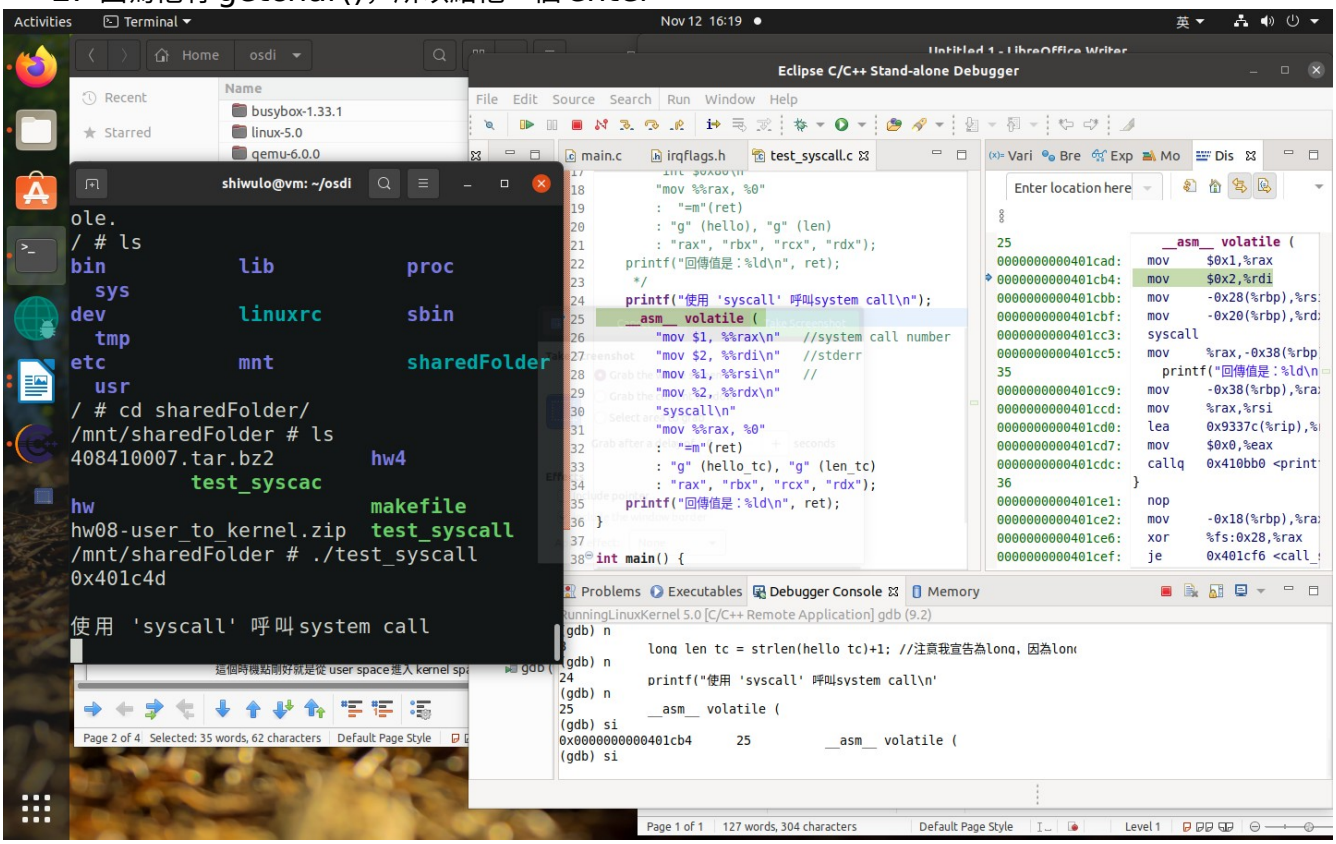
enter(跑出(gdb))，c 讓他開機，
執行./test_syscall 後，要 ctrl_c，讓 eclipse 告訴 demu 說我拿到控制權（送 signal）
file /home/shiwulo/osdi/sharedFolder/test_syscall
（沒有 start_kernel func，因為現在是對應用程式（test_syscall）做 debug，所以載入的是他的 symbol table。）
一：設定中斷點在 test_syscall 發出 system call 之前（例如：將中斷點設定在 call_sys），請在這個地方截圖

