作業 3 報告

408410098 資工三 蔡×祥

反組譯後，在 main 裡面執行第一行程式之前，有以下行為:

endbr64： 大概是說將選則 64bit 架構的這個分支

push %rbp： 將 rbp 裡面的值，為 main 函式的 return address 存入 stack
（rbp 暫存器指向這個 function 所擁有的 stack 的 high bit part;rsp 則指向 low bit part）

mov %rsp,%rbp：將 rsp 的值複製到 rbp，如此一來，rbp 就也指向 stack 的 low bit part

sub $0x28,%rsp：將 rsp 往 low bit part 的方向移動 28，則目前 rbp 跟 rsp 的空間就是 main 用到的 stack 空間

等到 main return 時:

add $0x28,%rsp 將 rsp 還原
retq 則依照 rsp 紀錄的 return address 返回。

（以上是我 google + 自己的理解，如果有錯還請麻煩老師、助教糾正我一下，謝謝）

而 sys_read 那段組語的意思則是:

rax 給 0 是 syscall 裡面的 sys_read
rdi 給 0 是指要從 stdin 讀取
rsi rdx 則分別給 buffer (buffer) 跟 buffersize (len)

最後回傳值放在 ret 裡面（從 rax output）

反組譯結果:

```
        int main(){
0x0000000000401ce5 <+0>:     endbr64
0x0000000000401ce9 <+4>:     push   %rbp
0x0000000000401cea <+5>:     mov    %rsp,%rbp
0x0000000000401ced <+8>:     push   %rbx
0x0000000000401cee <+9>:     sub    $0x28,%rsp
0x0000000000401cf2 <+13>:    mov    %fs:0x28,%rax
0x0000000000401cfb <+22>:    mov    %rax,-0x18(%rbp)
0x0000000000401cff <+26>:    xor    %eax,%eax

            char *buffer, *ret;
            long int len = 1;
0x0000000000401d01 <+28>:    movq   $0x1,-0x20(%rbp)


            __asm__ volatile(
0x0000000000401d09 <+36>:    mov    $0x0,%rax
0x0000000000401d10 <+43>:    mov    $0x0,%rdi
0x0000000000401d17 <+50>:    mov    -0x30(%rbp),%rsi
0x0000000000401d1b <+54>:    mov    -0x20(%rbp),%rdx
0x0000000000401d1f <+58>:    syscall
0x0000000000401d21 <+60>:    mov    %rax,-0x28(%rbp)


                "mov $0, %%rax\n" //sys_read: rax:0
                "mov $0, %%rdi\n"//rdi:file identity
                "mov %1, %%rsi\n"//rsi: char *buffer
                "mov %2, %%rdx\n"//rdx: buffer size
                "syscall\n"
                "mov %%rax, %0"
                : "=m"(ret)
                : "m" (buffer), "g" (len)
                : "rax", "rbx", "rcx", "rdx"
            );

            printf("讀入的字元為 \"%c\" \n", *buffer);
0x0000000000401d25 <+64>:    mov    -0x30(%rbp),%rax
0x0000000000401d29 <+68>:    movzbl (%rax),%eax
0x0000000000401d2c <+71>:    movsbl %al,%eax
0x0000000000401d2f <+74>:    mov    %eax,%esi
0x0000000000401d31 <+76>:    lea    0x932cc(%rip),%rdi        # 0x495004
0x0000000000401d38 <+83>:    mov    $0x0,%eax
0x0000000000401d3d <+88>:    callq  0x410bf0 <printf>

            return 0;
```

```
   0x0000000000401d42 <+93>:    mov    $0x0,%eax
}

   0x0000000000401d47 <+98>:    mov    -0x18(%rbp),%rdx
   0x0000000000401d4b <+102>:   xor    %fs:0x28,%rdx
   0x0000000000401d54 <+111>:   je     0x401d5b <main+118>
   0x0000000000401d56 <+113>:   callq  0x4542a0 <__stack_chk_fail_local>
   0x0000000000401d5b <+118>:   add    $0x28,%rsp
   0x0000000000401d5f <+122>:   pop    %rbx
   0x0000000000401d60 <+123>:   pop    %rbp
   0x0000000000401d61 <+124>:   retq
```