Pthread_Programming

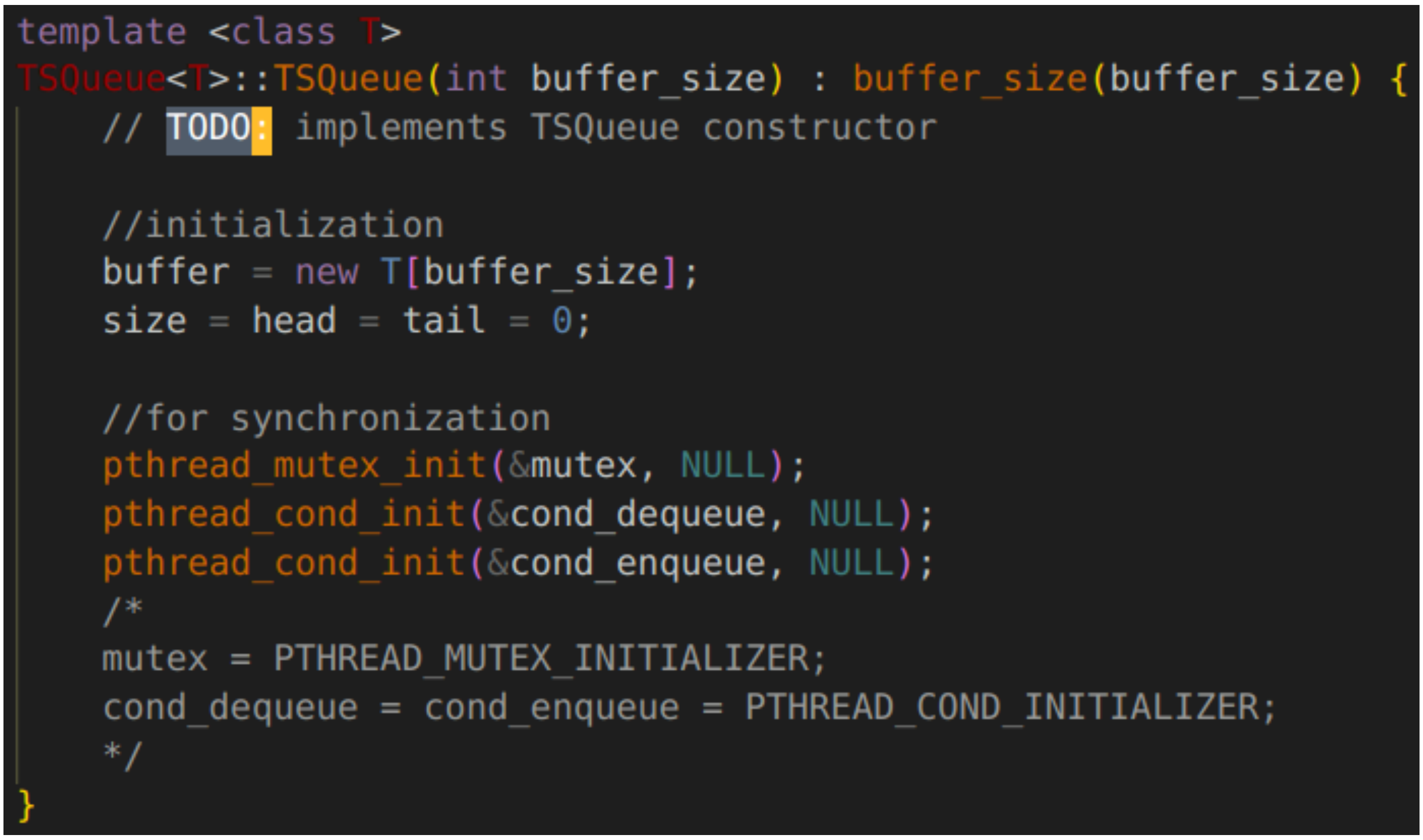
Cover Page

徐竣霆： 50%
江承紘： 50%

Implementation

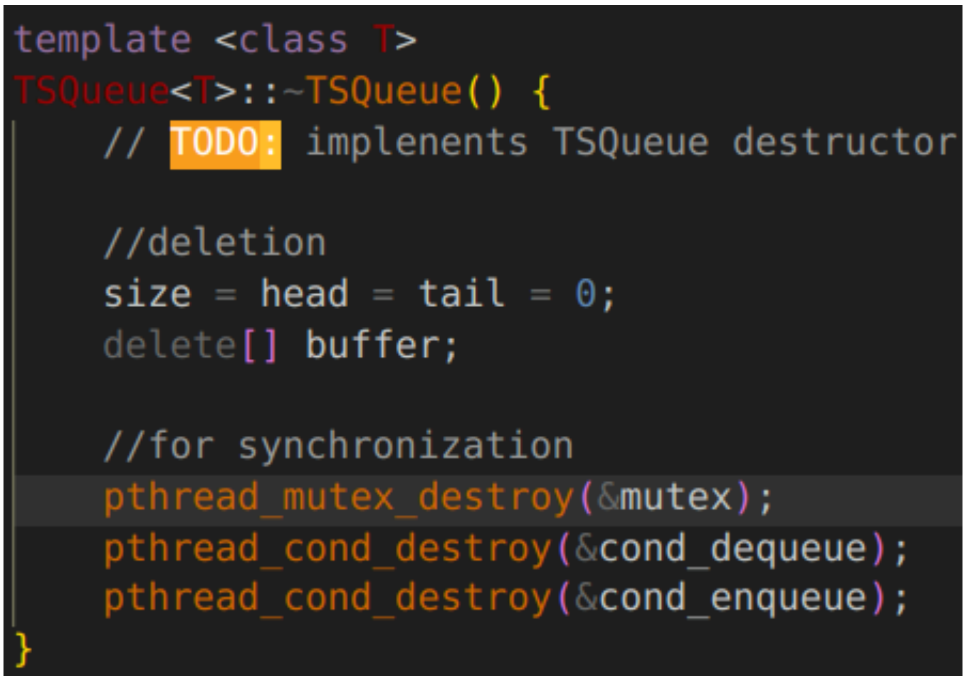
ts_queue.hpp

Constructor

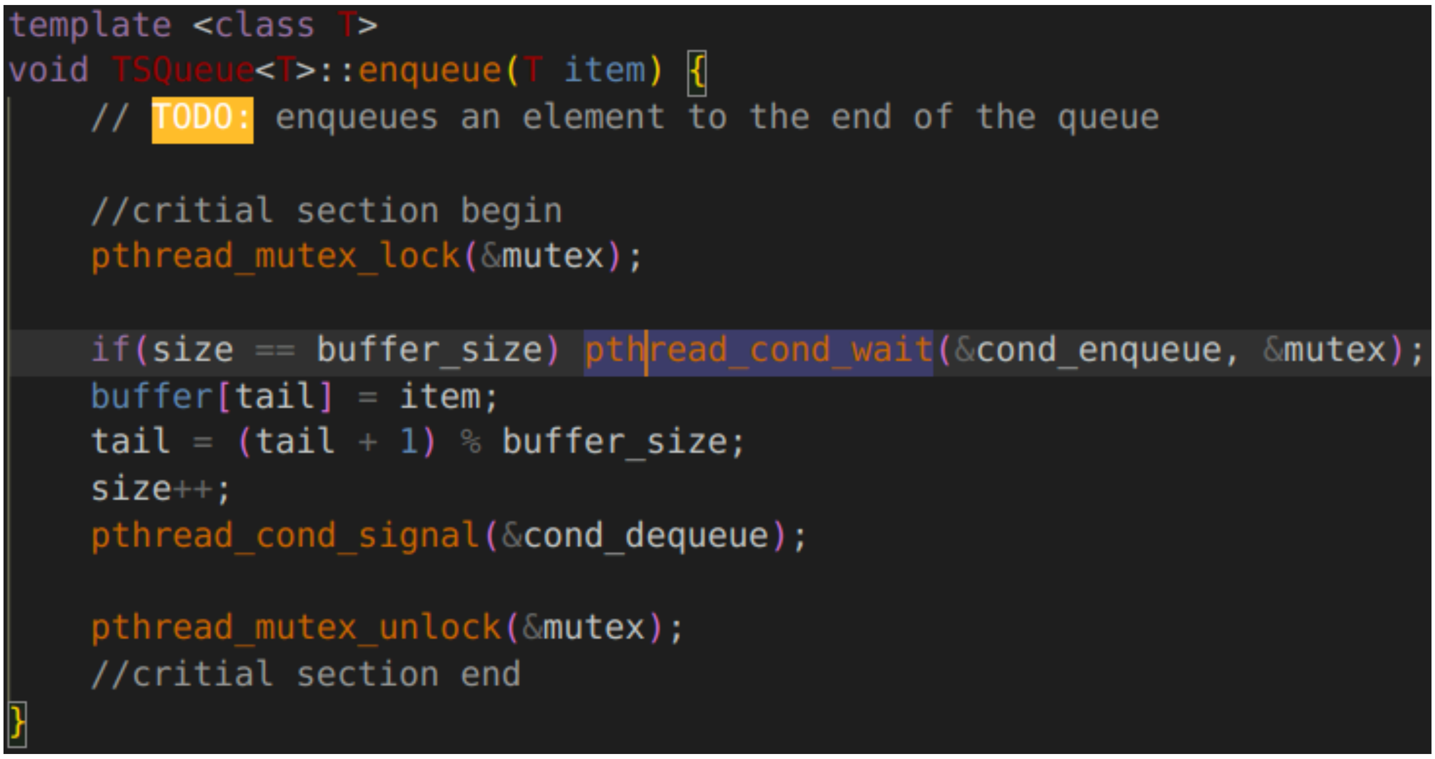


```
template <class T>
TSQueue<T>::TSQueue(int buffer_size) : buffer_size(buffer_size) {
    // TODO: implements TSQueue constructor

    //initialization
    buffer = new T[buffer_size];
    size = head = tail = 0;

    //for synchronization
    pthread_mutex_init(&mutex, NULL);
    pthread_cond_init(&cond_dequeue, NULL);
    pthread_cond_init(&cond_enqueue, NULL);
    /*
    mutex = PTHREAD_MUTEX_INITIALIZER;
    cond_dequeue = cond_enqueue = PTHREAD_COND_INITIALIZER;
    */
}
```

