408410007 鄭 O 云
甲:
執行環境:
cpu: Intel® Core™ i5-5250U CPU @ 1.60GHz
os:Ubuntu 20.04.2

乙:
1. ./peterson_trival-g 會不斷出現:

```
p0及 p1都在 critical section
p0及 p1都在 critical section
p0及 p1都在 critical section
p0及 p1都在 critical section
p0及 p1都在 critical section
p0及 p1都在 critical section
p0及 p1都在 critical section
p0及 p1都在 critical section
p0及 p1都在 critical section
p0及 p1都在 critical section
p0及 p1都在 critical section
p0及 p1都在 critical section
p0及 p1都在 critical section
p0及 p1都在 critical section
p0及 p1都在 critical section
p0及 p1都在 critical section
p0及 p1都在 critical section
p0及 p1?^C
```

可以得知 p0,p1 兩個都一直在 CS 裡面，修改下程式碼，就可以觀察 per_second（每秒鐘印出 P0 和 P1 進入 CS 的次數）

```
shiwulo@vm:~/downloads/osdi-hw7$ ./peterson_trival-g
p0: start
p1: start
進入次數（每秒） p0: 3947517, p1: 3943511, 分別執行於 core#3 及 core#1
進入次數（每秒） p0: 3923506, p1: 3917763, 分別執行於 core#3 及 core#1
進入次數（每秒） p0: 3971251, p1: 3966021, 分別執行於 core#3 及 core#1
進入次數（每秒） p0: 3946087, p1: 3944753, 分別執行於 core#3 及 core#1
進入次數（每秒） p0: 3967708, p1: 3967981, 分別執行於 core#3 及 core#1
進入次數（每秒） p0: 4005188, p1: 4005127, 分別執行於 core#3 及 core#1
進入次數（每秒） p0: 4013678, p1: 4013754, 分別執行於 core#3 及 core#1
^C
```

./peterson_trival-g

```
shiwulo@vm:~/downloads/osdi-hw7$ ./peterson_trival-O3
p0: start
p1: start
進入次數（每秒）p0: 15628, p1:      0, 分別執行於 core#1 及 core#0
進入次數（每秒）p0:      0, p1:      0, 分別執行於 core#1 及 core#0
進入次數（每秒）p0:      0, p1:      0, 分別執行於 core#1 及 core#0
進入次數（每秒）p0:      0, p1:      0, 分別執行於 core#1 及 core#0
進入次數（每秒）p0:      0, p1:      0, 分別執行於 core#1 及 core#0
進入次數（每秒）p0:      0, p1:      0, 分別執行於 core#1 及 core#0
進入次數（每秒）p0:      0, p1:      0, 分別執行於 core#1 及 core#0
進入次數（每秒）p0:      0, p1:      0, 分別執行於 core#1 及 core#0
進入次數（每秒）p0:      0, p1:      0, 分別執行於 core#1 及 core#0
進入次數（每秒）p0:      0, p1:      0, 分別執行於 core#1 及 core#0
進入次數（每秒）p0:      0, p1:      0, 分別執行於 core#1 及 core#0
進入次數（每秒）p0:      0, p1:      0, 分別執行於 core#1 及 core#0
進入次數（每秒）p0:      0, p1:      0, 分別執行於 core#1 及 core#0
進入次數（每秒）p0:      0, p1:      0, 分別執行於 core#1 及 core#0
進入次數（每秒）p0:      0, p1:      0, 分別執行於 core#1 及 core#0
進入次數（每秒）p0:      0, p1:      0, 分別執行於 core#1 及 core#0
^C
```

./peterson_correct-g

```
shiwulo@vm:~/downloads/osdi-hw7$ ./peterson_correct-g
start p0
start p1
進入次數（每秒）p0: 1936331, p1: 1900625, 分別執行於 core#0 及 core#3
進入次數（每秒）p0: 2438468, p1: 2461541, 分別執行於 core#0 及 core#3
進入次數（每秒）p0: 2452350, p1: 2458217, 分別執行於 core#0 及 core#3
進入次數（每秒）p0: 2420794, p1: 2436824, 分別執行於 core#0 及 core#3
進入次數（每秒）p0: 2504810, p1: 2483764, 分別執行於 core#0 及 core#3
進入次數（每秒）p0: 2444461, p1: 2445462, 分別執行於 core#0 及 core#3
進入次數（每秒）p0: 2441878, p1: 2453906, 分別執行於 core#0 及 core#3
進入次數（每秒）p0: 2437018, p1: 2455956, 分別執行於 core#0 及 core#3
進入次數（每秒）p0: 2481134, p1: 2475596, 分別執行於 core#0 及 core#3
進入次數（每秒）p0: 2488907, p1: 2515290, 分別執行於 core#0 及 core#3
進入次數（每秒）p0: 2447897, p1: 2460462, 分別執行於 core#0 及 core#3
進入次數（每秒）p0: 2326866, p1: 2345183, 分別執行於 core#0 及 core#3
進入次數（每秒）p0: 2277060, p1: 2269575, 分別執行於 core#0 及 core#3
進入次數（每秒）p0: 2409973, p1: 2396971, 分別執行於 core#0 及 core#3
^C
```

