作業0A：實現Peterson’s solution

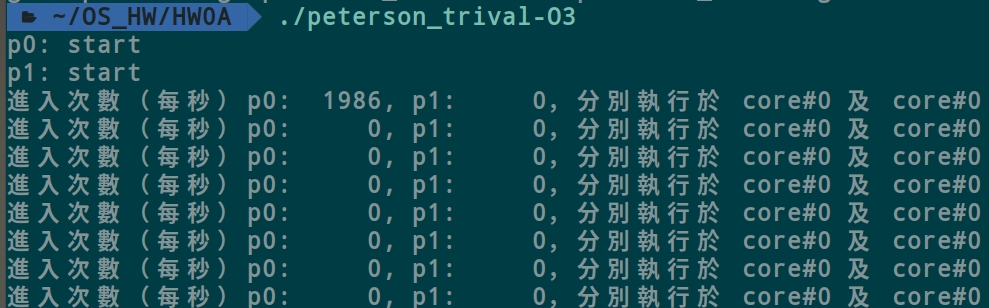
資工三 406410114 郭晏誠

1. 直接打make會得到四個執行檔案

(1)請將這四個執行檔案都執行過一次，請問你的執行節果為何？請附上畫面截圖（問題一）

(i) ./peterson\_trival-O3

結果錯誤 只有進去第一次



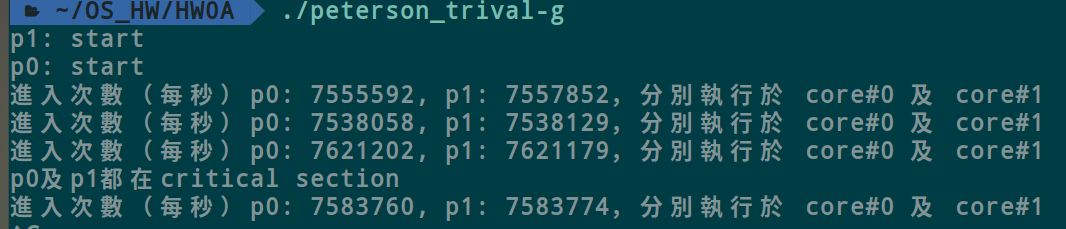
(ii) ./peterson\_trival-g

結果有時正確有時錯誤

正確如下：( 可能CPU運算有時候快有時慢 )



錯誤如下：



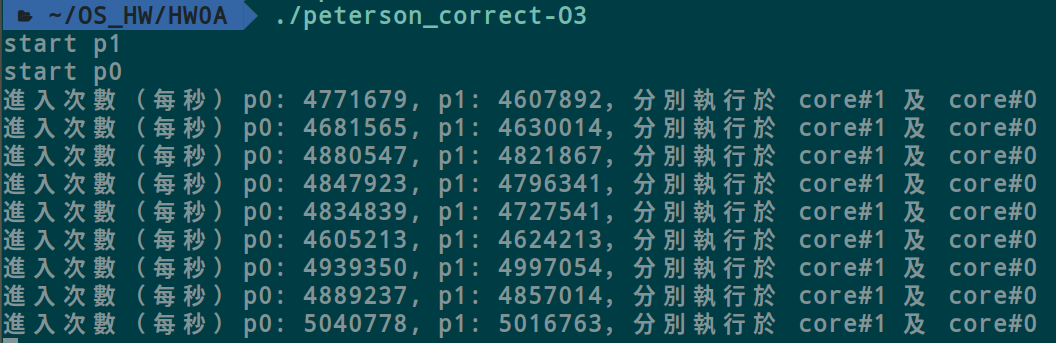
(iii) ./peterson\_correct-g

結果正確 (若多核心有可能換核心執行因為CPU比較燙)



