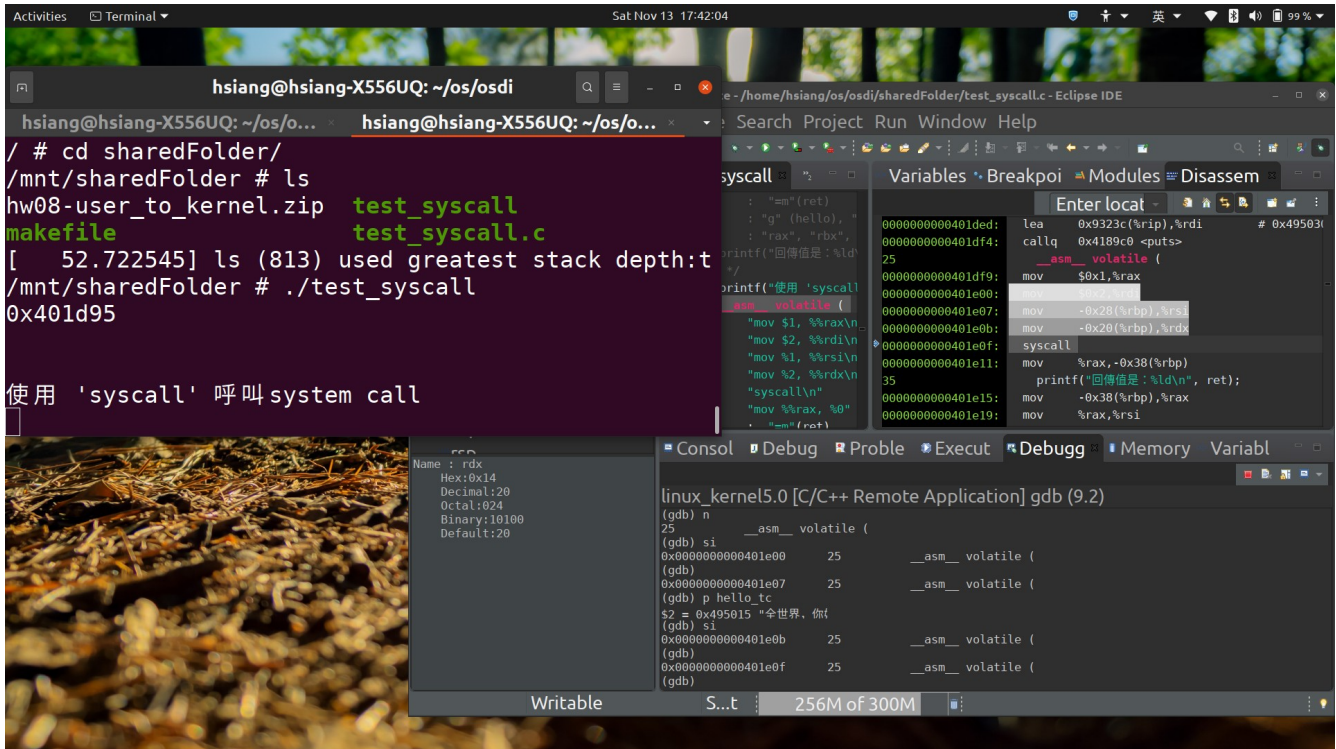


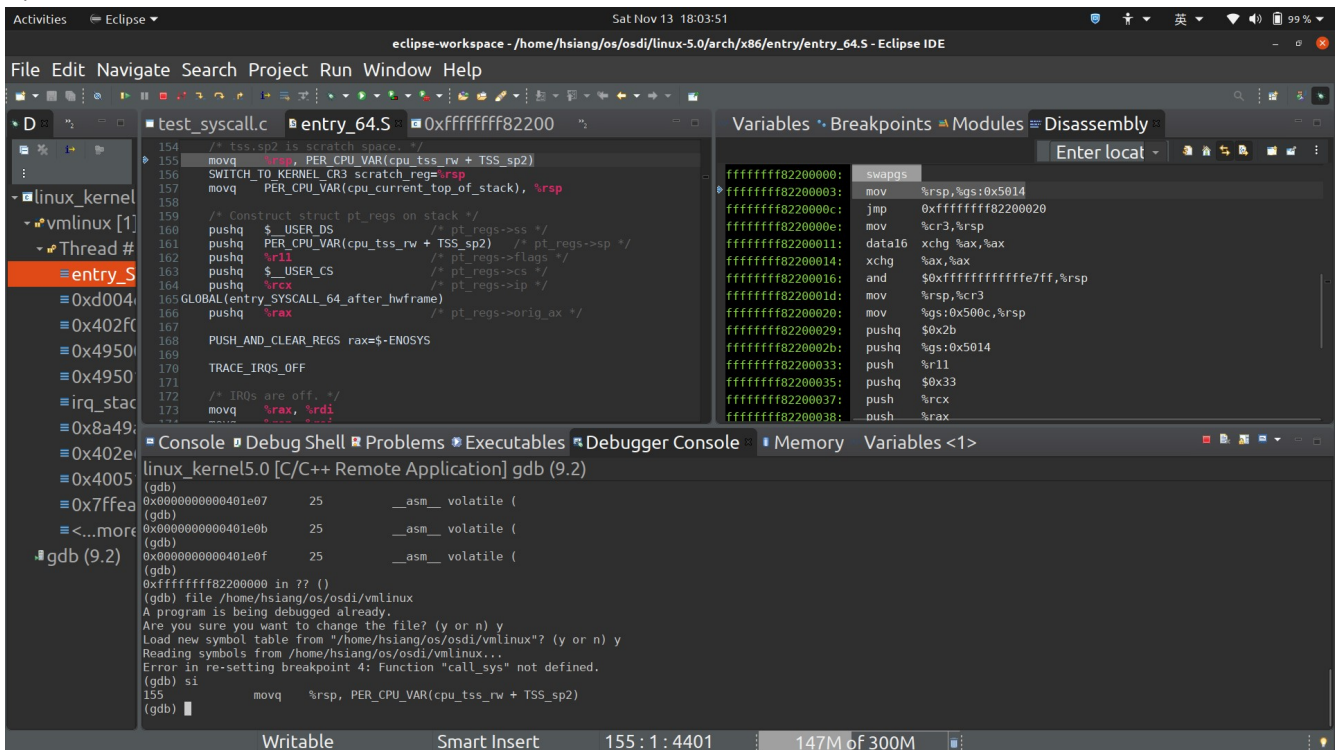
# HW8 報告

資工三 408410098 蔡×祥

1.



2.



3.

The screenshot shows the Eclipse IDE with the file `entry_64.S` open. The function `do_syscall_64` is visible, which takes an unsigned long `nr` and a struct `pt_regs` as arguments. The function's logic involves checking if the system call number is within the native x32 range, then using `sys_call_table` to find the appropriate handler. The assembly disassembly on the right shows the function's entry point at `0xffffffff8100747d`, where it sets up the stack and calls `enter_from_user_mode`. The console at the bottom shows the GDB session, with a breakpoint at `0xffffffff8100747d` and the instruction `call do_syscall_64` being executed.

