修改部分：

```c
void p0(void) {
    printf("start p0\n");
    while (1) {
        //Peteron's solution的進去部分的程式碼
        //atomic_store(&flag[0],1);
        atomic_store_explicit( &flag[0],1, memory_order_relaxed); // B
        atomic_thread_fence(memory_order_seq_cst);
        atomic_store_explicit(&turn,1, memory_order_relaxed);
        //atomic_store(&turn,1);
        while (atomic_load(&flag[1]) && atomic_load(&turn)==1)
                ;   //waiting


        //底下程式碼用於模擬在critical section
        cpu_p0 = sched_getcpu();
        in_cs++;        //計算有多少人在cs中
        nanosleep(&ts, NULL);
        if (in_cs == 2) fprintf(stderr, "p0及p1都在critical section\n");
        p0_in_cs++;     //P0在CS幾次
        nanosleep(&ts, NULL);
        in_cs--;        //計算有多少人在cs中
        //Peteron's solution的離開部分的程式碼
        atomic_store(&flag[0], 0);
    }
}
```

```c
void p1(void) {
    printf("start p1\n");
    while (1) {
        //atomic_store(&flag[1],1);
        atomic_store_explicit( &flag[1], 1, memory_order_relaxed);
        atomic_thread_fence(memory_order_seq_cst);
        //atomic_store(&turn, 0);
        atomic_store_explicit( &turn,0, memory_order_relaxed);
        while (atomic_load(&flag[0]) && atomic_load(&turn)==0)
                ;   //waiting
        //底下程式碼用於模擬在critical section
        cpu_p1 = sched_getcpu();
        in_cs++;
        nanosleep(&ts, NULL);
        if (in_cs == 2) fprintf(stderr, "p0及p1都在critical section\n");
        p1_in_cs++;
        nanosleep(&ts, NULL);
        in_cs--;
        //離開critical section
        atomic_store(&flag[1], 0);
    }
}
```

三條件：

1. mutual exclusive：

P0, P1 同時執行，並且都可以設定turn變數，因此turn不是為0就是為1。因為使用了memory_order_relaxed，而且turn 是共享的，所以turn 只能為0或1。如果不使用memory_order_relaxed 有可能p0 看到1，p1 看到為0。

當flag[0] == flag[1] == 1時，代表p0, p1同時都想進去cs，但是受制於turn == 0 or turn == 1，只有其中一個task可以進去。

2. bounded waiting：

當p0 進入cs之後，要再一次進入cs，會設定flag[0] ,與turn，但是turn ==1，p0讓p1，p1進入cs。

3. progress：

- 假設P0想進入CS，P1不想進入CS 。

(flag[0] == 1, flag[1]==0 turn == 1)

雖然p1 有進去cs 較大的優先權(turn == 1)，但是p1 不想進去(flag[1] == 0)，所以 while(flag[1] == true && turn ==1) ; 不成立，p0 可以進去cs。

證明：當cs沒有task時p0可以進去。

- 假設p0 想進入cs，p1 想進入cs，但是p1只執行到flag[1] = ture。

(flag[0]==1 flag[1] ==1 turn == 1)

因為turn == 1 對 p0 來說 while(flag[0] == true && turn ==0) 不成立，所以p1進入cs

證明：cs裡面沒有task正在執行時，如果p0也嘗試進入cs並且執行到「flag[1] = true;」，那麼p1可以進入cs

- 假設p0, p1 都想要進入cs 並且雙方都執行完flag和turn 設定

與mutual exclusion相同，只有一個task能夠進入cs