Pthread Report

組別:os21team34

成員:江咏宸108030001、蔡懿晨107070035

Contribution	
Tracecode與相關report	江咏宸
Implement與相關report	蔡懿晨

Explain your implementation as requested.

1. ts_queue.hpp

```
51 template <class T>
52 TSQueue<T>::TSQueue(int buffer_size) : buffer_size(buffer_size) {
53   // TODO: implements TSQueue constructor
54   buffer = new T[buffer_size];
55   size = 0;
56   head = 0;
57   tail = -1;
58   pthread_mutex_init(&mutex, NULL);
59   pthread_cond_init(&cond_enqueue, NULL);
60   pthread_cond_init(&cond_dequeue, NULL);
61 }
62
63 template <class T>
64 TSQueue<T>::~TSQueue() {
65   // TODO: implenents TSQueue destructor
66   delete [] buffer;
67   pthread_mutex_destroy(&mutex);
68   pthread_cond_destroy(&cond_enqueue);
69   pthread_cond_destroy(&cond_dequeue);
70 }
```

- 建構TSQueue的實作，首先先將buffer空間建立出來，variable初始值設好。這邊是將tail設為-1。
- 要使用mutex需先宣告58~60行
- 解構就須將buffer空間delete，並destroy mutex、condition variable。

```
72 template <class T>
73 void TSQueue<T>::enqueue(T item) {
74   // TODO: enqueues an element to the end of the queue
75   // chack mutex
76   pthread_mutex_lock(&mutex);
77
78   // if buffer is full, wait for deque
79   if (size == buffer_size)
80     pthread_cond_wait(&cond_enqueue, &mutex);
81
82   pthread_mutex_unlock(&mutex);
83   pthread_mutex_lock(&mutex);
84   //std::cout<<"In eneueue :: enqueue at position="<<tail<<std::endl;
85   // check cond var
86   tail++;
87   tail %= buffer_size;
88   buffer[tail] = item;
89   size++;
90
91   pthread_cond_signal(&cond_dequeue);
92   // unlock
93   pthread_mutex_unlock(&mutex);
94 }
```

- enque的實作，先呼叫lock把CS鎖住，當離開時呼叫unlock
- 如果buffer已滿，就呼叫wait等待enqueue
- 當可以加入時，也就是有其他thread call signal時就更新buffer、size、tail等資訊
- 最後要signal dequeue才能讓dequeue wake up

```cpp
template <class T>
T TSQueue<T>::dequeue() {
  // TODO: dequeues the first element of the queue
  pthread_mutex_lock(&mutex);

  // check cond var
  if (size == 0)
    pthread_cond_wait(&cond_dequeue, &mutex);
  pthread_mutex_unlock(&mutex);

  pthread_mutex_lock(&mutex);
  //std::cout<<"In dequeue :: dequeue at position="<<head<<std::endl;
  T out = buffer[head++];
  head %= buffer_size;
  size--;

  pthread_cond_signal(&cond_enqueue);
  // unlock
  pthread_mutex_unlock(&mutex);
  return out;
}
```

- dequeue的實作，先呼叫lock把CS鎖住，當離開時呼叫unlock
- 如果buffer為空，就呼叫wait，block dequeue
- 當可以dequeue時，也就是有其他thread call signal時就更新buffer、size、tail等資訊
- 最後要signal enqueue，使enqueue wake up

2. Producer.hpp

```cpp
void Producer::start() {
  // TODO: starts a Producer thread
  pthread_create(&t, 0, Producer::process, (void*)this);
}

void* Producer::process(void* arg) {
  // TODO: implements the Producer's work

  Producer* producer = (Producer*)arg;
  while (1){

    Item *item = producer->input_queue->dequeue();
   //std::cout<<item->key<<" "<<item->val<<"\n";
    item->val = producer->transformer->producer_transform(item->opcode, item->val);
   std::cout<<item->key<<" "<<item->val<<"\n";
    producer->worker_queue->enqueue(item);

    // finish and leave
    if (producer->input_queue->get_size() == 0 && producer->worker_queue->get_size() == 0)
      break;
  }
  return nullptr;
}

#endif // PRODUCER_HPP

```

- start呼叫了pthread_create(thread,attr,routine,arg)，thread，attr被用來設定thread attribution，通常用NULL；routine就是傳入要執行的function，這邊傳入我

們要用的process；arg就是傳入process所需要的arg，這邊要注意要包成void pointer。

- while裡面是在實作dequeue input queue，將取出的item經過transformer後得到新的value，將新的值enqueue進去worker queue。如果input queue為空，或是worker queue為空，就break，停止搬運，完成離開。

