**CS342301: Operating System**

**Pthreads**

組別: 8

組員:

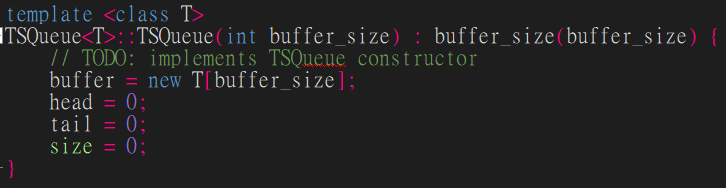
109062315洪聖祥

109062314張紘齊

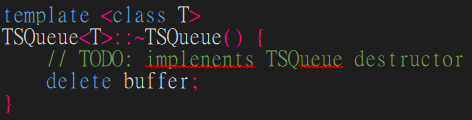
|  |  |
| --- | --- |
| 項目 | 分工 |
| **Implementation** |  |
| TSQueue、Producer、ConsumerController | 張紘齊 |
| Consumer、Writer、main.cc | 洪聖祥 |
| **Experiement** | 張紘齊、洪聖祥 |
| **Additional Experiment & Difficuties** | 張紘齊、洪聖祥 |

**Part I: Implementation**

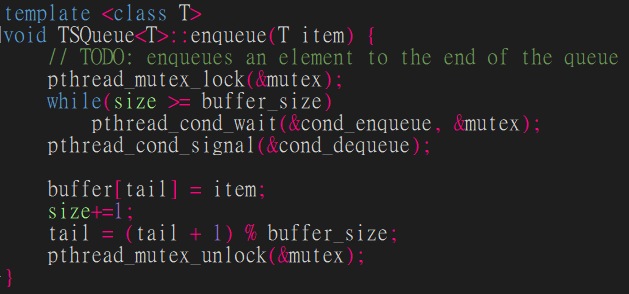
* TSQueue



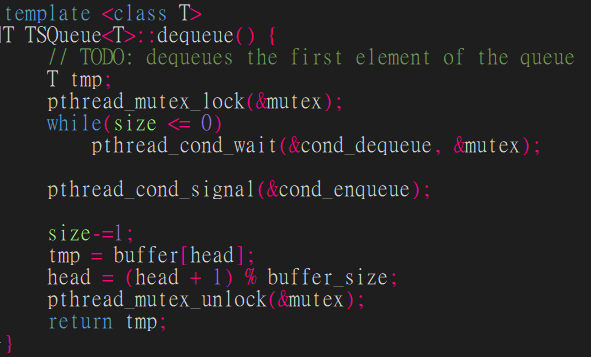
在Constructor，替buffer分配記憶體空間，然後將head (dequeue的位置)、tail (enqueue的位置)指向buffer中第一個位置，最後將buffer中當前的size設為0



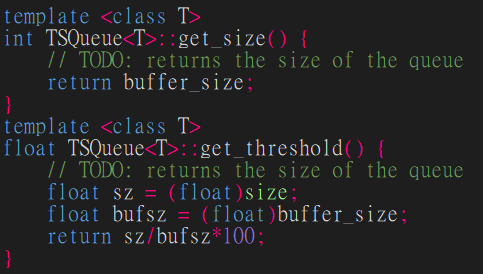
在Destructor，歸還先前替buffer分配的空間



每當要enqueue，要先取得mutex lock，接著判斷當前queue中是否已滿，若滿了，則透過condition variable等待別人(有要dequeue的thread)呼叫；若還沒滿，則可進行enqueue，步驟為先去呼叫(signal)一個正在等待dequeue的thread，接著將item放入buffer中tail的位置，然後將buffer裡當前的size+1，並將tail往前移一格，若超出buffer size，則指回到0，最後放掉一開始取得的mutex lock



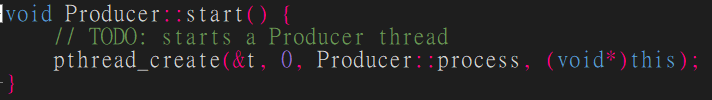
每當要dequeue，要先取得mutex lock，接著判斷當前queue中是否為空，若是空的，則透過condition variable等待別人(有要enqueue的thread)呼叫；若不是空的，則可進行dequeue，步驟為先去呼叫(signal)一個正在等待enqueue的thread，接著取出buffer中head的位置之item，然後將buffer裡當前的size-1，並將head往前移一格，若超出buffer size，一樣指回到0，最後放掉一開始取得的mutex lock，回傳剛剛取出的item