./peterson_correct-O3

```
shiwulo@vm:~/downloads/osdi-hw7$ ./peterson_correct-O3
start p0
start p1
進入次數（每秒） p0: 3007707, p1: 2945060, 分別執行於 core#2 及 core#0
進入次數（每秒） p0: 3042904, p1: 2970425, 分別執行於 core#2 及 core#0
進入次數（每秒） p0: 2386892, p1: 2388129, 分別執行於 core#2 及 core#0
進入次數（每秒） p0: 2470089, p1: 2442281, 分別執行於 core#2 及 core#0
進入次數（每秒） p0: 2399178, p1: 2391915, 分別執行於 core#2 及 core#0
進入次數（每秒） p0: 2439658, p1: 2400307, 分別執行於 core#2 及 core#0
進入次數（每秒） p0: 2481588, p1: 2483262, 分別執行於 core#2 及 core#0
進入次數（每秒） p0: 2433466, p1: 2409470, 分別執行於 core#2 及 core#0
進入次數（每秒） p0: 2602932, p1: 2605197, 分別執行於 core#2 及 core#0
進入次數（每秒） p0: 2529386, p1: 2480860, 分別執行於 core#2 及 core#0
進入次數（每秒） p0: 2854838, p1: 2824681, 分別執行於 core#2 及 core#0
進入次數（每秒） p0: 2447310, p1: 2439555, 分別執行於 core#2 及 core#0
進入次數（每秒） p0: 2479997, p1: 2486002, 分別執行於 core#2 及 core#0
進入次數（每秒） p0: 2443429, p1: 2387529, 分別執行於 core#2 及 core#0
進入次數（每秒） p0: 2357325, p1: 2328903, 分別執行於 core#2 及 core#0
^C
```

2.

```
(gdb) disass /m p0
Dump of assembler code for function p0:
   0x00000000000012d0 <+0>:     lea    0xd89(%rip),%rdi        # 0x2060
   0x00000000000012d7 <+7>:     sub    $0x8,%rsp
   0x00000000000012db <+11>:    callq  0x1040 <puts@plt>
   0x00000000000012e0 <+16>:    cmpl   $0x1,0x2d4d(%rip)        # 0x4034 <flag1>
   0x00000000000012e7 <+23>:    movl   $0x1,0x2d3f(%rip)        # 0x4030 <flag0>
   0x00000000000012f1 <+33>:    movl   $0x1,0x2d3d(%rip)        # 0x4038 <turn>
   0x00000000000012fb <+43>:    jne    0x1300 <p0+48>
   0x00000000000012fd <+45>:    jmp    0x12fd <p0+45>
   0x00000000000012ff <+47>:    nop
   0x0000000000001300 <+48>:    callq  0x10a0 <sched_getcpu@plt>
   0x0000000000001305 <+53>:    addl   $0x1,0x2d18(%rip)        # 0x4024 <p0_in_cs>
   0x000000000000130c <+60>:    mov    %eax,0x2d2a(%rip)        # 0x403c <cpu_p0>
   0x0000000000001312 <+66>:    jmp    0x12e0 <p0+16>
End of assembler dump.
```

本來程式碼為:
flag0 = 1;
turn = 1;
while (flag1==1 && turn==1)
        ;
<+16>那行:
但因為優化的目的是要把效能變好，速度變快，所以把比較慢的 cmp 往前移，但在 multi-thread 上這樣優化就會使其錯誤了。

3.
在我的電腦上，peterson_trival-g 會比 peterson_correct-O3 快。
peterson_trival-g 是錯的，他會讓兩個 process 都在 cs 裡面，相依於 gcc 版本或比較老的 cpu