3. consumer.hpp

```
void Consumer::start() {
  // TODO: starts a Consumer thread
  pthread_create(&t, 0, Consumer::process, (void*)this);
}

int Consumer::cancel() {
  // TODO: cancels the consumer thread
  is_cancel = true;

  return pthread_cancel(t);
}

void* Consumer::process(void* arg) {
  Consumer* consumer = (Consumer*)arg;

  pthread_setcanceltype(PTHREAD_CANCEL_DEFERRED, nullptr);

  while (!consumer->is_cancel) {
    pthread_setcancelstate(PTHREAD_CANCEL_DISABLE, nullptr);

    // TODO: implements the Consumer's work

    Item *item = consumer->worker_queue->dequeue();
    item->val = consumer->transformer->consumer_transform(item->opcode, item->val);
    consumer->output_queue->enqueue(item);

    pthread_setcancelstate(PTHREAD_CANCEL_ENABLE, nullptr);
  }

  //delete consumer;

  return nullptr;
}
```

- Consumer一樣有start，呼叫create去執行process
- process的while裡面是在實作dequeue worker queue，將取出的item經過transformer後得到新的value，將新的值enqueue進去output queue。
- pthread_setcanceltype()、pthread_setcancelstate()是prevent the thread being canceled at any time

4. consumer_controller.hpp

```cpp
void ConsumerController::start() {
  // TODO: starts a ConsumerController thread
  pthread_create(&t, 0, ConsumerController::process, (void*)this);
}

void* ConsumerController::process(void* arg) {
  // TODO: implements the ConsumerController's work
  ConsumerController* controller = (ConsumerController*)arg;

  while(1) {
    //check period
    usleep(controller->check_period);

    if (controller->worker_queue->get_size() < controller->low_threshold && controller->consumers.size() > 1) {
      // lower than lower threshold, need to scale down 1
      std::cout << "Scaling down consumers from " << controller->consumers.size() << " to " << controller->consumers.size() - 1 << "\n";
      controller->consumers[controller->consumers.size()-1]->cancel();
      controller->consumers.pop_back();
    }
    else if (controller->worker_queue->get_size() > controller->high_threshold || controller->consumers.empty()) {
      // greater than higher threshold, scale up 1
      std::cout << "Scaling up   consumers from " << controller->consumers.size() << " to " << controller->consumers.size() + 1 << "\n";
      controller->consumers.push_back(new Consumer(controller->worker_queue, controller->writer_queue, controller->transformer));
      controller->consumers[controller->consumers.size()-1]->start();
    }
```

- start一樣呼叫create去執行process
- Process呼叫usleep()去Check Work Queue status periodically，而if else是在檢查，當 Worker Queue size 小於 low_threshold，需要呼叫Consumer->cancel做cancel the newest Consumer，並將其pop出去consumer的queue；當Worker Queue size超過 high_threshold，就create a new Consumer thread，因為spec規定直到程式結束都要至少有一個consumer，所以必須確認consumer是否為空，如果是的話就一樣要create新的thread。

5. writer.hpp

```cpp
void Writer::start() {
  // TODO: starts a Writer thread
  pthread_create(&t, 0, Writer::process, (void*)this);
}

void* Writer::process(void* arg) {
  // TODO: implements the Writer's work
  Writer* writer = (Writer*)arg;

  while (writer->expected_lines--) {
    Item *item = writer->output_queue->dequeue();
    writer->ofs << *item;
  }

  return nullptr;
}
```

- start一樣呼叫create去執行process
- process就是把writer queue的item取出，印在output file

6. main.cpp

producer.hpp consumer.hpp main.cpp

```cpp
int main(int argc, char **argv)
{
  assert(argc == 4);

  int n = atoi(argv[1]);
  std::string input_file_name(argv[2]);
  std::string output_file_name(argv[3]);

  // TODO: implements main function

  TSQueue<Item *> *input_queue = new TSQueue<Item *>(READER_QUEUE_SIZE);
  TSQueue<Item *> *worker_queue = new TSQueue<Item *>(WORKER_QUEUE_SIZE);
  TSQueue<Item *> *output_queue = new TSQueue<Item *>(WRITER_QUEUE_SIZE);
  Transformer *transformer = new Transformer();
  ConsumerController *controller = new ConsumerController(worker_queue, output_queue, transformer, CONSUMER_CONTROLLER_CHECK_PERIOD,
CONSUMER_CONTROLLER_LOW_THRESHOLD_PERCENTAGE, CONSUMER_CONTROLLER_HIGH_THRESHOLD_PERCENTAGE);
  Reader *reader = new Reader(n, input_file_name, input_queue);
  Writer *writer = new Writer(n, output_file_name, output_queue);
  Producer *p1 = new Producer(input_queue, worker_queue, transformer);
  Producer *p2 = new Producer(input_queue, worker_queue, transformer);
  Producer *p3 = new Producer(input_queue, worker_queue, transformer);
  Producer *p4 = new Producer(input_queue, worker_queue, transformer);

  reader->start();
  p1->start();
  p2->start();
  p3->start();
  p4->start();
  controller->start();
  writer->start();
```

```cpp
  reader->join();
  writer->join();

  delete writer;
  delete reader;
  delete p1;
  delete p2;
  delete p3;
  delete p4;
  delete controller;
  return 0;
}

