

Assignment 2- uaccess

12212320 谭政

Checkpoint. 1

修改 xv6 的用户与内核的页表模型，

使得：

- 用户页表包含整个内核页表(kernel_pagetable)。(内核页表中的 PTE 均没有 PTE_U 权限位)
- 内核线程使用用户页表（此时它应该包含内核页表），而不是内核页表。同时，你需要修改 Context Switch 和 Trampoline：
- 在使用 swtch 进行 Context Switch 时，切换到内核线程各自的页表 (p->mm->pgt)。
- 通过 trampoline 在用户态与内核态之间切换时，保留用户页表，不需要切换到内核页表 kernel_satp。
- 对于没有用户空间的 scheduler，直接使用 kernel_pagetable 内核页表即可。

修改：

```
void mm_copy_kpgt(struct mm *mm) {  
  
    assert(holding(&mm->lock));  
  
    for (uint64 i = 0x100; i < 512; i++) {  
  
        pte_t *kpte = (pte_t *)&kernel_pagetable[i];  
  
        pte_t *upte = (pte_t *)&mm->pgt[i];  
  
        // Assignment 2: Your code here  
  
        if (*kpte & PTE_V) {  
  
            *upte = *kpte & ~PTE_U;  
  
        }  
    }  
}
```

```

    }

static int access_ok(struct mm* mm, uint64 addr, uint64 len) {

    // Assignment 2: Your code here

    if (addr >= USER_TOP)

        return 0;

    if (len > 0 && addr + len < addr) // 溢出

        return 0;

    if (addr + len > USER_TOP)

        return 0;

    return 1;

}

}

```

运行结果：

```

init: starting sh
sh >> test_uaccess 1
-> checkpoint 1 passed
sh > child 3 exit with code 0
sh >>

```

Checkpoint. 2

阅读特权级手册 4.1.1.2 Memory Privilege in sstatus Register 中关于 SUM 标志位的解释，理解为什么这里 会遇到 Page Fault，并且根据文档中的提示解决该问题。

因为内核在用户态下尝试直接访问用户空间内存，而没有正确设置权限位。通过

设置 sstatus 寄存器的 SUM 位，允许内核直接访问用户空间内存。在拷贝操作完成后，清除 SUM 位以恢复默认行为

修改：

```
static void begin_user_access() {  
    // Assignment 2: Your code here  
  
    w_sstatus(r_sstatus() | SSTATUS_SUM);  
}  
  
static void end_user_access() {  
    // Assignment 2: Your code here  
  
    w_sstatus(r_sstatus() & ~SSTATUS_SUM);  
}  
  
// Copy from kernel to user.  
  
// Copy len bytes from src to virtual address dstva in a given page table.  
  
// Return 0 on success, -1 on error.  
  
int copy_to_user(struct mm *mm, uint64 __user dstva, char *src, uint64 len) {  
    begin_user_access();  
  
    memmove((void *)dstva, src, len);  
  
    end_user_access();  
  
    return 0;  
}  
  
// Copy from user to kernel.  
  
// Copy len bytes to dst from virtual address srcva in a given page table.  
  
// Return 0 on success, -1 on error.  
  
int copy_from_user(struct mm *mm, char *dst, uint64 __user srcva, uint64 len) {  
    if (!access_ok(mm, srcva, len)) {
```

```

        return -1;
    }

    begin_user_access();

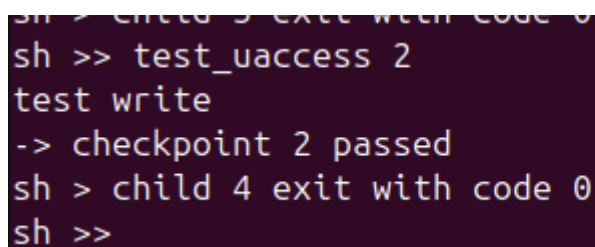
    memmove(dst, (void *)srcva, len);

    end_user_access();

    return 0;
}

```

运行结果：



```

sh > child 3 exit with code 0
sh >> test_uaccess 2
test write
-> checkpoint 2 passed
sh > child 4 exit with code 0
sh >>

```

Checkpoint. 3:

在 checkpoint3 中，任务是确保内核不会访问非法的用户空间地址。access_ok 函数通过检查地址范围来确保用户地址是合法的。

修改：

```

static int access_ok(struct mm* mm, uint64 addr, uint64 len) {

    // Assignment 2: Your code here

    if (addr >= USER_TOP)

        return 0;

    if (len > 0 && addr + len < addr) // 溢出

        return 0;

    if (addr + len > USER_TOP)

```

```

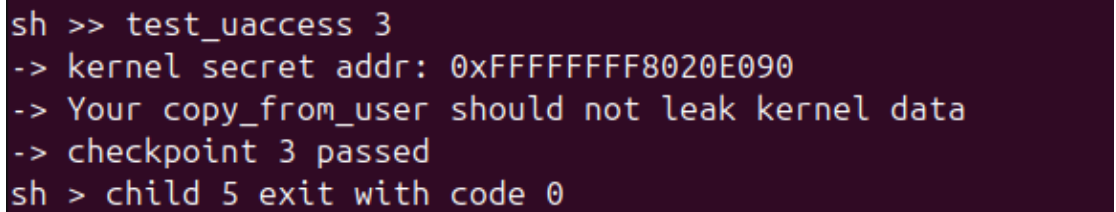
        return 0;

    return 1;

}

```

运行截图:



```

sh >> test_uaccess 3
-> kernel secret addr: 0xFFFFFFFF8020E090
-> Your copy_from_user should not leak kernel data
-> checkpoint 3 passed
sh > child 5 exit with code 0

```

Checkpoint. 4

```

修改: if (cause & SCAUSE_INTERRUPT) {

    // correctness checking:

    if (mycpu()->inkernel_trap > 1) {

        // should never have nested interrupt

        print_sysregs(true);

        print_ktrapframe(ktf);

        panic("nested kerneltrap");

    }

    if (panicked) {

        panic("other CPU has panicked");

    }

    // handle interrupt

    if (handle_intr() == 0) {

        errorf("unhandled interrupt: %d", cause);

        goto kernel_panic;

    }

} else {

```

```

        if (exception_code == 12 || exception_code == 13 || exception_code == 15)
        {
            struct proc *p = curr_proc();

            if (holding(&p->mm->lock)) {
                intr_off();

                release(&p->mm->lock);
            }

            p->killed = 1;

            mycpu()->inkernel_trap--;

            exit(-9);

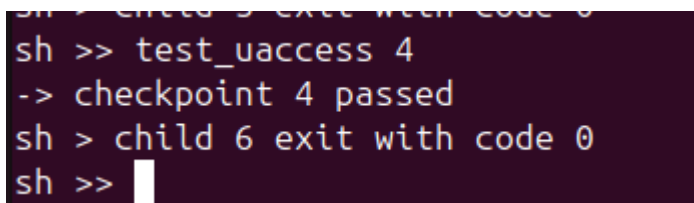
            return;
        }

        // kernel exception, unexpected.

        goto kernel_panic;
    }

```

运行结果：



```

sh >> test_uaccess 4
-> checkpoint 4 passed
sh > child 6 exit with code 0
sh >> 

```