munmap` region

Microbenchmark: Global

TODO

- Each thread `mmap` a 64KB region within a large region of memory
- All threads access all pages in random order

Memory Overhead

TODO

- TODO

Summary

TODO

- TODO

References

- Linux VM info from:
  `http://duartes.org/gustavo/blog/post/how-the-kernel-manages-your-memory/`

Attribution

- Virtual memory diagram by Ehamberg (Own work) [CC BY-SA 3.0 (http://creativecommons.org/licenses/by-sa/3.0) or GFDL (http://www.gnu.org/copyleft/fdl.html)], via Wikimedia Commons
- Address space diagram by Majenko (Own work) [CC BY-SA 4.0 (http://creativecommons.org/licenses/by-sa/4.0)], via Wikimedia Commons
- Patricia trie diagram by Saffles (Microsoft Visio) [GFDL (http://www.gnu.org/copyleft/fdl.html) or CC BY-SA 3.0 (http://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons