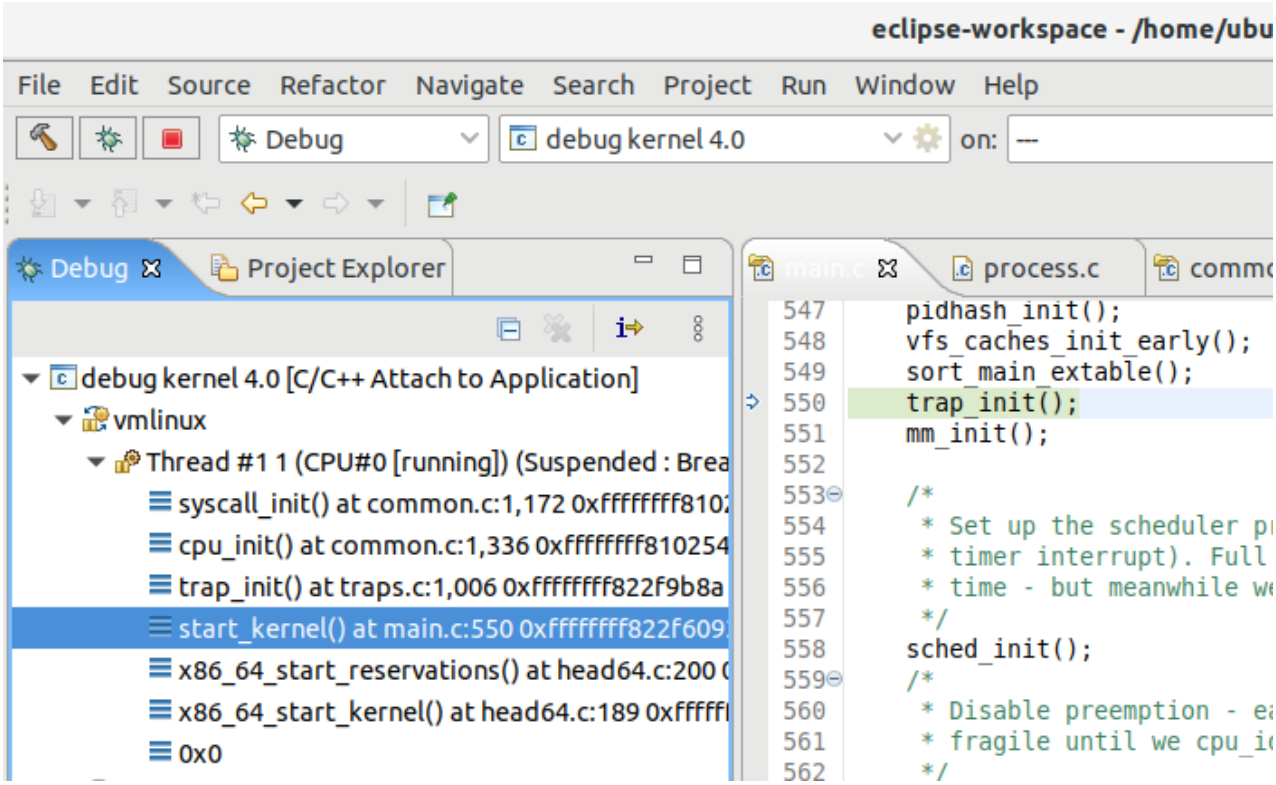
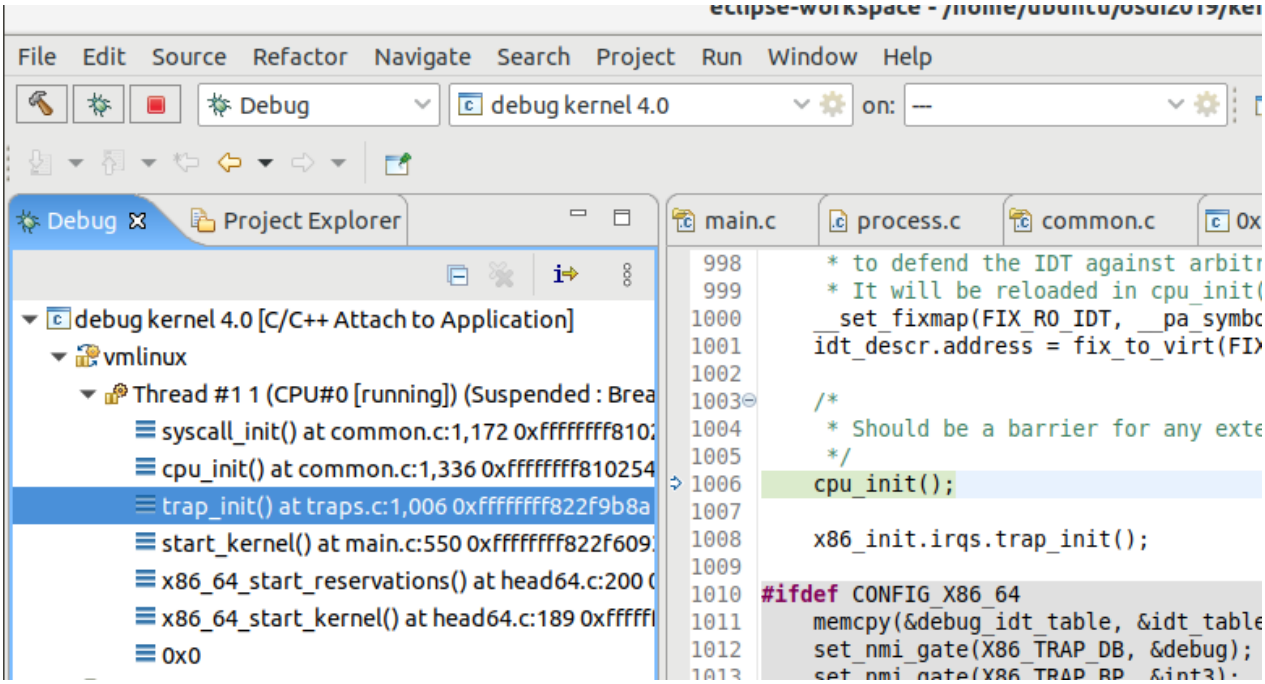
hw04. system call entry point 406410114 郭晏誠

