

2021 OS Pthread

TEAM15

108062217 傅詠軒

108062203江浩辰

Contribution:

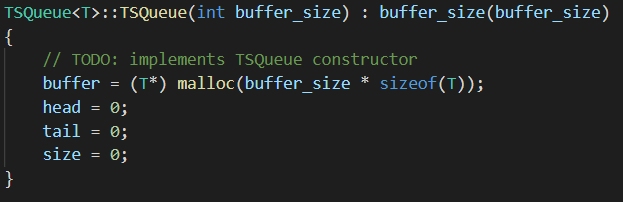
江浩辰50%

傅詠軒50%

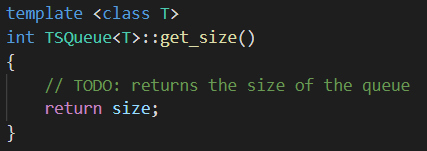
**Implementation:**

**TSQueue:**

在constructor內，先allocate space to buffer，並把head、tail、size等變數初始化為0。

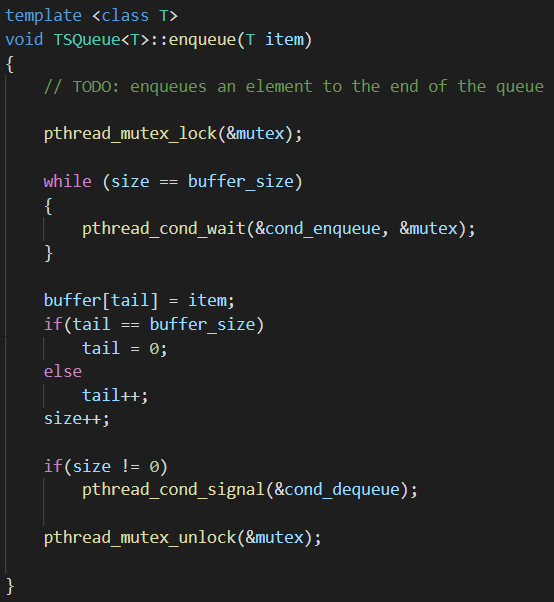


getsize()內，直接回傳size大小。



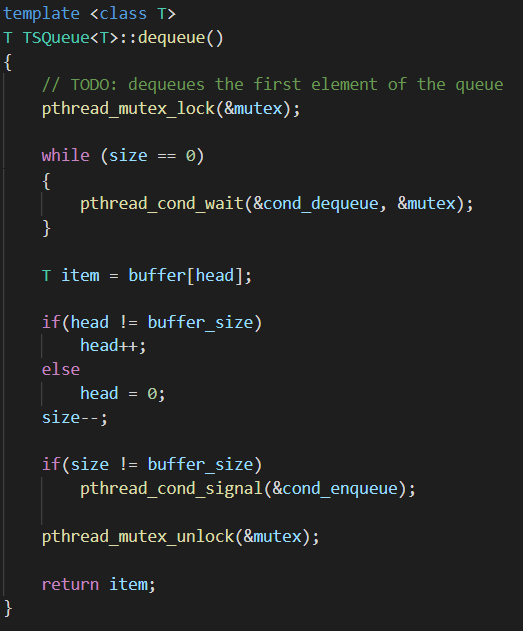
enqueue()內，將對condition variable的操作包在mutex lock內，如果目前buffer滿了，就放入conditional enqueue內等待dequeue釋出空間，接著有位子就放入buffer[tail]，並對tail進行操作跟判斷，如果到queue尾巴就移到0，

不然就++。



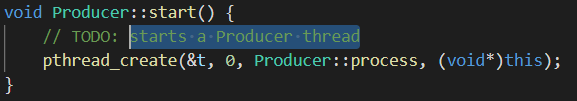
****dequeue與enqueue相似，如果queue內空的，就放入conditional dequeue內等待，有東西就把item從buffer[head]拿出來，接著判斷head是不是在尾巴，

如果是就把head移到0，不是就++，最後回傳取出的item。

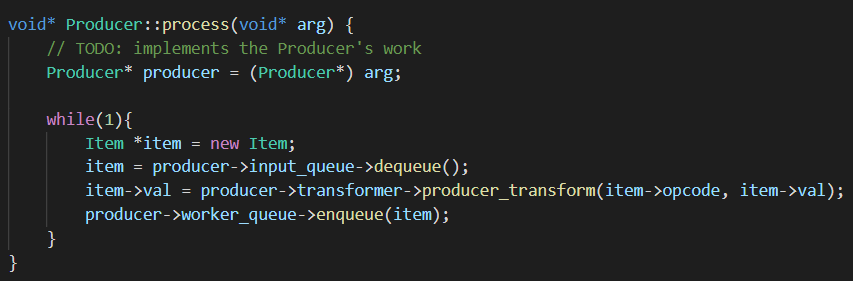


**Producer:**

在start()內starts a Producer thread。

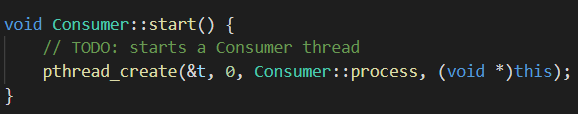


在process()內，在while loop內repeatly的從input\_queue內dequeue拿出item，之後用transformer的producer\_transform對item.val進行操作並存回，之後enqueue到worker\_queue內。

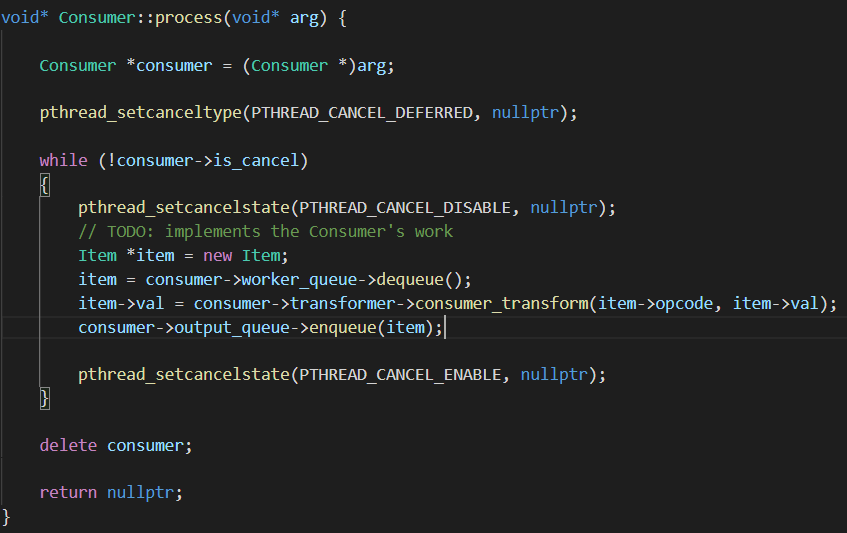


**Consumer:**

在start()內starts a Consumer thread。



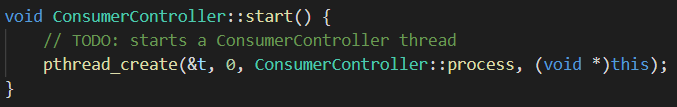
在process()內，如果不需要被cancel，則在while loop內repeatly的從worker\_queue內dequeue拿出item，之後用transformer的consumer\_transform對item.val進行操作並存回，之後enqueue到output\_queue內。如果出while代表要被cancel，將此consumer delete掉。



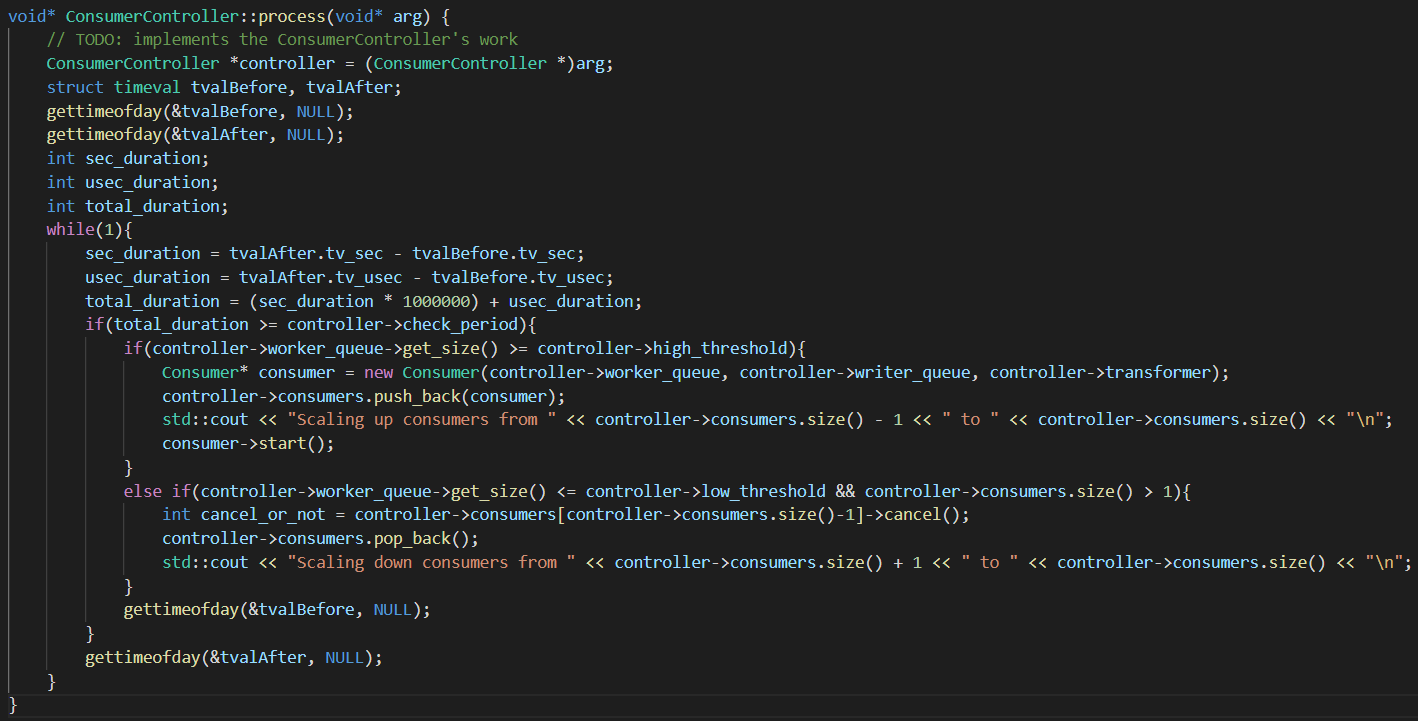
在cancel()內將is\_cancel設為true，代表此thread即將被cancel。

**ConsumerController:**

在start()內starts a ConsumerController thread。

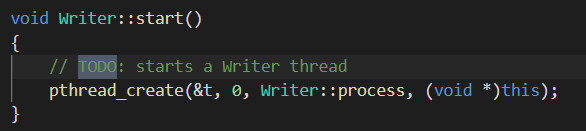


在process()內進行consumer threads的增加與減少的管理，在while loop內repeatly的檢查是否超過period了，用timeval來檢查是不是有超過設定的period，如果有，則判斷目前的worker\_queue的size，如果超過80%，就create一個consumer thread並放進controller管理的consumer thread list，如果低於20%且consumer的thread數至少有2個以上，就減少一個latest的consumer thread。

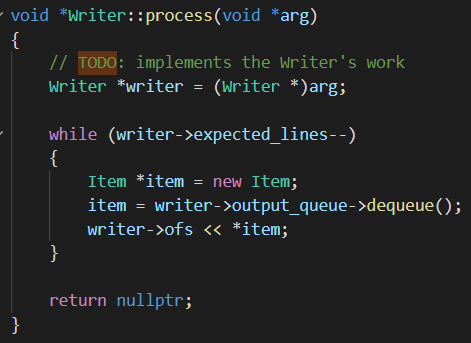
****

**Writer:**

在start()內starts a Writer thread。



在process()內，在while loop內，如果沒讀完所有的item，就repeatly從output\_queue dequeue item出來然後寫入ofstream。



