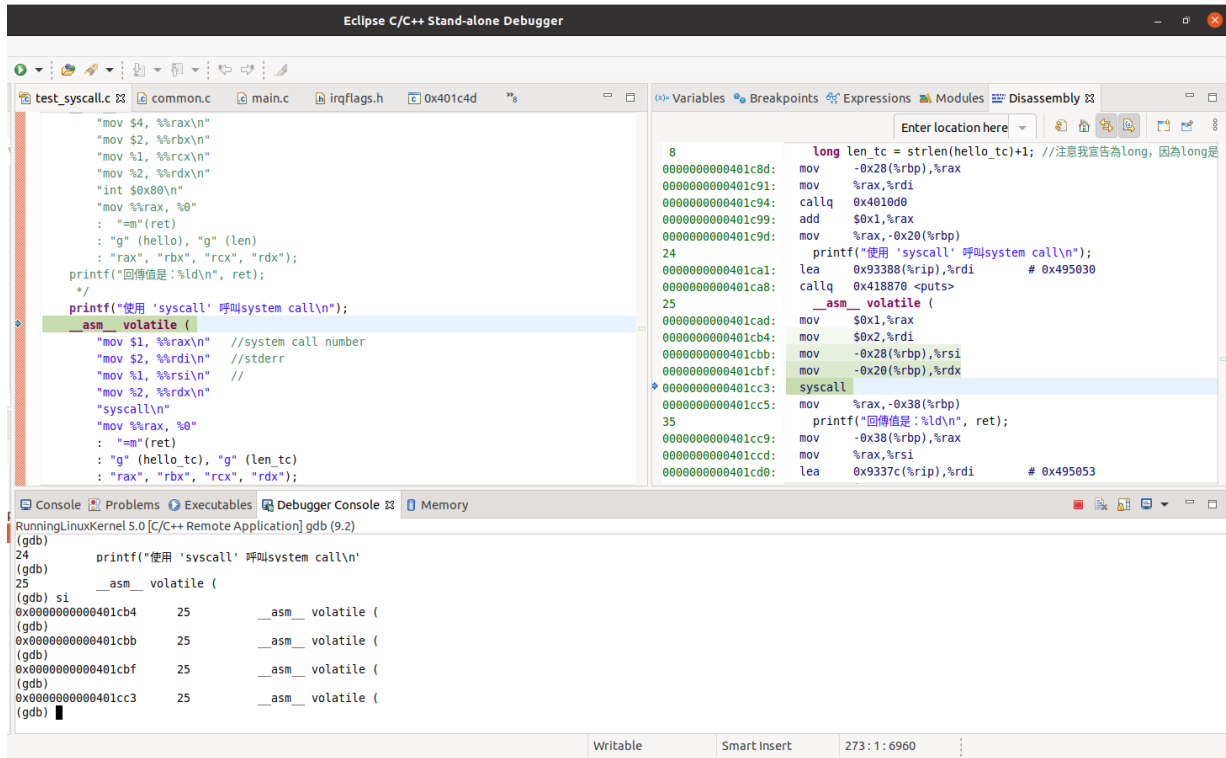


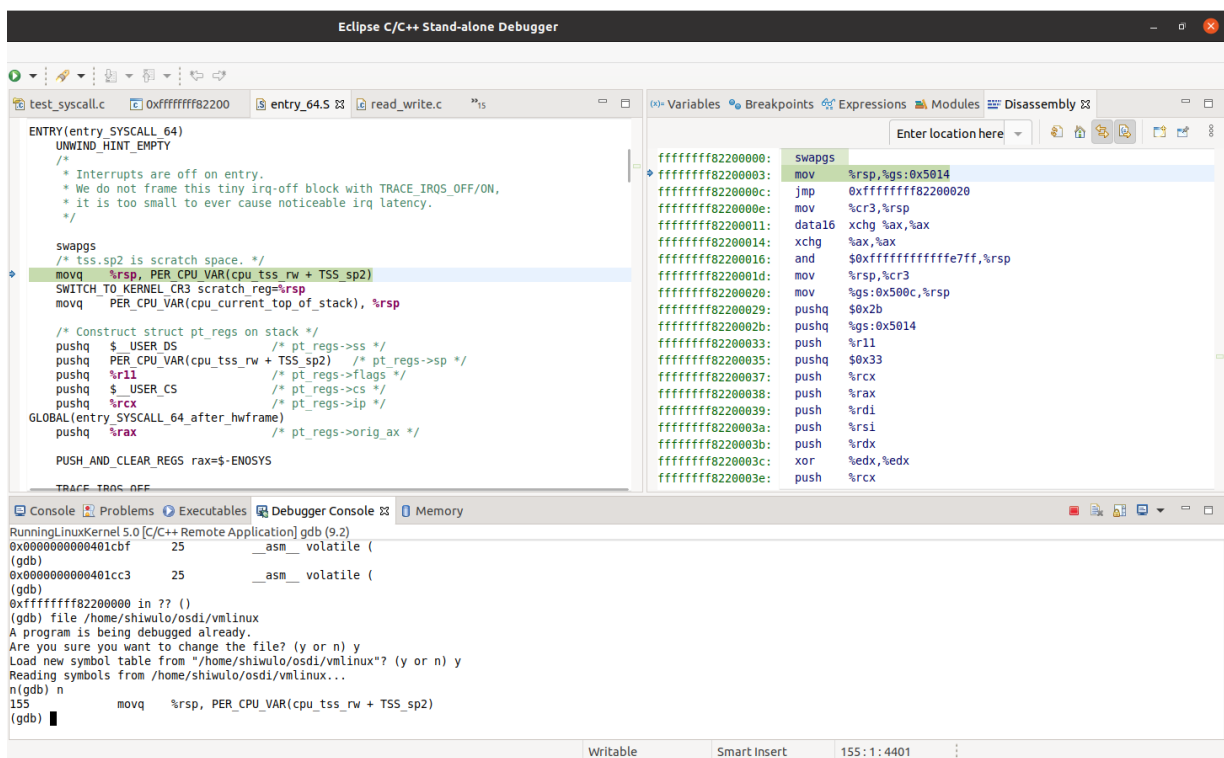
# 作業系統概論 hw8

學號: 408410113 姓名: 王 X 彥

## 1. 設定中斷點在test\_syscall發出system call之前，請在這個地方截圖



## 2. 使用單步追蹤 (si)，直到Linux kernel，請在進入Linux kernel時截圖



### 3. 請說明Linux kernel如何用RAX暫存器判斷要呼叫哪個Linux內部的函數

用RAX暫存器的值當作index，透過sys\_call\_table對應到Linux內部的函數。

The screenshot displays the Eclipse C/C++ Stand-alone Debugger interface. The main window shows the disassembly of the `do_syscall_64` function, which is responsible for dispatching system calls. The function uses the value in the `RAX` register to index into the `sys_call_table` to determine which kernel function to call.

```
#ifdef CONFIG_X86_64
visible void do_syscall_64(unsigned long nr, struct pt_regs *regs)
{
    struct thread_info *ti;
    enter_from_user_mode();
    local_irq_enable();
    ti = current_thread_info();
    if (READ_ONCE(ti->flags) & _TIF_WORK_SYSCALL_ENTRY)
        nr = syscall_trace_enter(regs);

    /*
     * NB: Native and x32 syscalls are dispatched from the same
     * table. The only functional difference is the x32 bit in
     * regs->orig_ax, which changes the behavior of some syscalls.
     */
    nr &= _SYSCALL_MASK;
    if (likely(nr < NR_syscalls)) {
        nr = array_index_nospec(nr, NR_syscalls);
        regs->ax = sys_call_table[nr](regs);
    }

    syscall_return_slowpath(regs);
}
#endif
```

The disassembly view shows the following instructions:

