Speculative page faults

Page Faults:

- Are normally executed under mmap_read_lock().
- Often need to allocate data pages,
  which might be slow depending on memory pressure.
- Thus, might hold up other threads that try to modify the VMA tree
  (i.e. creating additional mappings during application startup),
  even if the changes do not overlap with the fault address.

Speculative Page Faults:

- Try to execute without taking mmap lock in the common case
- Mmap writers are thus allowed to execute concurrently
- Behave as a transaction:
  - If the transaction succeeds, it must atomically commit its results (i.e. insert the new data page into the page tables)
  - If the transaction fails, either because of an interfering writer or because of hitting an unhandled case, it must be aborted before making any externally visible changes. The fault can then be retried using the usual, non-speculative code path.

Page Fault:

mmap_read_lock()
    find_vma()
    validate access permissions
    handle_mm_fault()
        (walks page tables, locates PTE, gets original PTE value)
    handle_pte_fault()
        (resolves fault based on VMA, original PTE, access type)
mmap_read_unlock()

handle_pte_fault(): do_anonymous_page() case

- original PTE is pte_none
- VMA is anonymous

Allocate a data page (or use zero page if access is a read)
Lock page table
    Verify the pte is still pte_none (avoid concurrent fault races)
    Set PTE to point to the new data page
Unlock page table

Implementing speculative page faults

Detecting interference with mmap writers:

- Done using a per-mm sequence counter
- Writers increment it to an odd value when taking mmap lock, increment it again to an even value when releasing it.
- This is added in mmap_write_lock() and mmap_write_unlock(), thus ensuring all lock sites are accounted for.

Implementing speculative page faults

Mmap writers might free vmas:

- Change VMA freeing to be RCU safe
- Make sure speculative page fault paths access VMAs under RCU read lock.
  If the sequence count is unmodified at the start of the RCU section, the VMA can be referenced until the end of the RCU section.

Implementing speculative page faults

Lock free VMA lookup:

- The existing find_vma() is called under RCU read lock.
- The mmap sequence counter is checked before and after looking up the VMA.
- The lookup succeeds if the counter is unchanged.
- If the counter changed, the fault is retried non-speculatively.

Implementing speculative page faults

Mmap writers might free page tables:

- Speculative page faults can protect against that using either RCU (in the CONFIG_MMU_GATHER_RCU_TABLE_FREE case), or by disabling local interrupts
- This is similar to what fast GUP already does.

Implementing speculative page faults

Atomically committing the page fault results (end of transaction):

- Existing code already takes page table lock to insert PTE
- Verifying the mmap sequence counter after taking the PT lock turns out to be sufficient.

handle_pte_fault(): wp_page_copy() case

- original PTE is present but not writable
- access is a write

Lock page table
    Verify the pte is still unchanged
    See that we must copy the original data page
    Take a reference on the original data page
Unlock page table
Allocate new data page
Copy from old to new data page
Lock page table
    Verify pte still unchanged
    Set PTE to point to the new data page
Unlock page table