1. 當應用程式發出 system call 時，Linux 會從哪裡開始執行

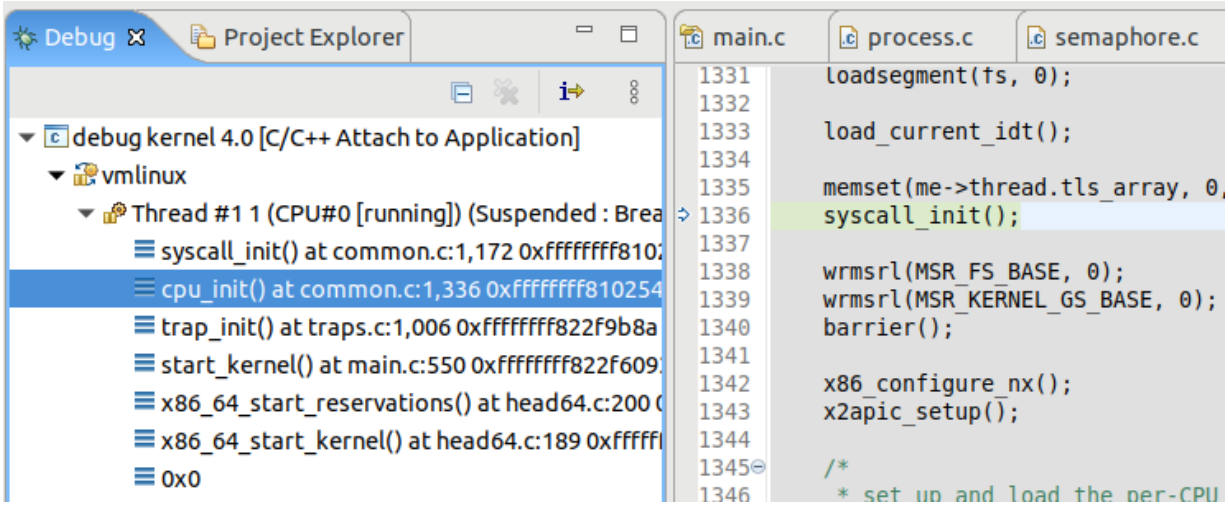
(i)start_kernel()呼叫 trap_init()