```

- 27~37行都是在new出新的物件，最後50~57則是刪掉這些有allocate的memory釋放掉。
- 根據spec，總共創了4個producer thread，p1~p4。
- Consumer_contoller這邊比較特別的是傳進去的後兩個參數是用來調整consumer個數的low_threshold、high_threshold，依照spec會是20%、80%的worker size，所以我是傳入CONSUMER_CONTROLLER_LOW_THRESHOLD_PERCENTAGE*WORKER_QUEUE_SIZE/100, CONSUMER_CONTROLLER_HIGH_THRESHOLD_PERCENTAGE*WORKER_QUEUE_SIZE/100
- 呼叫start來開始每一個thread
- 呼叫join，執行Pthread_join(threadId，status)，等待reader、writer thread 的work結束。

Explain what experiments you have done and what are the results. You are encouraged to do more experiments.

1. Different values of CONSUMER_CONTROLLER_CHECK_PERIOD.
   原本CONSUMER_CONTROLLER_CHECK_PERIOD=1000000

```
[os21team34@localhost NTHU-OS-Pthreads]$ ./main 200 ./tests/00.in ./tests/00.out
Scaling up   consumers from 0 to 1
Scaling up   consumers from 1 to 2
Scaling up   consumers from 2 to 3
[os21team34@localhost NTHU-OS-Pthreads]$
```

CONSUMER_CONTROLLER_CHECK_PERIOD=10000

```
[os21team34@localhost NTHU-OS-Pthreads]$ ./main 200 ./tests/00.in ./tests/00.out
Scaling up   consumers from 0 to 1
Scaling up   consumers from 1 to 2
Scaling up   consumers from 2 to 3
Scaling up   consumers from 3 to 4
Scaling up   consumers from 4 to 5
Scaling up   consumers from 5 to 6
Scaling down consumers from 6 to 5
Scaling down consumers from 5 to 4
Scaling down consumers from 4 to 3
Scaling down consumers from 3 to 2
Scaling down consumers from 2 to 1
[os21team34@localhost NTHU-OS-Pthreads]$ ./scripts/verify --output ./tests/00.out --answer ./tests/00.ans
```

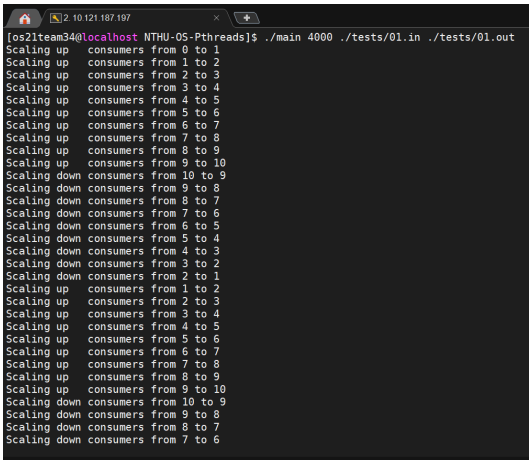
CONSUMER_CONTROLLER_CHECK_PERIOD=10000000

```
[os21team34@localhost NTHU-OS-Pthreads]$ ./main 200 ./tests/00.in ./tests/00.out
Scaling up   consumers from 0 to 1
[os21team34@localhost NTHU-OS-Pthreads]$
```

因為在Consumer_controller會呼叫usleep()，然後傳入check period，所以period改小的話會讓調整的次數變多一點，因為檢查的次數變多了；相反的如果把檢查的時間拉長，就會較少去調整consumer的數量。

