Operating System

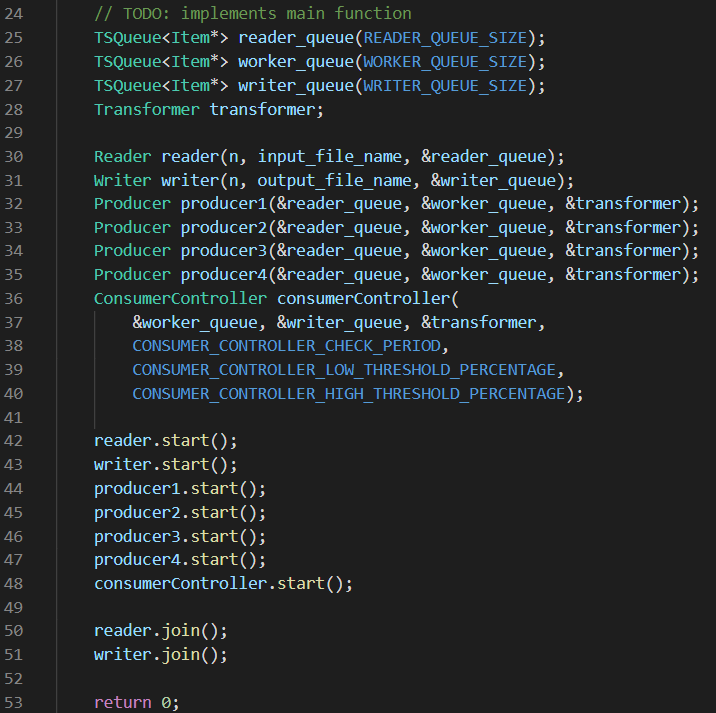
Pthreads Report

組別：25

陳凱揚：Implement、Experiment、Report  
簡佩如：Implement、Experiment、Report

1. Implement
   1. main.cpp

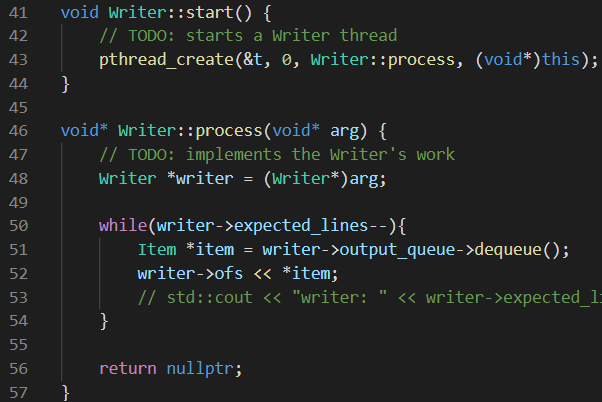
首先在main裡面（圖1-1），我先建立了所有需要的物件，包括3個queue、transformer、reader、writer、4個producer、consumerController等，並傳入對應參數至建構元，其中controller會根據worker\_queue的儲存數量控制consumer的數量。接著將這些thread都呼叫start()開始運行，最後需要呼叫reader和writer的join()，等待這2個thread運行完畢，再結束程式。此外，這邊因為thread都使用local變數，在最後main結束後，釋放local變數後，便同時會釋放運行中的thread，但如果這些thread是由new所開啟的，則一定要將所有東西delete掉，否則其他thread仍會持續進行。



▲ 圖1-1

* 1. writer.hpp

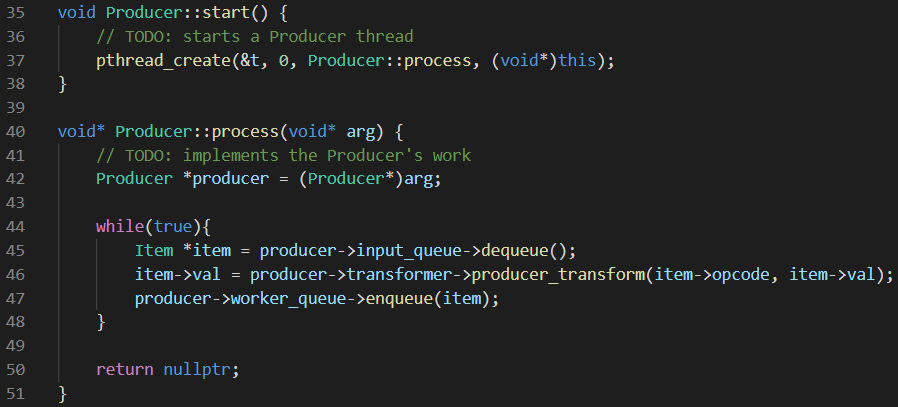
在Wirter中，需要完成2個函式（圖1-2）。在start()中只要呼叫pthread\_create()，參數放入Write::process和this，開始運行這個thread即可；在process()中與Reader的process()類似，需要不斷從output\_queue中呼叫dequeue()取出1個item，並寫進ofs中，直到所有item寫完就結束。



