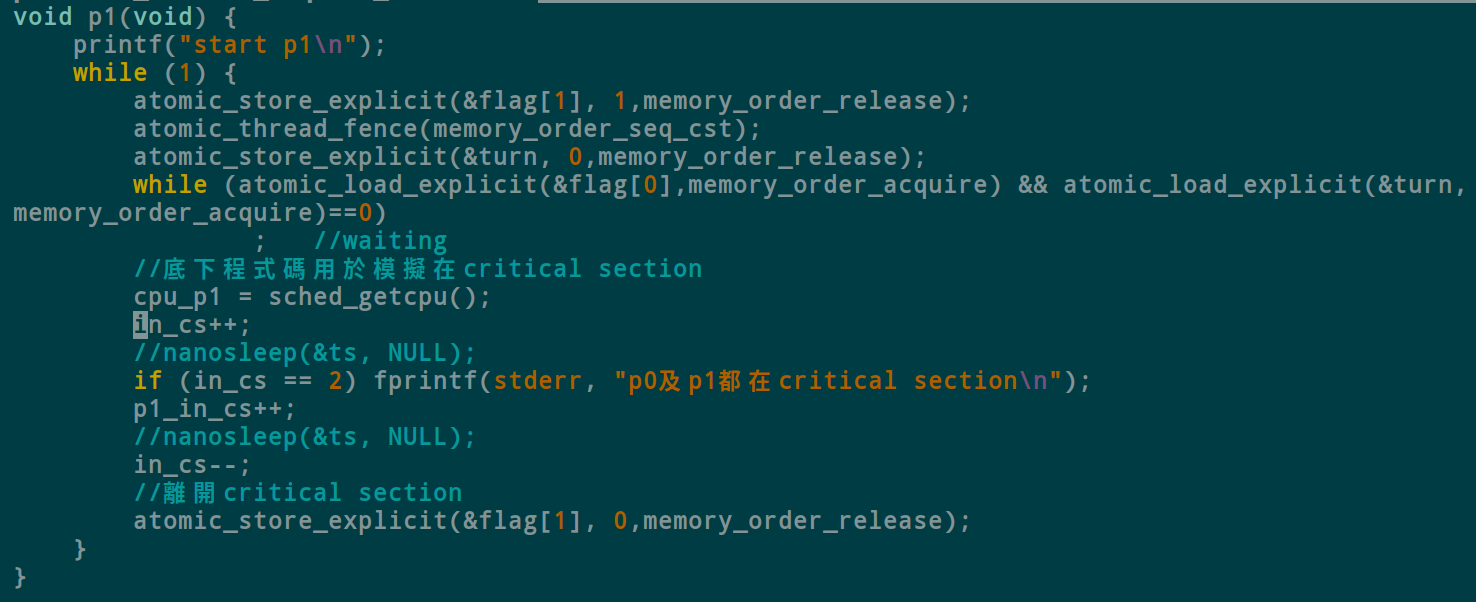
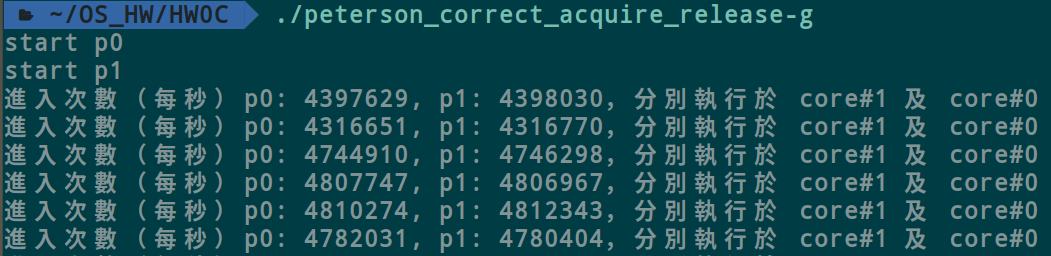
hw05.syswrite