2. Different values of CONSUMER_CONTROLLER_LOW_THRESHOLD_PERCENTAGE and CONSUMER_CONTROLLER_HIGH_THRESHOLD_PERCENTAGE.

   原本:20/80



   10/90

```
g++ -o ts_queue_test -static -std=c++11 -O3 -pthread ts_queue_test.cpp transformer
[os21team34@localhost NTHU-OS-Pthreads]$ ./main 4000 ./tests/01.in ./tests/01.out
Scaling up   consumers from 0 to 1
Scaling up   consumers from 1 to 2
Scaling up   consumers from 2 to 3
Scaling up   consumers from 3 to 4
Scaling up   consumers from 4 to 5
Scaling up   consumers from 5 to 6
Scaling up   consumers from 6 to 7
Scaling up   consumers from 7 to 8
Scaling up   consumers from 8 to 9
Scaling down consumers from 9 to 8
Scaling down consumers from 8 to 7
Scaling down consumers from 7 to 6
Scaling down consumers from 6 to 5
Scaling down consumers from 5 to 4
Scaling down consumers from 4 to 3
Scaling down consumers from 3 to 2
Scaling down consumers from 2 to 1
Scaling up   consumers from 1 to 2
Scaling up   consumers from 2 to 3
Scaling up   consumers from 3 to 4
Scaling up   consumers from 4 to 5
Scaling up   consumers from 5 to 6
Scaling up   consumers from 6 to 7
Scaling up   consumers from 7 to 8
Scaling up   consumers from 8 to 9
Scaling up   consumers from 9 to 10
Scaling down consumers from 10 to 9
Scaling down consumers from 9 to 8
Scaling down consumers from 8 to 7
Scaling down consumers from 7 to 6
```

40/60

```
[os21team34@localhost NTHU-OS-Pthreads]$ ./main 4000 ./tests/01.in ./tests/01.out
Scaling up   consumers from 0 to 1
Scaling up   consumers from 1 to 2
Scaling up   consumers from 2 to 3
Scaling up   consumers from 3 to 4
Scaling up   consumers from 4 to 5
Scaling up   consumers from 5 to 6
Scaling up   consumers from 6 to 7
Scaling up   consumers from 7 to 8
Scaling up   consumers from 8 to 9
Scaling up   consumers from 9 to 10
Scaling down consumers from 10 to 9
Scaling down consumers from 9 to 8
Scaling down consumers from 8 to 7
Scaling down consumers from 7 to 6
Scaling down consumers from 6 to 5
Scaling down consumers from 5 to 4
Scaling down consumers from 4 to 3
Scaling down consumers from 3 to 2
Scaling down consumers from 2 to 1
Scaling up   consumers from 1 to 2
Scaling up   consumers from 2 to 3
Scaling up   consumers from 3 to 4
Scaling up   consumers from 4 to 5
Scaling up   consumers from 5 to 6
Scaling up   consumers from 6 to 7
Scaling up   consumers from 7 to 8
Scaling up   consumers from 8 to 9
Scaling up   consumers from 9 to 10
Scaling up   consumers from 10 to 11
Scaling down consumers from 11 to 10
Scaling down consumers from 10 to 9
```

小於low_threshold要delete，大於high_threshold要create。所以如果把兩者區間拉大，比較不會create或delete，如果把區間縮小就容易create、delete。

3. Different values of WORKER_QUEUE_SIZE.

   原本200

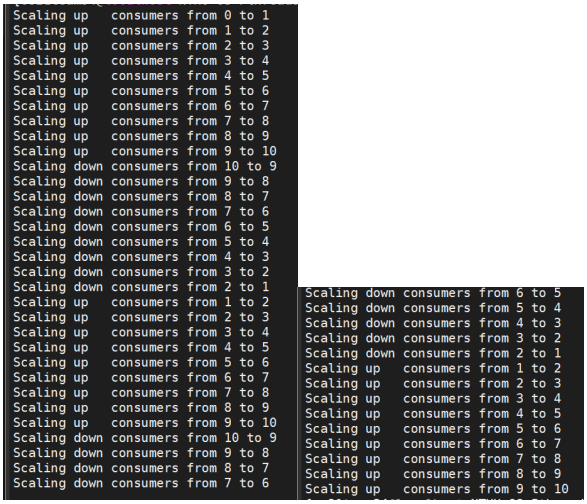
```
2. 10.121.187.197
[os21team34@localhost NTHU-OS-Pthreads]$ ./main 4000 ./tests/01.in ./tests/01.out
Scaling up   consumers from 0 to 1
Scaling up   consumers from 1 to 2
Scaling up   consumers from 2 to 3
Scaling up   consumers from 3 to 4
Scaling up   consumers from 4 to 5
Scaling up   consumers from 5 to 6
Scaling up   consumers from 6 to 7
Scaling up   consumers from 7 to 8
Scaling up   consumers from 8 to 9
Scaling up   consumers from 9 to 10
Scaling down consumers from 10 to 9
Scaling down consumers from 9 to 8
Scaling down consumers from 8 to 7
Scaling down consumers from 7 to 6
Scaling down consumers from 6 to 5
Scaling down consumers from 5 to 4
Scaling down consumers from 4 to 3
Scaling down consumers from 3 to 2
Scaling down consumers from 2 to 1
Scaling up   consumers from 1 to 2
Scaling up   consumers from 2 to 3
Scaling up   consumers from 3 to 4
Scaling up   consumers from 4 to 5
Scaling up   consumers from 5 to 6
Scaling up   consumers from 6 to 7
Scaling up   consumers from 7 to 8
Scaling up   consumers from 8 to 9
Scaling up   consumers from 9 to 10
Scaling down consumers from 10 to 9
Scaling down consumers from 9 to 8
Scaling down consumers from 8 to 7
Scaling down consumers from 7 to 6
```

