WEEK10 WIL

6조 정재혁 유흥국 남청우

PROJECT3 VM: Copy-on-write

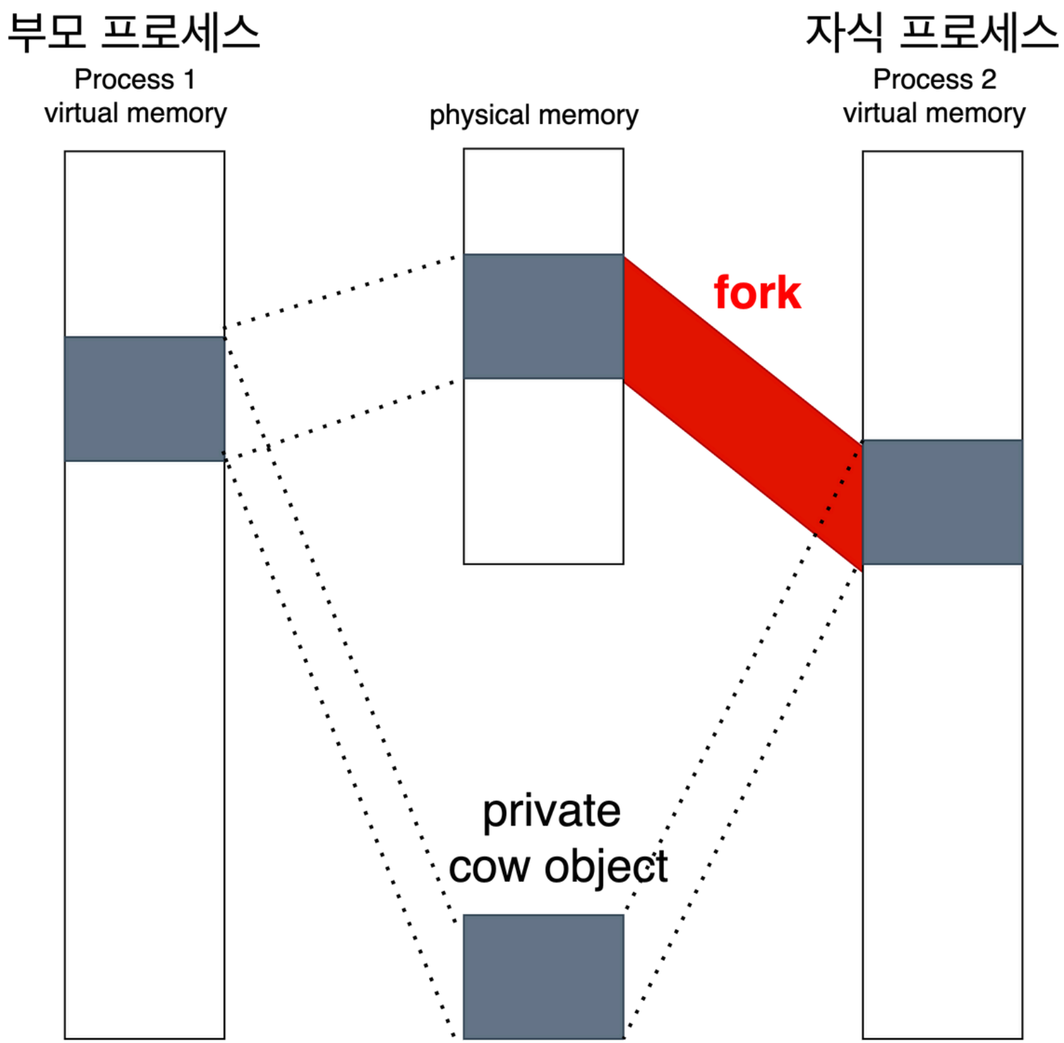
목차

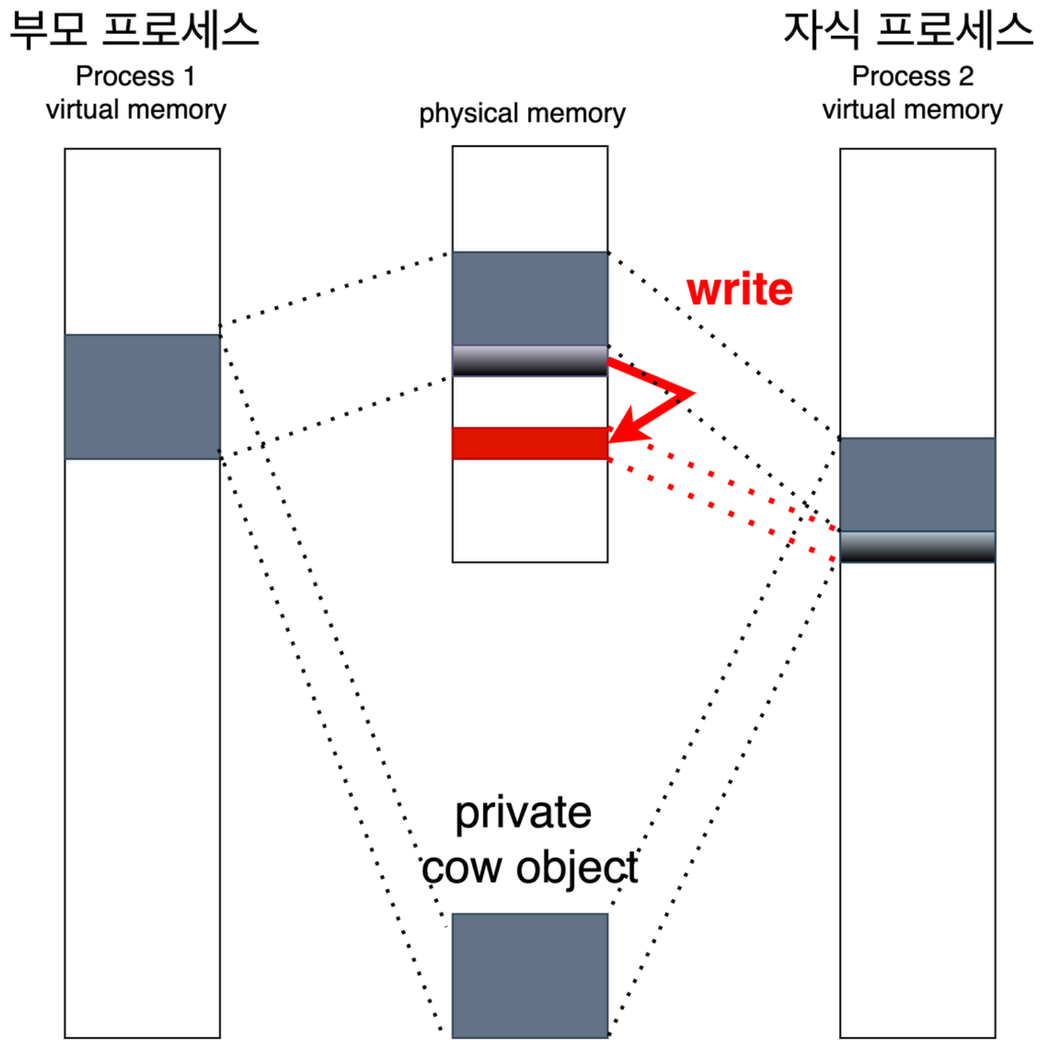
정의

Copy-On-Write (COW)는 **동일한 물리 페이지 인스턴스**를 이용해
더 빠른 복제 작업을 가능하게 하는 리소스 관리 기술입니다.

여러 프로세스가 동일한 리소스를 사용할 때, 충돌을 방지하기 위해 각 프로세스에
고유한 복사본이 필요하지만, **리소스가 읽기 전용인 경우에는** 여러 복사본이 필요하지 않습니다.
COW는 **리소스가 수정될 때만 복사본을 생성**하여 메모리 효율성을 높입니다.

Copy-on-write 개념





accessible 추가

쓰기 시도를 탐지하기 위해서 쓰기 보호 매커니즘이 필요

- write를 시도하면 page fault를 발생시키기 위해 **fork**시 **writable**을 **false**로 복제
- 대신 **새로운 속성(accessible)**을 만들어서 수정 가능한 page인지 구분

```
struct page {
    ...
    /** Project 3: Memory Management - Your implementation */
    struct hash_elem hash_elem;
    bool writable;
    bool accessible; /** Project 3: Copy On Write (Extra) */
    ...
};
```

supplemental_page_table_copy 수정

기존 코드

```
case VM_ANON:                                   // src 타입이 anon인 경우
    if (!vm_alloc_page(type, upage, writable))  // UNINIT 페이지 생성 및 초기화
        goto err;

    if (!vm_claim_page(upage))  // 물리 메모리와 매핑하고 initialize
        goto err;
    struct page *dst_page = spt_find_page(dst, upage);  // 대응하는 물리 메모리 데이터 복제
    memcpy(dst_page->frame->kva, src_page->frame->kva, PGSIZE);

    break;
```