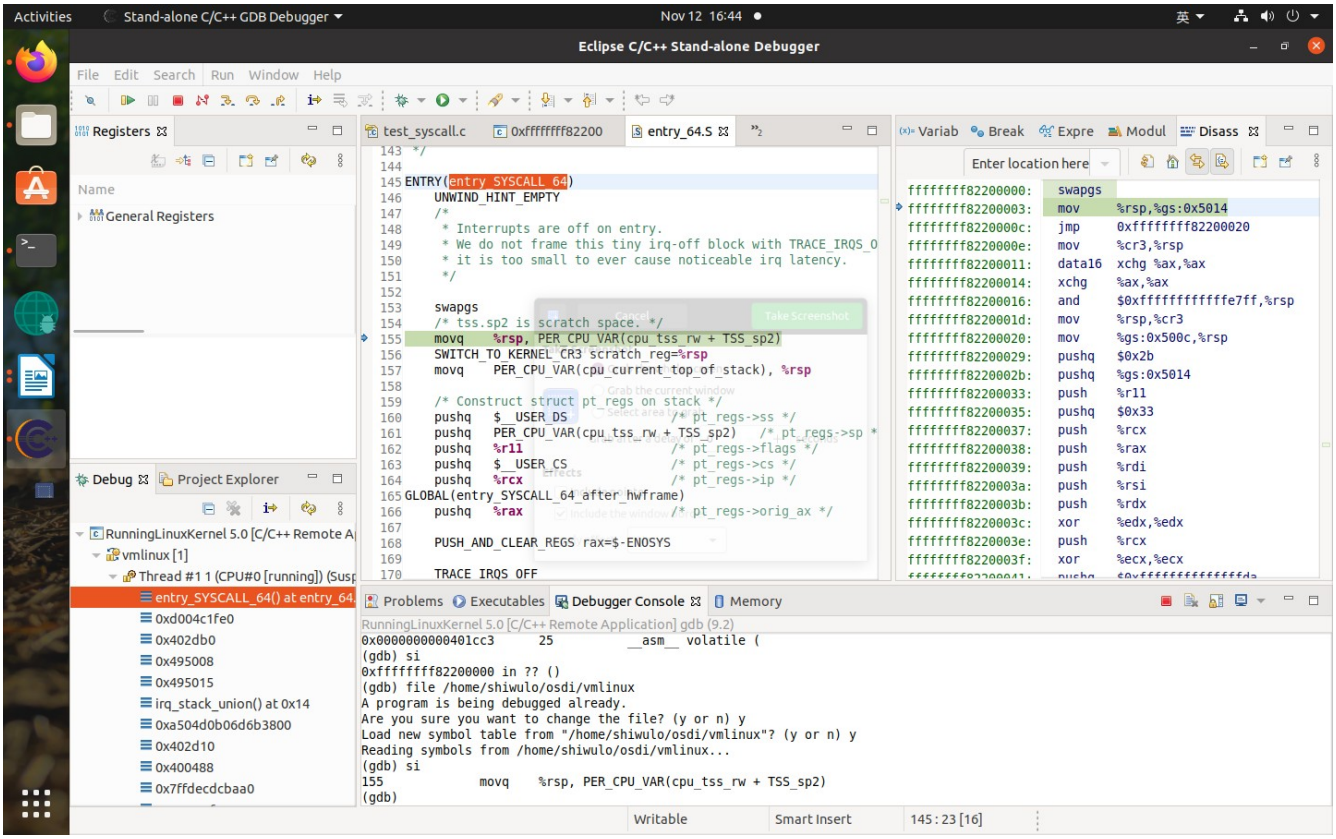
1. b *(0x401c4d)，按一個 c
2. 因為他有 getchar()，所以給他一個 enter



二：
使用單步追蹤（si），直到 syscall 後（注意：syscall 之後就是跳入到 Linux kernel），先在 Debugger Console 輸入「file /home/shiwulo/osdi/vmlinux」，然後對 Eclipse 拍照。這個時機點剛好就是從 user space 進入 kernel space 時。