Destructor

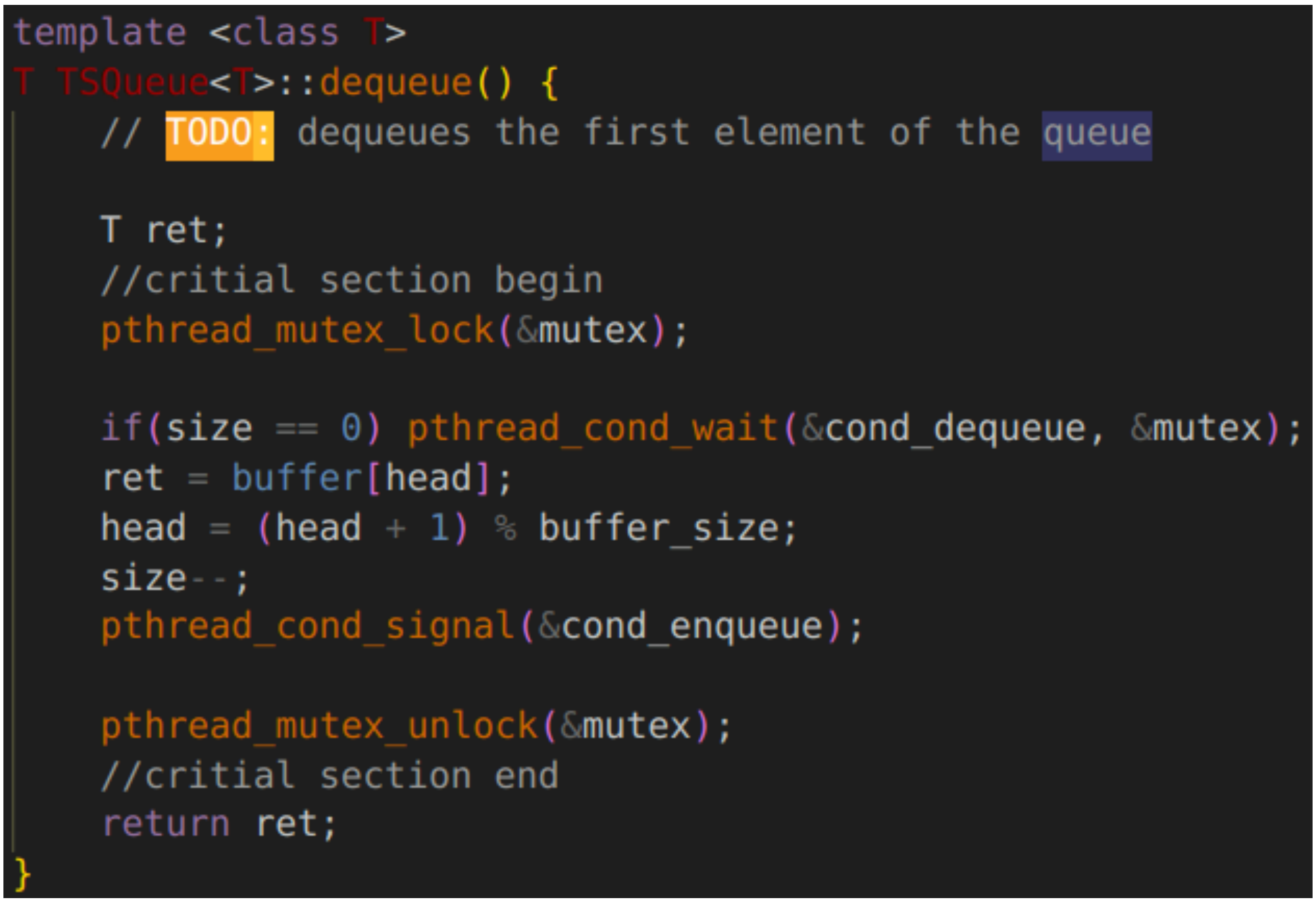

```
template <class T>
TSQueue<T>::~TSQueue() {
    // TODO: implements TSQueue destructor

    //deletion
    size = head = tail = 0;
    delete[] buffer;

    //for synchronization
    pthread_mutex_destroy(&mutex);
    pthread_cond_destroy(&cond_dequeue);
    pthread_cond_destroy(&cond_enqueue);
}
```

Enqueue


```
template <class T>
void TSQueue<T>::enqueue(T item) {
    // TODO: enqueues an element to the end of the queue

    //critial section begin
    pthread_mutex_lock(&mutex);

    if(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);
    //critial section end
}
```

Since it will modify shared data(size, buffer, tail)(critical section), use mutex lock & unlock to enclose them.
If buffer is full, call condition wait, else push item in buffer, modify the size and tail.
Signal(dequeue).

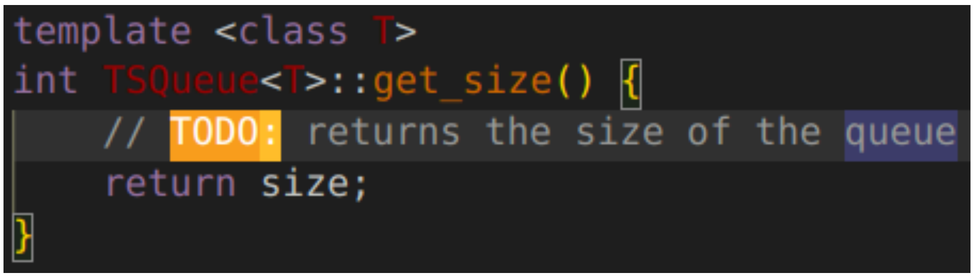
Dequeue



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

    T ret;
    //critial section begin
    pthread_mutex_lock(&mutex);

    if(size == 0) pthread_cond_wait(&cond_dequeue, &mutex);
    ret = buffer[head];
    head = (head + 1) % buffer_size;
    size--;
    pthread_cond_signal(&cond_enqueue);

    pthread_mutex_unlock(&mutex);
    //critial section end
    return ret;
}
```

Since it will modify shared data(size, buffer, head)(critical section), use mutex lock & unlock to enclose them.
If buffer is empty, call condition wait, else pop item at head, modify the size and head. Signal(enqueue).

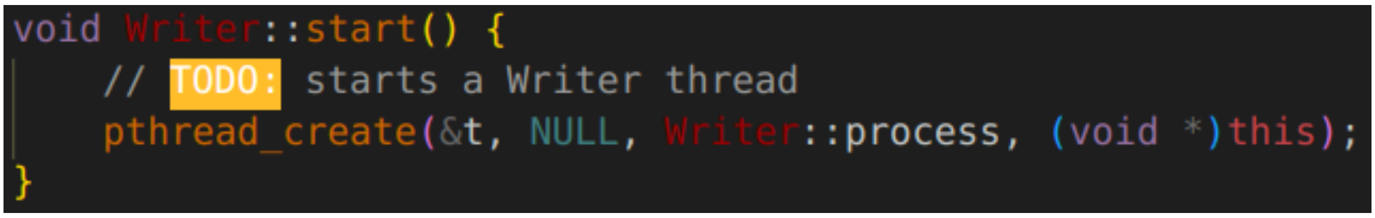
get_size()



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

writer.hpp

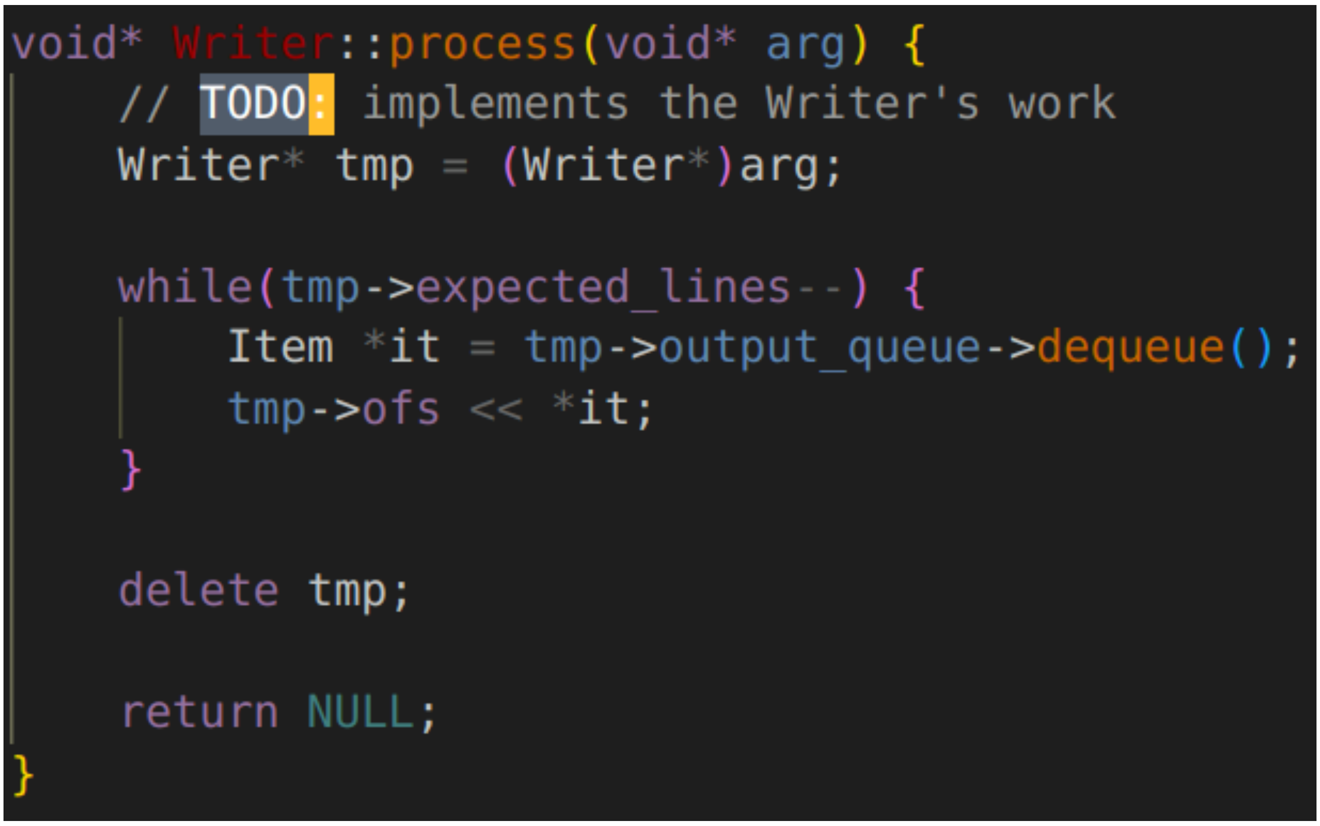
Writer::start()



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

Create a new thread running the Writer::process.
The arg passed in is the whole object itself.

Writer::process(void* arg)

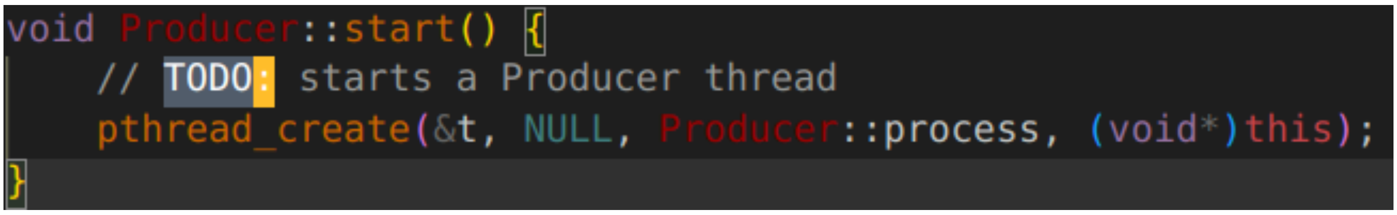


```
void* Writer::process(void* arg) {
    // TODO: implements the Writer's work
    Writer* tmp = (Writer*)arg;