수정 코드

```
case VM_ANON:                                   // src 타입이 anon인 경우
    if (!vm_alloc_page(type, upage, writable))  // UNINIT 페이지 생성 및 초기화
        goto err;

    /** Project 3: Copy On Write (Extra) - 메모리에 load된 데이터를 write하지 않는 이상 똑같은 메모리를 사용하는데
     *  2개의 복사본을 만드는 것은 메모리가 낭비가 난다. 따라서 write 요청이 들어왔을 때만 해당 페이지에 대한 물리메모리를
     *  할당하고 맵핑하면 된다. */
    if (!vm_copy_claim_page(dst, upage, src_page->frame->kva, writable))  // 물리 메모리와 매핑하고 initialize
        goto err;

    break;
```

vm_copy_claim_page 생성

```c
/** Project 3: Copy On Write (Extra) - VA에 할당된 페이지를 복제. */
static bool vm_copy_claim_page(struct supplemental_page_table *dst, void *va, void *kva, bool writable) {
    struct page *page = spt_find_page(dst, va);

    if (page == NULL)
        return false;

    struct frame *frame = (struct frame *)malloc(sizeof(struct frame));

    if (!frame)
        return false;

    /* Set links */
    page->accessible = writable;  // 접근 권한 설정
    frame->page = page;
    page->frame = frame;
    frame->kva = kva;

    if (!pml4_set_page(thread_current()->pml4, page->va, frame->kva, false)) {
        free(frame);
        return false;
    }

    list_push_back(&frame_table, &frame->frame_elem);  // frame table에 추가

    return swap_in(page, frame->kva);
}
```

1. 접근 권한을 writable로 설정
2. 기존의 물리메모리 매핑
3. 페이지를 매핑할 때 writable을 false로 설정

handle_page_fault 수정

```c
/** Project 3: Memory Management - Return true on success */
bool vm_try_handle_fault(struct intr_frame *f UNUSED, void *addr UNUSED, bool user UNUSED, bool write UNUSED, bool not_present UNUSED) {
    struct supplemental_page_table *spt UNUSED = &thread_current()->spt;
    struct page *page = spt_find_page(&thread_current()->spt, addr);

    /* TODO: Validate the fault */
    if (addr == NULL || is_kernel_vaddr(addr))
        return false;

    /** Project 3: Copy On Write (Extra) - 접근한 메모리의 page가 존재하고 write 요청인데 write protected인 경우라 발생한 fault일 경우*/
    if (!not_present && write)
        return vm_handle_wp(page);

    /** Project 3: Copy On Write (Extra) - 이전에 만들었던 페이지인데 child가 먼저 종료되어서 spt에서 삭제하였을 때 stack_growth 대신 claim_page를 하기 위함 */
    if (!page) {
        /** Project 3: Stack Growth - stack growth로 처리할 수 있는 경우 */
        /* stack pointer 아래 8바이트는 페이지 폴트 발생 & addr 위치를 USER_STACK에서 1MB로 제한 */
        void *stack_pointer = user ? f->rsp : thread_current()->stack_pointer;
        if (stack_pointer - 8 <= addr && addr >= STACK_LIMIT && addr <= USER_STACK) {
            vm_stack_growth(thread_current()->stack_bottom - PGSIZE);
            return true;
        }
    }

    return vm_claim_page(addr);  // demand page 수행
}
```

vm_handle_wp 추가

```c
/** Project 3: Copy On Write (Extra) - Handle the fault on write_protected page */
bool vm_handle_wp(struct page *page UNUSED) {
    if (!page->accessible)
        return false;

    void *kva = page->frame->kva;

    page->frame->kva = palloc_get_page(PAL_USER);

    if (page->frame->kva == NULL)
        page->frame = vm_evict_frame();  // Swap Out 수행

    memcpy(page->frame->kva, kva, PGSIZE);

    if (!pml4_set_page(thread_current()->pml4, page->va, page->frame->kva, page->accessible))
        return false;

    return true;
}
```