藉由傳入的參數 `nr` 來從函數指標 array 找到要用的 system call 的函式。

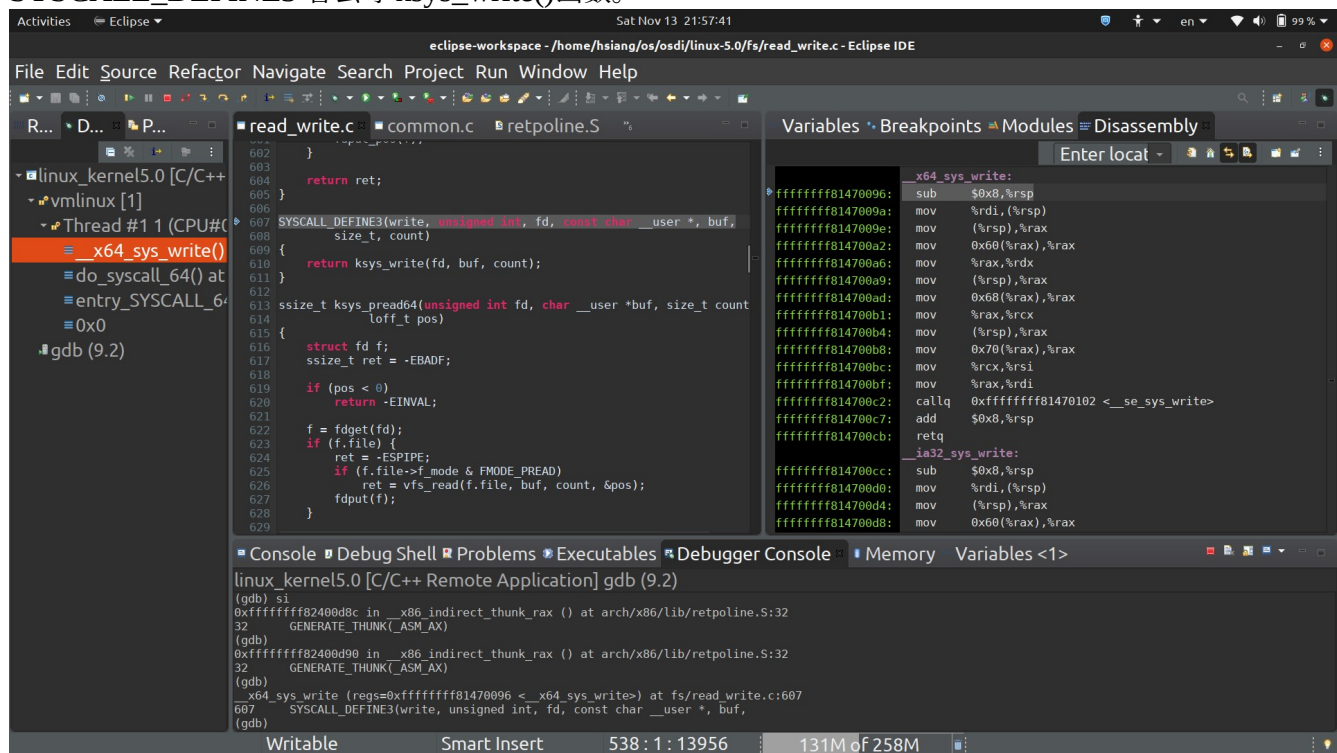
The screenshot shows the Eclipse IDE with the file `read_write.c` open. The function `_x64_sys_write` is visible, which takes an unsigned int `fd`, a char `__user` `buf`, and a size\_t `count` as arguments. The function's logic involves checking if the file descriptor is valid, then using `vfs_read` to read the data into the buffer. The assembly disassembly on the right shows the function's entry point at `0xffffffff81470096`, where it sets up the stack and calls `fs/read_write.c: No such file or directory`. The console at the bottom shows the GDB session, with a breakpoint at `0xffffffff81470096` and the instruction `callq 0xffffffff81470102 <__se_sys_write>` being executed.