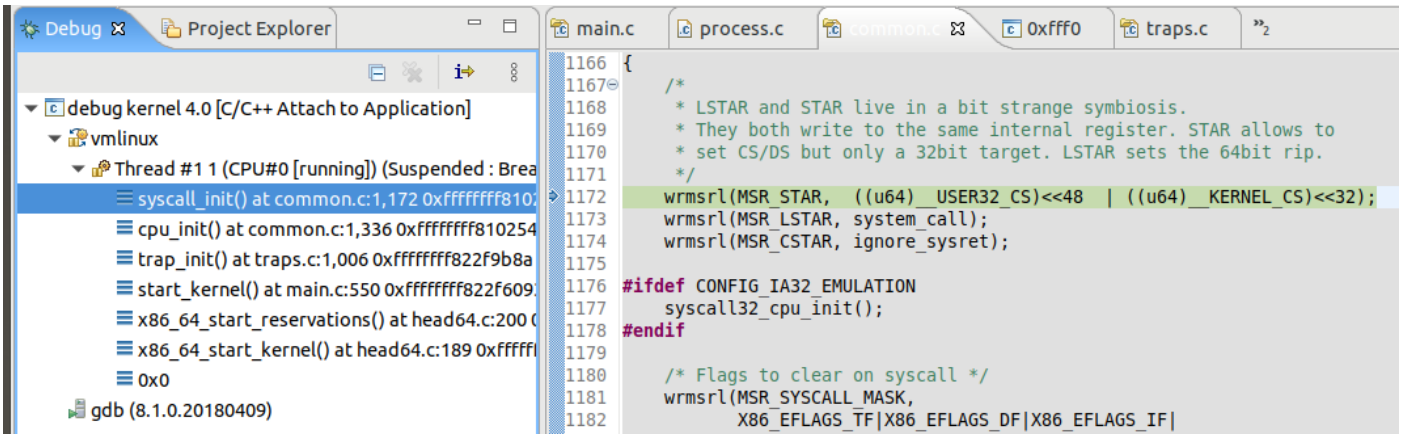
(ii) trap_init()呼叫 cpu_inti()



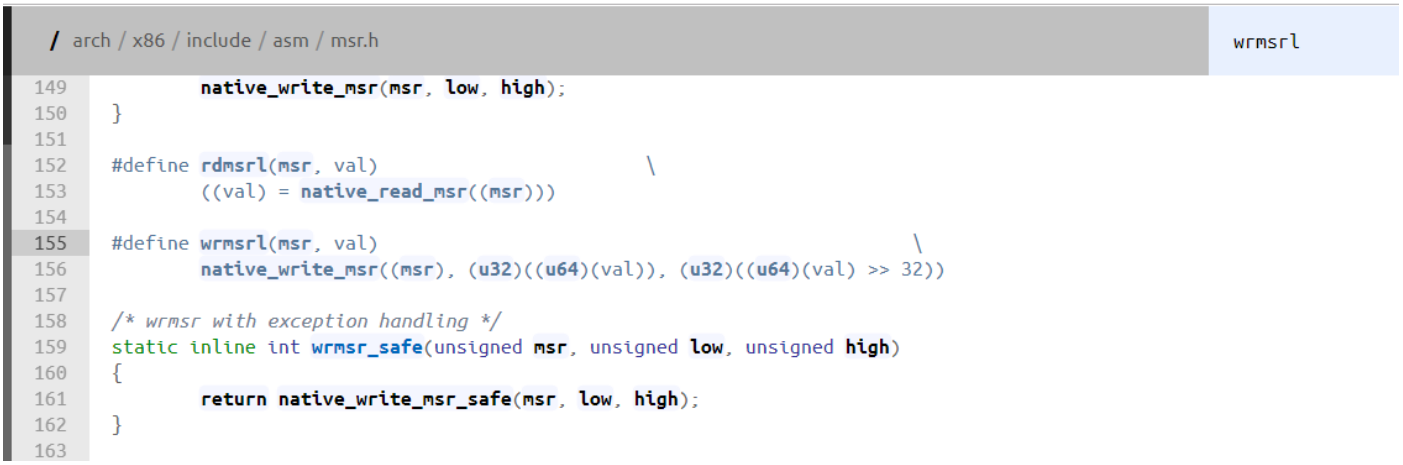
(iii)在 cpu_init()呼叫 syscall_init()