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

    delete tmp;

    return NULL;
}
```

Keep dequeuing items, write them into file, until all lines have been written.

producer.hpp

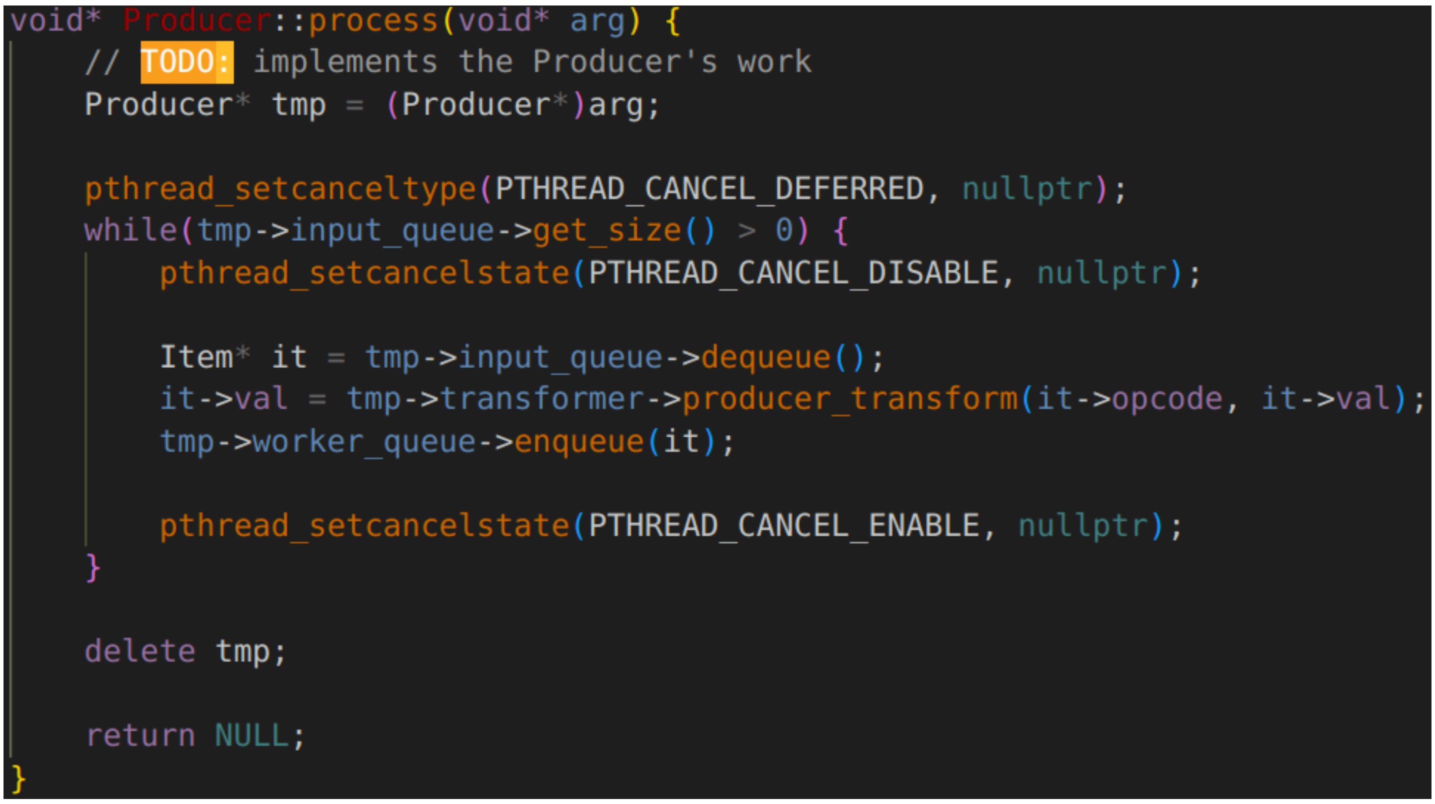
Producer::start()



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

Create a new thread running the Producer::process.
The arg passed in is the whole object itself.

Producer::process(void* arg)



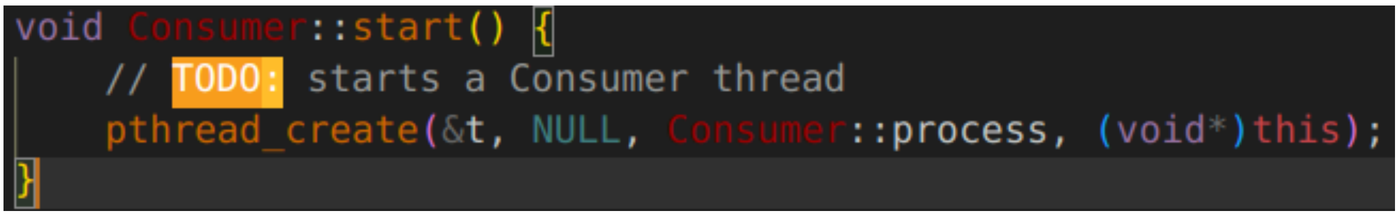
```
void* Producer::process(void* arg) {
    // TODO: implements the Producer's work
    Producer* tmp = (Producer*)arg;

    pthread_setcanceltype(PTHREAD_CANCEL_DEFERRED, nullptr);
    while(tmp->input_queue->get_size() > 0) {
        pthread_setcancelstate(PTHREAD_CANCEL_DISABLE, nullptr);

        Item* it = tmp->input_queue->dequeue();
        it->val = tmp->transformer->producer_transform(it->opcode, it->val);
        tmp->worker_queue->enqueue(it);

        pthread_setcancelstate(PTHREAD_CANCEL_ENABLE, nullptr);
    }

    delete tmp;

    return NULL;
}
```

If the input queue is not empty, keep dequeuing from input queue, transform, and enqueuing to worker queue.

The "cancelstates" are to make sure will not be interrupted during process.

consumer.hpp

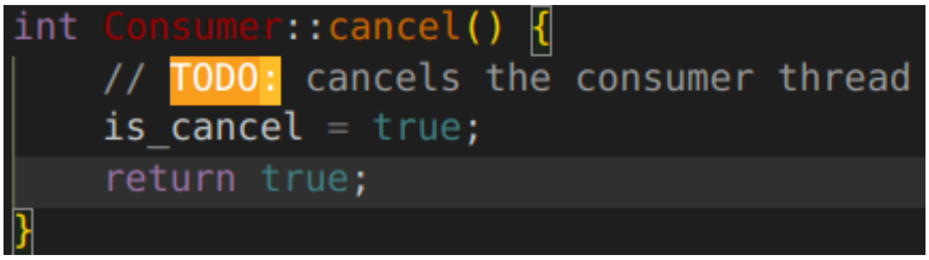
Consumer::start()



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

Create a new thread running the Consumer::process.
The arg passed in is the whole object itself.

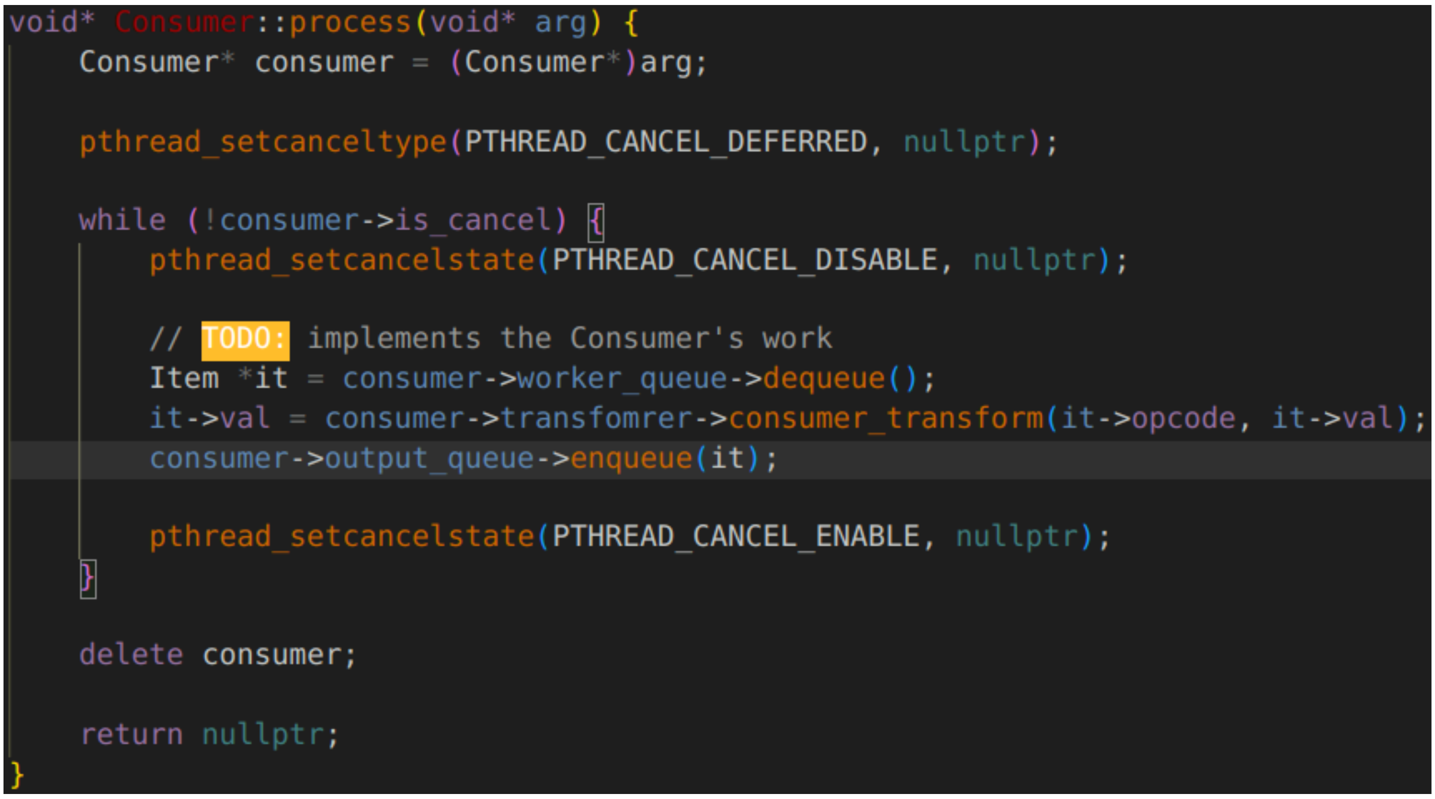
Consumer::cancel()



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

Set is_cancel = true, return true

Consumer::process(void* arg)



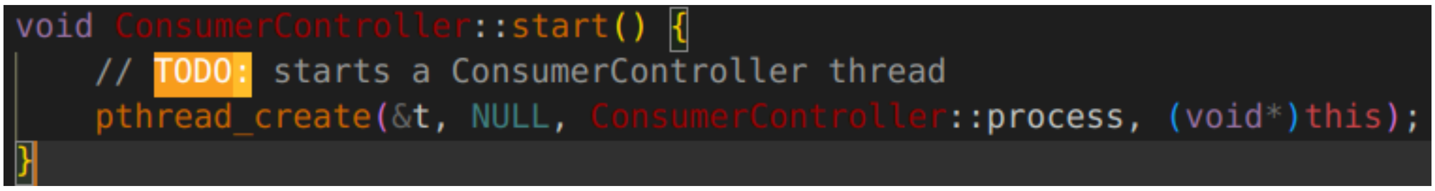
```cpp
void* Consumer::process(void* arg) {
    Consumer* consumer = (Consumer*)arg;

    pthread_setcanceltype(PTHREAD_CANCEL_DEFERRED, nullptr);