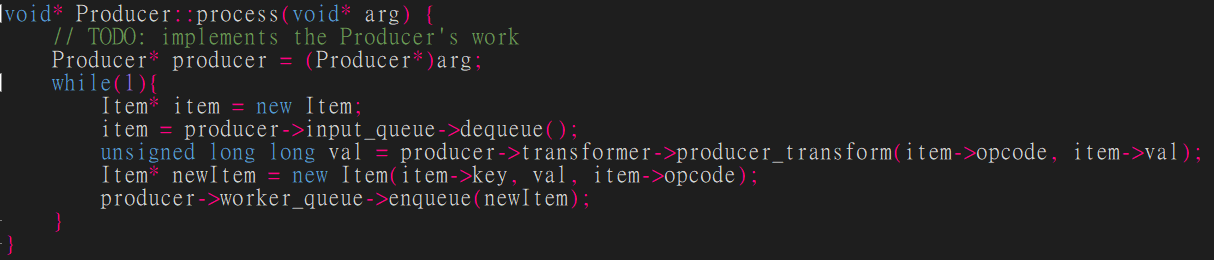
get\_size，回傳這個queue最大的容量

get\_threshold，回傳當前queue中的使用率\*100，要注意使用float

* Producer

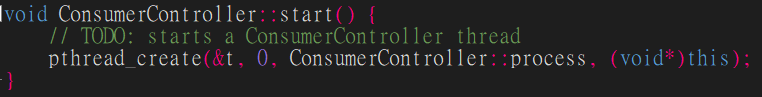


在start function中，create一個Producer的thread，並傳入process function，也就是Producer要進行的工作

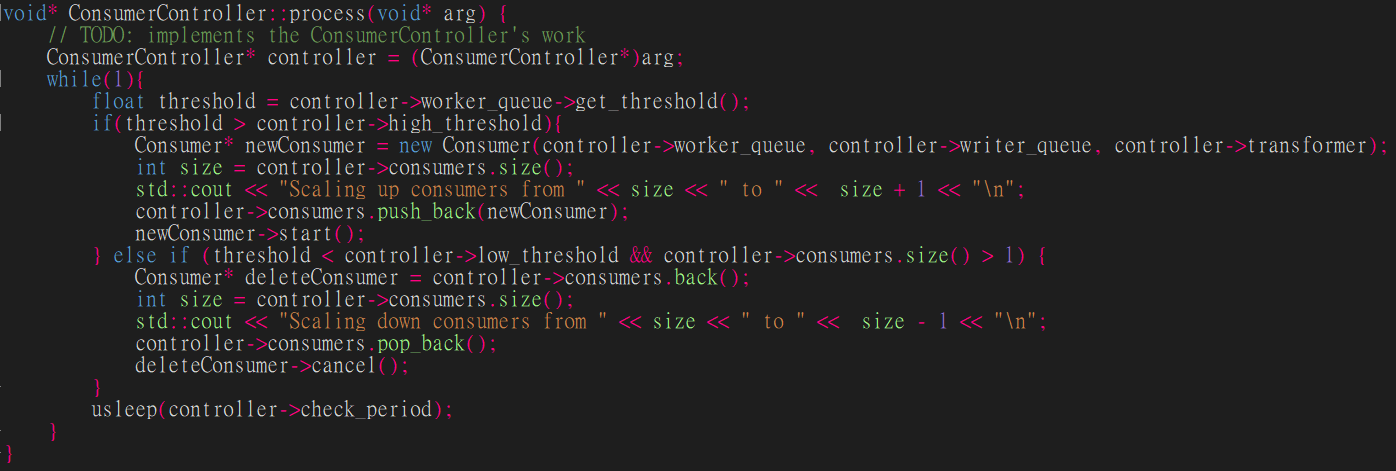


Producer要做的事為，不斷地: 從input\_queue中拿出item，接著將item透過transformer產生一個新的value，並新增一個item物件使用這個value，最後將此item放入到worker queue中

* ConsumerController

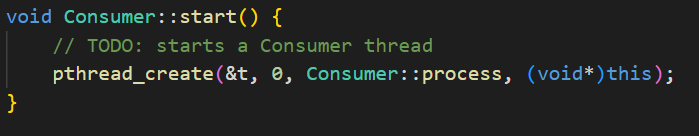


在start function中，create一個ConsumerController的thread，並傳入process function，也就是ConsumerController要進行的工作