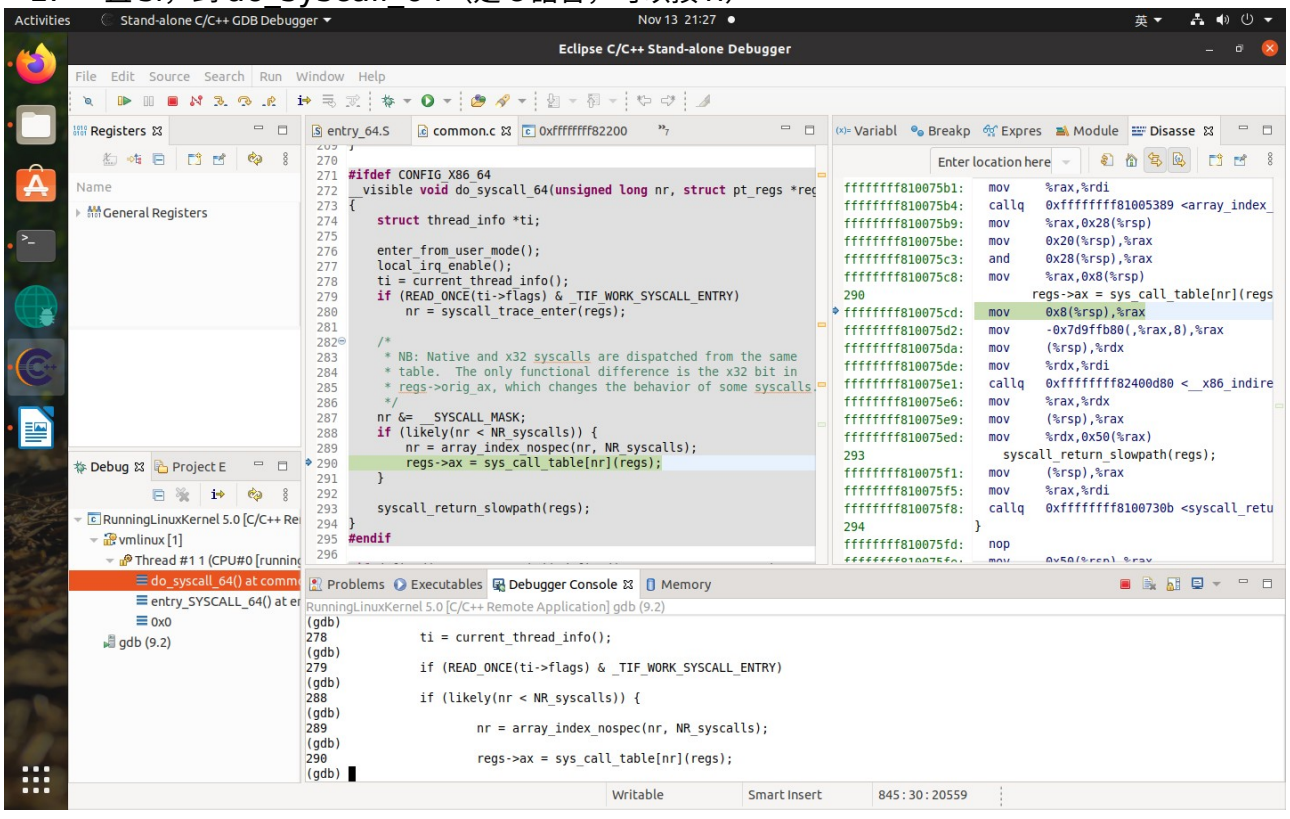
//這裡一開始的時候，就是把暫存器 push 到堆疊裡面。

1. 先進 swapgs，輸入「file /home/shiwulo/osdi/vmlinux」，把 debug sybmol 把他切換成 linux kernel 的 debug symbol
2. 之後要再 si 一次，才看得到左側程式碼



