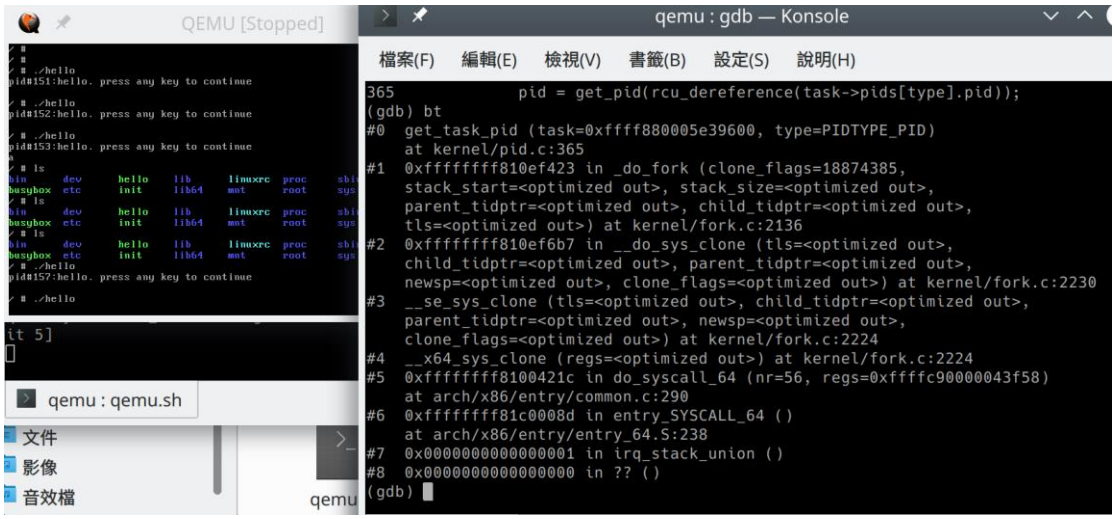
資工三 405235035 王博輝

作業系統概論 作業三

問題：請任選一個你覺得有趣的地方，列出該函數的呼叫者是誰，該函數又呼叫了哪些函數，並且說明該函數中各個程式區塊的約略功能。



設定完 gdb 及 QEMU 後，我嘗試追蹤了 get_pid 這個函數。

```
(gdb) bt
#0  get_task_pid (task=0xffff880005e39600, type=PIDTYPE_PID)
    at kernel/pid.c:365
#1  0xffffffff810ef423 in _do_fork (clone_flags=18874385,
    stack_start=<optimized out>, stack_size=<optimized out>,
    parent_tidptr=<optimized out>, child_tidptr=<optimized out>,
    tls=<optimized out>) at kernel/fork.c:2136
#2  0xffffffff810ef6b7 in __do_sys_clone (tls=<optimized out>,
    child_tidptr=<optimized out>, parent_tidptr=<optimized out>,
    newsp=<optimized out>, clone_flags=<optimized out>) at kernel/fork.c:2230
#3  __se_sys_clone (tls=<optimized out>, child_tidptr=<optimized out>,
    parent_tidptr=<optimized out>, newsp=<optimized out>,
    clone_flags=<optimized out>) at kernel/fork.c:2224
#4  __x64_sys_clone (regs=<optimized out>) at kernel/fork.c:2224
#5  0xffffffff8100421c in do_syscall_64 (nr=56, regs=0xffffc90000043f58)
    at arch/x86/entry/common.c:290
#6  0xffffffff81c0008d in entry_SYSCALL_64 ()
    at arch/x86/entry/entry_64.S:238
#7  0x0000000000000001 in irq_stack_union ()
#8  0x0000000000000000 in ?? ()
```

中斷點設在 get_pid 之後，在 debugee 執行 hello 時被觸發了。

Getpid 這個系統呼叫是由 entry_SYSCALL_64 出發

```
(gdb) up
#6  0xffffffff81c0008d in entry_SYSCALL_64 ()
    at arch/x86/entry/entry_64.S:238
238             call    do_syscall_64           /* returns with IRQs disabled
*/
```

經由 do_syscall_64

```
#5  0xffffffff8100421c in do_syscall_64 (nr=56, regs=0xffffc90000043f58)
    at arch/x86/entry/common.c:290
290                     regs->ax = sys_call_table[nr](regs);
```

將暫存器中的值先儲存起來

```
#4  __x64_sys_clone (regs=<optimized out>) at kernel/fork.c:2224
2224    SYSCALL_DEFINE5(clone, unsigned long, clone_flags, unsigned long, news
p,
```

```
#3  __se_sys_clone (tls=<optimized out>, child_tidptr=<optimized out>,
    parent_tidptr=<optimized out>, newsp=<optimized out>,
    clone_flags=<optimized out>) at kernel/fork.c:2224
2224    SYSCALL_DEFINE5(clone, unsigned long, clone_flags, unsigned long, news
```

```
#2  0xffffffff810ef6b7 in __do_sys_clone (tls=<optimized out>,
    child_tidptr=<optimized out>, parent_tidptr=<optimized out>,
    newsp=<optimized out>, clone_flags=<optimized out>) at kernel/fork.c:2230
2230            return _do_fork(clone_flags, newsp, 0, parent_tidptr, child_ti
dptr, tls);
```

呼叫__x64_sys_clone、__se_sys_clone、__do_sys_clone，想要複製一個行程出來。

```
#1  0xffffffff810ef423 in _do_fork (clone_flags=18874385,
    stack_start=<optimized out>, stack_size=<optimized out>,
    parent_tidptr=<optimized out>, child_tidptr=<optimized out>,
    tls=<optimized out>) at kernel/fork.c:2136
2136            pid = get_task_pid(p, PIDTYPE_PID);
```

```
Breakpoint 1, get_task_pid (task=0xffff880005e39600, type=PIDTYPE_PID)
    at kernel/pid.c:365
365             pid = get_pid(rcu_dereference(task->pids[type].pid));
```

複製出一個行程後，就讓這個行程緊接著呼叫 get_task_pid，完成此次的系統呼叫。

經過這兩次的作業發現，每個系統呼叫其實都會經由_do_fork 所產生，因為每一個呼叫都是一個行程，自然呼要產生一個子行程來管

理。