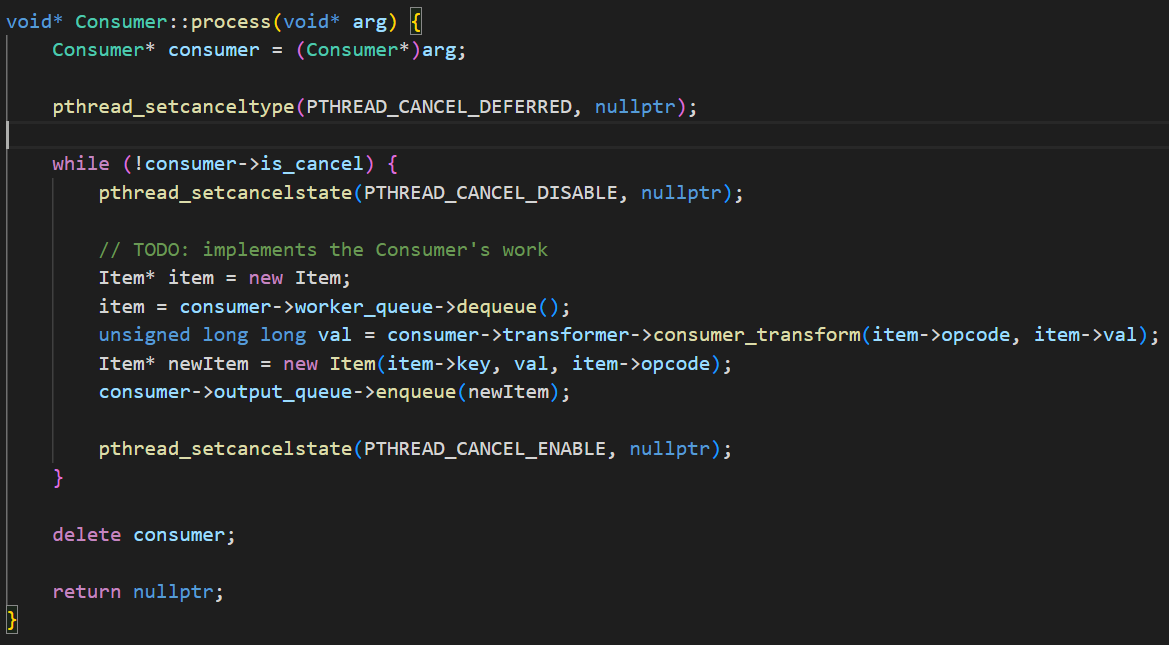


ConsumerController要做的事為，不斷地: 檢查worker queue的使用率，若使用率高於high threshold，則新增一個Consumer thread，在Consumer list紀錄它並執行它，然後輸出Consumer的數量變化(少->多)；若使用率低於low threshold且當前存在的Consumer至少有一個，則將Consumer list中最近一個新增的Consumer thread cancel掉並移除，然後輸出Consumer的數量變化(多->少)。

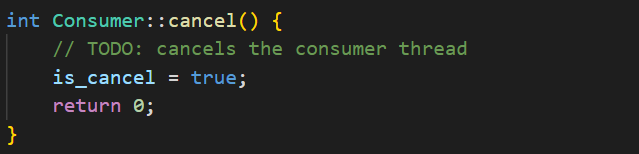
每次判斷的時間間隔為check\_period， 透過-usleep呈現

* Consumer 

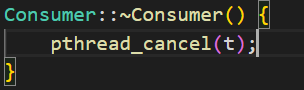
在create Consumer thread會呼叫Consumer::start()這個method，並且這個thread會執行void\* Consumer::process(void\*)。

****

在Consumer::process(void\* arg)，因為consumer thread的數量是由CosumerController控制的，所以有可能thread 被cancel。如果沒有被cancel，也就是is\_cancel = False，Consumer會一直從Worker Queue拿 Item，用transformer transform Item的值，最後把新的Item放到Output Queue中。如果在while loop中，ConsumerController決定要cancel consumer thread，但consumer thread還在做事情(把Worker Queue 拿出Item，放到Output Queue中)，所以在while loop 最前面把cancel state設為disable，然後在做完事情後，在把cancel state設為enable。如果ConsumerController決定要cancel掉consumer (is\_cancel = true)，會跳出迴圈然後delete 掉consumer。

