OS Pthread Report

Team member:

110062204 呂宜嫻: (正在努力寫MP4) **110062239 侯茹文:** all!

Implement

main.cpp

```
// TODO: implements main function
TSQueue<Item*> input_queue(READER_QUEUE_SIZE);
TSQueue<Item*> worker_queue(WORKER_QUEUE_SIZE);
TSQueue<Item*> output_queue(WRITER_QUEUE_SIZE);
Transformer transformer;

Reader reader(n, input_file_name, &input_queue);
Writer writer(n, output_file_name, &output_queue);
Producer producer1(&input_queue, &worker_queue, &transformer);
Producer producer2(&input_queue, &worker_queue, &transformer);
Producer producer3(&input_queue, &worker_queue, &transformer);
Producer producer4(&input_queue, &worker_queue, &transformer);
ConsumerController consumer_controller(&worker_queue, &output_queue, &transformer,
                    CONSUMER_CONTROLLER_CHECK_PERIOD,
                    CONSUMER_CONTROLLER_LOW_THRESHOLD_PERCENTAGE *
WORKER_QUEUE_SIZE / 100,
                    CONSUMER_CONTROLLER_HIGH_THRESHOLD_PERCENTAGE *
WORKER_QUEUE_SIZE / 100);

reader.start();
writer.start();
producer1.start();
producer2.start();
producer3.start();
producer4.start();
consumer_controller.start();

reader.join();
writer.join();
```

> 一開始先建構需要的物件，並將其所需的參數傳入。而此處需要注意的地方是傳入consumer_controller的low threshold以及high threshold，因為我在consumer controller中採用直接比較的方式，所以這裡要將原本的80%轉為實際可用的量。
>
> 而後將需要運行的部分start()，再下面的join()是讓運行可以等待reader、writer運行完畢後，再結束程式。

ts_queue.hpp

```
TSQueue<T>::TSQueue(int buffer_size) : buffer_size(buffer_size) {
    // TODO: implements TSQueue constructor
    size = 0;
    head = 0;
    tail = 0;

    buffer = new T[buffer_size];
    pthread_mutex_init(&mutex, nullptr);
    pthread_cond_init(&cond_enqueue, nullptr);
    pthread_cond_init(&cond_dequeue, nullptr);
}
```

> 這裡先將一些需要用到的參數進行初始化，並分配buffer_size給buffer使用，然後初始化接下來可能需要用到的lock。

```
TSQueue<T>::~TSQueue() {
    // TODO: implenents TSQueue destructor
    delete[] buffer;

    pthread_cond_destroy(&cond_enqueue);
    pthread_cond_destroy(&cond_dequeue);
    pthread_mutex_destroy(&mutex);
}
```

> 在結束的時候刪除buffer，並destroy所用的lock。

```
void TSQueue<T>::enqueue(T item) {
    // TODO: enqueues an element to the end of the queue
    pthread_mutex_lock(&mutex);

    while(size == buffer_size){
        pthread_cond_wait(&cond_enqueue, &mutex);
    }
    buffer[tail] = item;
    tail =  (tail + 1) % buffer_size;
    size++;
    pthread_cond_signal(&cond_dequeue);
    pthread_mutex_unlock(&mutex);
}
```

> 因為接下來的操作都是critical section，所以使用mutex lock。而因為是要放入queue，先檢查是否已經滿了，滿了就讓他wait()，並等signal()。
>
> 若是還沒滿，就將其放入buffer的尾端，並將尾端後移(若是在最後一格，就要移去第一格)，並且增加size。然後signal() dequeue，提醒有東西被放入了，最後將lock unlock 因為結束了。

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

    while(size == 0){
        pthread_cond_wait(&cond_dequeue, &mutex);
    }
    T item = buffer[head];
    head = (head + 1) % buffer_size;
    size--;
    pthread_cond_signal(&cond_enqueue);
    pthread_mutex_unlock(&mutex);

    return item;
}
```

> 一樣會先lock，並且檢查現在queue中是否有東西，若是沒有就需要wait()，等待上述enqueue的通知。若是有東西，就會讀取buffer的head，並將head位置下移，然後更改size，最後提醒enqueue有東西被移走了，再結束lock。

```
int TSQueue<T>::get_size() {
    // TODO: returns the size of the queue
    return size;
}
```

> 單純的將size回傳。

---

producer.hpp

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

> 創建一個producer thread。

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

    while(true){
        Item *item = producer->input_queue->dequeue();

        item->val = producer->transformer->producer_transform(item->opcode, item-
>val);
```

```
        producer->worker_queue->enqueue(item);
    }

    delete producer;

    return nullptr;
}
```

> process會無限的運作，就是不停地從input_queue取出item來運行。然後再利用producer_transform將op code和val進行轉換，並且將item放入worker_queue。