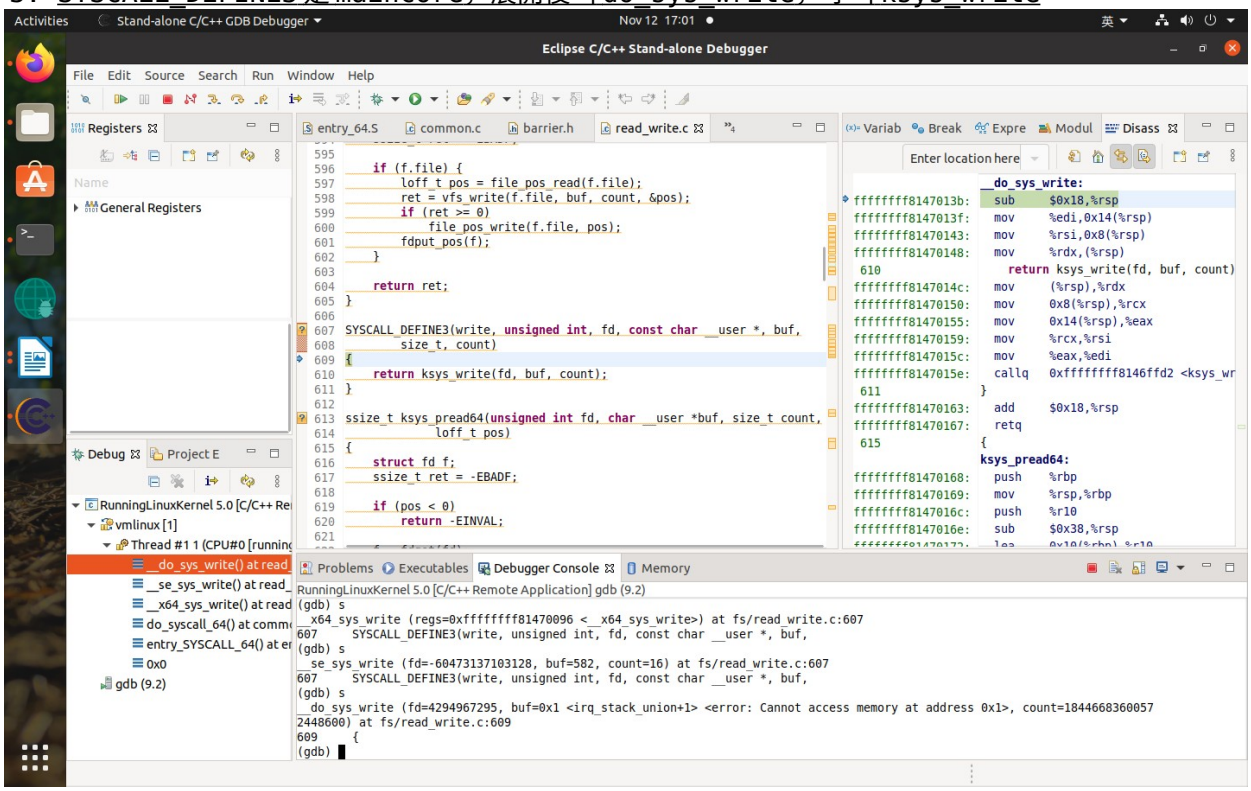
三：請說明 Linux kernel 如何用 RAX 暫存器判斷要呼叫哪個 Linux 內部的函數，請說明該函數的名稱

1. 一直 si，到 do_syscall_64（是 c 語言，可以按 n）



2. sys_call_table 是函數指標，指向一開始 push 的那些暫存器，regs 是那些 reg

3. SYSCALL_DEFINE3 是 maincore，展開後叫 do_sys_write，呼叫 ksys_write




```
    nr = array_index_nospec(nr, NR_syscalls);
    regs->ax = sys_call_table[nr](regs);
```