`write` 是呼叫 `SYSCALL_DEFINE3` 這個函數。

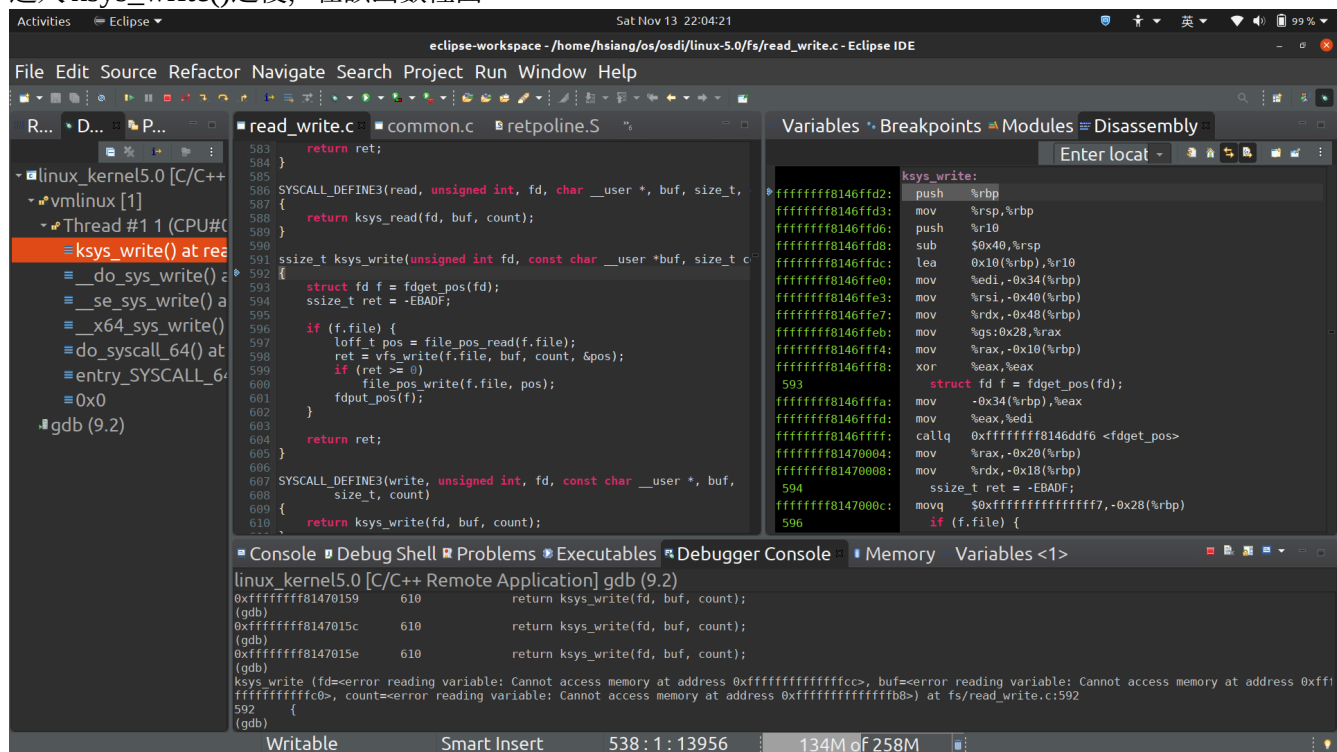


4.

SYSCALL\_DEFINE3 會去呼 ksys\_write() 函數。



進入 ksys\_write() 之後，在該函數裡面：



首先使用 `fdget_pos()`。

將 `int` 型態的轉成 `struct fd`

該函數鎖定要寫入的檔案

內部機制：

```
extern void __f_unlock_pos(struct file *);
static inline struct fd __to_fd(unsigned long v)
{
    return (struct fd){(struct file *) (v & ~3), v & 3};
}
static inline struct fd fdget(unsigned int fd)
{
    return __to_fd(__fdget(fd));
}
static inline struct fd fdget_raw(unsigned int fd)
{
    return __to_fd(__fdget_raw(fd));
}
static inline struct fd fdget_pos(int fd)
{
    return __to_fd(__fdget_pos(fd));
}
static inline void fdput_pos(struct fd f)
{
    if (f.flags & FDPUT_POS_UNLOCK)
        __f_unlock_pos(f.file);
    fdput(f);
}
```

```
fdget_pos:
ffffff8146ddf6: push    %rbp
ffffff8146ddf7: mov     %rsp,%rbp
ffffff8146ddf8: push    %r10
ffffff8146ddf9: sub     $0x8,%rsp
ffffff8146dde0: mov     %edi,-0xc(%rbp)
70:         return __to_fd(__fdget_pos(fd));
ffffff8146dde7: mov     -0xc(%rbp),%eax
ffffff8146dde8: callq   0xffffffff814b719c <__fdget_pos>
ffffff8146dde9: mov     %rax,%rdi
ffffff8146ded0: callq   0xffffffff8146dd84 <__to_fd>
71:         add     $0x8,%rsp
ffffff8146ded1: pop     %r10
ffffff8146ded2: pop     %rbp
72:         retq
73:         fdput_pos:
ffffff8146ded3: push    %rbp
```

```
linux_kernel5.0 [C/C++ Remote Application] gdb (9.2)
0xffffffff8146ddfc 69 {
(gdb)
0xffffffff8146de00 69 {
(gdb)
0xffffffff8146de04 69 {
(gdb)
70         return __to_fd(__fdget_pos(fd));
(gdb)
0xffffffff8146de0a 70         return __to_fd(__fdget_pos(fd));
(gdb)
```

