Lab06: Copy-on-Write Fork for xv6

Implement Copy-On-Write (Hard)

1) 实验目的

本实验的目的是在 xv6 中实现写时复制（Copy-On-Write, COW）机制，从而优化 `fork()` 系统调用的内存使用。通过实现 COW，我们可以延迟物理内存的分配，直到写操作发生。目标是使得内核能够通过 `cowtest` 和 `usertests` 的所有测试。

2) 实验步骤

1. **修改 `uvmcopy()` 以支持 COW**：
   - **在 `kernel/vm.c` 中修改 `uvmcopy()`**，使其在 `fork()` 时将父进程的物理页面映射到子进程，并清除 `PTE_W` 标志：

```
int
uvmcopy(pagetable_t old, pagetable_t new, uint64 sz)
{
    uint64 i, pa;
    pte_t *pte;
    char *mem;

    for(i = 0; i < sz; i += PGSIZE){
        pte = walk(old, i, 0);
        if(pte == 0 || !(*pte & PTE_V))
            panic("uvmcopy: pte should exist");

        pa = PTE2PA(*pte);
        if(uvmmap(new, i, pa, PGSIZE, PTE_R | PTE_X | PTE_U | PTE_COW)
!= 0)
            return -1;
    }
    return 0;
}
```

2. **修改 `usertrap()` 以处理 COW 页面错误**：
   - **在 `kernel/trap.c` 中修改 `usertrap()`**，处理 COW 页面错误：

```
void
usertrap(void)
{
    uint64 va;
    pte_t *pte;
    struct proc *p = myproc();
```

```
    // 检查是否是页面错误
    if (r_scause() == 13 || r_scause() == 15) {
        va = r_stval();
        va = PGROUNDDOWN(va);

        pte = walk(p->pagetable, va, 0);
        if (pte && (*pte & PTE_COW)) {
            char *mem = kalloc();
            if (mem == 0) {
                printf("usertrap(): out of memory\n");
                exit(-1);
            }
            memmove(mem, (char*)PTE2PA(*pte), PGSIZE);
            if (mappages(p->pagetable, va, PGSIZE, (uint64)mem, PTE_R |
PTE_W | PTE_U) != 0) {
                printf("usertrap(): mappages failed\n");
                exit(-1);
            }
            *pte = PA2PTE((uint64)mem) | PTE_R | PTE_W | PTE_U;
            sfence_vma();
            return;
        }
    }
    // 处理其他异常
    // ...
}
```

3. **在 `kalloc()` 中实现物理页的引用计数**:

   - **在 `kernel/kalloc.c` 中添加引用计数机制**:

```
#define MAX_PHYS_PAGES 1024
static int refcount[MAX_PHYS_PAGES];

void
kinit(void)
{
    // 初始化物理页面
    memset(refcount, 0, sizeof(refcount));
}

char*
kalloc(void)
{
    // 分配页面并更新引用计数
    char *p = kalloc_impl();  // 你需要实现这个函数
    if (p) {
        int idx = PTE2PA(p) / PGSIZE;
        refcount[idx] = 1;
    }
    return p;
}
```

```
void
kfree(char *p)
{
    int idx = PTE2PA(p) / PGSIZE;
    if (--refcount[idx] == 0) {
        kfree_impl(p);  // 你需要实现这个函数
    }
}
```

4. **修改 `copyout()` 以支持 COW**:
   - **在 `kernel/sysproc.c` 中修改 `copyout()`**，处理 COW 页面:

```
int
copyout(pagetable_t pagetable, uint64 va, void *p, uint64 len)
{
    uint64 i;

    for(i = 0; i < len; i += PGSIZE){
        if (walk(pagetable, i, 0) && (*walk(pagetable, i, 0) &
PTE_COW)) {
            // Handle COW pages
            if (mappages(pagetable, va + i, PGSIZE,
PA2PTE(PTE2PA(*walk(pagetable, i, 0))), PTE_R | PTE_W | PTE_U) != 0)
                return -1;
        }
    }

    return copyout_impl(pagetable, va, p, len);
}
```

5. **测试和验证**:
   - **编译并运行 `cowtest`**，确保所有测试通过。
   - **编译并运行 `usertests`**，确保所有测试通过。
   - **检查输出是否符合预期**，验证内核的稳定性和 COW 功能的正确性。

3) 实验中遇到的困难和解决办法

1. **物理页面的引用计数**:
   - **困难描述**：确保物理页面的引用计数正确更新，避免内存泄漏或错误释放。
   - **解决办法**：仔细检查 `kalloc()` 和 `kfree()` 的实现，确保引用计数的正确更新和物理页的正确管理。

2. **COW 页面处理**:
   - **困难描述**：在 `usertrap()` 和 `copyout()` 中正确处理 COW 页面，避免错误的内存操作。

- **解决办法**：仔细检查页面错误处理和页面映射的逻辑，确保在发生 COW 页面错误时正确分配和映射新页面。

3. **系统调用的修改**：

- **困难描述**：修改系统调用以支持 COW 可能会引入新的问题或错误。
- **解决办法**：使用调试工具（如 `gdb`）进行逐步检查和验证，确保所有系统调用的实现都符合预期。

4) 实验心得

通过本次实验，我深入理解了写时复制（COW）机制在操作系统中的应用，特别是在内存管理和进程复制方面的优化。实现 COW 使得内存使用更加高效，避免了不必要的内存复制，提高了系统的性能。解决实验中的问题让我提高了对内核内存管理和系统调用的理解，也增强了调试和问题解决的能力。