

作業七： 實現Peterson's solution



中正大學 作業系統實驗室
指導**大叔**：羅習五

羅習五

創作共用-姓名 標示-非商業性-相同方式分享
CC-BY-NC-SA



作業目標及負責助教

作業目標：

- 了解Peterson's solution如何實現
- 了解如何正確的實現spinlock
- 從組合語言的角度去思考對的和錯的Peterson's solution的實現方法

Peterson's solution的直觀實現方法

- 🍏 底下二頁是Peterson's solution的直觀實現方法
- 🍏 對照到我們的程式碼是「peterson_trival.c」

P0

```
1. while(1) { /*代表P0週而復始地想進入CS*/
2.     flag[0] = true;      /*告訴大家, P0想進入CS*/
3.     turn = 1;            /*如果P1也想進入CS, 讓P1先進去*/
4.     while(flag[1] == true AND turn ==1) /*P1進入權比較高, 且P1想進入CS*/
5.         ;//用while loop實作出wait
6.
7.     /*critical section, 可以於此更新共享資料結構*/
8.
9.     flag[0] = false;     /*離開CS, 因此將flag[0]設定為false。
10.                        /*表示P0目前不需要待在CS了*/
11. }
```

註：AND即為C語法中的&&

P1

```
1. while(1) { /*代表P0週而復始地想進入CS*/
2.     flag[1] = true;      /*告訴大家, P1想進入CS*/
3.     turn = 0;           /*如果P0也想進入CS, 讓P0先進去*/
4.     while(flag[0] == true AND turn == 0) /*P0進入權比較高, 且P0想進入CS*/
5.         ;//用while loop實作出wait
6.
7.     /*critical section, 可以於此更新共享資料結構*/
8.
9.     flag[1] = false;     /*離開CS, 因此將flag[1]設定為false。
10.                        /*表示P1目前不需要待在CS了*/
11. }
```

註：AND即為C語法中的&&

Peterson's solution的正確實現方法

🍏 下一頁是Peterson's solution的正確實現方法

🍏 對照到我們的程式碼是 「peterson_correct.c」

正確的方法 下圖是p0

```
1.  atomic_int turn=0;
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);
7.          atomic_thread_fence(memory_order_seq_cst);
8.          atomic_store(&turn, 1);
9.          while (atomic_load(&flag[1]) && atomic_load(&turn)==1)
10.             ;    //waiting
11.         //底下程式碼用於模擬在critical section
12.         in_cs++;
13.         nanosleep(&ts, NULL);
14.         if (in_cs == 2) fprintf(stderr, "p0及p1都在critical section\n");
15.         p0_in_cs++;
16.         nanosleep(&ts, NULL);
17.         in_cs--;
18.         //離開critical section
19.         atomic_store(&flag[0], 0);
20.     }
21. }
```

編譯程式碼

- 🍏 直接打make會得到四個執行檔案
- 🍏 請將這四個執行檔案都執行過一次
- 🍏 請問你的執行結果為何？請附上畫面截圖（問題一）

反組譯

- 🍏 你可以修改「peterson_trival.c」盡量讓程式碼精簡
- 🍏 請反組譯「peterson_trival-O3」中的P0函數
- 🍏 請試著解釋「為什麼」peterson_trival-O3 的執行結果是錯的
(問題二)

反組譯

- 🍏 在老師的電腦上「peterson_trival-g」的執行結果是對的
- 🍏 「peterson_trival-g」的速度比「peterson_correct-O3」還要來得快
- 🍏 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（問題三）
- 🍏 請使用反組譯解釋前述之結果（問題四，盡可能的解釋即可，最起碼附上每一行組語的意義），你可以修改原始程式碼，讓反組譯後的組合語言比較好懂

反組譯的方法

以peterson_trival-g為例

```
$ gdb ./peterson_trival-g
```

```
(gdb) disassemble p0
```

```
Dump of assembler code for function p0:
```

```
0x0000000000000121f <+0>:  push    %rbp
0x00000000000001220 <+1>:  mov     %rsp,%rbp
0x00000000000001223 <+4>:  lea     0xe36(%rip),%rdi          # 0x2060
0x0000000000000122a <+11>: callq   0x1040 <puts@plt>
0x0000000000000122f <+16>: movl    $0x1,0x2dfb(%rip)        # 0x4034 <flag0>
0x00000000000001239 <+26>: movl    $0x1,0x2de9(%rip)        # 0x402c <turn>
0x00000000000001243 <+36>:  nop
```

提示

- 🍏 可以參考上課的講義
- 🍏 就附在這份投影片的後面 (15~17)

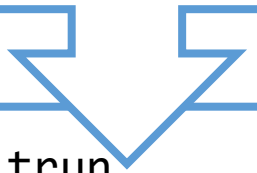
錯誤版，反組譯

```
1.  cmpl    $0x1,0x2015f5(%rip)    # 0x20203c <flag1>
2.  movl    $0x1,0x2015e7(%rip)    # 0x202038 <flag0>
3.  movl    $0x1,0x2015e5(%rip)    # 0x202040 <turn>
4.  jne     0xa60 <p0+64>
5.  jmp     0xa5d <p0+61>
```

因為compare是效能瓶頸，因此編譯器將compare往前挪，挪到設定flag0之前

錯誤版，反組譯（進一步解釋）

```
flag0 = 1;  
turn = 1;  
while (flag1==1 && turn==1)
```



```
if (flag1 != 1) jmp = true;  
flag0 = 1;  
turn = 1;  
//while (flag1==1 && turn==1) 這一行不見了  
//因為compiler看到第3行是「turn=1」，  
//因此compiler認為while loop的結果只相依於flag1  
if (jmp == true) goto out;  
out:  
//critical section
```

正確版, 反組譯

```
1. 23      atomic_store(&flag[0], 1);
2.      lea    0x2016ad(%rip),%rax          # 0x555555756038 <flag>
3.      mov    %rax,-0x30(%rbp)
4.      movl   $0x1,-0x34(%rbp)
5.      mov    -0x34(%rbp),%eax
6.      mov    %eax,%edx
7.      mov    -0x30(%rbp),%rax
8.      mov    %edx,(%rax)
9.      mfence
10. 24      atomic_thread_fence(memory_order_seq_cst);
11.      mfence
12. 25      atomic_store(&turn, 1);
13.      lea    0x201682(%rip),%rax          # 0x555555756030 <turn>
14.      mov    %rax,-0x28(%rbp)
15.      movl   $0x1,-0x34(%rbp)
16.      mov    -0x34(%rbp),%eax
17.      mov    %eax,%edx
18.      mov    -0x28(%rbp),%rax
19.      mov    %edx,(%rax)
20.      mfence
```

作業繳交

🍏 請參考「作業07-Peterson's solution.docx」