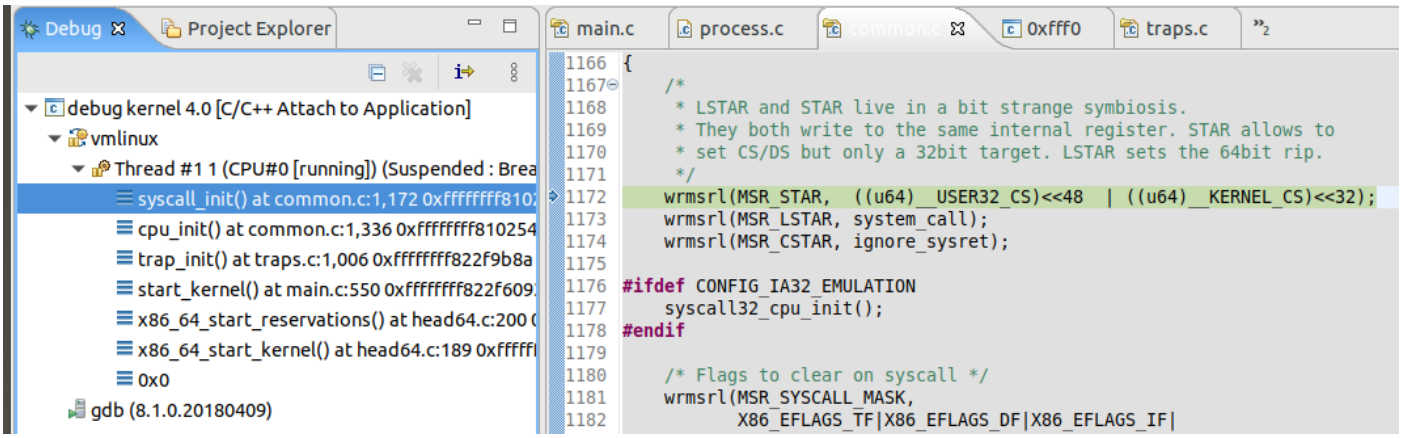
(iv) syscall_init()用 wrmsrl(MSR_STAR, ((u64)__USER32_CS)<<48 | ((u64)__KERNEL_CS)<<32);



