Lab 5 Design Doc: Copy-on-write

Overview

The goal of this lab is to implement a copy-on-write mechanism in fork, to avoid duplicating unnecessary memory when forking a process.

Major Parts

Change fork: The child process is given a page table that points to the same memory pages as the parent, and the PTE of both processes are changed to be read-only.

Copy-on-write: If either process tries to alter the content of their memory, the kernel will make a copy of the memory page.

In-depth Analysis and Implementation

Copy-on-write

kernel/vpmap:vpmap_cow_copy

- Update parent's page table entry permission to be read-only.
    - Bitwise set bits of PTE_W to 0.
- Set the child PTE to the same content as the parent's.
- Increment the reference count for each physical page (pmem_inc_refcnt(PTE_ADDR(addr))).

kernel/mm/vm:memregion_copy_internal

- Call vpmap_cow_copy instead of vpmap_copy.
- Call vpmap_flush_tlb() at the end.

kernel/pgfault:handle_page_fault

- If fault_addr is present, write is 1 and mem region is valid with write permission:
    - Allocate physical page
    - Copy data from the copy-on-write page.
    - Set the permissions of the virtual page to be read and write (vpmap_set_perm).

    - Get the physical address (vpmap_lookup_vaddr using pg_round_down(virtual_addr))
    - Decrement the reference count of the read only page (pmem_dec_refcnt)
    - Return

Risk Analysis

Unanswered Questions

- How do we change the physical address of the parent's PTE because it is not a pointer?

Staging of Work

First, we will implement the copy-on-write mechanism in vpmap_cow_copy(). Then we will update memregion_copy_internal() to use vpmap_cow_copy() instead vpmap_copy(). After that, we will update page_fault_handler() to incorporate copy-on-write

Time Estimation

- vpmap_cow_copy (1-2 hours)
- Memregion_copy_internal (1 minute)
- handle_page_fault (30 min - 1 hours)
- Edge cases and error handling (5-7 hours)