800:

```
[os21team34@localhost NTHU-OS-Pthreads]$ ./main 4000 ./tests/01.in ./tests/01.out
Scaling up   consumers from 0 to 1
Scaling up   consumers from 1 to 2
Scaling up   consumers from 2 to 3
Scaling up   consumers from 3 to 4
Scaling up   consumers from 4 to 5
Scaling up   consumers from 5 to 6
Scaling up   consumers from 6 to 7
Scaling up   consumers from 7 to 8
Scaling down consumers from 8 to 7
Scaling down consumers from 7 to 6
Scaling down consumers from 6 to 5
Scaling down consumers from 5 to 4
Scaling down consumers from 4 to 3
Scaling down consumers from 3 to 2
Scaling up   consumers from 2 to 3
Scaling up   consumers from 3 to 4
Scaling up   consumers from 4 to 5
Scaling up   consumers from 5 to 6
Scaling down consumers from 6 to 5
Scaling down consumers from 5 to 4
Scaling down consumers from 4 to 3
Scaling down consumers from 3 to 2
Scaling down consumers from 2 to 1
[os21team34@localhost NTHU-OS-Pthreads]$
```

把worker queue size調大，就會影響low、high_threshold。當high_threshold比較大就比較不會create出新的consumer。

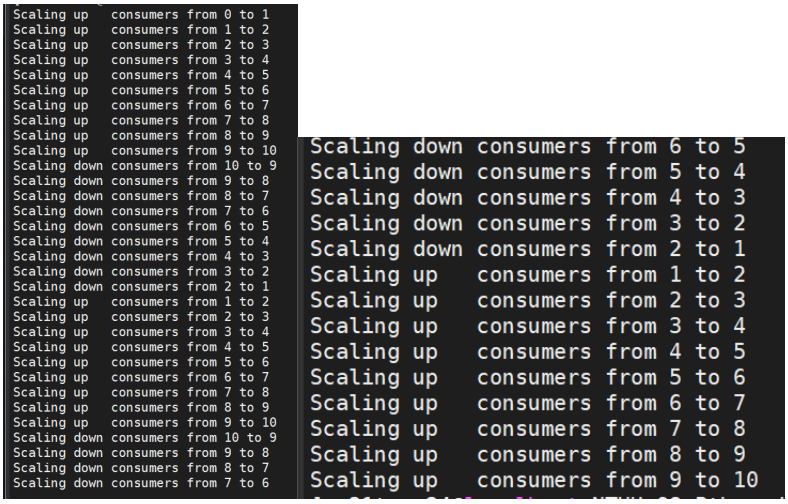
4. What happens if WRITER_QUEUE_SIZE is very small?



會發現做比較多次的調整，是因為writer queue如果太小，會不能裝太多東西、很容易滿，滿的時候 consumer 就不能放東西，因此 consumer不能動，不過producer還是可以一直放東西到worker queue，因為worker queue東西多，就會需要提高consumer。

5. What happens if READER_QUEUE_SIZE is very small?

調整成5:



沒有太大的差異，因為reader比較滿也不會影響後面的worker queue

What difficulties did you encounter when implementing this assignment?

一開始看不太懂但之後回去看了PPT相關章節後，大概清楚要怎麼做。在寫producer的時候忘記加while loop，程式一直停下來，想了一天才發現QQ

最後還有發生一個問題是tests00會過01卻有問題，去PPT重新看了一次cond的部分，才發現我把while寫成if，在跑太大的測資時會有item key重複讀取的問題，改好之後就過了。這個作業蠻好玩的但是真的好累。