Google 結果是 msr 是 machine specific register ##不是每個 CPU 都有

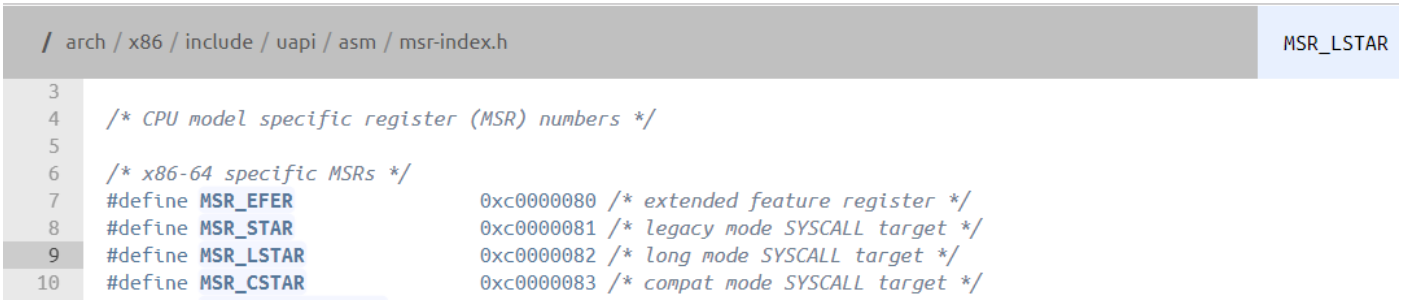


(v)wrmsrl(MSR_LSTAR, system_call);// 將 system_call 放到 MSR_LSTAR

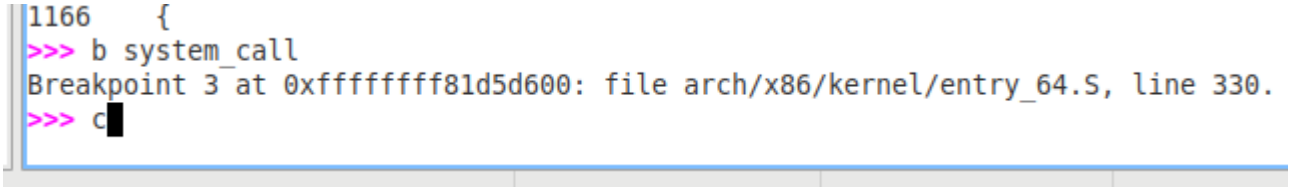


Google 發現進入點在#define MSR_LSTAR 0xc0000082 /* long mode SYSCALL target */

Long mode 64 位元 ；compat mode 為 32 位元

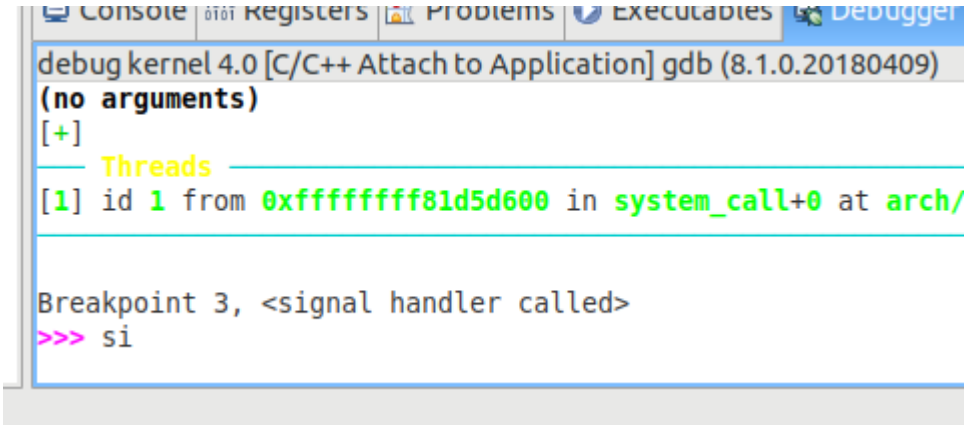


(vi)b system_call



2. Linux kernel 如何使用 rax 暫存器呼叫對應的 system call 處理函數？

(i)輸入 si 看組合語言



(ii)看到右邊有 330 system_call

```
Variables  Breakpoints  Expressions  Modules  Disassembly
Enter location here
ffffffff81d5d5e3:   mov     0x20(%rsp),%rbp
ffffffff81d5d5e8:   mov     0x28(%rsp),%rbx
ffffffff81d5d5ed:   add     $0x30,%rsp
 288                  jmp int_ret_from_sys_call
ffffffff81d5d5f1:   jmpq    0xffffffff81d5d81b <system_call+539>
ffffffff81d5d5f6:   nopw    %cs:0x0(%rax,%rax,1)
 330                  SWAPGS_UNSAFE_STACK
                  system_call:
ffffffff81d5d600:   swapgs
 338                  movq    %rsp,PER_CPU_VAR(old_rsp)
ffffffff81d5d603:   mov     %rsp,%gs:0xa180
 339                  movq    PER_CPU_VAR(kernel_stack),%rsp
ffffffff81d5d60c:   mov     %gs:0xa940,%rsp
 344                  ENABLE_INTERRUPTS(CLBR_NONE)
ffffffff81d5d615:   sti
 345                  SAVE_ARGS 8, 0, rax_enosys=1
ffffffff81d5d616:   sub     $0x50,%rsp
```

(iii)把 rdi rsi rdx 放到 stack

```
 345                  SAVE_ARGS 8, 0, rax_enosys=1
ffffffff81d5d616:   sub     $0x50,%rsp
ffffffff81d5d61a:   mov     %rdi,0x40(%rsp)
ffffffff81d5d61f:   mov     %rsi,0x38(%rsp)
ffffffff81d5d624:   mov     %rdx,0x30(%rsp)
```

(iiii)檢查 rax 是否超過 355

```
 353                  cmpq $__NR_syscall_max,%rax
ffffffff81d5d660:   cmp     $0x142,%rax
 358                  ja ret_from_sys_call  /* and return regs->ax */
ffffffff81d5d666:   ja      0xffffffff81d5d677 <system_call+119>
 359                  movq %r10,%rcx
```

(iv)看 sys_call_table

```
ffffffff81d5d668:   mov     %r10,%rcx
 360                  call *sys_call_table(,%rax,8)  # XXX:    rip relative
ffffffff81d5d66b:   callq   *-0x7e1fe180(,%rax,8)
 361                  movq %rax,RAX-ARGOFFSET(%rsp)
```