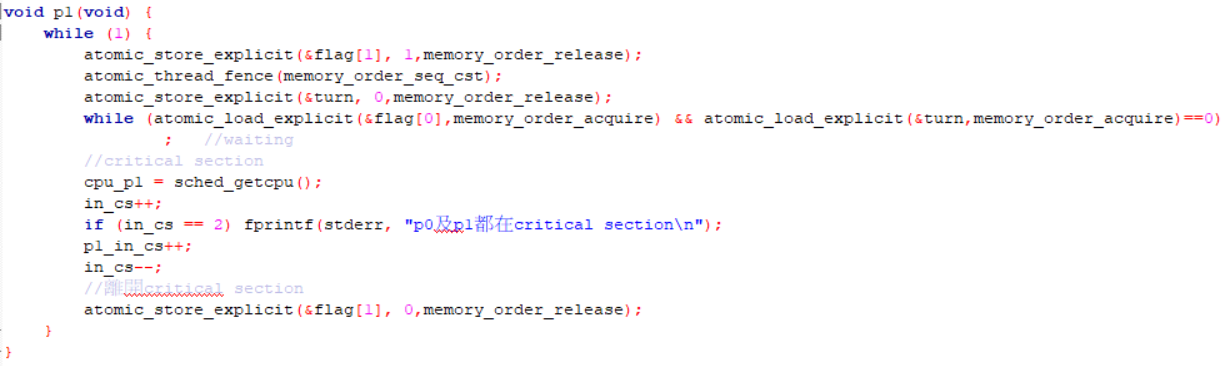
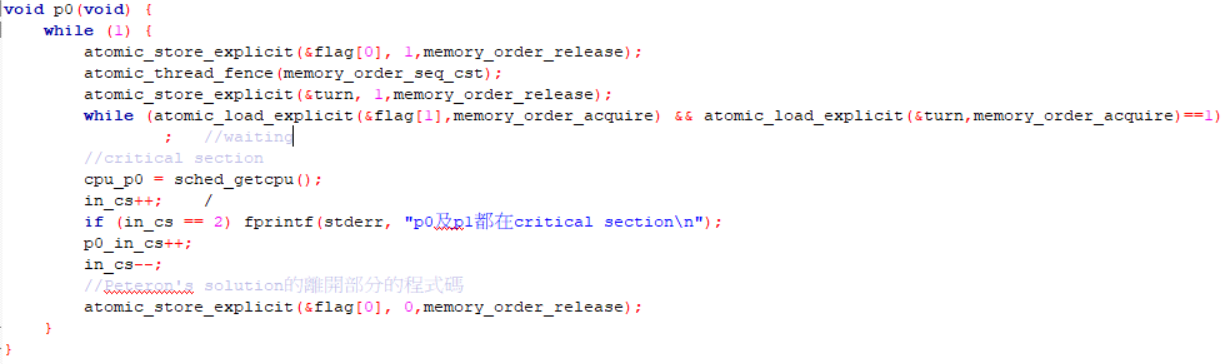
1. 結果( 用證明的方式證明是對的 )

(1) store : memory\_order\_release、load : memory\_order\_acquire 結果正確





1. 相關說明與證明

使用store : memory\_order\_release、load : memory\_order\_acquire會達到以下的效果

1. P0 load-acquire 之後可以看見 P1 store-release 之前寫入的資料；
2. P1 load-acquire 之後可以看見 P0 store-release 之前寫入的資料。
3. load-acquire 之後的 read/write 不能搬到 load-acquire 之前。
4. store-release 之前的 read/write 不能搬到 store-release 之後。

證明如下：

1. mutual exclusion:

* 執行順序存取的保證性

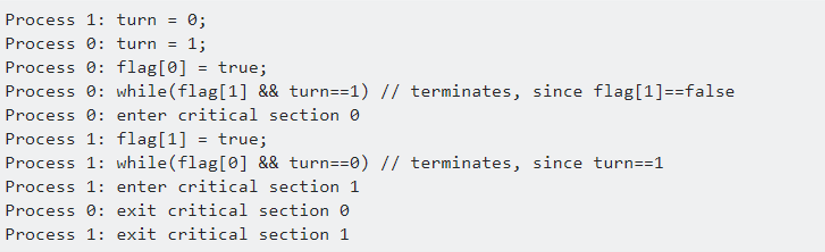
由於(4) store-release 之前的 read/write 不能搬到 store-release 之後

保證會先設定flag才會設定turn，並且進入critical section後才會修改flag設定

* P0和P1執行的互斥性

如果P0在CS內，那麼flag[1]為false(意味著P1已經離開CS)，或者turn為0(意味著P1只能在while等待，不能進入CS)

並且保證會先設定turn再設定flag不會造成此現象



1. progress:

* CS為空，想進去的thread可以進去性

假設P0拿到CS做完後，CS為空會用store-release 修改flag，

若P1在while等待會用load-acquire看到後，P1就可以進入CS；

否則P1還沒開始等，若P1先設定讓先，P0後設定讓先，則P1進入CS

否則P1進入等待，P0一做完P1就可以進去CS，證明了P0一離開CS，CS為空P1就有機會進去；P0的證明同理，故達到progress

1. bound waiting

* 等待有限次性

假設P0先拿到CS，P0做完後會改成設定讓P1先進去，在等待P1就可以等1次P0就進入CS，P1也同理，故達到bound waiting