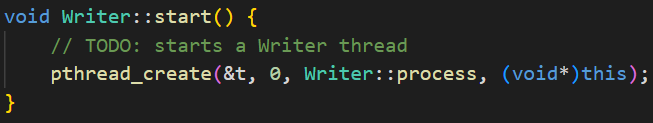


Consumer::cancel()，ConsumerController會控制Consumer的數量，低於low threshold會cancel Consumer thread，呼叫Consumer::cancel()。將is\_cancel 設為true，這樣Consumer::process才會跳出迴圈。

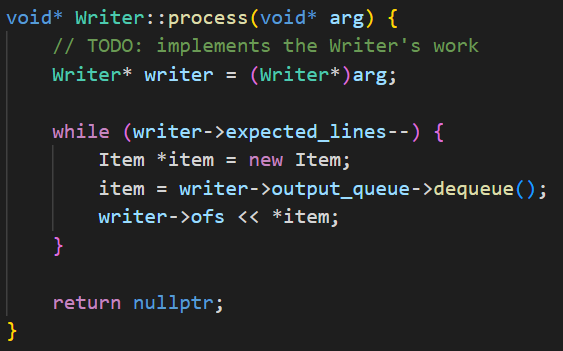


Destructor被執行的情況下為Consumer::process()中while loop條件不成立，也就是is\_cancel = true，所以呼叫pthread\_cancel(t)。

* Writer

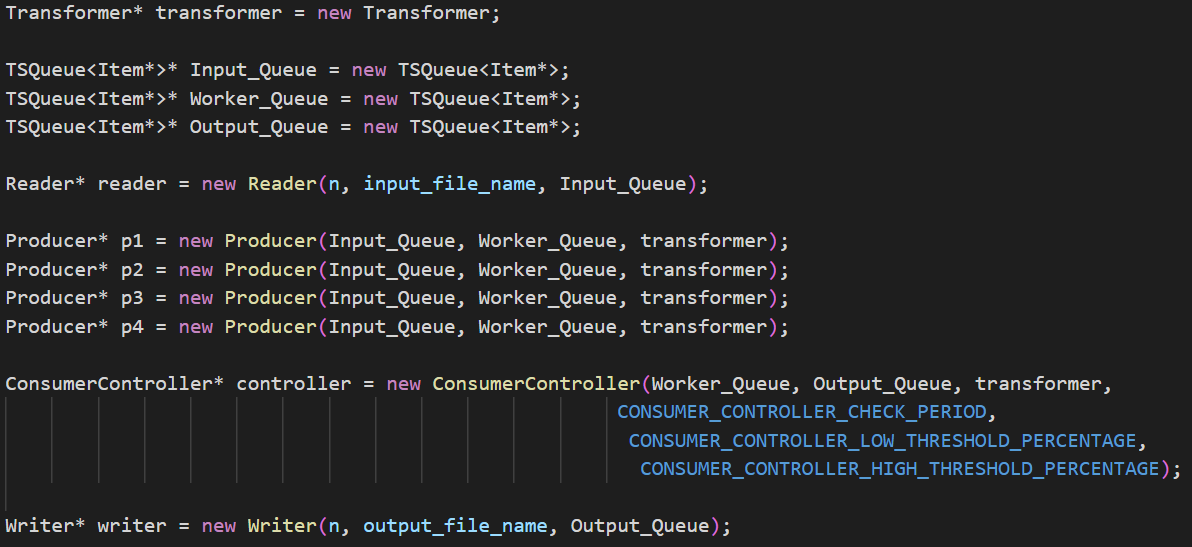


在create Consumer thread會呼叫Writer::start()這個method，並且這個thread會執行void\* Consumer::process(void\*)。

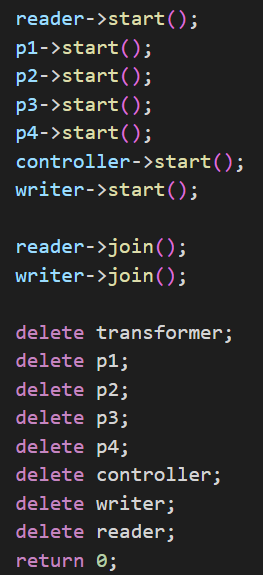


Writer::process(void\* arg) 是當expected\_lines > 0時要一直從Output Queue中拿Item，並把item 寫進output file。