調用\_\_to\_fd(), \_\_to\_fd()是接\_\_fdget\_light()的回傳值。

\_\_fdget\_light():

```
static unsigned long __fdget_light(unsigned int fd, fmode_t mask)
{
    struct files_struct *files = current->files;
    struct file *file;
    if (atomic_read(&files->count) == 1) {
        file = __fcheck_files(files, fd);
        if (!file || unlikely(files->f_mode & mask))
            return 0;
        return (unsigned long)file;
    } else {
        file = __fdget(fd, mask);
        if (!file)
            return 0;
        return FDPUT_FPUT | (unsigned long)file;
    }
}
unsigned long __fdget(unsigned int fd)
{
    return __fdget_light(fd, FMODE_PATH);
}
```

```
__fdget_light:
ffffff814b6fcc: sub     $0x58,%rsp
ffffff814b6fd0: mov     %edi,0x4(%rsp)
ffffff814b6fd4: mov     %esi,%rsp
ffffff814b6fd7: mov     %gs:0x28,%rax
ffffff814b6fe0: mov     %rax,0x50(%rsp)
ffffff814b6fe5: xor     %eax,%eax
15:         return this_cpu_read_stable(current_task);
ffffff814b6fe7: mov     %gs:0x14cc,%rax
ffffff814b6ff0: mov     %rax,0x40(%rsp)
ffffff814b6ff5: mov     0x40(%rsp),%rax
761:         struct files_struct *files = current->files;
ffffff814b6ffa: mov     0x668(%rax),%rax
ffffff814b7001: mov     %rax,0x10(%rsp)
764:         if (atomic_read(&files->count) == 1) {
ffffff814b7006: mov     0x10(%rsp),%rax
ffffff814b700b: mov     %rax,0x20(%rsp)
ffffff814b7010: mov     0x20(%rsp),%rax
ffffff814b7015: mov     $0x4,%esi
ffffff814b701a: mov     %rax,%rdi
ffffff814b701d: callq   0xffffffff814b352f <kasan_check_read>
```

```
linux_kernel5.0 [C/C++ Remote Application] gdb (9.2)
0xffffffff814b716e 778         return __fdget_light(fd, FMODE_PATH);
(gdb)
0xffffffff814b7173 778         return __fdget_light(fd, FMODE_PATH);
(gdb)
0xffffffff814b7175 778         return __fdget_light(fd, FMODE_PATH);
(gdb)
__fdget_light (fd=0, mask=0) at fs/file.c:760
760 {
(gdb)
```

其中，裡面的if(atomic\_read(&file->count)==1)大致上的意思是要確定該 atomic 模式的 file 只有這個 task 目前開啟這個檔案。

如果是的話就透過 fd 找到該檔案(\_\_fcheck\_files)，若檔案存在就回傳檔案(unsigned long 型態)

再來，使用 `file_pos_read()` 取的檔案得偏移值。

之後，使用 `vfs_write()` 來寫入，成功的話則 `pos` 更新到 `f.file` 裡面。( `if(ret>=0){ file_pos_write() }` )

最後，呼叫 `fdput_pos()` 解開 `file` 的鎖定（即讀取這個 `file` 的 `task -1` ）