tags: OS

HW7

408410094 資工三 葉X勛

1. 執行make，將產生的四個執行檔案的執行結果截圖。
2. 「確實的」解釋「為什麼」peterson_trival-O3的執行結果是錯的。
3. 請問在你的電腦上「peterson_trival-g」的速度比「peterson_correct-O3」快或者是慢?上述兩個程式的正確與否?
4. 「確實的」解釋「3.」，某個程式比另一個程式快或者慢的理由。

執行環境:Win11的WSL CPU:Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz(8 CPUs), ~1.8GHz

1.執行make，將產生的四個執行檔案的執行結果截圖。

peterson_trival-g

```
❯ ./peterson_trival-g
p0: start
p1: start
進入次數(每秒)p0: 6735147, p1: 6729483，分別執行於 core#3 及 core#7
進入次數(每秒)p0: 13874316, p1: 7138670，分別執行於 core#3 及 core#7
進入次數(每秒)p0: 8070733, p1: 8071578，分別執行於 core#3 及 core#7
進入次數(每秒)p0: 7071483, p1: 7070764，分別執行於 core#3 及 core#7
進入次數(每秒)p0: 8026686, p1: 8026673，分別執行於 core#3 及 core#7
進入次數(每秒)p0: 8245484, p1: 8245529，分別執行於 core#3 及 core#7
進入次數(每秒)p0: 8365181, p1: 8365064，分別執行於 core#3 及 core#7
進入次數(每秒)p0: 8005162, p1: 8005216，分別執行於 core#3 及 core#7
進入次數(每秒)p0: 8117753, p1: 8117466，分別執行於 core#3 及 core#7
進入次數(每秒)p0: 16564909, p1: 8446906，分別執行於 core#3 及 core#7
進入次數(每秒)p0: 9063213, p1: 9063218，分別執行於 core#3 及 core#7
進入次數(每秒)p0: 7067052, p1: 7067053，分別執行於 core#3 及 core#7
進入次數(每秒)p0: 7901578, p1: 7901856，分別執行於 core#3 及 core#7
進入次數(每秒)p0: 8905140, p1: 8905357，分別執行於 core#3 及 core#7
進入次數(每秒)p0: 7682617, p1: 7680447，分別執行於 core#3 及 core#7
進入次數(每秒)p0: 7666107, p1: 7666486，分別執行於 core#3 及 core#7
```

:執行到後期會一直有p0,p1同時在critical section的情況發生。

peterson_trival-O3

```
❯ ./peterson_trival-O3
p0: start
p1: start
進入次數(每秒)p0:   411, 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
進入次數(每秒)p0:     0, p1:     0，分別執行於 core#1 及 core#0
```

:產生deadlock。

peterson_correct-g

```
❯ ./peterson_correct-g
start p0
start p1
進入次數(每秒)p0: 4162513, p1: 4153920，分別執行於 core#1 及 core#3
進入次數(每秒)p0: 4380686, p1: 4379251，分別執行於 core#1 及 core#3
進入次數(每秒)p0: 4241228, p1: 4244185，分別執行於 core#1 及 core#3
進入次數(每秒)p0: 4743709, p1: 4736815，分別執行於 core#1 及 core#3
進入次數(每秒)p0: 4086939, p1: 4088620，分別執行於 core#1 及 core#3
進入次數(每秒)p0: 4283831, p1: 4280849，分別執行於 core#1 及 core#3
進入次數(每秒)p0: 4541127, p1: 4537634，分別執行於 core#1 及 core#3
進入次數(每秒)p0: 4383726, p1: 4378944，分別執行於 core#1 及 core#3
進入次數(每秒)p0: 4114427, p1: 4122975，分別執行於 core#1 及 core#3
進入次數(每秒)p0: 4157818, p1: 4139777，分別執行於 core#1 及 core#3
進入次數(每秒)p0: 4486941, p1: 4481785，分別執行於 core#1 及 core#3
進入次數(每秒)p0: 4454339, p1: 4453787，分別執行於 core#1 及 core#3
進入次數(每秒)p0: 4556540, p1: 4557812，分別執行於 core#1 及 core#3
進入次數(每秒)p0: 4819590, p1: 4820360，分別執行於 core#1 及 core#3
進入次數(每秒)p0: 4457490, p1: 4458842，分別執行於 core#1 及 core#3
進入次數(每秒)p0: 4373797, p1: 4377847，分別執行於 core#1 及 core#3
進入次數(每秒)p0: 4620563, p1: 4619606，分別執行於 core#1 及 core#3
```