**main:**

首先宣告三個TSQueue Input\_Queue, Worker\_Queue, Writer\_Queue，並new objects，以及宣告transformer, reader, writer, controller, 及四個producer threads，把low\_threshold跟high\_threshhold各是幾個item算出來，並new objects，之後對每個new出來的threads都呼叫start函數，並在最後加上reader->join()、writer->join()，表示等到做完才會去delete並end program。

**Experiments:**

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

**以testcase 00做測試:**

1000000(initial): consumer threads，最多只會scale up to 2。

10000: consumer threads，會scale up to 6。

1000: consumer threads，會scale up to 6。

100: consumer threads，會scale up to 5。

10: consumer threads，會scale up to 6。

10000000: consumer thread，只會create一個。

* Check\_Period如果太大，則在check時Queue內的item，就會先被前面create的consumer threads取出來了，所以此時會小於80%，就不會create新的consumer threads。反之如果Check\_Period較小的話，則在不斷check時可能還是大於80%，就會一直create新的consumer threads。

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

**以testcase 00做測試，並把CHECK\_PERIOD調成10000方便觀察:**

LOW 20 and HIGH 80: consumer threads，會scale up to 6。

LOW 40 and HIGH 80: consumer threads，會scale up to 6，但consumer threads開始減少的速度會快一些。

LOW 20 and HIGH 60: 會scale up to 6，有時會scale up to 7，開始增加的速度也會比較快。

LOW 40 and HIGH 60: consumer threads，會scale up to 6，有時會scale up to 7，consumer threads開始減少的速度會快一些，開始增加的速度也會比較快。

* 如果把LOW\_THRESHOLD調高，則consumer threads會比較快開始scale down。如果把HIGJ\_THRESHOLD調高，則consumer threads會比較快開始scale up。

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

**以testcase 00做測試，並把CHECK\_PERIOD調成500000方便觀察:**

200: consumer threads，會scale up to 3。

50: consumer threads，會scale up to 5，且很快開始scale up，且沒有scale down。

20: consumer threads，會scale up to 6，且很快開始scale up，且沒有scale down。

250: consumer threads，只會scale up to 1。

* Worker\_Queue size如果越小，那HIGH THRESHOLD就越小，consumer threads會較快開始scale up，且個數也可能比較多，而LOW THRESHOLD也****會更小，因此剛過LOW THRESHOLD有可能在下個CHECK\_PERIOD還沒到就dequeue完了，因此不會scale down consumer thread。
* 而如果越大，HIGH THRESHOLD就越大，則會越慢開始scale up，數量也可能較少，如果剛好HIGHTHRESHOLD = Item數量，則只會scale up一次。

**4. What happens if WRITER\_QUEUE\_SIZE is very small?**

**以testcase 00做測試，並把CHECK\_PERIOD調成500000方便觀察:**

把WRITER\_QUEUE\_SIZE設為1，則consumer threads每次要enqueue都要等writer把Items從queue內dequeue出來，會比較慢結束，且會比較慢開始scale down consumer threads。

**5. What happens if READER\_QUEUE\_SIZE is very small?**

**以testcase 00做測試，並把CHECK\_PERIOD調成500000方便觀察:**

把READER\_QUEUE\_SIZE設為1，則每次reader讀完一個item後，只有一個producer thread會去dequeue，再來reader才會去讀下一個item，重複循環，因此WORKER QUEUE內的Items會累積的較慢，因此consumer threads會很慢才開始scale up，整個程式的執行時間也會較長。

**Difficulties:**

1. 一開始在看TSQueue的定義跟實作的時候為了釐清conditional variable的使用花了不少時間。

2. ConsumerController中要check在每段period檢查一次worker queue size，clock()會不準，花了一段時間才找到Linux可以用timeval的方法。