1. 새로운 물리 페이지 할당
2. 원본 kva의 내용을 복사
3. 가상 메모리와 물리 메모리를 매핑

문제점

```c
/* Free the current process's resources. */
static void process_cleanup(void) {
    struct thread *curr = thread_current();

#ifdef VM
    supplemental_page_table_kill(&curr->spt);
#endif
    uint64_t *pml4;
    pml4 = curr->pml4;
    if (pml4 != NULL) {
        curr->pml4 = NULL;
        pml4_activate(NULL);
        pml4_destroy(pml4);
    }
}
```

pml4 를 destroy 하면 공유했던 페이지에
다른 프로세스가 접근할 때 fault 발생

```
cow-simple: dying due to interrupt 0x03 (#BP Breakpoint Exception).
Interrupt 0x03 (#BP Breakpoint Exception) at rip=4038d0
 cr2=0000000000406200 error=               0
rax 0000000000000000 rbx 0000000000000000 rcx 00000000004038cf rdx 0000000000000000
rsp 0000000004747ff18 rbp 0000000004747ff60 rsi 0000000000000000 rdi 0000000000000004
rip 00000000004038d0 r8 0000000000000000  r9 0000000000000000 r10 0000000000000000
r11 0000000000000216 r12 00008004249f1000 r13 0000010424900000 r14 0000800424900000
r15 0000800422de3800 rflags 00000216
es: 001b ds: 001b cs: 0023 ss: 001b
Kernel PANIC at ../../threads/palloc.c:310 in palloc_free_multiple(): assertion `bit
map_all(pool->used_map, page_idx, page_cnt)' failed.
```

해결방안 1

```
static void pt_destroy(uint64_t *pt) {
    for (unsigned i = 0; i < PGSIZE / sizeof(uint64_t *); i++) {
        uint64_t *pte = ptov((uint64_t *)pt[i]);
        if (((uint64_t)pte) & PTE_P) {
            struct page *page = spt_find_page(&thread_current()->spt, (void *)PTE_ADDR(pte));
            if (!page->writable && page->accessible)
                continue;

                palloc_free_page((void *)PTE_ADDR(pte));
        }
    }
    palloc_free_page((void *)pt);
}
```

1. 페이지를 destroy할 때 부모로부터 fork된 페이지라면 삭제를 막음

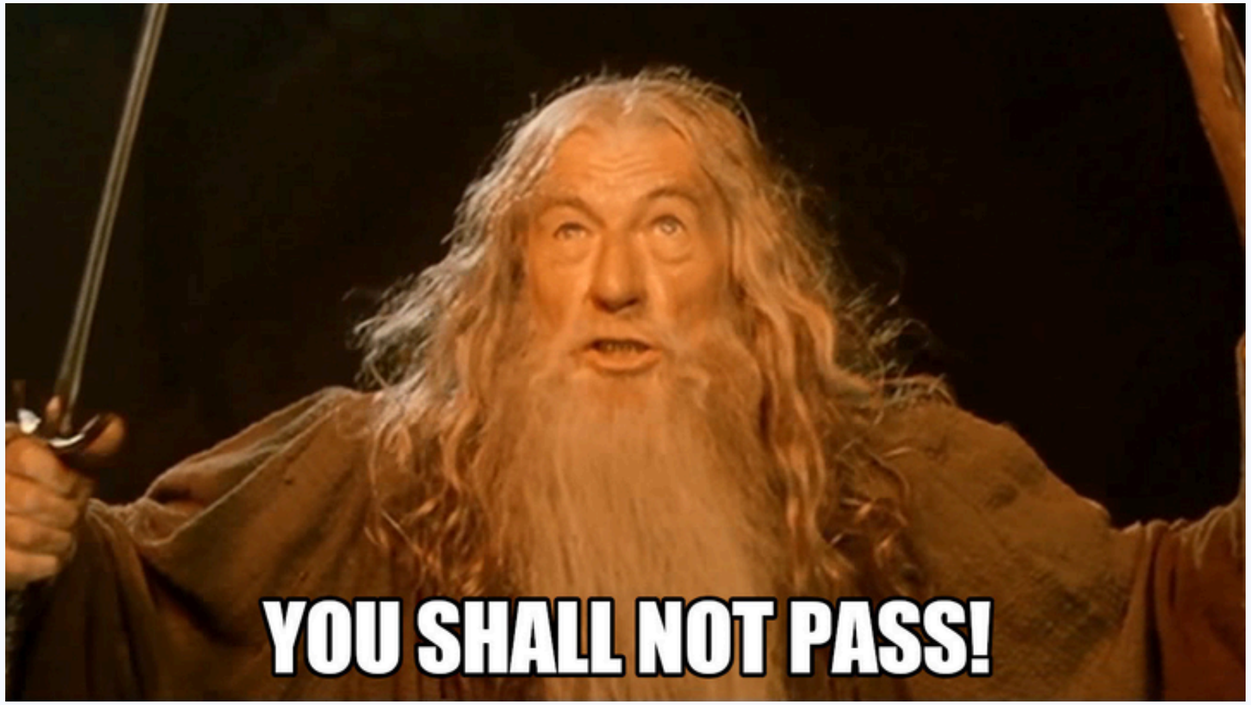
2. Page의 SPT Kill을 뒤로 미루고 pml4를 검사해서 fork 되지 않은 페이지만 삭제