* Main.cpp



Main.cpp 中主要先construct 1個Transformer class、3個TSQueue class (分別當成Input Queue、Worker Queue、Output Queue) 、 1個 Reader class、4個Producer class、1個Consumer Controller class和1個Writer class。



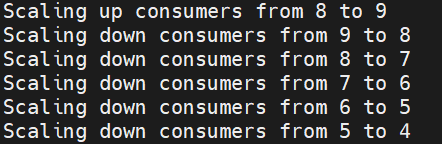
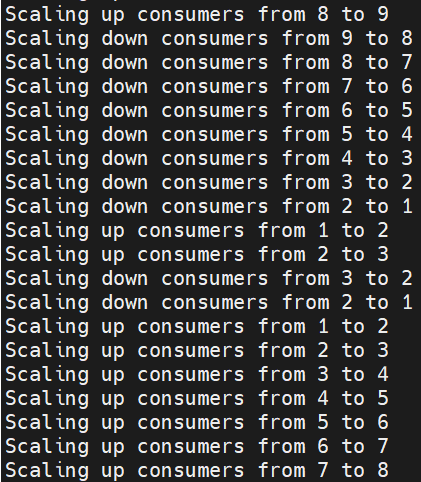
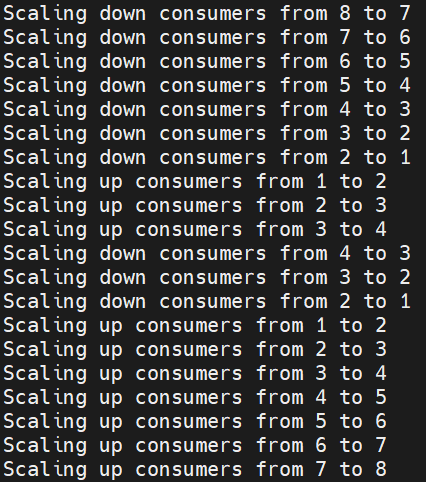
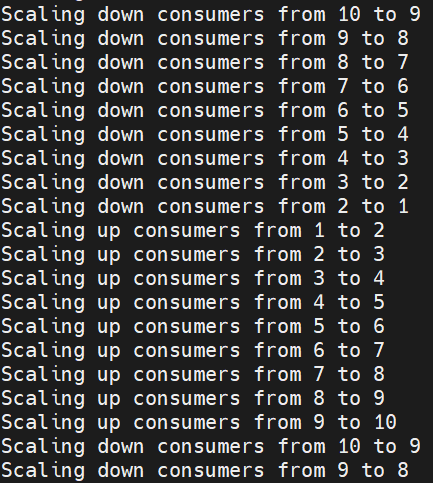
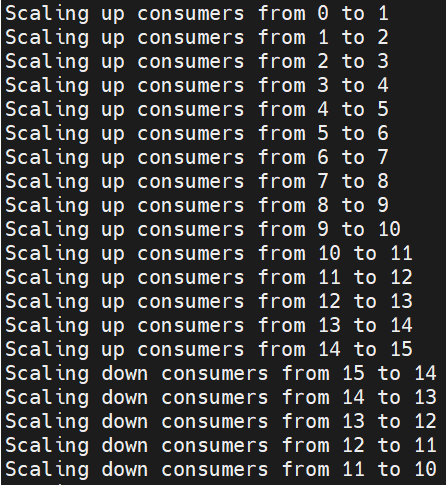
接下來從剛剛建立的class instance呼叫start() method，表示要create thread。Create 完代表要執行每個class 中的 process() method。那結束的時機為reader讀完input file的資料和writer寫完output file。所以最後要呼叫reader->join() 、 writer->join()，表示main thread要等 reader thread完成和writer thread完成才能執行收尾的動作。最後就是把 class instance delete。