4. **以下都要用 s**，因為不是組合語言

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

四：請用 50~200 個「有意義的文字」大致說明作業系統如何處理該 system call（例如：可以將註解英翻中，或者列出呼叫流程。請盡可能表達。這一題會依照所寫的內容的完整性給分）。

```
/*
* Lightweight file lookup - no refcnt increment if fd table isn't shared.
 *
 * You can use this instead of fget if you satisfy all of the following
 * conditions:
 * 1) You must call fput_light before exiting the syscall and returning control
 *    to userspace (i.e. you cannot remember the returned struct file * after
 *    returning to userspace).
 * 2) You must not call filp_close on the returned struct file * in between
 *    calls to fget_light and fput_light.
 * 3) You must not clone the current task in between the calls to fget_light
 *    and fput_light.
 *
 * The fput_needed flag returned by fget_light should be passed to the
 * corresponding fput_light.
 */
```

```
759 static unsigned long __fget_light(unsigned int fd, fmode_t mask)
760 {
761     struct files_struct *files = current->files;
762     struct file *file;
763
764     if (atomic_read(&files->count) == 1) {
765         file = __fcheck_files(files, fd);
766         if (!file || unlikely(file->f_mode & mask))
767             return 0;
768         return (unsigned long)file;
769     } else {
770         file = __fget(fd, mask);
771         if (!file)
772             return 0;
773         return FDPUT_FPUT | (unsigned long)file;
774     }
775 }
776 unsigned long __fdget(unsigned int fd)
777 {
778     return __fget_light(fd, FMODE_PATH);
779 }
780 EXPORT_SYMBOL(__fdget);
781
782 unsigned long __fdget_raw(unsigned int fd)
786
787 unsigned long __fdget_pos(unsigned int fd)
788 {
789     unsigned long v = __fdget(fd);
790     struct file *file = (struct file *)(v & ~3);
791
792     if (file && (file->f_mode & FMODE_ATOMIC_POS)) {
793         if (file_count(file) > 1) {
794             v |= FDPUT_POS_UNLOCK;
795             mutex_lock(&file->f_pos_lock);
796         }
797     }
798     return v;
```

在__fget_light 裡回傳的 file，給__fdget(上層呼叫的)，最後是 v。把回傳值變成指標，用 fd 去查詢，得到描述這個檔案的一個資料結構，file 這個指標。(在__fdgets_pos)

■ **在 ksys_write 中首先呼叫 fdget_pos()，這個函數似乎是「鎖定」這個檔案，同時他會回傳一個資料結構用以代表「檔案」，這個資料結構包含 fields 及 function pointers。**

```
591 ssize_t ksys_write(unsigned int fd, const char __user *buf, size_t count)
592 {
593     struct fd f = fdget_pos(fd);
594     ssize_t ret = -EBADF;
595
596     if (f.file) {
597         loff_t pos = file_pos_read(f.file);
598         ret = vfs_write(f.file, buf, count, &pos);
599         if (ret >= 0)
600             file_pos_write(f.file, pos);
601         fdput_pos(f);
602     }
603
604     return ret;
605 }
```

597.598 行：拿到檔案的資料結構，再拿到檔案的位置，位置往後讀，要把資料讀到 buf，buf 前面是定義一個 __user,所以這個指標是指到 user space 的指標。

- **呼叫 vfs_write(f.file, buf, count, &pos)做真正的寫入。因為每個檔案可能對應到不同的裝置，因此 vfs_write 的行為，取決於 f.file**
  - **這個地方應該繼續往下追，vfs_write 應該是呼叫 f 裡面的某個函數**

```
533 ssize_t vfs_write(struct file *file, const char __user *buf, size_t count, loff_t *pos)
534 {
535     ssize_t ret;
536
537     if (!(file->f_mode & FMODE_WRITE))
538         return -EBADF;
539     if (!(file->f_mode & FMODE_CAN_WRITE))
540         return -EINVAL;
541     if (unlikely(!access_ok(buf, count)))
542         return -EFAULT;
543
544     ret = rw_verify_area(WRITE, file, pos, count);
545     if (!ret) {
546         if (count > MAX_RW_COUNT)
547             count =  MAX_RW_COUNT;
548         file_start_write(file);
549         ret =  __vfs_write(file, buf, count, pos);
550         if (ret > 0) {
551             fsnotify_modify(file);
552             add_wchar(current, ret);
553         }
554         inc_syscw(current);
555         file_end_write(file);
556     }
557
558     return ret;
559 }
```

EBADF: fd 不是有效的文件描述符或未打開寫入

EINVAL: fd 附加到不適合寫入的對像上；或者文件是使用 O_DIRECT 標誌打開的，並且 buf 中指定的地址、count 中指定的值或文件偏移量未適當對齊。

EFAULT: buf 超出您的可訪問地址空間。

- **如果上述的 write 成功，那麼會將 pos 更新到 f.file**

```
566 static inline void file_pos_write(struct file *file, loff_t pos)
567 {
568     file->f_pos = pos;
569 }
```

- 因為已經更新完畢，呼叫 **fdput_pos**，解開「鎖定」

```
73 static inline void fdput_pos(struct fd f)
74 {
75     if (f.flags & FDPUT_POS_UNLOCK)
76         __f_unlock_pos(f.file);
77     fdput(f);
78 }
```