問題一：執行結果

peterson_trival-O3

```
➜ ~ ./peterson_trival-O3
p0: start
p1: start
進入次數 (每秒) p0:  1025, p1:     0，分別執行於 core#0 及 core#0
進入次數 (每秒) p0:     0, p1:     0，分別執行於 core#0 及 core#0
進入次數 (每秒) p0:     0, p1:     0，分別執行於 core#0 及 core#0
進入次數 (每秒) p0:     0, p1:     0，分別執行於 core#0 及 core#0
進入次數 (每秒) p0:     0, p1:     0，分別執行於 core#0 及 core#0
進入次數 (每秒) p0:     0, p1:     0，分別執行於 core#0 及 core#0
```

peterson_trival-g

```
➜ ~ ./peterson_trival-g
p0: start
p1: start
進入次數 (每秒) p0: 7641628, p1: 7639028，分別執行於 core#0 及 core#1
進入次數 (每秒) p0: 8839657, p1: 8841647，分別執行於 core#0 及 core#1
進入次數 (每秒) p0: 8052182, p1: 8058833，分別執行於 core#0 及 core#1
進入次數 (每秒) p0: 9009540, p1: 9009573，分別執行於 core#0 及 core#1
進入次數 (每秒) p0: 7765154, p1: 7765379，分別執行於 core#0 及 core#1
進入次數 (每秒) p0: 8308299, p1: 8311669，分別執行於 core#0 及 core#1
```

peterson_correct-O3

```
➜ ~ ./peterson_correct-O3
start p0
start p1
進入次數 (每秒) p0: 5067232, p1: 5021216，分別執行於 core#0 及 core#1
進入次數 (每秒) p0: 5048634, p1: 5073092，分別執行於 core#0 及 core#1
進入次數 (每秒) p0: 5394884, p1: 5441859，分別執行於 core#0 及 core#1
進入次數 (每秒) p0: 5139053, p1: 5139863，分別執行於 core#0 及 core#1
進入次數 (每秒) p0: 4290660, p1: 4349524，分別執行於 core#0 及 core#1
進入次數 (每秒) p0: 4605892, p1: 4620329，分別執行於 core#0 及 core#1
```

peterson_correct-g

```
➜ ~ ./peterson_correct-g
start p0
start p1
進入次數 (每秒) p0: 5309824, p1: 5296728，分別執行於 core#1 及 core#0
進入次數 (每秒) p0: 5010626, p1: 4965119，分別執行於 core#1 及 core#0
進入次數 (每秒) p0: 4738781, p1: 4714339，分別執行於 core#1 及 core#0
進入次數 (每秒) p0: 5957443, p1: 5951334，分別執行於 core#1 及 core#0
進入次數 (每秒) p0: 5036427, p1: 4979946，分別執行於 core#1 及 core#0
進入次數 (每秒) p0: 5501036, p1: 5471138，分別執行於 core#1 及 core#0
進入次數 (每秒) p0: 4918802, p1: 4881639，分別執行於 core#1 及 core#0
```