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

        // TODO: implements the Consumer's work
        Item *it = consumer->worker_queue->dequeue();
        it->val = consumer->transfomrer->consumer_transform(it->opcode, it->val);
        consumer->output_queue->enqueue(it);

        pthread_setcancelstate(PTHREAD_CANCEL_ENABLE, nullptr);
    }

    delete consumer;

    return nullptr;
}
```

If consumer is not canceled, keep dequeuing from worker queue, transform, and enqueuing to output queue.

The “cancelstates” are to make sure will not be interrupted during process.

consumer_controller.hpp

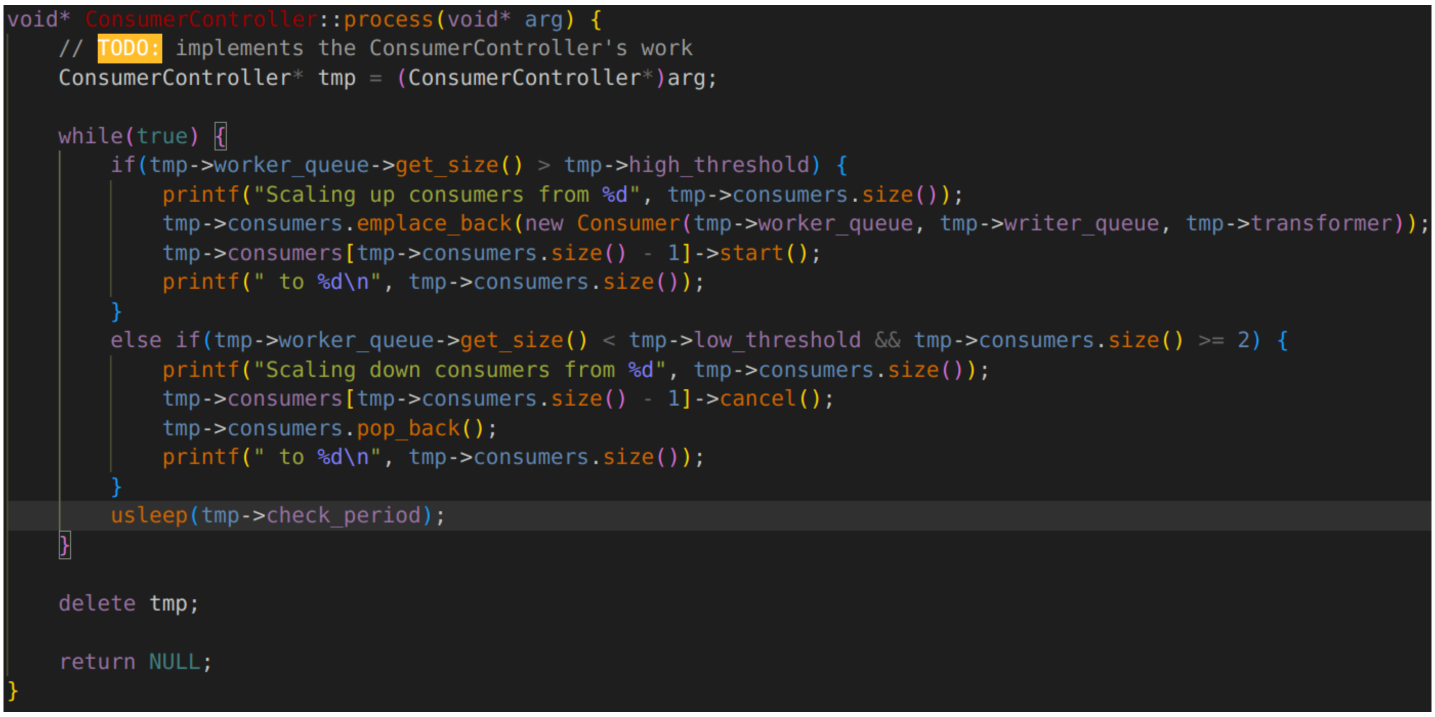
ConsumerController::start()



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

Create a new thread running the ConsumerController::process.
The arg passed in is the whole object itself.

ConsumerController::process(void* arg)


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

    while(true) {
        if(tmp->worker_queue->get_size() > tmp->high_threshold) {
            printf("Scaling up consumers from %d", tmp->consumers.size());
            tmp->consumers.emplace_back(new Consumer(tmp->worker_queue, tmp->writer_queue, tmp->transformer));
            tmp->consumers[tmp->consumers.size() - 1]->start();
            printf(" to %d\n", tmp->consumers.size());
        }
        else if(tmp->worker_queue->get_size() < tmp->low_threshold && tmp->consumers.size() >= 2) {
            printf("Scaling down consumers from %d", tmp->consumers.size());
            tmp->consumers[tmp->consumers.size() - 1]->cancel();
            tmp->consumers.pop_back();
            printf(" to %d\n", tmp->consumers.size());
        }
        usleep(tmp->check_period);
    }

    delete tmp;

    return NULL;
}
```

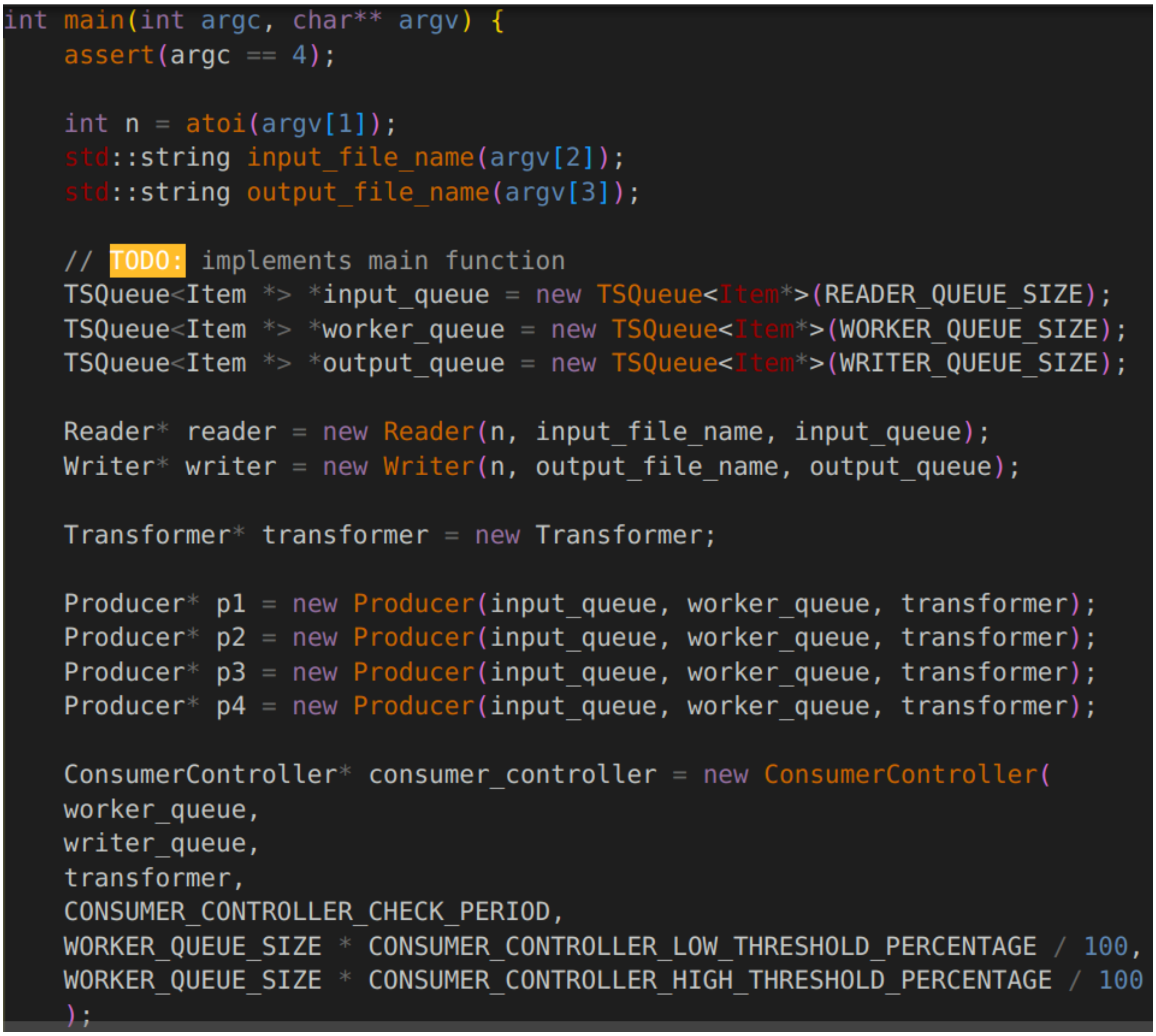
Since sleep() works in seconds, we use usleep() instead.

A while(true) loop and usleep() for periodic checking.

If size > high_threshold, add a new Consumer and boost it.

If size < low_threshold and size > 1, cancel the newest Consumer.

main.cpp

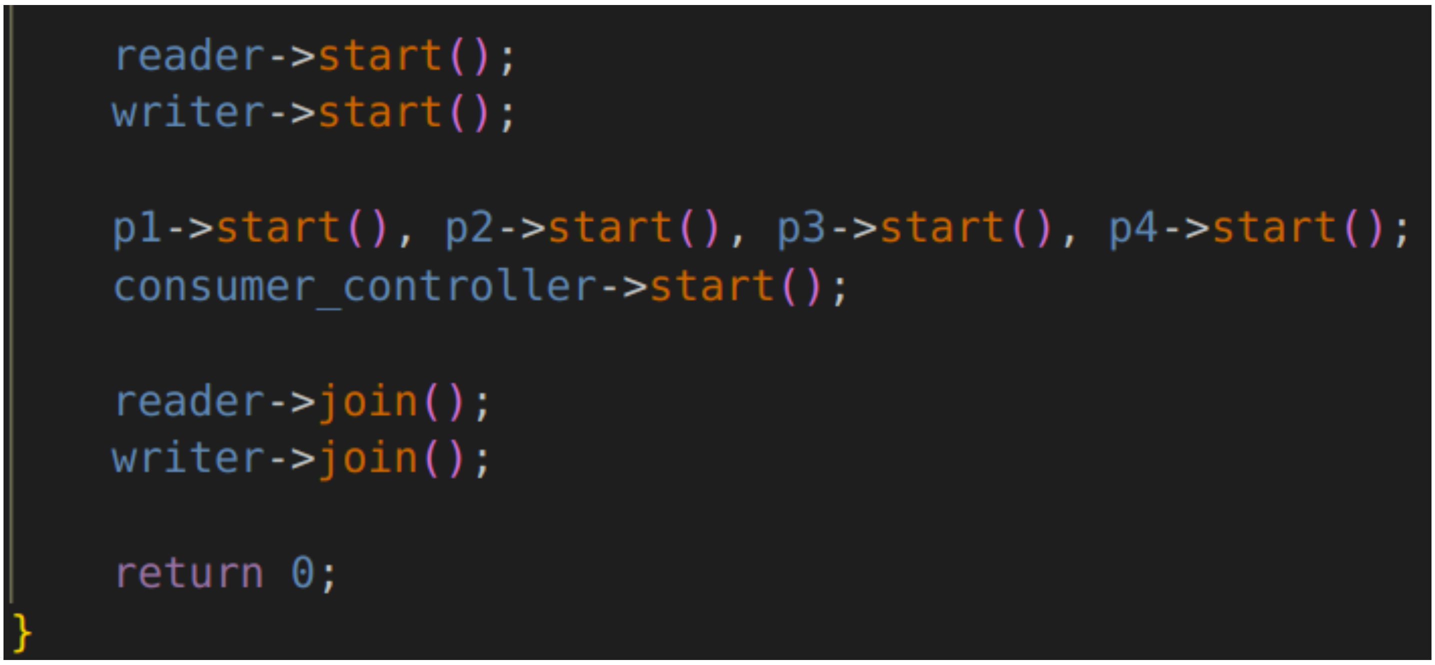


```
int main(int argc, char** argv) {
    assert(argc == 4);

    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);

    Reader* reader = new Reader(n, input_file_name, input_queue);
    Writer* writer = new Writer(n, output_file_name, output_queue);

    Transformer* transformer = new Transformer;

    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);

    ConsumerController* consumer_controller = new ConsumerController(
    worker_queue,
    writer_queue,
    transformer,
    CONSUMER_CONTROLLER_CHECK_PERIOD,
    WORKER_QUEUE_SIZE * CONSUMER_CONTROLLER_LOW_THRESHOLD_PERCENTAGE / 100,
    WORKER_QUEUE_SIZE * CONSUMER_CONTROLLER_HIGH_THRESHOLD_PERCENTAGE / 100
    );
```

Initialization:

We need an Input Queue, a Worker Queue, an Output Queue.