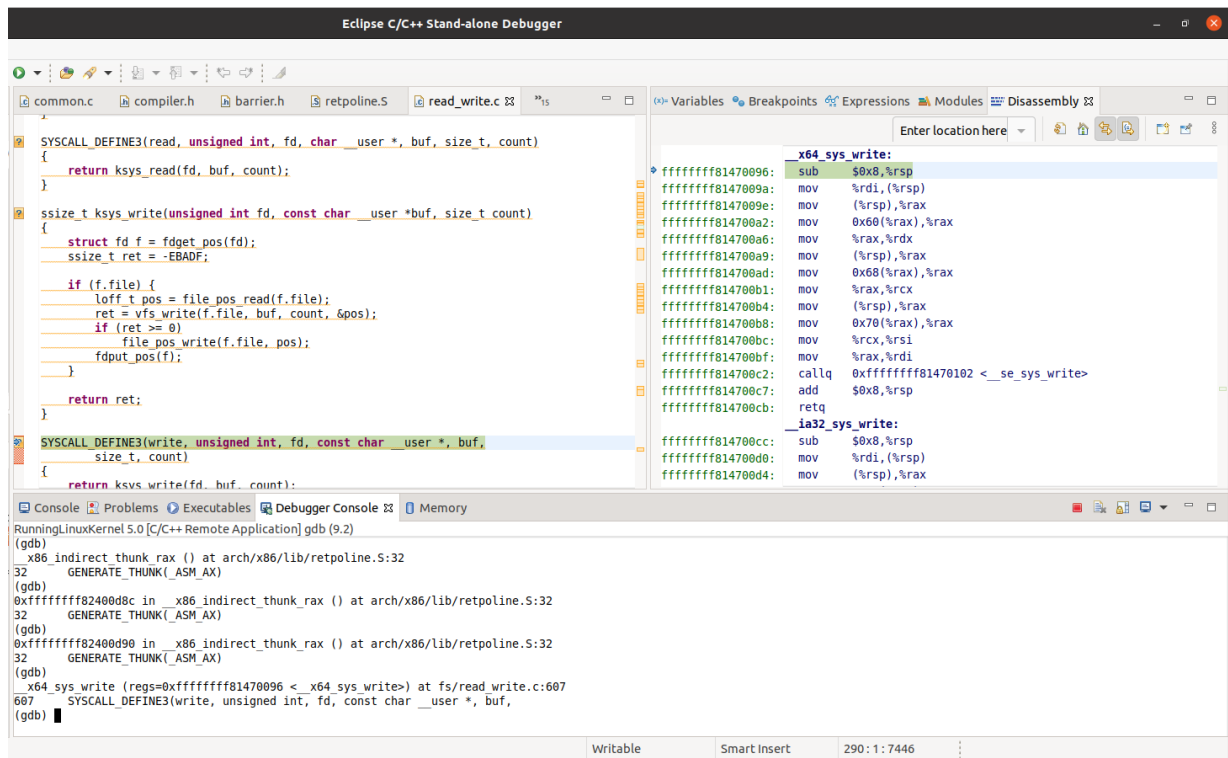
```
do_syscall_64:
ffffffffff8100747d: sub    $0x58,%rsp
ffffffffff81007481: mov    %rdi,0x8(%rsp)
ffffffffff81007486: mov    %rsi,(%rsp)
ffffffffff8100748a: mov    %gs:0x28,%rax
ffffffffff81007493: mov    %rax,0x50(%rsp)
ffffffffff81007498: xor    %eax,%eax
276:      enter_from_user_mode();
ffffffffff8100749a: callq  0xffffffff81006bc6 <enter_from_user_mode>
277:      local_irq_enable();
ffffffffff8100749f: callq  0xffffffff810053d1 <arch_local_irq_enable>
15:      return this_cpu_read_stable(current_task);
ffffffffff810074a4: mov    %gs:0x14cc0,%rax
ffffffffff810074ad: mov    %rax,0x30(%rsp)
ffffffffff810074b2: mov    0x30(%rsp),%rax
278:      ti = current_thread_info();
ffffffffff810074b7: mov    %rax,0x18(%rsp)
279:      if (READ_ONCE(ti->flags) & _TIF_WORK_SYSCALL_ENTRY)
ffffffffff810074bc: mov    0x18(%rsp),%rax
ffffffffff810074c1: mov    %rax,0x38(%rsp)
```

The console window shows the following output:

```
RunningLinuxKernel 5.0 [C/C++ Remote Application] gdb (9.2)
(gdb)
0xffffffff8220006e in entry_SYSCALL_64 () at arch/x86/entry/entry_64.S:168
168      PUSH_AND_CLEAR_REGS rax=$-ENOSYS
(gdb)
173      movq    %rax,%rdi
(gdb)
174      movq    %rsp,%rsi
(gdb)
175      call   do_syscall_64          /* returns with IRQs disabled */
(gdb) si
do_syscall_64 (nr=1, regs=0xffffffff900003ffff58) at arch/x86/entry/common.c:273
273 {
(gdb)
```

#### 4. 請大致說明作業系統如何處理write。

呼叫 `ksys_write`，`ksys_write` 內部先取得 file offset 後，呼叫 `vfs_write`，調用 file 的 inode 節點供 write 使用，更新 file offset。