▲ 圖1-2

* 1. producer.hpp

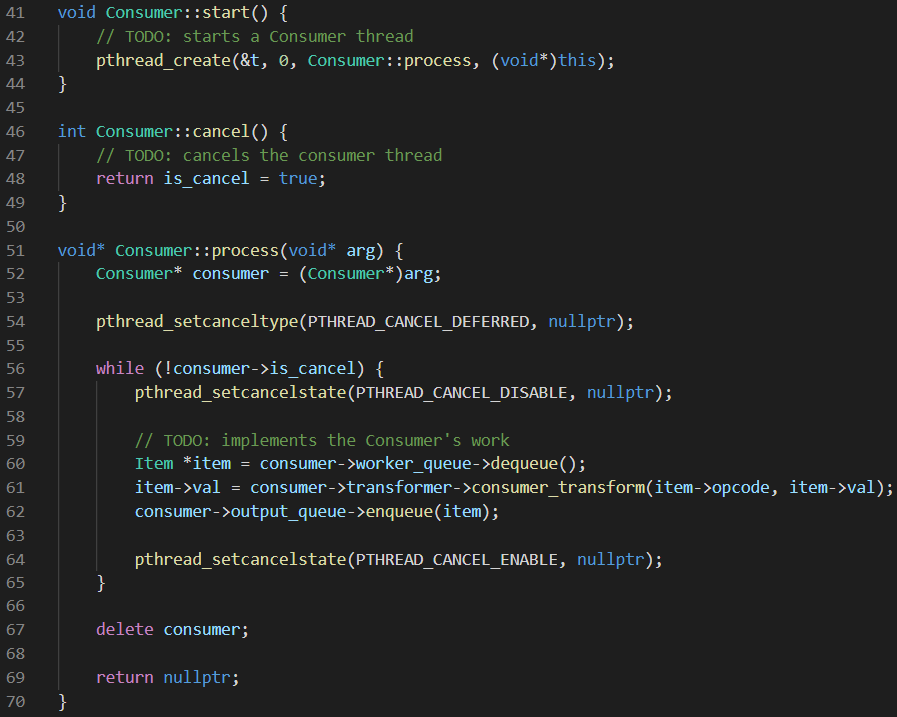
在Producer中（圖1-3），start()呼叫pthread\_create()開始運行thread；而process()會有一個無限迴圈，不斷從input\_queue取出1個item，並使用producer\_transform()，傳入item的opcode和val來決定新的val值，最後再把item放進worker\_queue裡。



▲ 圖1-3

* 1. consumer.hpp

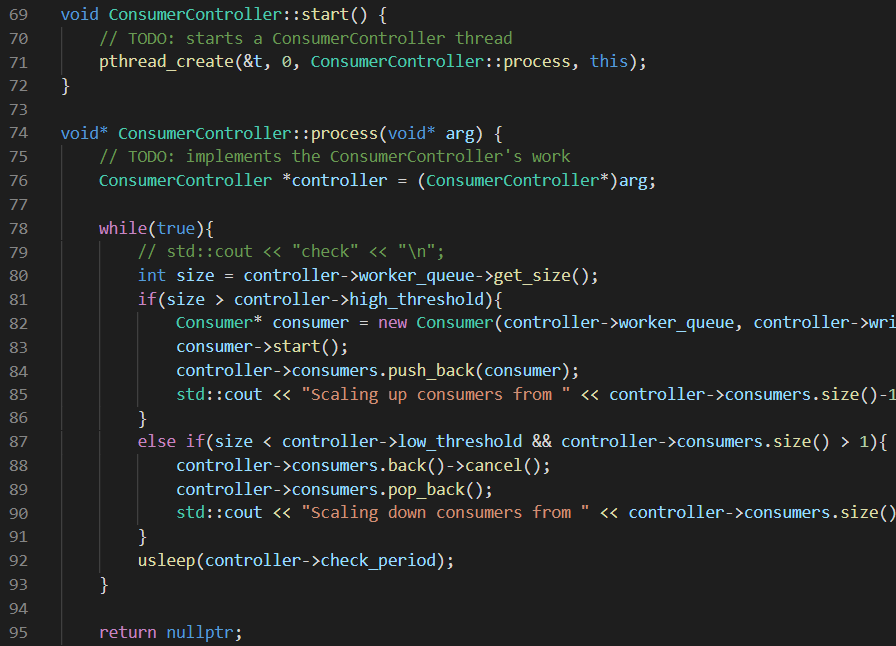
Consumer與Producer大致相同（圖1-4），但多了cancel()函式，用來給ConsumerController呼叫來結束這個thread，cancel()會將is\_cancel設為true，即可跳出process()的迴圈，結束此thread。此外，為了防止thread在搬運item的過程中被直接結束，會先set canceltype為DEFERRED，也就是同步的意思，並在搬運前set calcelstate為DISABLE，直到搬運結束再設回ENABLE。