(v)p sys_call_table # 看 sys_call_table

```
debug kernel 4.0 [C/C++ Attach to Application] gdb (8.1.0.20180409)
[1] id 1 from 0xffffffff81d5d603 in system_call+3 at arch/x86/kernel/entry_64.S:338

<signal handler called>
>>> p sys_call_table
$1 = {[0] = 0xffffffff81290835 <SyS_read>, [1] = 0xffffffff812908fe <SyS_write>, [2] = 0xffffffff8128ec31 <SyS_open>, [3] = 0xffffffff8128ee30 <SyS_close>, [4] = 0xffff
fff81298259 <SyS_newstat>, [5] = 0xffffffff812983dd <SyS_newfstat>, [6] = 0xffffffff812982d0 <SyS_newlstat>, [7] = 0xffffffff812b09f4 <SyS_poll>, [8] = 0xffffffff8128fa
5 <SyS_lseek>, [9] = 0xffffffff8100f4a2 <SyS_mmap>, [10] = 0xffffffff81248eb7 <SyS_mprotect>, [11] = 0xffffffff8124617f <SyS_munmap>, [12] = 0xffffffff812421b1 <SyS_brk
, [13] = 0xffffffff810a8654 <SyS_rt_sigaction>, [14] = 0xffffffff810a64d8 <SyS_rt_sigprocmask>, [15] = 0xffffffff81d5dc40 <stub_rt_sigreturn>, [16] = 0xffffffff812ad40e
```

(vi) info registers eax #印出 rax 63 號 sys_newuname 切換使用者模式

```
>>> info registers eax
eax            0x3f     63
>>>
```

```
[63] = 0xffffffff810ac660 <SyS_newuname>,
```

3. 去看 HW3 syscall 一開始是呼叫幾號 syscall 並且印出來

(i)先在 eclipse 輸入 file /home/ubuntu/vmlinux 按下 c

```
Memory Browser  Memory  Debugger Console
Debug Linux [C/C++ Attach to Application] gdb (8.1.0.20180409)
New UI allocated
GNU gdb (Ubuntu 8.1-0ubuntu3) 8.1.0.20180409-git
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb) 0x000000000000fff0 in ?? ()

>>> file /home/ubuntu/vmlinux
A program is being debugged already.
Are you sure you want to change the file? (y or n) y
Reading symbols from /home/ubuntu/vmlinux...done.
>>> c
```

(ii)dgb_qemu.sh 會開始跑到可以打字的位置，先準備好要跑的程式 但是先不要按下 enter

```
Please press Enter to activate this console.
/ #
/ # ls
bin             lib             proc            sys
dev             linuxrc         sbin            tmp
etc             mnt             sharedFolder    usr
/ # cd mnt/sharedFolder/hw03.syscall.asm/
/mnt/sharedFolder/hw03.syscall.asm # ./syscall
```

(iii)回到 eclipse 按下 ctrl+c 並輸入要中斷的位置 (b system_call)再輸入 c

```
debug kernel 4.0 [C/C++ Attach to Application] gdb (8.1.0.20180409)
Output/messages
```

```
debug kernel 4.0 [C/C++ Attach to Application] gdb (8.1.0.20180409)
[+]
Threads
[1] id 1 from 0xffffffff81017ebf in default_idle+79 at arch/x86/kernel/process.c:

default_idle () at arch/x86/kernel/process.c:315
315             trace_cpu_idle_rcuidle(PWR_EVENT_EXIT, smp_processor_id());
>>> b system_call
Breakpoint 1 at 0xffffffff81d5d600: file arch/x86/kernel/entry_64.S, line 330.
>>> c
```

(iv)回到 cmd 按 enter 會再跳回 eclipse
(v)輸入 c 按下 enter　再利用 info registers eax 檢查 rax 看是哪一號 syscall
(vi) 連續操作三次可得下圖
第一次 rax 為 0 是 sys_read（讀取輸入的指令）
第二次 rax 為 1 是 sys_write(將指令寫到其他位置，例如 history)
第三次 rax 為 2 是 sys_open（利用 sys_open 開啟檔案）

故執行後的第一個 syscall 是 sys_open

```
Breakpoint 1, <signal handler called>
>>> info registers eax
eax            0x0      0
```

```
Breakpoint 1, <signal handler called>
>>> info registers eax
eax            0x1      1
>>>
```

```
Breakpoint 1, <signal handler called>
>>> info registers eax
eax            0x2      2
>>>
```