peterson_correct-O3

```
❯ ./peterson_correct-O3
start p0
start p1
進入次數(每秒) p0: 4507874, p1: 4498479, 分別執行於 core#3 及 core#1
進入次數(每秒) p0: 4150350, p1: 4166119, 分別執行於 core#3 及 core#1
進入次數(每秒) p0: 4299683, p1: 4287841, 分別執行於 core#3 及 core#1
進入次數(每秒) p0: 4442867, p1: 4450773, 分別執行於 core#3 及 core#1
進入次數(每秒) p0: 4465949, p1: 4459374, 分別執行於 core#3 及 core#1
進入次數(每秒) p0: 4460128, p1: 4460977, 分別執行於 core#3 及 core#1
進入次數(每秒) p0: 4079133, p1: 4072900, 分別執行於 core#3 及 core#1
進入次數(每秒) p0: 3600323, p1: 3594930, 分別執行於 core#3 及 core#1
進入次數(每秒) p0: 3805836, p1: 3801336, 分別執行於 core#3 及 core#1
進入次數(每秒) p0: 4370521, p1: 4401386, 分別執行於 core#3 及 core#1
進入次數(每秒) p0: 4563749, p1: 4566399, 分別執行於 core#3 及 core#1
進入次數(每秒) p0: 4193360, p1: 4191530, 分別執行於 core#3 及 core#1
進入次數(每秒) p0: 4291781, p1: 4287179, 分別執行於 core#3 及 core#1
進入次數(每秒) p0: 4234108, p1: 4223651, 分別執行於 core#3 及 core#1
進入次數(每秒) p0: 4163627, p1: 4160847, 分別執行於 core#3 及 core#1
進入次數(每秒) p0: 4022315, p1: 4015785, 分別執行於 core#3 及 core#1
```

2.「確實的」解釋「為什麼」peterson_trival-O3的執行結果是錯的。

由1.的截圖，可發現當p1進來的時候，p0、p1同時進入deadlock。以下為反組譯後p0的組語:

```
Reading symbols from peterson_trival-O3...
(No debugging symbols found in peterson_trival-O3)
(gdb) disass /m p0
Dump of assembler code for function p0:
   0x0000000000001360 <+0>:     endbr64
   0x0000000000001364 <+4>:     push   %rbx
   0x0000000000001365 <+5>:     lea    0xd13(%rip),%rdi        # 0x207f
   0x000000000000136c <+12>:    lea    0xced(%rip),%rbx        # 0x2060
   0x0000000000001373 <+19>:    callq  0x10e0 <puts@plt>
   0x0000000000001378 <+24>:    cmpl   $0x1,0x2cbd(%rip)        # 0x403c <flag1>
   0x000000000000137f <+31>:    movl   $0x1,0x2caf(%rip)        # 0x4038 <flag0>
   0x0000000000001389 <+41>:    movl   $0x1,0x2cad(%rip)        # 0x4040 <turn>
   0x0000000000001393 <+51>:    jne    0x13e4 <p0+132>
   0x0000000000001395 <+53>:    nopl   (%rax)
   0x0000000000001398 <+56>:    jmp    0x1398 <p0+56>
   0x000000000000139a <+58>:    nopw   0x0(%rax,%rax,1)
   0x00000000000013a0 <+64>:    mov    0x2c79(%rip),%rcx        # 0x4020 <stderr@@GLIBC_2.2.5>
   0x00000000000013a7 <+71>:    mov    $0x1e,%edx
   0x00000000000013ac <+76>:    mov    $0x1,%esi
   0x00000000000013b1 <+81>:    mov    %rbx,%rdi
   0x00000000000013b4 <+84>:    callq  0x1150 <fwrite@plt>
   0x00000000000013b9 <+89>:    addl   $0x1,0x2c6c(%rip)        # 0x402c <p0_in_cs>
   0x00000000000013c0 <+96>:    subl   $0x1,0x2c6d(%rip)        # 0x4034 <in_cs>
   0x00000000000013c7 <+103>:   cmpl   $0x1,0x2c6e(%rip)        # 0x403c <flag1>
   0x00000000000013ce <+110>:   movl   $0x1,0x2c60(%rip)        # 0x4038 <flag0>
   0x00000000000013d8 <+120>:   movl   $0x1,0x2c5e(%rip)        # 0x4040 <turn>
   0x00000000000013e2 <+130>:   je     0x1398 <p0+56>
   0x00000000000013e4 <+132>:   callq  0x1140 <sched_getcpu@plt>
   0x00000000000013e9 <+137>:   mov    0x2c45(%rip),%edx        # 0x4034 <in_cs>
   0x00000000000013ef <+143>:   mov    %eax,0x2c4f(%rip)        # 0x4044 <cpu_p0>
   0x00000000000013f5 <+149>:   lea    0x1(%rdx),%eax
   0x00000000000013f8 <+152>:   mov    %eax,0x2c36(%rip)        # 0x4034 <in_cs>
```

:可以發現在<+24>那行，gcc將turn是否=1的判斷優化掉了，只判斷flag1是否=1，其原因在於原本程式在進行判斷前就已經先將turn=1，而由於gcc不會知道這個程式為multithreading的程式，因此在優化時gcc會將turn=1進行優化，只剩下判斷flag1=1。在反組譯的組語中得知，當flag1為1的時候，會跑到<+56>: jmp 0x1398 <p0+56>這行，會一直跳回自己這行，因此發生了deadlock的現象。

3.請問在你的電腦上「peterson_trival-g」的速度比「peterson_correct-O3」快者是慢?上述兩個程式的正確與否?