---

consumer.hpp

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

> 創建一個consumer thread。

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

    return is_cancel;
}
```

> 將現在的is_cancel值設為true並回傳。

```
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;
}
```

> 這一部分與producer相似，從worker queue取出一個item，並且用transform轉換值後，再將item放入output queue。

---

consumer_controller.hpp

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

> 開一個thread給ConsumerController。

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

    while(true){
        int now_size = controller->worker_queue->get_size();
        if(now_size > controller->high_threshold){
            std::cout << "Scaling up consumers from " << controller-
>consumers.size() << " to " << controller->consumers.size()+1 << std::endl;

            Consumer *new_consumer = new Consumer(controller->worker_queue,
controller->writer_queue, controller->transformer);
            controller->consumers.push_back(new_consumer);
            new_consumer->start();
        }
        else if(now_size < controller->low_threshold && controller-
>consumers.size() > 1){
            std::cout << "Scaling down consumers from " << controller-
>consumers.size() << " to " << controller->consumers.size()-1 <<std::endl;

            Consumer *delete_consumer = controller->consumers.back();
            delete_consumer->cancel();
            controller->consumers.pop_back();
        }
        usleep(controller->check_period); //wait
    }

    return nullptr;
}
```

> 會持續的檢查now_size是否大於high threshold，若是大於high threshold，就會創建一個consumer，並且放到consumers vector的尾端，然後讓它start()。
>
> 而若是小於low threshold，並且現在有>1的consumer，就會刪除尾端的consumer，讓它不要再運作，並且pop掉。
>
> 最後會運作usleep，讓其可以週期性的(check_period的週期)，檢查now_size。

---

writer.hpp

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

> 開一個thread給writer。

```
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;
}
```

> writer會持續運作值到expect_line(預計處理數量)為零，並且會持續從output_queue取出item，放入輸出檔案中(ofs == ofstream)。

Experiment

1. Different values of CONSUMER_CONTROLLER_CHECK_PERIOD

原本: 1000000 結果如下:

```
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
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
```

嘗試: 100000 結果如下:

```
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 up consumers from 10 to 11
```

```
Scaling up consumers from 11 to 12
Scaling up consumers from 12 to 13
Scaling up consumers from 13 to 14
Scaling up consumers from 14 to 15
Scaling down consumers from 15 to 14
Scaling down consumers from 14 to 13
Scaling down consumers from 13 to 12
Scaling down consumers from 12 to 11
Scaling down consumers from 11 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
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 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 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 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 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
```

嘗試: 10000000 結果如下:

```
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 down consumers from 4 to 3
Scaling up consumers from 3 to 4
Scaling down consumers from 4 to 3
Scaling up consumers from 3 to 4
```

- 可能原因應該是較短的CONSUMER_CONTROLLER_CHECK_PERIOD會讓他頻繁的去檢查是否需要增加worker queue，因此變動會較多，可以再剛到達high threshold/low threshold 就馬上發現；而較小的check period會因為要隔很長時間才會去check一次，所以變動較少。

---

2. Different values of CONSUMER_CONTROLLER_LOW_THRESHOLD_PERCENTAGE and CONSUMER_CONTROLLER_HIGH_THRESHOLD_PERCENTAGE.

原本: 20/80 結果就是第一個，不重複了

嘗試: 45/80 結果如下:

```
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
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
```

> 其實只多了最後一行，但我們仍可以看出他讓調降的threshold變小後，讓其更容易下降，也因此才會多出了最後一行(讓調降更早發生)。

嘗試: 20/50 結果如下:

```
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
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
```

> 可以發現其中出現了"11"，這是原本的沒有出現的。發生原因可能是因為調降了high threshold，導致更容易需要新的consumer，也因此讓最多consumer數量增到11。(跑起來也感覺比較快)

- 由上述可知，調升調降 low/high threshold 會影響製造/減少consumer的條件。而consumer的多寡也會影響運作的效率以及速度。

3. Different values of WORKER_QUEUE_SIZE.

原本: 200 結果如上

嘗試: 20 結果如下:

```
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 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 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 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 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 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 down consumers from 9 to 8
```

嘗試: 500 結果:

```
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 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 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 down consumers from 7 to 6
Scaling down consumers from 6 to 5
```

- 可以發現worker queue size若越小，更容易超過/小於threshold，也更常會需要製造或減少consumer。而若是worker queue size較大，consumer數量的變化就較小。

---

4. What happens if WRITER_QUEUE_SIZE is very small?

原本: 4000 結果如上

嘗試: 1 結果:

```
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
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
```

- 雖然得出了相同的結果，但是很明顯有感受到其速度變慢了。這可能是因為當writer queue過小，consumer就有較多機會會等在其外面(等待enqueue writer queue)。因此會讓worker queue 消耗(dequeue)變慢，也導致了整個運作變慢。

---

5. What happens if READER_QUEUE_SIZE is very small?

原本: 200 結果如上

嘗試: 2 結果:

```
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
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
```

- 結果也相同，但同樣的速度也較慢。可能是因為當input queue變小後，讓producer可以消耗的變少了，可能也讓一些producer閒置了，讓整體速度變慢。

Difficulties & Feedback

- 這次操作起來其實沒有太大的問題(跟MP4比起來)，但當時跑test case有點搞混，0 1分別要跑200 4000這點。我一開始拿了testcase0 跑200，想說為甚麼一直錯哈哈哈。
- 然後還有一個部分是在writer寫時，不知道為甚麼我一開始寫`writer->ofs << writer->output_queue->dequeue();`不行，直到我改成`Item *item = writer->output_queue->dequeue(); writer->ofs << *item;`才可以順利運作。

> 感謝助教、教授這一個學期的辛勞!!雖然很累但也學到很多東西!!!(但好累)