**Part II Experiment**

* Different values of **CONSUMER\_CONTROLLER\_CHECK\_PERIOD**

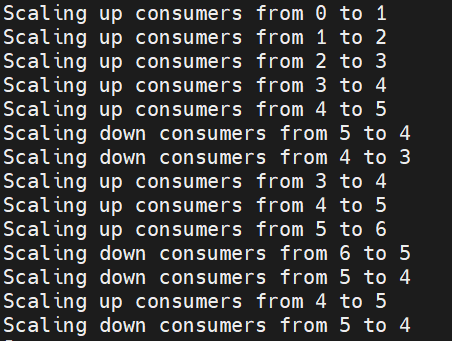


Check period = 0.1 sec



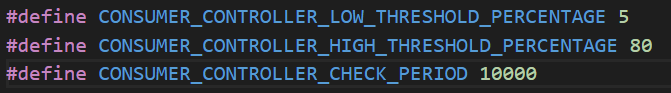


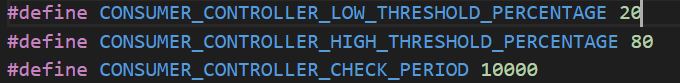
Check period = 5 sec

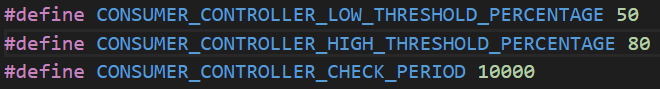


我們實驗了check period = 0.1 sec 、1 sec (原本預設值) 和 5 sec的情況。我們發現隨著check period 秒數越小，scaling up and down 的次數越來越多，而且consumer最多的數量也隨著check period 越小而越多。因此我們覺得當check period 越小，consumer controller 更能動態調整consumer的數量。

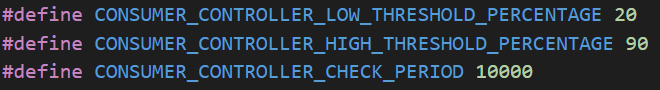
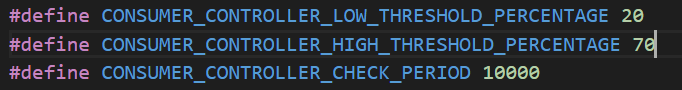
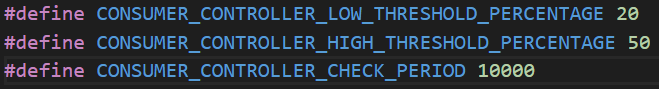
* Different values of **CONSUMER\_CONTROLLER\_LOW\_THRESHOLD\_PERCENTAGE** and different values of **CONSUMER\_CONTROLLER\_HIGH\_THRESHOLD\_PERCENTAGE**







我們根據low threshold = 5、20、和50做實驗，發現low threshold 越大，時間越長，low threshold = 5、20、50，時間分別是58 sec、54 sec 和50 sec。可能原因為當low threshold 越大 consumer 數量越容易變少。



我們根據high threshold = 50、70、和90做實驗，發現high threshold 越大，時間越長，high threshold = 50、70、90，時間分別是53 sec、54 sec 和55 sec。可能原因為當high threshold 越大 consumer 數量越不容易變大。而且我們發現當high threshold 越大，consumer 最大數量越多，原因可能是因為buffer累積太多，所以會需要更多consumer。

* Different values of **WORKER\_QUEUE\_SIZE**







我們發現worker queu size = 5、50、200的執行時間和scale up and down 的結果三者完全一樣。

* What happens if **WRITER\_QUEUE\_SIZE** is very small

我們實驗writer queue size = 1、200、4000，發現scale up and down的結果完全一樣，且完成時間也一樣。時間一樣的原因可能是因為儘管writer queue裡item的數量不同但一直有東西，而writer只有一個所以writer拿東西的速度決定完成時間。

* What happens if **READER\_QUEUE\_SIZE** is very small



我們實驗reader queue size = 1，發現scale up and down的結果完全一樣，且完成時間也一樣。原因可能是因為reader (1個) 放到reader queue的速度和producer拿的速度是一樣的，好比說同個時間裡4個producer中只有1個在做事。

**Part III Additional Experiment & Difficulties**

* Difficulties

我們覺得Experiment的結果因為有太多變因，所以很難說某個結果是哪個特定便因造成的。然後這是我們第一次接觸有multithread的程式，所以需要花時間去熟悉，且條件判斷要考慮更多事情，因為同一時間下有很多thread在執行。

* Additional Experiment

我們的目標是嘗試找出執行時間最小的參數設定，根據default參數設定，原本執行時間為54s(test case 01)。我們認為low threshold、high threshold愈小，consumer數量能變更多且維持這個數量更久，然後input queue應該設愈大愈好，因為即使worker queue滿了input queue越大，reader能放的數量越多。然後output queue 越大可以讓worker queue不容易滿。Check period 會影響很多因素，所以我們一個個測試，最後結果check period = 0.01 sec 時間最短。經過我們的測試，執行時間可以在50 秒執行完成。