We also need 1 Reader thread, 1 Writer thread, and 4 Producer threads. We also initialize a ConsumerController thread for further Consumer thread creation/deletion.



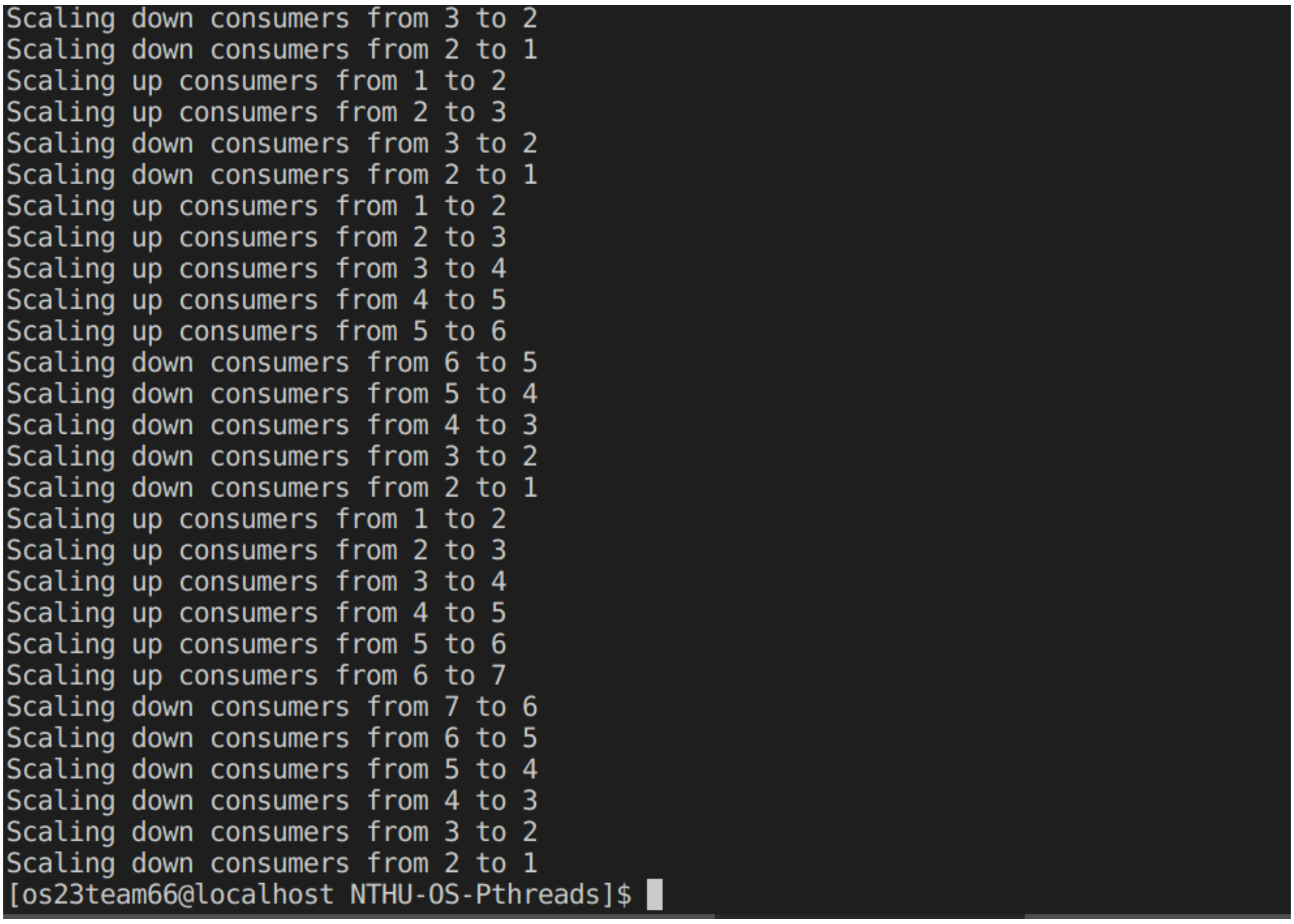
```
    reader->start();
    writer->start();

    p1->start(), p2->start(), p3->start(), p4->start();
    consumer_controller->start();

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

    return 0;
}
```

For each thread, call start() to start working.
Also, we should call join() for reader and writer, since the main function have to wait these two to complete then terminate.

Experiments

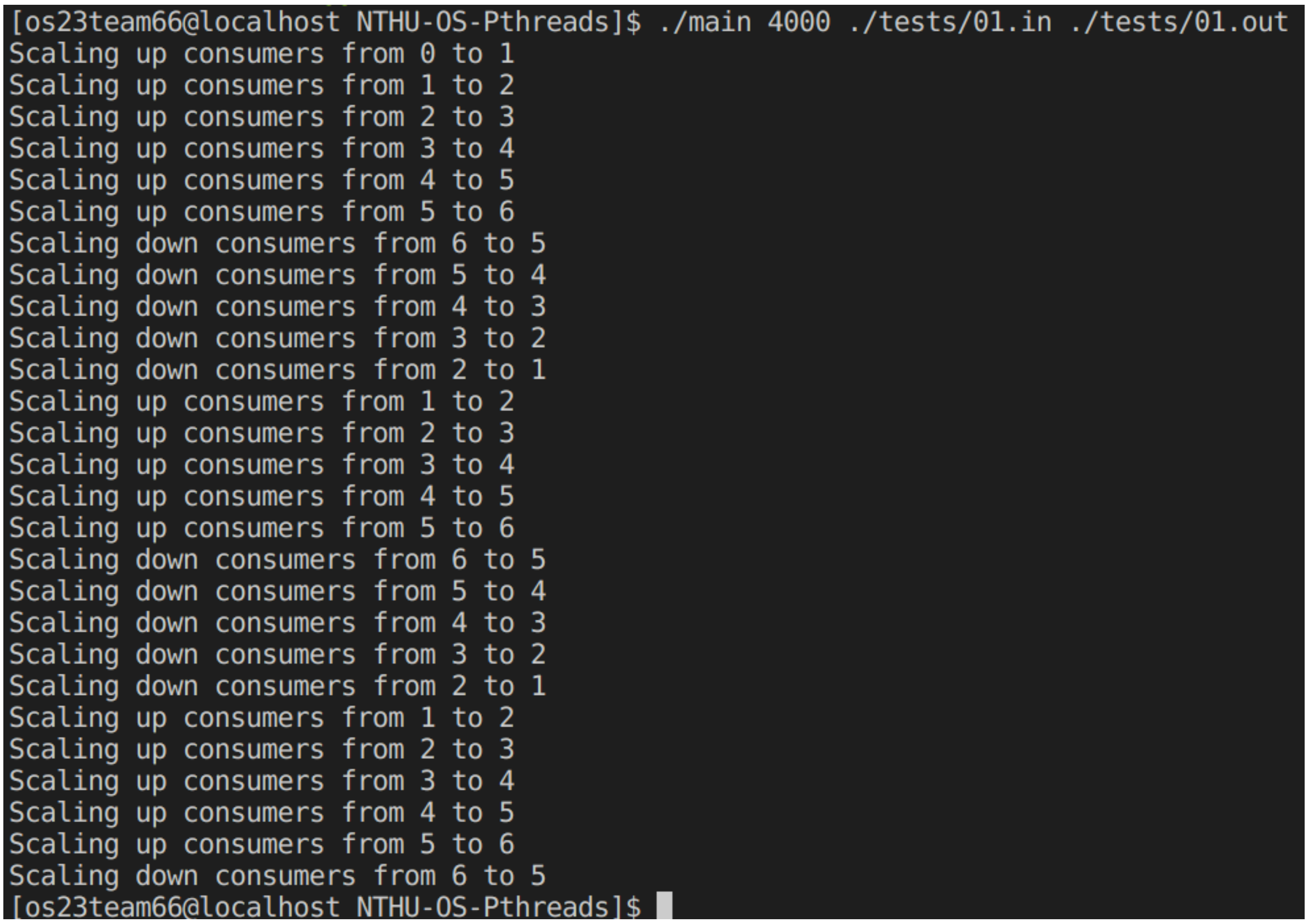
Different values of CONSUMER_CONTROLLER_CHECK_PERIOD.

- Period == 500000(test01)



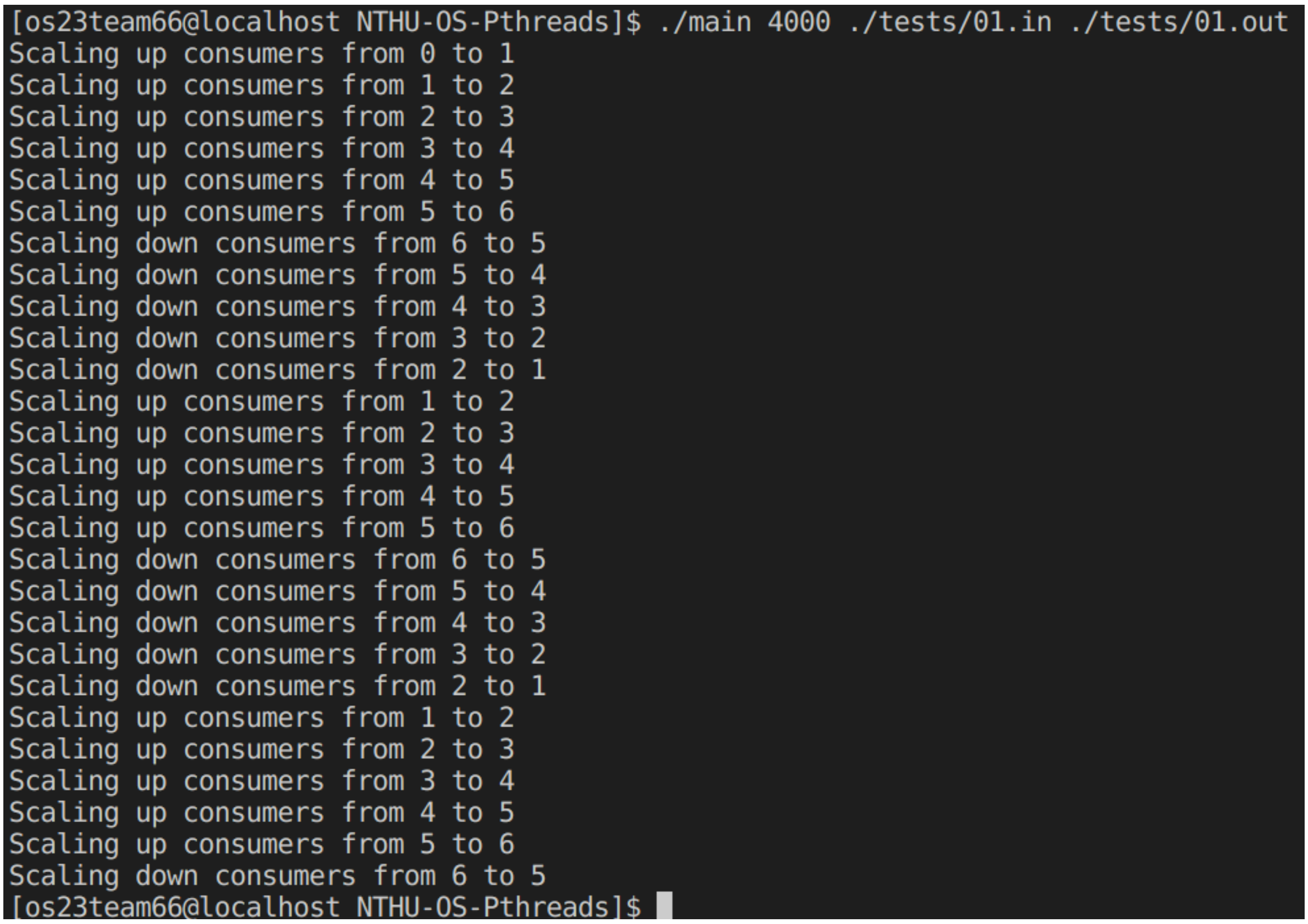
```
Scaling down consumers from 3 to 2
Scaling down consumers from 2 to 1
Scaling up consumers from 1 to 2
Scaling up consumers from 2 to 3
Scaling down consumers from 3 to 2
Scaling down consumers from 2 to 1
Scaling up consumers from 1 to 2
Scaling up consumers from 2 to 3
Scaling up consumers from 3 to 4
Scaling up consumers from 4 to 5
Scaling up consumers from 5 to 6
Scaling 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
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
[os23team66@localhost NTHU-OS-Pthreads]$
```

