HW4

反組譯後: 408410094 葉X勛

```
10          __asm__ volatile (
   0x0000000000401d77 <+66>:    lea    -0x420(%rbp),%rsi
   0x0000000000401d7e <+73>:    mov    $0x0,%rax
   0x0000000000401d85 <+80>:    mov    $0x0,%rdi
   0x0000000000401d8c <+87>:    mov    %rsi,%rsi
   0x0000000000401d8f <+90>:    mov    -0x42c(%rbp),%rdx
--Type <RET> for more, q to quit, c to continue without paging--
   0x0000000000401d96 <+97>:    syscall
   0x0000000000401d98 <+99>:    mov    %rax,-0x428(%rbp)

11              "mov $0, %%rax\n"    //system call number:sys_read
12              "mov $0, %%rdi\n"    //stdin
13              "mov %1, %%rsi\n"    //buffer
14              "mov %2, %%rdx\n"    //max_buffer_size
15              "syscall\n"
16              "mov %%rax, %0"      //save return to %0
17              :  "=m"(ret)
18              : "g" (hello), "g" (len)
19              : "rax", "rbx", "rcx", "rdx");
```

:可發現反組譯後的執行順序與本來執行順序沒什麼變，其中rbp為堆疊的開頭、rbp-0X420為hello的位址、rbp-0X42c為len的位址，而rbp-0X428為ret的位址。<+66>對應到原本程式的13行，而<+73>、<+80>、<+90>分別對應到11、12、14行。

```
21          printf("輸入的字元為:%c\n",hello[0]);
   0x0000000000401d9f <+106>:   movzbl -0x420(%rbp),%eax
   0x0000000000401da6 <+113>:   movsbl %al,%eax
   0x0000000000401da9 <+116>:   mov    %eax,%esi
   0x0000000000401dab <+118>:   lea    0x93279(%rip),%rdi
   0x0000000000401db2 <+125>:   mov    $0x0,%eax
   0x0000000000401db7 <+130>:   callq  0x410c80 <printf>
```

:反組譯後可發現實際上會藉由callq去call C libraries來進行write的動作，而也可以發現在組語中write的編號及所用的暫存器與sys_write有所差異:前者為將0存入eax暫存器，後者為將1存入rax暫存器，兩者不一致。