▲ 圖1-4

* 1. consumer\_controller.hpp

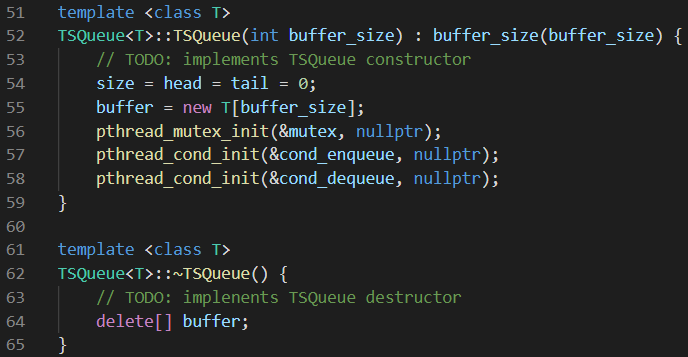
在ConsumerController中（圖1-5），會有一個無限迴圈，週期的不斷檢查worker\_queue的大小，當大小超過high\_threshold時，建立一個新的Consumer且呼叫start()開始運行，並放置到vector consumers的最後面，同時輸出前後變化；當大小小於low\_threshold且目前consumers的大小大於1時（避免Consumer數量減為0），呼叫cancel()將最近建立的consumer結束，並將consumers pop\_back()，同時輸出前後變化。最後會使用usleep()，並傳入check\_period，代表等待幾微秒後再繼續向下執行，來達到週期性檢查的效果。



▲ 圖1-5

* 1. ts\_queue.hpp

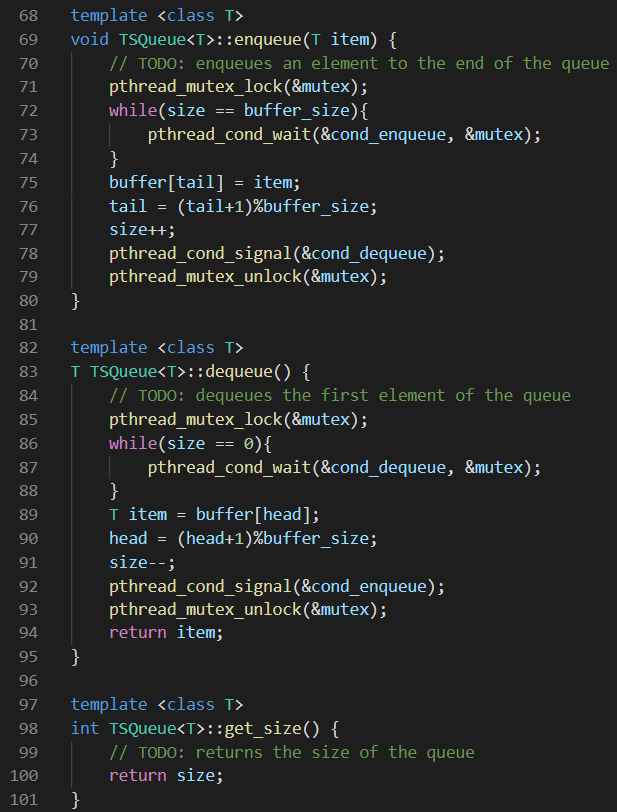
在TSQueue的建構元中（圖1-6），會將size、head、tail都設為0，並acllocate 大小為buffer\_size的陣列給buffer，接著把mutex、cond\_enqueue、cond\_dequeue初始化。而在解構元中，只需delete掉分配給buffer的空間。



▲ 圖1-6

在enqueue()和dequeue()中（圖1-7），這是這次作業最重要的部分，因為會同時有多個thread來存取相同的記憶體，而產生synchronization的問題，必須使用mutex、condition variable來達成mutual exclusion。而get\_size()只需直接回傳size，因為size的值在enqueue和dequeue中都會做更新。