- Period == 1000000(test01)



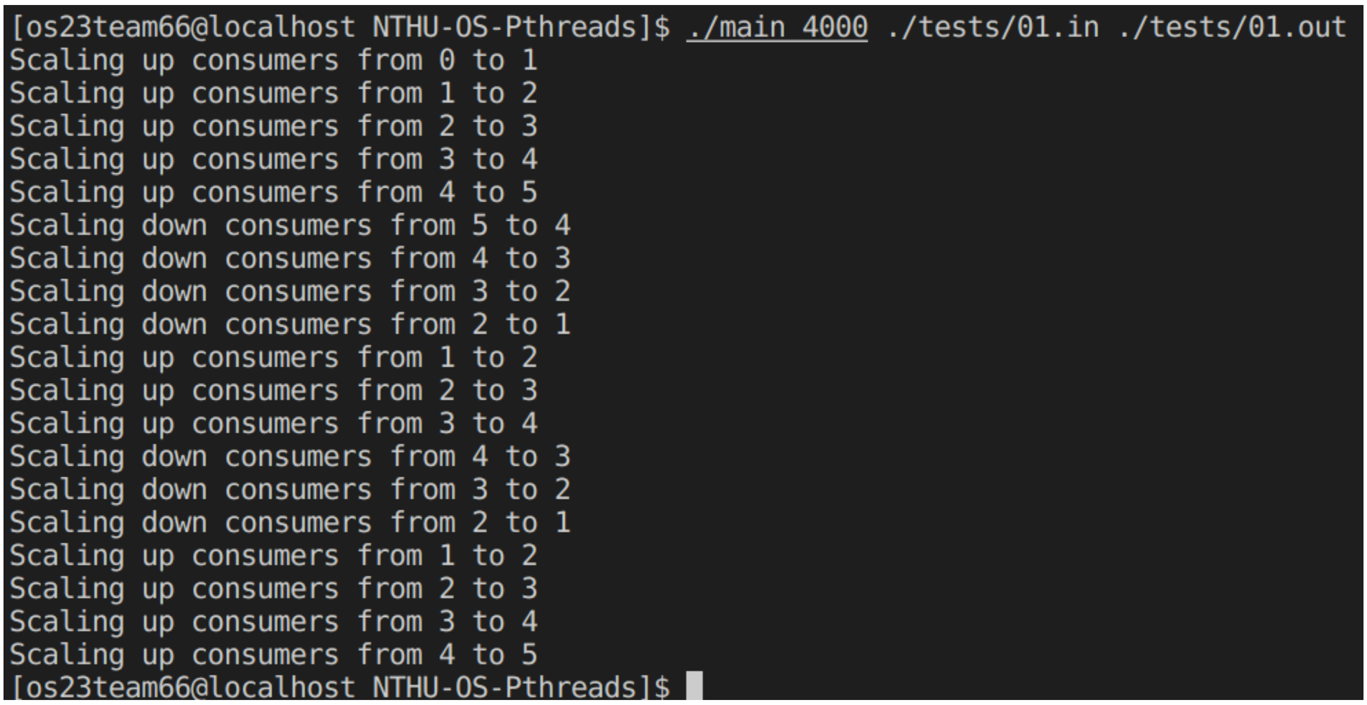
```
[os23team66@localhost NTHU-OS-Pthreads]$ ./main 4000 ./tests/01.in ./tests/01.out
Scaling up consumers from 0 to 1
Scaling up consumers from 1 to 2
Scaling up consumers from 2 to 3
Scaling up consumers from 3 to 4
Scaling up consumers from 4 to 5
Scaling up consumers from 5 to 6
Scaling down consumers 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 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 down consumers from 6 to 5
[os23team66@localhost NTHU-OS-Pthreads]$
```

- Period == 2000000(test01)



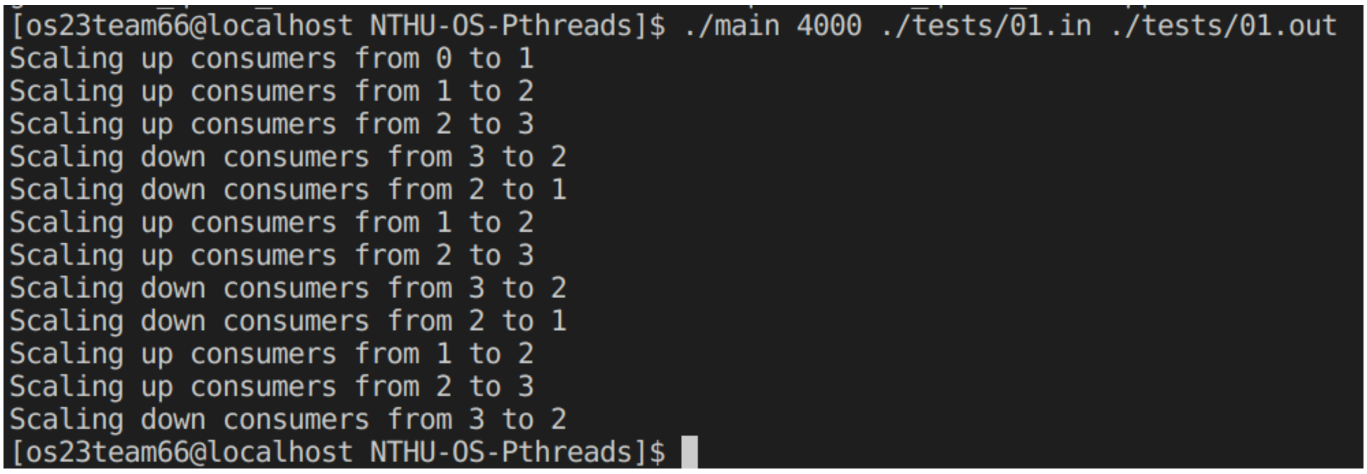
```
[os23team66@localhost NTHU-OS-Pthreads]$ ./main 4000 ./tests/01.in ./tests/01.out
Scaling up consumers from 0 to 1
Scaling up consumers from 1 to 2
Scaling up consumers from 2 to 3
Scaling up consumers from 3 to 4
Scaling up consumers from 4 to 5
Scaling up consumers from 5 to 6
Scaling down consumers 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 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 down consumers from 6 to 5
[os23team66@localhost NTHU-OS-Pthreads]$
```

- Period == 4000000(test01)



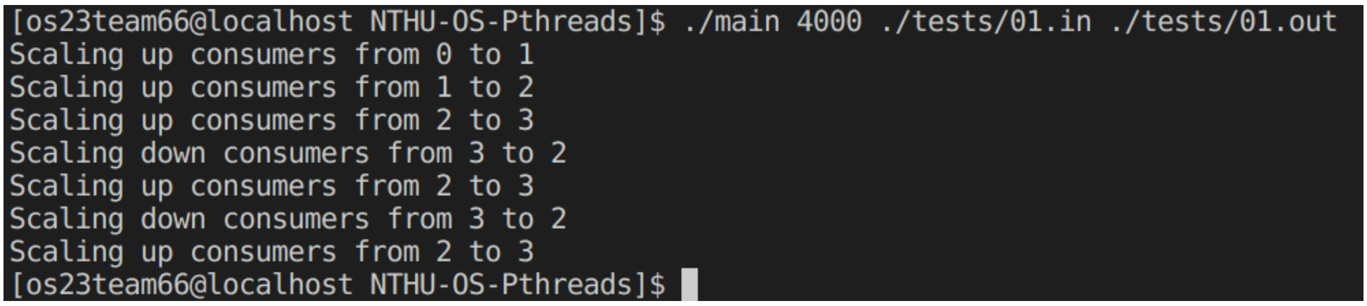
```
[os23team66@localhost NTHU-OS-Pthreads]$ ./main 4000 ./tests/01.in ./tests/01.out
Scaling up consumers from 0 to 1
Scaling up consumers from 1 to 2
Scaling up consumers from 2 to 3
Scaling up consumers from 3 to 4
Scaling up consumers from 4 to 5
Scaling down consumers from 5 to 4
Scaling down consumers 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 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
[os23team66@localhost NTHU-OS-Pthreads]$
```

- Period == 8000000(test01)



```
[os23team66@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 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
[os23team66@localhost NTHU-OS-Pthreads]$
```

- Period == 16000000(test01)