(iv) ./peterson\_correct-O3

結果正確

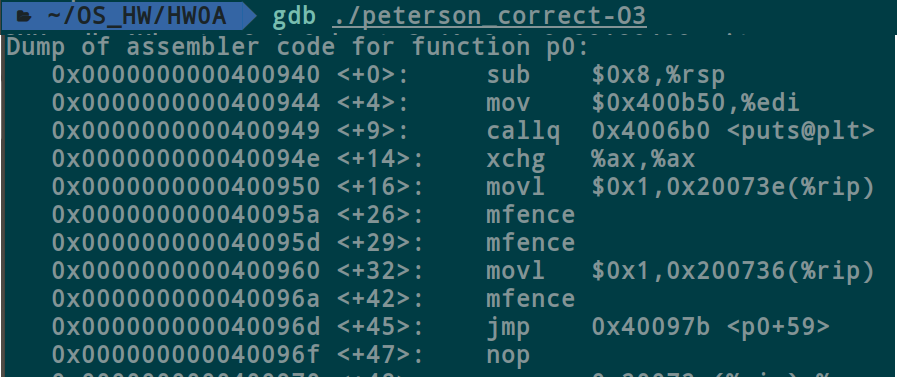
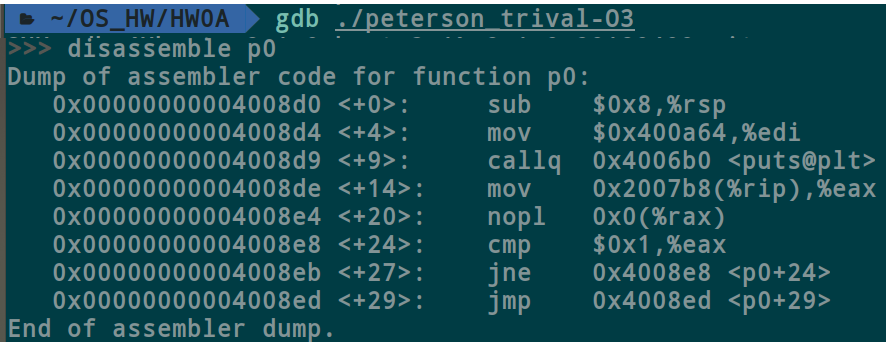


2. 請試著解釋「為什麼」 peterson\_trival-O3 的執行結果是錯的

* 請反組譯「 peterson\_trival-O3 」中的P0函數
* 你可以修改「peterson\_trival.c」盡量讓程式碼精簡

1. gdb peterson\_trival-O3 和gdb peterson\_correct-O3可發現correct多了mfence

▼gdb peterson\_trival-O3 ▼gdb peterson\_correct-O3



1. peterson\_trival-O3 的執行結果是錯是因為沒有mfence讓讀寫串行化，導致p0跟p1可能通時進入critical section進行修改全域資源，可能造成錯誤。
2. mfence補充

(i)定義：保證Memory access的讀寫都串行化，其實是在一系列Memory access中添加若干延遲(屏障)，可用來保證此指令的Memory access完成後，後面的指令才可Memory access，相關指令還有還有lfence(讀串行化)、sfence(寫串行化)

(ii) x86 mfence實作概念

(a) sfence/mfenceg會將store buffer中緩存的修改刷入L1 cache中，使其他cpu核可以觀察到這些修改

(b) sfence/mfence之後的寫操作，不會被調度到sfence/mfence之前。

(iii)效能優化

(a) 新的Intel處理器為了提升效能，新增了memory order

(b) 允許預讀取操作先前存到load buffer中。

(c) 允許store buffer的緩存，讓先寫再讀操作變成先讀再寫。

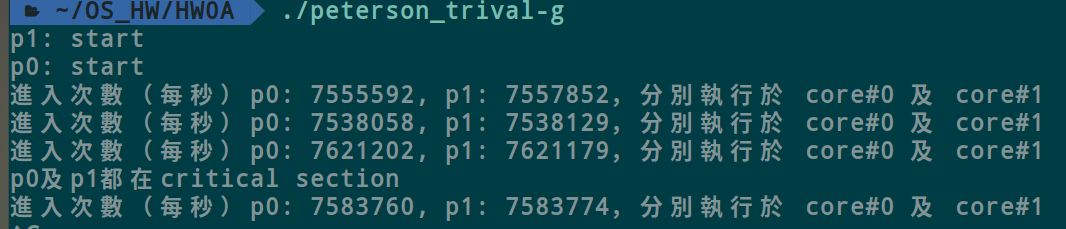
(d) string操作指令和繞過cache的write指令（MOVNTI, MOVNTDQ等）有priority。