4.
反組譯 peterson_trival-g

```
(gdb) disass /m p0
Dump of assembler code for function p0:
36      void p0(void) {
   0x0000000000001234 <+0>:     push   %rbp
   0x0000000000001235 <+1>:     mov    %rsp,%rbp

37          printf("p0: start\n");
   0x0000000000001238 <+4>:     lea    0xe21(%rip),%rdi        # 0x2060
   0x000000000000123f <+11>:    callq  0x1040 <puts@plt>

38          while (1) {
39              //🐉 🐲 🌵 🎄 🌲 🌳 🌴 🌱 🌿 ☘🍀
40                      //Peteron's solution的進去部分的程式碼
41              flag0 = 1;
   0x0000000000001244 <+16>:    movl   $0x1,0x2dde(%rip)        # 0x402c <flag0>
   0x00000000000012b1 <+125>:   jmp    0x1244 <p0+16>

42              turn = 1;
   0x000000000000124e <+26>:    movl   $0x1,0x2dcc(%rip)        # 0x4024 <turn>

43              while (flag1==1 && turn==1)
   0x0000000000001258 <+36>:    nop
   0x0000000000001259 <+37>:    mov    0x2dc9(%rip),%eax        # 0x4028 <flag1>
   0x000000000000125f <+43>:    cmp    $0x1,%eax
   0x0000000000001262 <+46>:    jne    0x126f <p0+59>
   0x0000000000001264 <+48>:    mov    0x2dba(%rip),%eax        # 0x4024 <turn>
   0x000000000000126a <+54>:    cmp    $0x1,%eax
   0x000000000000126d <+57>:    je     0x1259 <p0+37>

44                  ;   //waiting
```

反組譯 peterson_correct-O3

```
(gdb) disass /m p0
Dump of assembler code for function p0:
   0x00000000000012f0 <+0>:     lea    0xd88(%rip),%rdi        # 0x207f
   0x00000000000012f7 <+7>:     sub    $0x8,%rsp
   0x00000000000012fb <+11>:    callq  0x1040 <puts@plt>
   0x0000000000001300 <+16>:    movl   $0x1,0x2d36(%rip)        # 0x4040 <flag>
   0x000000000000130a <+26>:    mfence
   0x000000000000130d <+29>:    mfence
   0x0000000000001310 <+32>:    movl   $0x1,0x2d2e(%rip)        # 0x4048 <turn>
   0x000000000000131a <+42>:    mfence
   0x000000000000131d <+45>:    jmp    0x132b <p0+59>
   0x000000000000131f <+47>:    nop
   0x0000000000001320 <+48>:    mov    0x2d22(%rip),%eax        # 0x4048 <turn>
   0x0000000000001326 <+54>:    cmp    $0x1,%eax
   0x0000000000001329 <+57>:    jne    0x1335 <p0+69>
   0x000000000000132b <+59>:    mov    0x2d13(%rip),%eax        # 0x4044 <flag+4>
   0x0000000000001331 <+65>:    test   %eax,%eax
   0x0000000000001333 <+67>:    jne    0x1320 <p0+48>
   0x0000000000001335 <+69>:    callq  0x10a0 <sched_getcpu@plt>
   0x000000000000133a <+74>:    mov    %eax,0x2d14(%rip)        # 0x4054 <cpu_p0>
   0x0000000000001340 <+80>:    mov    0x2cf2(%rip),%eax        # 0x4038 <in_cs>
   0x0000000000001346 <+86>:    add    $0x1,%eax
   0x0000000000001349 <+89>:    cmp    $0x2,%eax
   0x000000000000134c <+92>:    mov    %eax,0x2ce6(%rip)        # 0x4038 <in_cs>
   0x0000000000001352 <+98>:    je     0x1378 <p0+136>
   0x0000000000001354 <+100>:   addl   $0x1,0x2cd5(%rip)        # 0x4030 <p0_in_cs>
   0x000000000000135b <+107>:   sub    $0x1,%eax
   0x000000000000135e <+110>:   mov    %eax,0x2cd4(%rip)        # 0x4038 <in_cs>
   0x0000000000001364 <+116>:   movl   $0x0,0x2cd2(%rip)        # 0x4040 <flag>
   0x000000000000136e <+126>:   mfence
   0x0000000000001371 <+129>:   jmp    0x1300 <p0+16>
```

從反組譯可看出， peterson_correct-O3 會多使用 atomic_store 及 mfence（可以確保在 mfence 前面與後面的程式碼不會因為被優化而延後或提前，保持程式碼順序）。
並且老師在上課提到，memory_order_seq_cst 是個很強的 fence，對硬體的負擔太重，所以導致 O3 會比較慢。