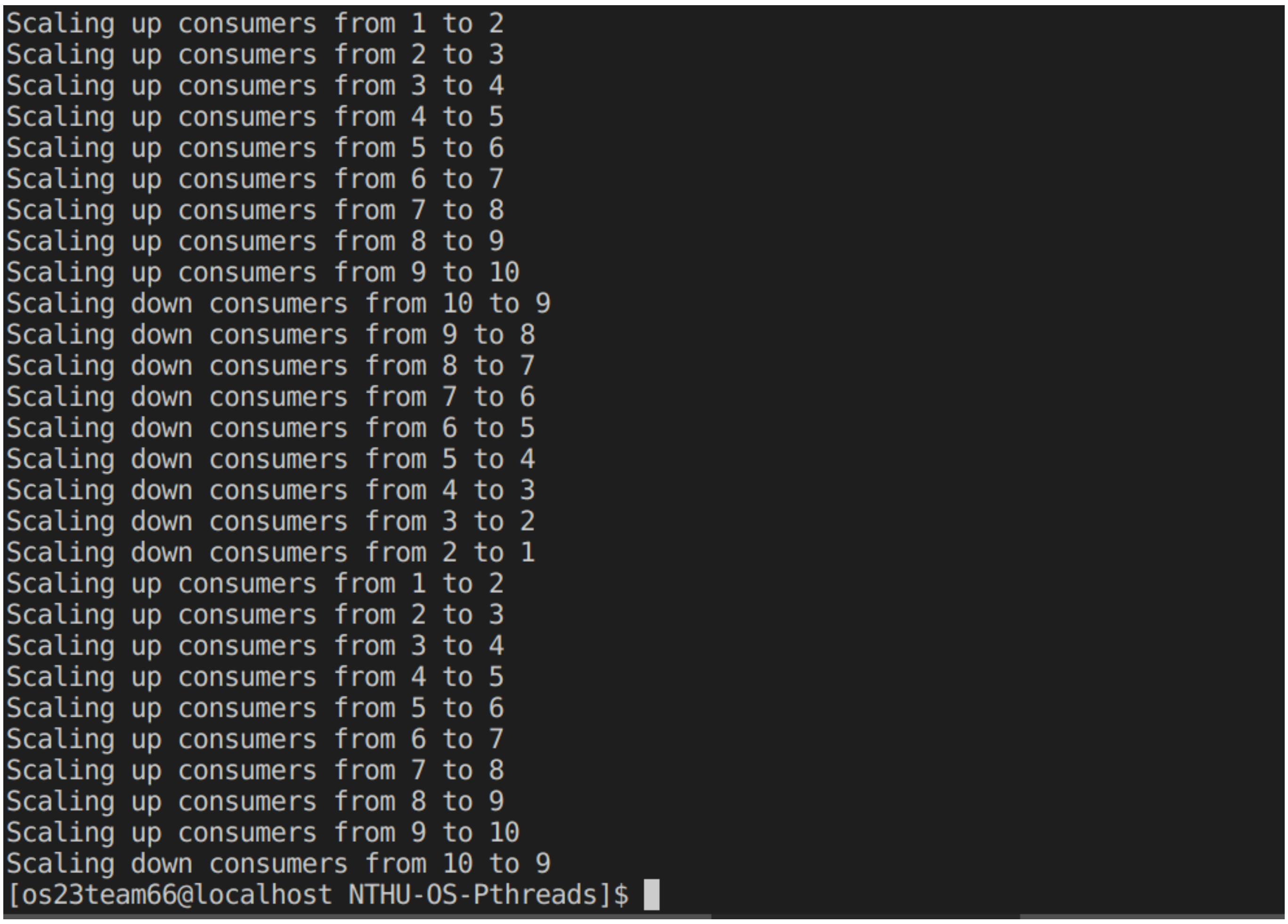


```
[os23team66@localhost NTHU-OS-Pthreads]$ ./main 4000 ./tests/01.in ./tests/01.out
Scaling up consumers from 0 to 1
Scaling up consumers from 1 to 2
Scaling up consumers from 2 to 3
Scaling down consumers from 3 to 2
Scaling up consumers from 2 to 3
Scaling down consumers from 3 to 2
Scaling up consumers from 2 to 3
[os23team66@localhost NTHU-OS-Pthreads]$
```

We can see as period get bigger, the times do scaling get less(the execution time get longer also), which means as period get smaller, the consumer_controller become more flexible and responsive.

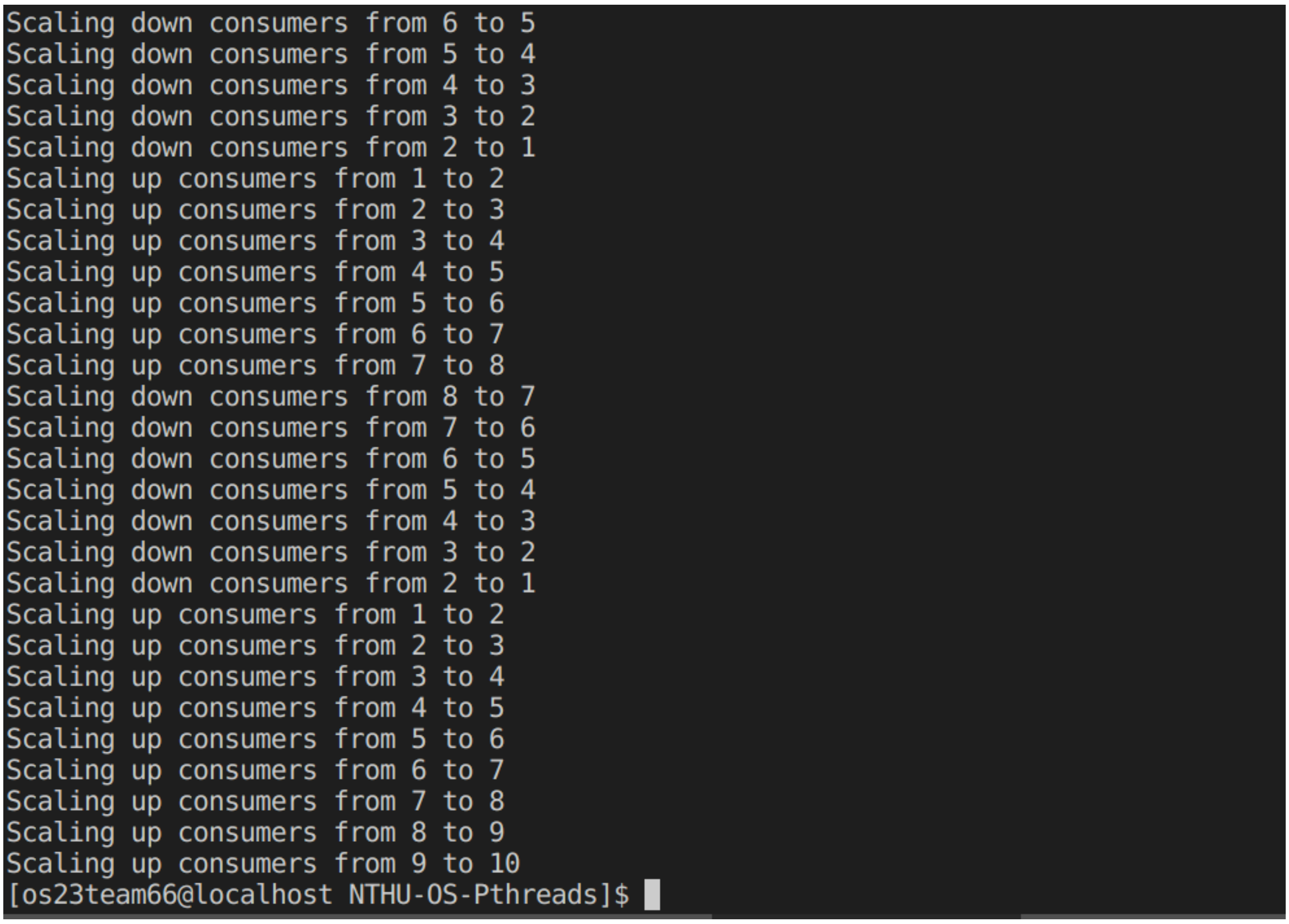
Different values of CONSUMER_CONTROLLER_LOW_THRESHOLD_PERCENTAGE and CONSUMER_CONTROLLER_HIGH_THRESHOLD_PERCENTAGE.

- low / high = 20/80(test01)



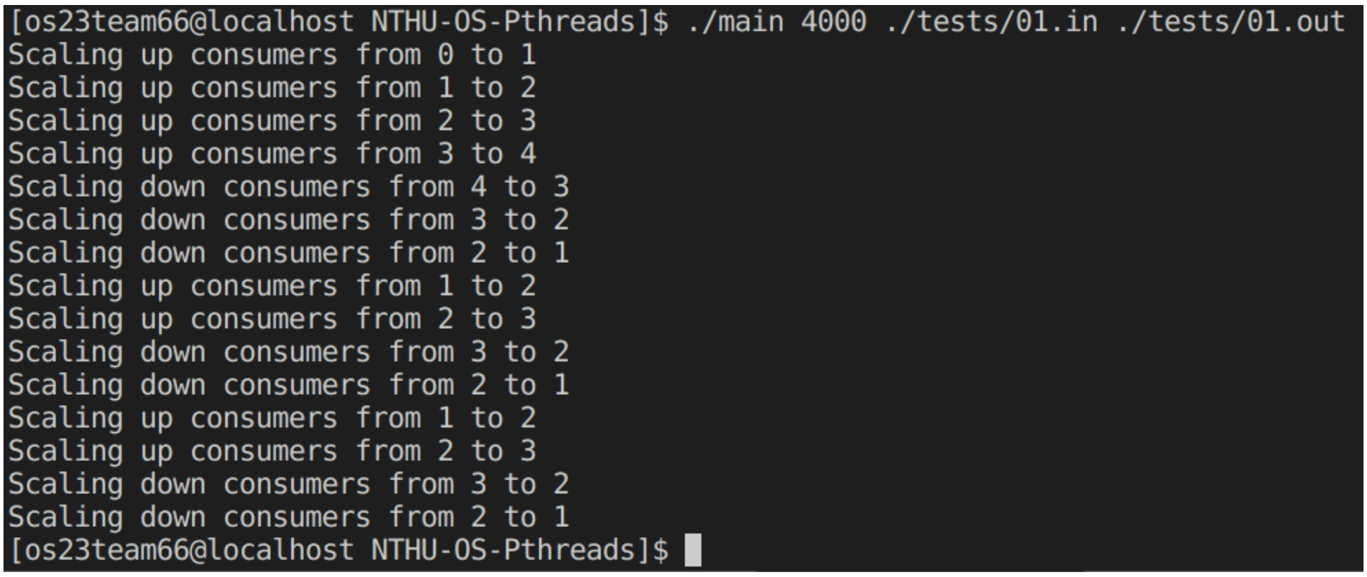
```
Scaling up consumers from 1 to 2
Scaling up consumers from 2 to 3
Scaling up consumers from 3 to 4
Scaling up consumers from 4 to 5
Scaling up consumers from 5 to 6
Scaling up consumers from 6 to 7
Scaling up consumers from 7 to 8
Scaling up consumers from 8 to 9
Scaling up consumers from 9 to 10
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
[os23team66@localhost NTHU-OS-Pthreads]$ 
```

- low / high = 30/70(test01)