3. 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

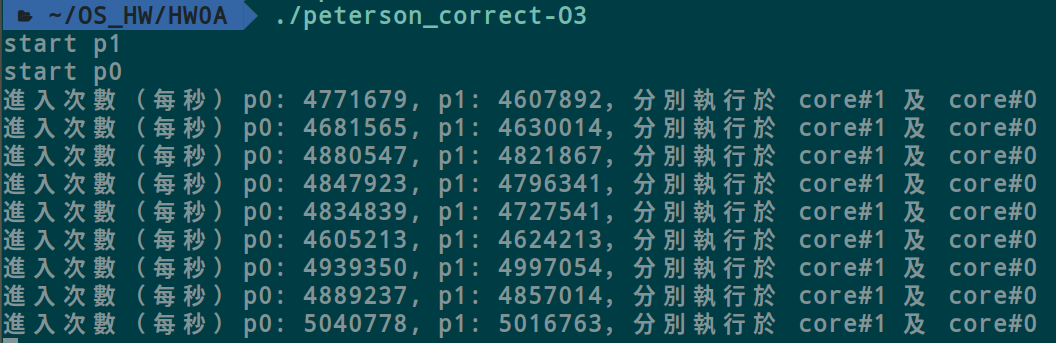
* 「peterson\_trival-g」的速度比「 peterson\_correct-O3」還要來得快(excel畫成表)
* 需附上cpu型號與gcc版本

1. peterson\_trival-g是錯的，peterson\_correct-O3是正確的

peterson\_trival-g錯誤如下：



peterson\_correct-O3運行正確：



1. 速度比較

(i)明顯peterson\_trival-g比peterson\_correct-O3快1.53倍

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | peterson\_trival-g |  | peterson\_correct-O3 |  |
| 實驗次數 | P0 | P1 | P0 | P1 |
| 1 | 8297021 | 8287943 | 4815676 | 4839527 |
| 2 | 8320601 | 8321127 | 4797272 | 4800218 |
| 3 | 7410145 | 7410987 | 4785693 | 4808459 |
| 4 | 7180751 | 7181048 | 4819306 | 4836898 |
| 5 | 7239093 | 7245478 | 5021556 | 5040582 |
| 6 | 6617149 | 6617073 | 4782907 | 4808973 |
| 7 | 7385823 | 7385823 | 4874496 | 4900021 |
| 8 | 7379195 | 7379111 | 5000973 | 5018152 |
| 9 | 7345502 | 7347220 | 4561879 | 4639658 |
| 10 | 7450935 | 7458881 | 4853873 | 4844018 |
| 平均 | 7462622 | 7463469 | 4831363 | 4853651 |

(ii)gcc version： 4.9.3

(iii)CPU型號



4. 盡可能的解釋即可，最起碼附上每一行組語的意義

(1) gdb ./peterson\_correct-O3 因為mfence變慢了

#每一行意思

#使O3變慢的原因

>>> disassemble p0

Dump of assembler code for function p0:

0x0000000000400940 <+0>: sub $0x8,%rsp # stack frame

0x0000000000400944 <+4>: mov $0x400b50,%edi #用edi保存地址 0x400b50

#printf("start p0\n");

0x0000000000400949 <+9>: callq 0x4006b0 <puts@plt> #找到puts函數地址0x4006b0

0x000000000040094e <+14>: xchg %ax,%ax # NOP

#atomic\_store(&flag[0], 1);

0x0000000000400950 <+16>: movl $0x1,0x20073e(%rip) #設定flag rip相對定址

# 0x601098 <flag>

0x000000000040095a <+26>: mfence #內存屏障 atomic\_store

# atomic\_thread\_fence(memory\_order\_seq\_cst);

0x000000000040095d <+29>: mfence #內存屏障 atomic\_thread\_fence

#atomic\_store(&turn, 1);

0x0000000000400960 <+32>: movl $0x1,0x200736(%rip) #設定turn rip相對定址

# 0x6010a0 <turn>

0x000000000040096a <+42>: mfence #內存屏障 atomic\_store

0x000000000040096d <+45>: jmp 0x40097b <p0+59> #無條件跳轉

0x000000000040096f <+47>: nop #不改變任何程式可存取的暫存器

#while (atomic\_load(&flag[1]) && atomic\_load(&turn)==1)