The screenshot shows the Eclipse C/C++ Stand-alone Debugger interface. The left pane displays the source code for `read_write.c`, specifically the `ksys_write` function. The right pane shows the assembly code for `_x64_sys_write`. The console at the bottom shows the execution flow, including the call to `ksys_write` and the subsequent call to `vfs_write`.

```
SYSCALL_DEFINE3(read, unsigned int, fd, char __user *, buf, size_t, count)
{
    return ksys_read(fd, buf, count);
}

ssize_t ksys_write(unsigned int fd, const char __user *buf, size_t count)
{
    struct fd f = fdget_pos(fd);
    ssize_t ret = -EBADF;

    if (f.file) {
        loff_t pos = file_pos_read(f.file);
        ret = vfs_write(f.file, buf, count, &pos);
        if (ret >= 0)
            file_pos_write(f.file, pos);
        fdput_pos(f);
    }

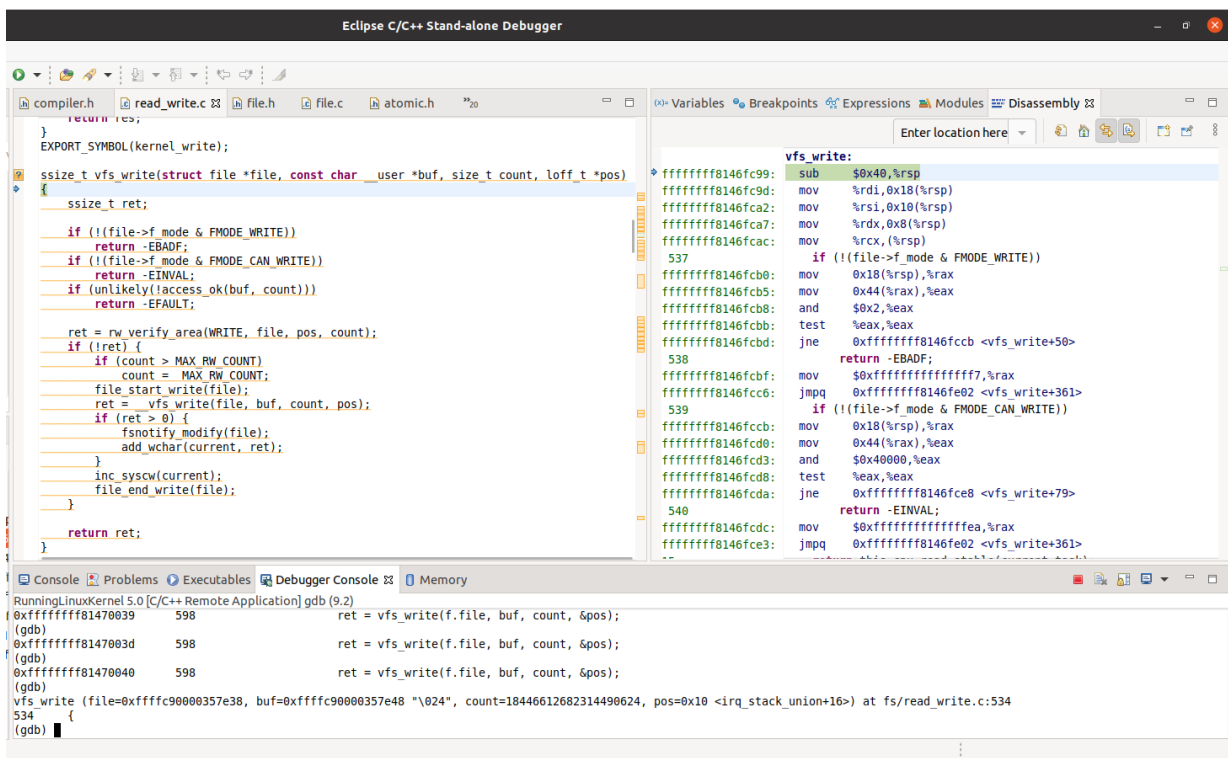
    return ret;
}

SYSCALL_DEFINE3(write, unsigned int, fd, const char __user *, buf,
size_t, count)
{
    return ksys_write(fd, buf, count);
}
```

```
_x64_sys_write:
ffffffffff81470096: sub    $0x8,%rsp
ffffffffff8147009a: mov    %rdi,%rax
ffffffffff8147009e: mov    (%rsp),%rax
ffffffffff814700a2: mov    0x60(%rax),%rax
ffffffffff814700a6: mov    %rax,%rdx
ffffffffff814700a9: mov    (%rsp),%rax
ffffffffff814700ad: mov    0x68(%rax),%rax
ffffffffff814700b1: mov    %rax,%rcx
ffffffffff814700b4: mov    (%rsp),%rax
ffffffffff814700b8: mov    0x70(%rax),%rax
ffffffffff814700bc: mov    %rcx,%rsi
ffffffffff814700bf: mov    %rax,%rdi
ffffffffff814700c2: callq 0xffffffff81470102 <__se_sys_write>
ffffffffff814700c7: add    $0x8,%rsp
ffffffffff814700cb: retq

_ia32_sys_write:
ffffffffff814700cc: sub    $0x8,%rsp
ffffffffff814700d0: mov    %rdi,%rsp
ffffffffff814700d4: mov    (%rsp),%rax
```