그러나 잔여물 때문에 kill 발생!!

```
Interrupt 0x0d (#GP General Protection Exception) at rip=800421f8e0
 cr2=0000000000000028 error=               0
rax cccccccccccccccc rbx 0000008004a22000 rcx 0000000000000000 rdx 0000008004268000
rsp 0000008004267be0 rbp 0000008004267c00 rsi 0000008004249020 rdi 0000000000000000
rip 000000800421f8e0 r8 0000008004267b58  r9 000000800421d0a4 r10 0000000000000000
r11 0000000000000206 r12 000000800421ede7 r13 0000010424900000 r14 0000800424900000
r15 0000800422de3800 rflags 00000286
es: 0010 ds: 0010 cs: 0008 ss: 0010
Kernel PANIC at ../../userprog/exception.c:92 in kill(): Kernel bug - unexpected inter
rupt in kernel
```

해결방안 2

1. fork할 때마다 페이지를 marking
2. 부모 프로세스의 exit가 수행될 때까지 marking된 페이지의 destroy 를 차단



해결방안 3

anon 페이지 destroy 시 pml4_clear 수행

```c
/** Project 3: Swap In/Out - Destroy the anonymous page. PAGE will be freed by the caller. */
static void anon_destroy(struct page *page) {
    struct anon_page *anon_page = &page->anon;

    /** Project 3: Swap In/Out - 점거중인 bitmap 삭제 */
    if (anon_page->slot != BITMAP_ERROR)
        bitmap_reset(swap_table, anon_page->slot);

    /** Project 3: Anonymous Page - 점거중인 frame 삭제 */
    if (page->frame) {
        list_remove(&page->frame->frame_elem);
        page->frame->page = NULL;
        free(page->frame);
        page->frame = NULL;
    }

    /** Project 3: Copy on Write (Extra) - destroy 시 pml4 clear하여 참조하던 kva들을 모두 해제한다.
     * 그렇지 않으면 자식에서 `exit`시 참조한 부모의 kva가 파괴되어 자식이 부모에서 해당 kva에 접근할 수 없기 때문이다. */
    pml4_clear_page(thread_current()->pml4, page->va);
}
```

```
pass tests/filesys/base/sm-seq-block
pass tests/filesys/base/sm-seq-random
pass tests/filesys/base/syn-read
pass tests/filesys/base/syn-remove
pass tests/filesys/base/syn-write
pass tests/threads/alarm-single
pass tests/threads/alarm-multiple
pass tests/threads/alarm-simultaneous
pass tests/threads/alarm-priority
pass tests/threads/alarm-zero
pass tests/threads/alarm-negative
pass tests/threads/priority-change
pass tests/threads/priority-donate-one
pass tests/threads/priority-donate-multiple
pass tests/threads/priority-donate-multiple2
pass tests/threads/priority-donate-nest
pass tests/threads/priority-donate-sema
pass tests/threads/priority-donate-lower
pass tests/threads/priority-fifo
pass tests/threads/priority-preempt
pass tests/threads/priority-sema
pass tests/threads/priority-condvar
pass tests/threads/priority-donate-chain
pass tests/vm/cow/cow-simple
All 141 tests passed.
```

감사합니다