我的實作方法為先按照一般的方式，在enqueue()中，將item放入buffer[tail]裡，並更新tail和size的值；在dequeue()中，將buffer[head]取出，並更新head和size的值。接著在enqueue()中，buffer滿了時，就呼叫cond\_wait()，參數為cond\_enqueue，將目前這個thread放入等待中，並在完成enqueue後呼叫cond\_signal()，參數為cond\_dequeue，告訴等待dequeue的thread可以開始dequeue，若沒有thread在等待中，則沒有影響，最後使用mutex的lock和unlock把整個函式包起來。而在dequeue()中與enqueue()相似，首先在buffer為空時，呼叫cond\_wait()進入cond\_dequeue中等待，並在完成dequeue後呼叫cond\_signal()，告訴等待enqueue的thread可以開始enqueue，最後同樣將整個函式以mutex的lock和unlock包起來。



▲ 圖1-7

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

原本CONSUMER\_CONTROLLER\_CHECK\_PERIOD為 1000000。

一張含有 文字 的圖片

自動產生的描述

由1000000開始向上乘兩倍去觀察變化，2000000、4000000、… 以此類推，發現scaling up 的次數會減半，直到剩下一次。

一張含有 文字 的圖片

自動產生的描述 一張含有 文字 的圖片

自動產生的描述

一張含有 文字 的圖片

自動產生的描述 

由1000000開始向下除兩倍觀察變化，一開始會在 scaling up 5~6跟6~7之間跑。

一張含有 文字 的圖片

自動產生的描述 一張含有 文字 的圖片

自動產生的描述

一張含有 文字 的圖片

自動產生的描述 一張含有 文字, 電子用品, 鍵盤 的圖片

自動產生的描述

一張含有 文字 的圖片

自動產生的描述 一張含有 文字, 電子用品, 鍵盤 的圖片

自動產生的描述

一張含有 文字, 電子用品, 鍵盤 的圖片

自動產生的描述 一張含有 文字 的圖片

自動產生的描述

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

固定CONSUMER\_CONTROLLER\_LOW\_THRESHOLD\_PERCENTAGE，則scaling的數量會隨著CONSUMER\_CONTROLLER\_HIGH\_THRESHOLD\_PERCENTAGE上升而下降。

一張含有 文字 的圖片

自動產生的描述 一張含有 文字 的圖片

自動產生的描述

一張含有 文字, 電子用品, 鍵盤 的圖片

自動產生的描述 一張含有 文字 的圖片

自動產生的描述

固定CONSUMER\_CONTROLLER\_HIGH\_THRESHOLD\_PERCENTAGE，  
當CONSUMER\_CONTROLLER\_LOW\_THRESHOLD\_PERCENTAGE小於17就不會scaling down了。

一張含有 文字 的圖片

自動產生的描述 一張含有 文字 的圖片

自動產生的描述

固定CONSUMER\_CONTROLLER\_HIGH\_THRESHOLD\_PERCENTAGE，Scaling down的次數會隨著CONSUMER\_CONTROLLER\_LOW\_THRESHOLD\_PERCENTAGE增加而增加。

一張含有 文字 的圖片

自動產生的描述 一張含有 文字 的圖片

自動產生的描述

一張含有 文字 的圖片

自動產生的描述

* 1. Different values of WORKER\_QUEUE\_SIZE

實驗方法是200\*2^n, n = 2,3,4,…，會發現scaling up的次數基本上是固定不動的，而scaling down的次數會在1~2次之間跳，主要是以1次為主。

一張含有 文字, 電子用品, 鍵盤 的圖片

自動產生的描述 一張含有 文字, 電子用品, 鍵盤 的圖片

自動產生的描述

基本上 WORKER\_QUEUE\_SIZE 小於80就跑不動了，所以不能太小。

一張含有 文字, 電子用品, 鍵盤 的圖片

自動產生的描述 一張含有 文字 的圖片

自動產生的描述

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

test00只有在size等於1的時候會多一次scaling down。因為test00效果不明顯，因此使用test01去測試，整體來說是scaling up跟scaling down的次數會減少。

一張含有 文字, 電子用品, 鍵盤 的圖片

自動產生的描述

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

在test00中，不管READER\_QUEUE\_SIZE是4000還是1，scaling的次數都是一樣的。在test01中，READER\_QUEUE\_SIZE是4000或1也沒有顯著影響，只有4000 的scaling 次數稍微多一點。

一張含有 文字, 電子用品, 鍵盤 的圖片

自動產生的描述

1. Difficulties and Feedback

這次作業中由於都有TODO的引導，也有很多相似且寫好的程式，讓整個implementation簡單不少，唯一比較遇到問題的只有週期性的檢查consumer的地方，一開始使用clock()來得到時間檢查，但一直都比預期的快，後來改採usleep()就順利解決問題了。此外，這也是第一次實作使用Pthread的函式庫，增加了不少對synchronization解決方法的熟悉度。