RunningLinuxKernel 5.0 [C/C++ Remote Application] gdb (9.2)  
(gdb) x/8i indirect\_thunk\_rax () at arch/x86/lib/retpoline.S:32  
32 GENERATE\_THUNK(ASM\_AX)  
(gdb) 0xffffffff82400d8c in \_x86\_indirect\_thunk\_rax () at arch/x86/lib/retpoline.S:32  
32 GENERATE\_THUNK(ASM\_AX)  
(gdb) 0xffffffff82400d90 in \_x86\_indirect\_thunk\_rax () at arch/x86/lib/retpoline.S:32  
32 GENERATE\_THUNK(ASM\_AX)  
(gdb) x/8i sys\_write (regs=0xffffffff81470096 <\_x64\_sys\_write>) at fs/read\_write.c:607  
607 SYSCALL\_DEFINE3(write, unsigned int, fd, const char \_\_user \*, buf,  
(gdb)



The screenshot shows the Eclipse C/C++ Stand-alone Debugger interface. The left pane displays the source code for `file.c`, specifically the `vfs_write` function. The right pane shows the assembly code for `vfs_write`. The console at the bottom shows the execution flow, including the call to `vfs_write` and the subsequent call to `fs_write`.

```
EXPORT_SYMBOL(kernel_write);

ssize_t vfs_write(struct file *file, const char __user *buf, size_t count, loff_t *pos)
{
    ssize_t ret;

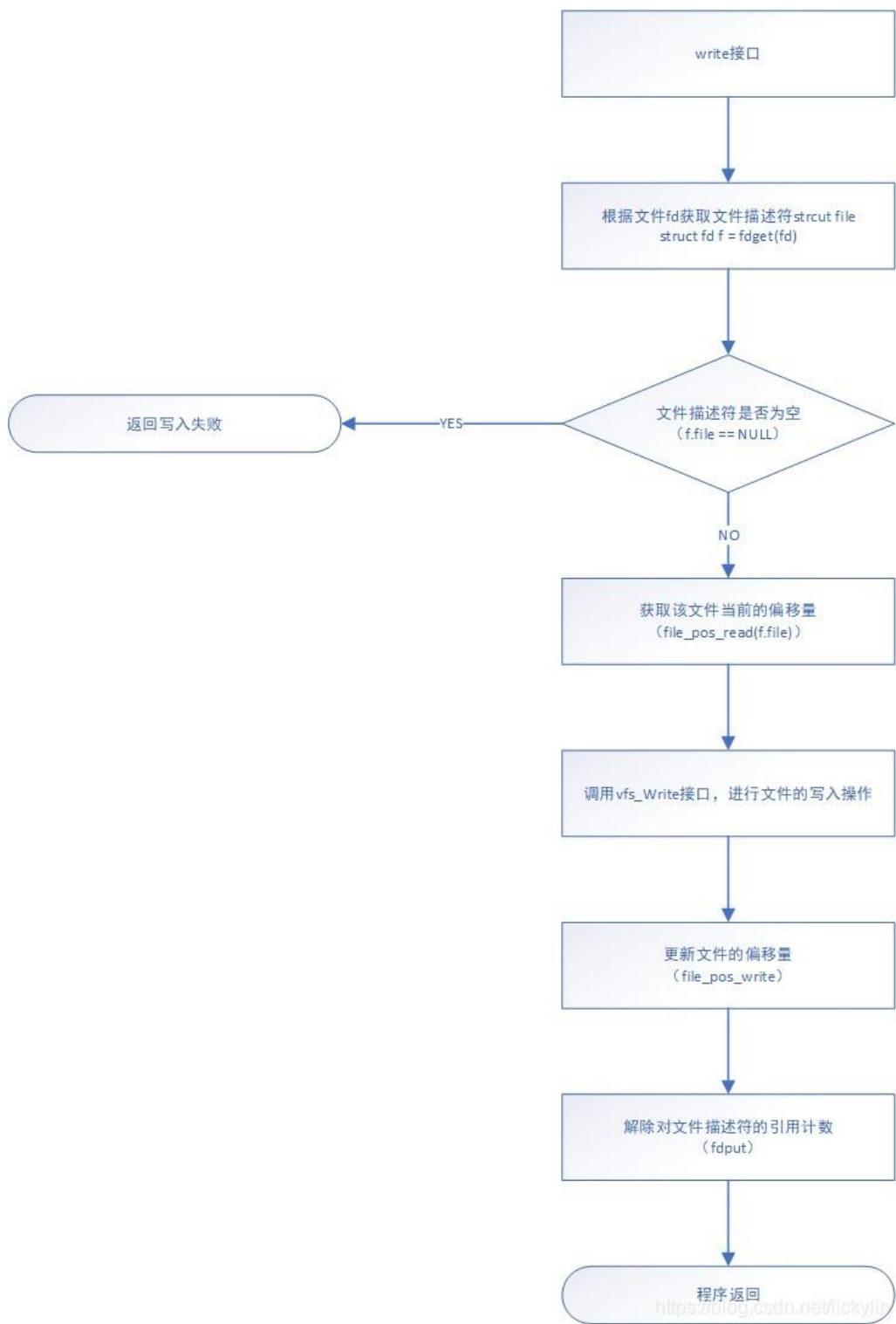
    if (!file->f_mode & FMODE_WRITE)
        return -EBADF;
    if (!file->f_mode & FMODE_CAN_WRITE)
        return -EINVAL;
    if (unlikely(!access_ok(buf, count)))
        return -EFAULT;

    ret = rw_verify_area(WRITE, file, pos, count);
    if (!ret) {
        if (count > MAX_RW_COUNT)
            count = MAX_RW_COUNT;
        file_start_write(file);
        ret = fs_write(file, buf, count, pos);
        if (ret > 0) {
            fsnotify_modify(file);
            add_wchar(current, ret);
        }