由1.的截圖很明顯的可以發現前者的速度比後者還要快得多，但是peterson_trival-g是個錯誤的程式，因為他沒有使用atomic operation 來保護global varaible，導致turn在不同的thread中有不同的值，p0看到的turn可能是1，而p1看到的turn可能是0。而在peterson_correct-O3中，因為有使用了atomic operation對turn進行寫入，因此在不同的thread中看到的turn的值是一致的，確保程式的正確性。

4.「確實的」解釋「3.」，某個程式比另一個程式快或者慢的理由。

peterson_correct-O3使用了atomic operation來指定turn的值，如此每個thread(每顆CPU)看到的turn都會一樣。若是沒有用atomic operation的話，當CPU_0剛修改完turn的值後，會先將最新的數值寫到自己的cache，等待cache line被標註為invalidate後才會寫到主記憶體，若同時間有另一顆CPU_1欲存取主記憶體中同一位置的資料，由於自己的cache以及主記憶體中都沒有最新的資料，因此會讀到舊的turn值，造成程式錯誤。因此需要用atomic operation，在某顆CPU_0更改某個位置的資料時，都要先確保其他CPU已經invalid同一位置的cache(欲寫入的CPU_0廣播invalidate，其他CPU回invalidate ack)，清除掉其他CPU cache內的值後，才能將資料寫入自己的cache，稍後補寫回主記憶體。如此一來就能確保其他CPU能讀到更新後的值，但這樣的保護機制同時會使得程式的負擔加大(像是CPU_0要廣播invalidate，並且等其他CPU回ack)，造成執行速度變慢。 以下為CPU_0欲寫入時，需等待CPU_1回ack，中間產生了stall，讓CPU_0閒置了一段時間，導致程式執行變慢。

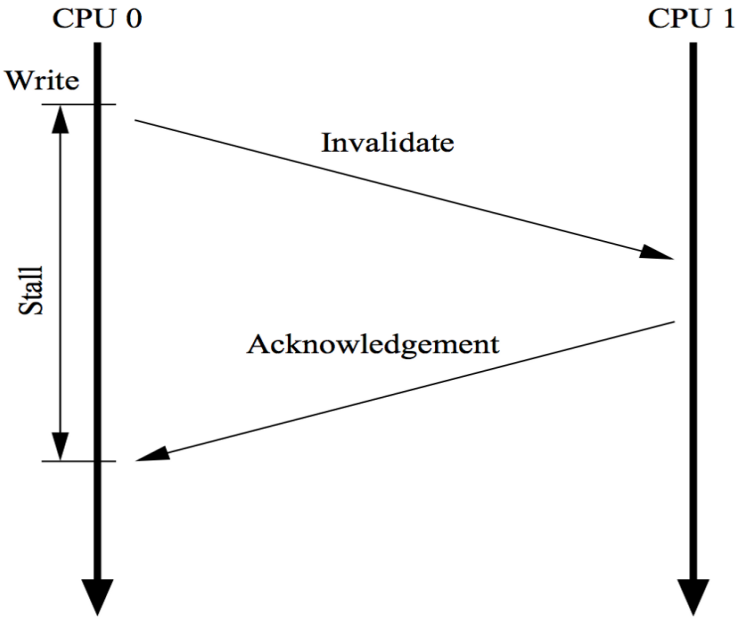

Figure 4: Writes See Unnecessary Stalls

反組譯後，可以看到peterson_correct-O3多了一些mfence的指令，來確保gcc不會打亂程式執行的順序，但由於使用了mfence的關係，在mfence指令前的讀寫操作必須在mfence指令後的讀寫操作前完成，讓gcc不能對這部分的

程式進行memory reordering，限制了gcc對其的優化，造成程式的執行速度下降。

```
Dump of assembler code for function p0:
   0x0000000000001370 <+0>:     endbr64
   0x0000000000001374 <+4>:     push   %rax
   0x0000000000001375 <+5>:     pop    %rax
   0x0000000000001376 <+6>:     lea    0xd02(%rip),%rdi        # 0x207f
   0x000000000000137d <+13>:    sub    $0x8,%rsp
   0x0000000000001381 <+17>:    callq  0x10e0 <puts@plt>
   0x0000000000001386 <+22>:    nopw   %cs:0x0(%rax,%rax,1)
   0x0000000000001390 <+32>:    movl   $0x1,0x2ca6(%rip)        # 0x4040 <flag>
   0x000000000000139a <+42>:    mfence
   0x000000000000139d <+45>:    mfence
   0x00000000000013a0 <+48>:    movl   $0x1,0x2c9e(%rip)        # 0x4048 <turn>
   0x00000000000013aa <+58>:    mfence
   0x00000000000013ad <+61>:    jmp    0x13bb <p0+75>
   0x00000000000013af <+63>:    nop
   0x00000000000013b0 <+64>:    mov    0x2c92(%rip),%eax        # 0x4048 <turn>
```