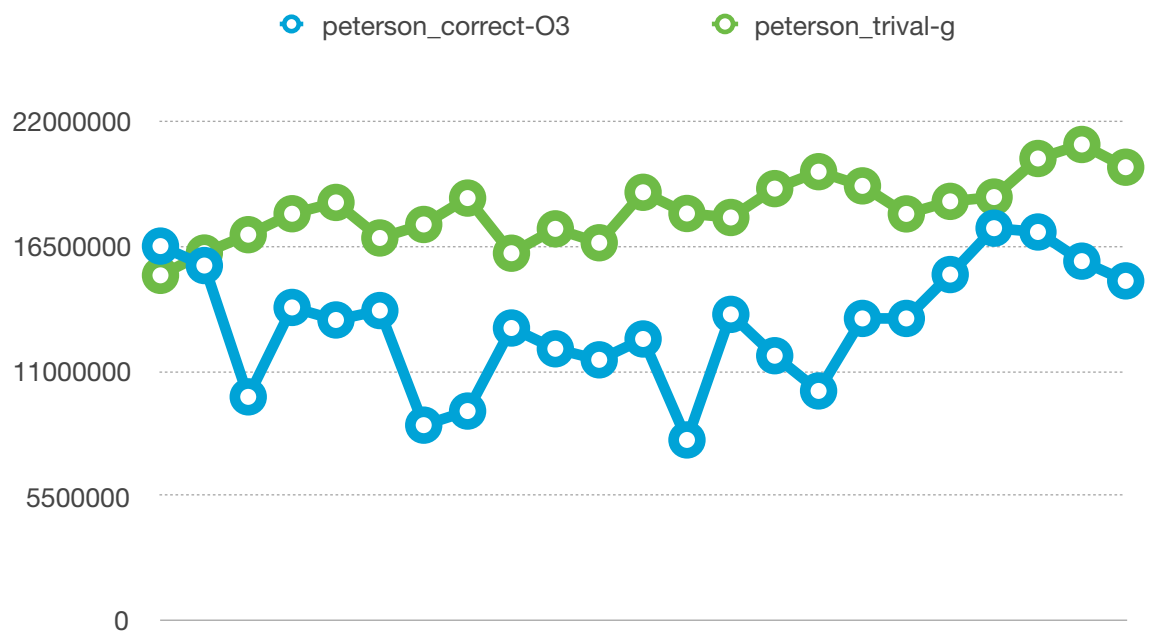
問題二：「為什麼」 peterson_trival-O3 的執行結果是錯的？

開啟-O3參數會錯是因為cmp是效能瓶頸，在組譯的時候會被往前移到設定的flag0前面，而造成執行結果錯誤。

問題三：請說明在你的電腦上，上述二個程式的正確與否，並說明速度的快慢

CPU:intel Core i5-7360U　　GCC:gcc-7.4.0

兩組程式都正確，根據電腦執行結果，peterson_trival-g 較 peterson_correct-O3快



問題四：使用反組譯解釋前述之結果，盡可能的解釋即可， 最起碼附上每一行組語的意義

原本CPU或是編譯器對記憶體操作時會採用亂序執行的方式已達到提高效能的目的，但是 peterson_correct-O3 使用了mfence，讓CPU或記憶體按照順序來執行，因而造成peterson_trival-g 較 peterson_correct-O3 快的結果。

peterson_trival-g

```
0x00000000000009b4 <+0>:    push   %rbp
0x00000000000009b5 <+1>:    mov    %rsp,%rbp
0x00000000000009b8 <+4>:    lea    0x271(%rip),%rdi        # 0xc30
0x00000000000009bf <+11>:   callq  0x7b0 <puts@plt>
0x00000000000009c4 <+16>:   movl   $0x1,0x201666(%rip)        # 0x202034 <flag0>        flag0=1;
0x00000000000009ce <+26>:   movl   $0x1,0x201654(%rip)        # 0x20202c <turn>         turn = 1
0x00000000000009d8 <+36>:   nop
0x00000000000009d9 <+37>:   mov    0x201651(%rip),%eax        # 0x202030 <flag1>        while(flag1==1&& turn==1) ;
0x00000000000009df <+43>:   cmp    $0x1,%eax
0x00000000000009e2 <+46>:   jne    0x9ef <p0+59>
0x00000000000009e4 <+48>:   mov    0x201642(%rip),%eax        # 0x20202c <turn>
0x00000000000009ea <+54>:   cmp    $0x1,%eax
0x00000000000009ed <+57>:   je     0x9d9 <p0+37>
0x00000000000009ef <+59>:   movl   $0x0,0x20163b(%rip)        # 0x202034 <flag0>.       flag0 =0
0x00000000000009f9 <+69>:   jmp    0x9c4 <p0+16>
```

peterson_correct-O3

```
0x0000000000000a80 <+0>:  lea    0x209(%rip),%rdi        # 0xc90
0x0000000000000a87 <+7>:  sub    $0x8,%rsp
0x0000000000000a8b <+11>:  callq  0x7d0 <puts@plt>
0x0000000000000a90 <+16>:  movl   $0x1,0x2015a6(%rip)        # 0x202040 <flag>  atomic_store(&flag[0], 1);
0x0000000000000a9a <+26>:  mfence
0x0000000000000a9d <+29>:  mfence                           atomic_thread_fence(memory_order_seq_cst);
0x0000000000000aa0 <+32>:  movl   $0x1,0x20159e(%rip)        # 0x202048 <turn> atomic_store(&turn, 1);
0x0000000000000aaa <+42>:  mfence
0x0000000000000aad <+45>:  jmp    0xabb <p0+59>
0x0000000000000aaf <+47>:   nop
0x0000000000000ab0 <+48>:  mov    0x201592(%rip),%eax       # 0x202048 <turn>  while(atomic_load(&flag[1])&&
                                                                    atomic_load(&turn)==1);
0x0000000000000ab6 <+54>:  cmp    $0x1,%eax
0x0000000000000ab9 <+57>:  jne    0xac5 <p0+69>
0x0000000000000abb <+59>:  mov    0x201583(%rip),%eax       # 0x202044 <flag+4>
0x0000000000000ac1 <+65>:  test   %eax,%eax
0x0000000000000ac3 <+67>:  jne    0xab0 <p0+48>
0x0000000000000ac5 <+69>:  movl   $0x0,0x201571(%rip)        # 0x202040 <flag>  atomic_store(&flag[0],0);
0x0000000000000acf <+79>:   mfence
0x0000000000000ad2 <+82>:  jmp    0xa90 <p0+16>
```