        inc_syscw(current);
        file_end_write(file);
    }

    return ret;
}
```

```
vfs_write:
ffffffffff8146fc99: sub    $0x40,%rsp
ffffffffff8146fc9d: mov    %rdi,0x18(%rsp)
ffffffffff8146fca2: mov    %rsi,0x10(%rsp)
ffffffffff8146fca7: mov    %rdx,0x8(%rsp)
ffffffffff8146fcac: mov    %rcx,%rsp
537     if (!file->f_mode & FMODE_WRITE)
ffffffffff8146fcb0: mov    0x18(%rsp),%rax
ffffffffff8146fcb5: mov    0x44(%rax),%eax
ffffffffff8146fcb8: and    $0x2,%eax
ffffffffff8146fcbb: test   %eax,%eax
ffffffffff8146fcbb: jne    0xffffffff8146fccb <vfs_write+50>
538     return -EBADF;
ffffffffff8146fcfb: mov    $0xfffffffffffffff7,%rax
ffffffffff8146fcc6: jmpq   0xffffffff8146fe02 <vfs_write+361>
539     if (!file->f_mode & FMODE_CAN_WRITE)
ffffffffff8146fccb: mov    0x18(%rsp),%rax
ffffffffff8146fcd0: mov    0x44(%rax),%eax
ffffffffff8146fcd3: and    $0x40000,%eax
ffffffffff8146fcd8: test   %eax,%eax
ffffffffff8146fcda: jne    0xffffffff8146fce8 <vfs_write+79>
540     return -EINVAL;
ffffffffff8146fcdc: mov    $0xfffffffffffffeaf,%rax
ffffffffff8146fce3: jmpq   0xffffffff8146fe02 <vfs_write+361>
```

RunningLinuxKernel 5.0 [C/C++ Remote Application] gdb (9.2)  
(gdb) 0xffffffff81470039 598 ret = vfs\_write(f.file, buf, count, &pos);  
(gdb) 0xffffffff8147003d 598 ret = vfs\_write(f.file, buf, count, &pos);  
(gdb) 0xffffffff81470040 598 ret = vfs\_write(f.file, buf, count, &pos);  
(gdb) vfs\_write (file=0xffffffff90000357e38, buf=0xffffc90000357e48 "024", count=18446612682314490624, pos=0x10 <irq\_stack\_union+16>) at fs/read\_write.c:534  
534 {  
(gdb)



參考資料: <https://blog.csdn.net/lickylin/article/details/101715048>