0x0000000000400970 <+48>: mov 0x20072a(%rip),%eax #把turn讀出來

#0x6010a0 <turn>

0x0000000000400976 <+54>: cmp $0x1,%eax #提早做compare因為是效能瓶頸

0x0000000000400979 <+57>: jne 0x400985 <p0+69> #符合條件時跳轉

0x000000000040097b <+59>: mov 0x20071b(%rip),%eax # flag[1]# 0x60109c <flag+4>

0x0000000000400981 <+65>: test %eax,%eax #&&運算

0x0000000000400983 <+67>: jne 0x400970 <p0+48> #符合條件時跳轉

0x0000000000400985 <+69>: movl $0x0,0x200709(%rip)# (&flag[0],0)

#0x601098 <flag>

0x000000000040098f <+79>: mfence #內存屏障 atomic\_store

0x0000000000400992 <+82>: jmp 0x400950 <p0+16> #無條件跳轉回到while上面

End of assembler dump.

(2) gdb ./peterson\_trival-g

#每一行意思

#使peterson\_trival-g錯的原因

>>> disassemble p0

Dump of assembler code for function p0:

0x000000000040087b <+0>: push %rbp # stack frame

0x000000000040087c <+1>: mov %rsp,%rbp #rsp =rbp

0x000000000040087f <+4>: mov $0x400af0,%edi #用edi保存地址 0x400b50

0x0000000000400884 <+9>: callq 0x400690 <puts@plt> #printf("start p0\n");

0x0000000000400889 <+14>: movl $0x1,0x200801(%rip)#設定flag0# 0x601094 <flag0>

0x0000000000400893 <+24>: movl $0x1,0x2007ef(%rip) #設定turn# 0x60108c <turn>

0x000000000040089d <+34>: nop #不改變任何程式可存取的暫存器

##while (flag1==1 && turn==1) 不見了

0x000000000040089e <+35>: mov 0x2007ec(%rip),%eax#拿flag1 #0x601090 <flag1>

0x00000000004008a4 <+41>: cmp $0x1,%eax #比對

0x00000000004008a7 <+44>: jne 0x4008b4 <p0+57> #符合條件時跳轉

0x00000000004008a9 <+46>: mov 0x2007dd(%rip),%eax #turn = 1 # 0x60108c <turn>

0x00000000004008af <+52>: cmp $0x1,%eax #比對

0x00000000004008b2 <+55>: je 0x40089e <p0+35> #符合條件時跳轉

0x00000000004008b4 <+57>: movl $0x0,0x2007d6(%rip) # flag0=0 # 0x601094 <flag0>

0x00000000004008be <+67>: jmp 0x400889 <p0+14> #無條件跳轉回到while上面

End of assembler dump.

// youtube筆記

正確的P0

1. atomic\_int turn=0; //要用atomic\_int 才可傳入atomic\_store
2. atomic\_int flag[2] = {0, 0};
3. void p0(void) {
4. printf("start p0\n");
5. while (1) {
6. atomic\_store(&flag[0], 1); //address type: atomic\_int
7. atomic\_thread\_fence(memory\_order\_seq\_cst);//避免第6和

// 第8對調

1. atomic\_store(&turn, 1);
2. while (atomic\_load(&flag[1]) && atomic\_load(&turn)==1)
3. ;   //waiting
4. //底下程式碼用於模擬在critical section
5. in\_cs++; //進入critical section++
6. nanosleep(&ts, NULL);
7. if (in\_cs == 2) fprintf(stderr, "p0及p1都在critical section\n");
8. p0\_in\_cs++;
9. nanosleep(&ts, NULL);
10. in\_cs--;//出去critical section++
11. //離開critical section
12. atomic\_store(&flag[0], 0); //把0存到flag
13. }
14. }

gcc -pthread -O3 peterson\_correct.c -o peterson\_correct-O3 ##-O3優化

gcc -pthread -g peterson\_correct.c -o peterson\_correct-g ##-g =-O0不優化

gcc -pthread -O3 peterson\_trival.c -o peterson\_trival-O3

gcc -pthread -g peterson\_trival.c -o peterson\_trival-g