MADV_LAZYFREE

“rcu_free” for user memory

Rik van Riel

MADV_DONTNEED overhead

TLB flush CPU overhead is … bad

- In MADV_DONTNEED calling thread: 50% of CPU time spent on TLB flush
- Add in TLB flushing on other CPUs: **90-97%**
- >90% of TLB flushes from MADV_DONTNEED & MADV_FREE
- Numbers with 10s of CPUs, newer systems have 100s of CPUs

RCU provides inspiration

- Make data inaccessible
- Users go away over time
- Freeing is cheap without users left

Userspace API

API for use by malloc libraries and runtimes. API was simplified in collaboration with Jemalloc developers. Most user programs never need to touch this directly.

MADV_LAZYFREE

- Mark a memory range (start, end) for lazy freeing
- Program needs to ask the kernel before reusing the range

MADV_REUSE

- Make a previously MADV_LAZYFREEd virtual memory range (start, end) available for reuse
- Return values (open to better ideas)
  - 0: memory is available for reuse, the process has other MADV_LAZYFREE ranges
  - ENOMEM: memory is available for reuse, no more MADV_LAZYFREE ranges present

MADV_LAZYFREE pseudo-code

madvise(..., MADV_LAZYFREE)

- Kernel can discard data in this memory range, on its own schedule
- Advance mm_struct tlb_gen number
  - Currently in arch/x86, move to generic code?
- Store MADV_LAZYFREE memory range
  - start, end, tlb_gen
  - Some tree attached to mm_struct
  - Link mm into MADV_LAZYFREE range shrinker
- Mark PTEs non-present
  - PMD_NONE, or some swap type like migration?
  - Keeps CPUs from loading valid page mappings for this address range

MADV_REUSE pseudo-code

madvise(..., MADV_REUSE)

- Data in this range will be preserved after MADV_REUSE returns
- Mark PTEs present again
  - Pages may have been reclaimed already (nothing to do)
  - Pages might still be around if memory pressure is low
- Remove MADV_LAZYFREE memory range from mm tree
- No need to flush TLBs
  - If CPU tlb_gen >= MADV_LAZYFREE tlb_gen, it has no mapping for this range
  - If CPU tlb_gen < MADV_LAZYFREE tlb_gen, there was no reclaim, and some of the same pages might still be mapped

MADV_LAZYFREE TLB flush requirements

No TLB flushes in MADV_LAZYFREE or MADV_REUSE themselves.

The TLB needs to be flushed on reuse, if:

- The virtual memory range is reused for something else
- A physical memory page is freed

If reuse is “far enough” into the future, the TLB flushes can happen naturally at context switch time, and we maybe able to avoid sending IPIs.

Context switch pseudo-code

Piggyback on existing lazy TLB code

- Check whether CPU tlb_gen < mm_struct tlb_gen
  - If CPU tlb_gen is behind, flush the TLB
- The lazy TLB code on x86 already does this today
- MADV_LAZYFREE expands lazy TLB mode, minor change needed

Reclaim & migration pseudo-code

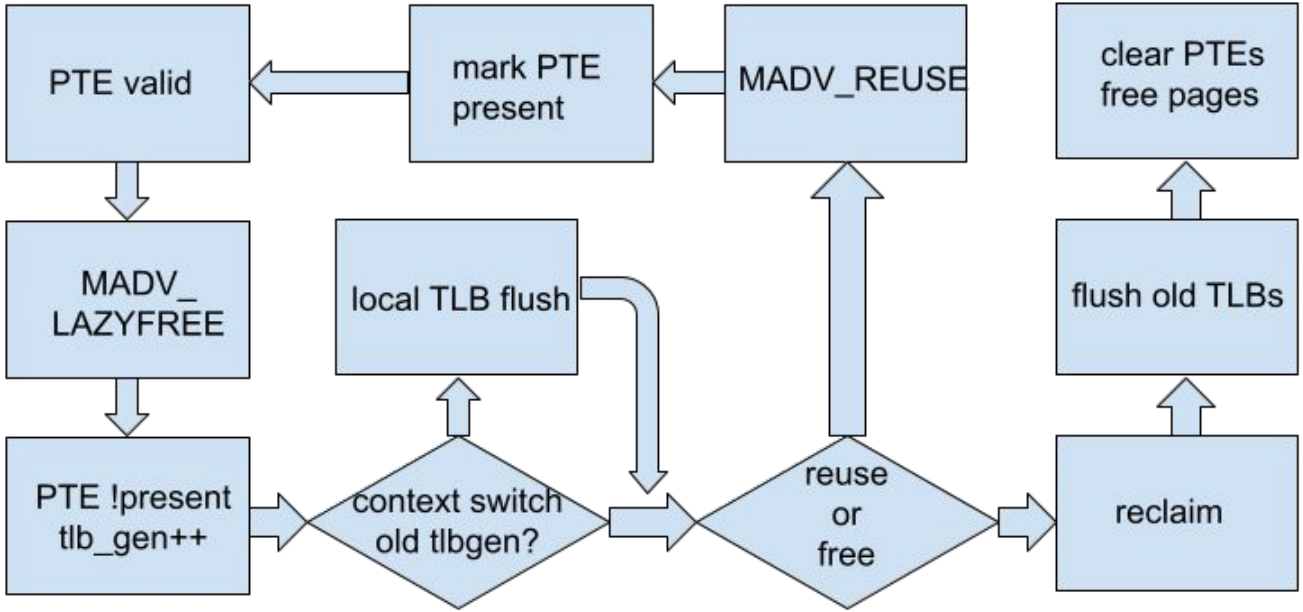
- Check whether page is part of MADV_LAZYFREE range
- Look up tlb_gen of MADV_LAZYFREE range
- Clear (already non-present) PTE
- Iterate over CPUs with mm loaded
  - CPU tlb_gen < MADV_LAZYFREE tlb_gen? Send TLB flush IPI
  - CPU tlb_gen >= MADV_LAZYFREE tlb_gen? Nothing to do
- Free page

MADV_LAZYFREE potential optimizations

Potential optimizations

- Keep a small buffer of flushed ranges
  - Do partial flushes at context switch time, instead of full flushes
  - Gives 1-2% IPC improvement in tests (without MADV_LAZYFREE)
  - Effect reduced when system has less idle time
- Use lazy freeing for munmap() too?
  - Reduce mmap_lock contention
- Integrate with Byungchul Park's Lazy Unmap Flush project?

MADV_LAZYFREE lifetime rules



MADV_LAZYFREE

Questions?

Suggestions?

Ideas?