- 

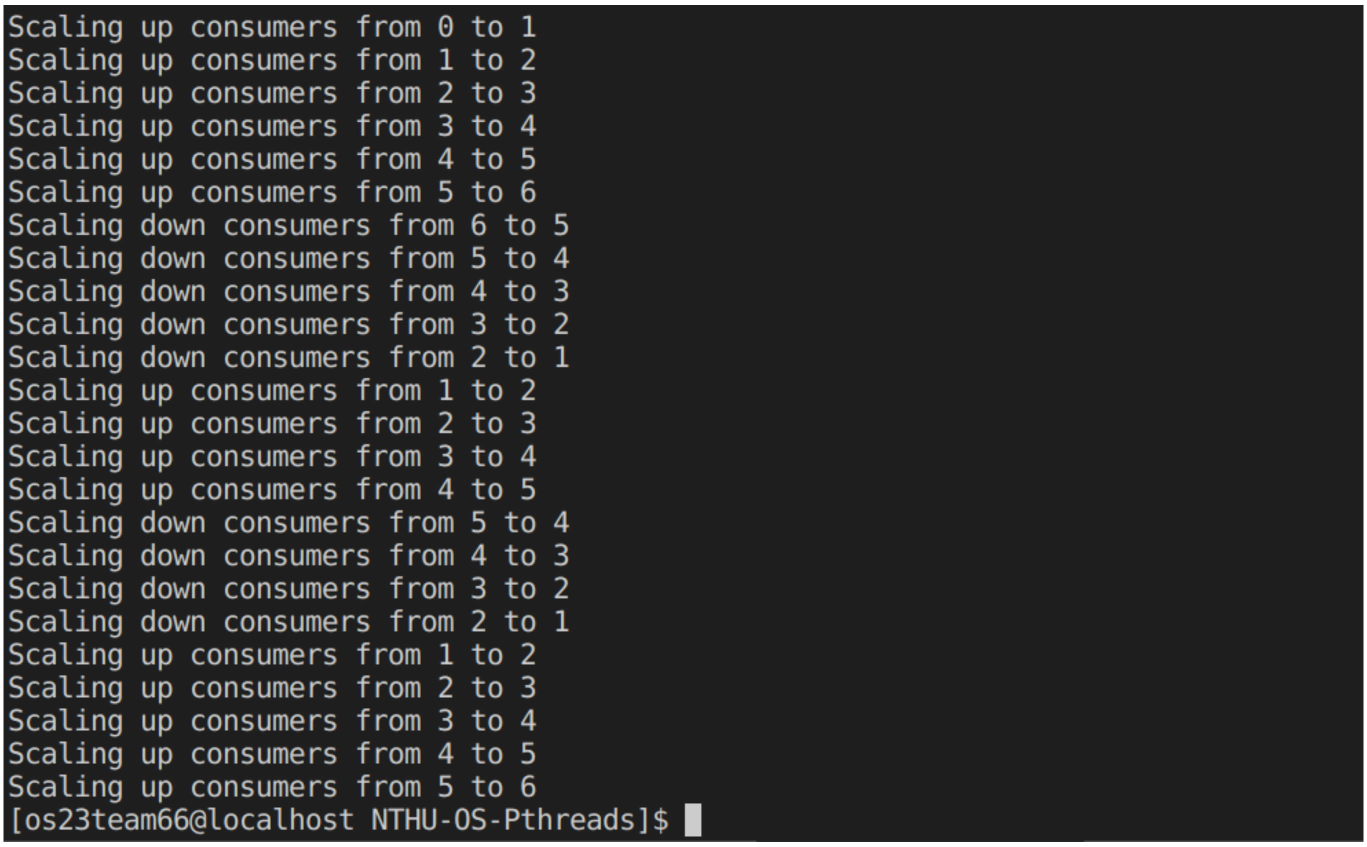
```
Scaling down consumers from 6 to 5
Scaling down consumers from 5 to 4
Scaling down consumers from 4 to 3
Scaling down consumers from 3 to 2
Scaling down consumers from 2 to 1
Scaling up consumers from 1 to 2
Scaling up consumers from 2 to 3
Scaling up consumers from 3 to 4
Scaling up consumers from 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 up consumers from 7 to 8
Scaling up consumers from 8 to 9
Scaling up consumers from 9 to 10
[os23team66@localhost NTHU-OS-Pthreads]$
```

- low / high = 10/90(test01)



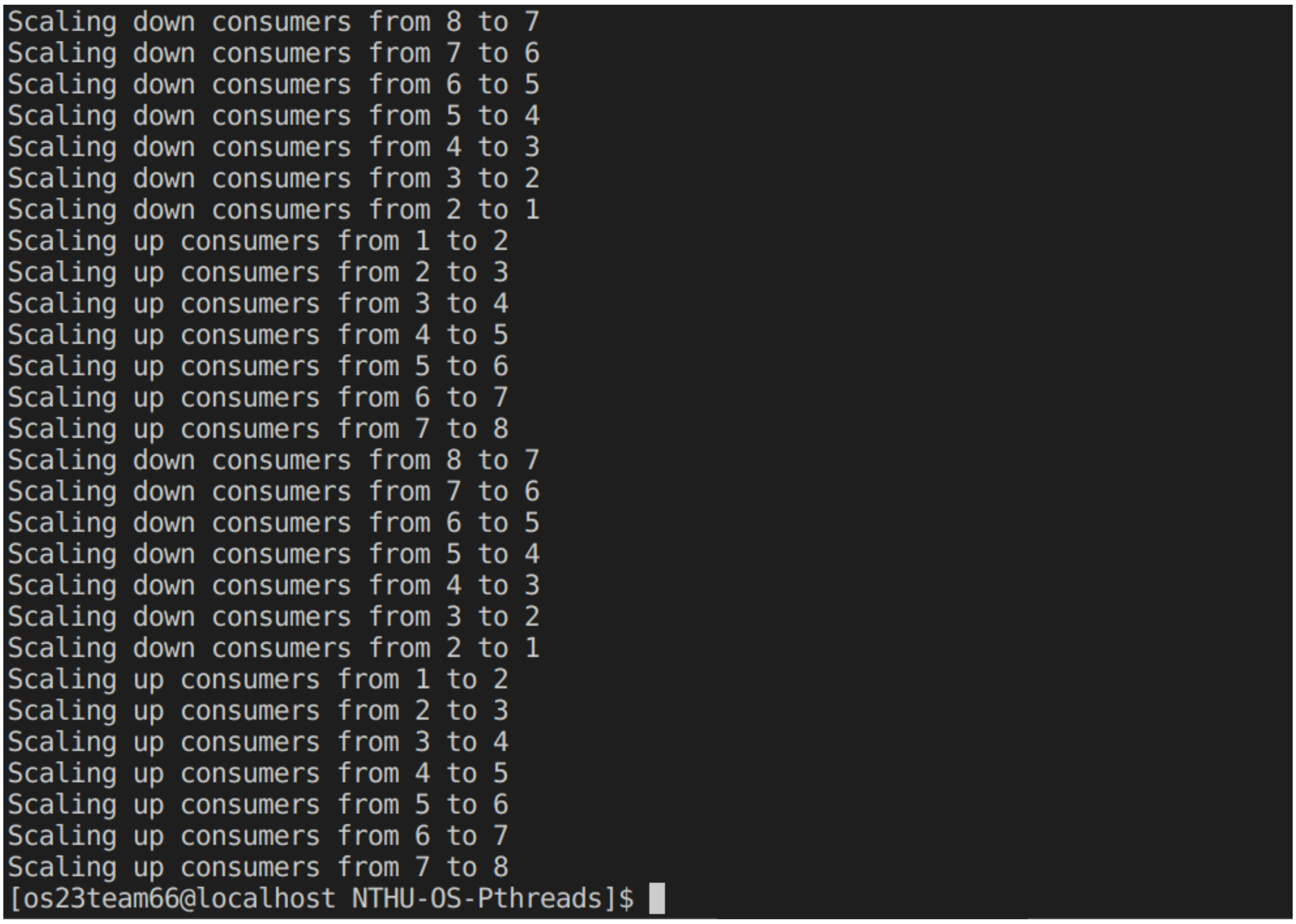
```
[os23team66@localhost NTHU-OS-Pthreads]$ ./main 4000 ./tests/01.in ./tests/01.out
Scaling up consumers from 0 to 1
Scaling up consumers from 1 to 2
Scaling up consumers from 2 to 3
Scaling up consumers from 3 to 4
Scaling down consumers from 4 to 3
Scaling down consumers from 3 to 2
Scaling down 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
[os23team66@localhost NTHU-OS-Pthreads]$
```

- low / high = 0/80(test01)



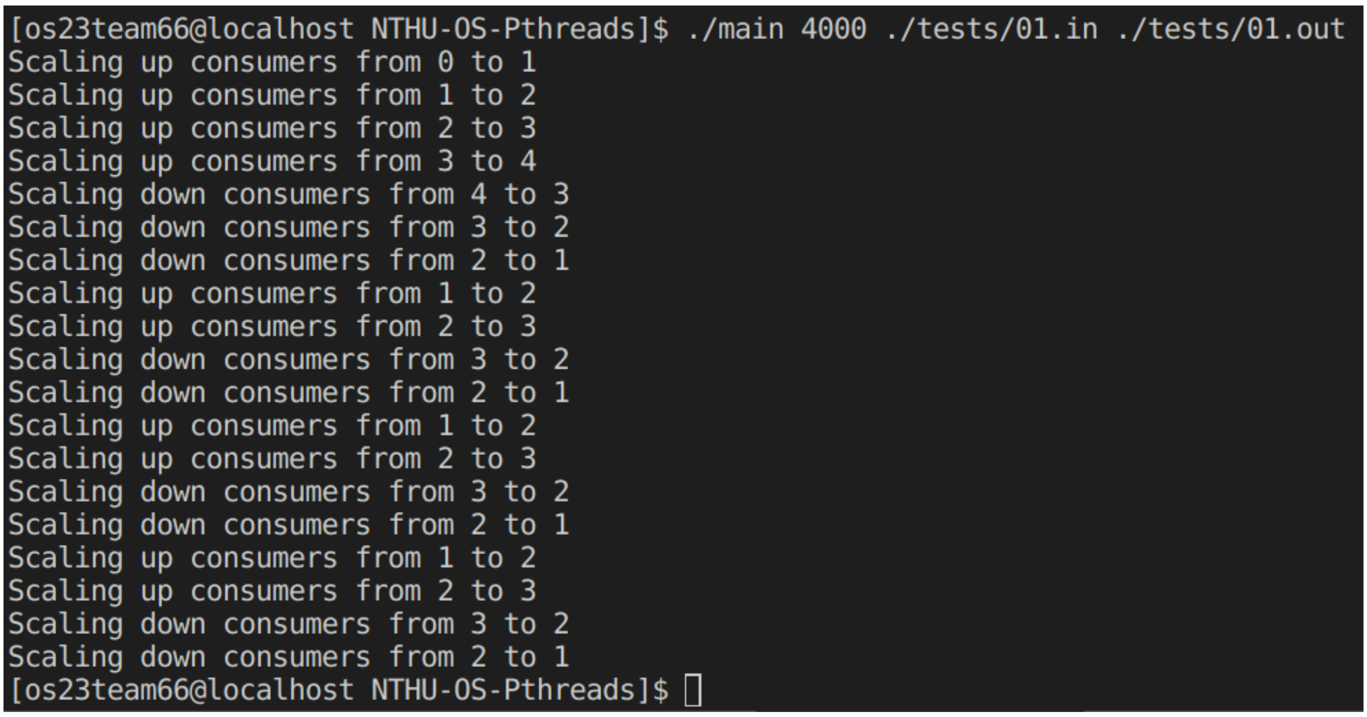
```
Scaling up consumers from 0 to 1
Scaling up consumers from 1 to 2
Scaling up consumers from 2 to 3
Scaling up consumers from 3 to 4
Scaling up consumers from 4 to 5
Scaling up consumers from 5 to 6
Scaling down consumers from 6 to 5
Scaling down consumers from 5 to 4
Scaling down consumers from 4 to 3
Scaling down consumers from 3 to 2
Scaling down consumers from 2 to 1
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 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
[os23team66@localhost NTHU-OS-Pthreads]$
```

- low / high = 10/80(test01)



```
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 up consumers from 7 to 8
[os23team66@localhost NTHU-OS-Pthreads]$
```

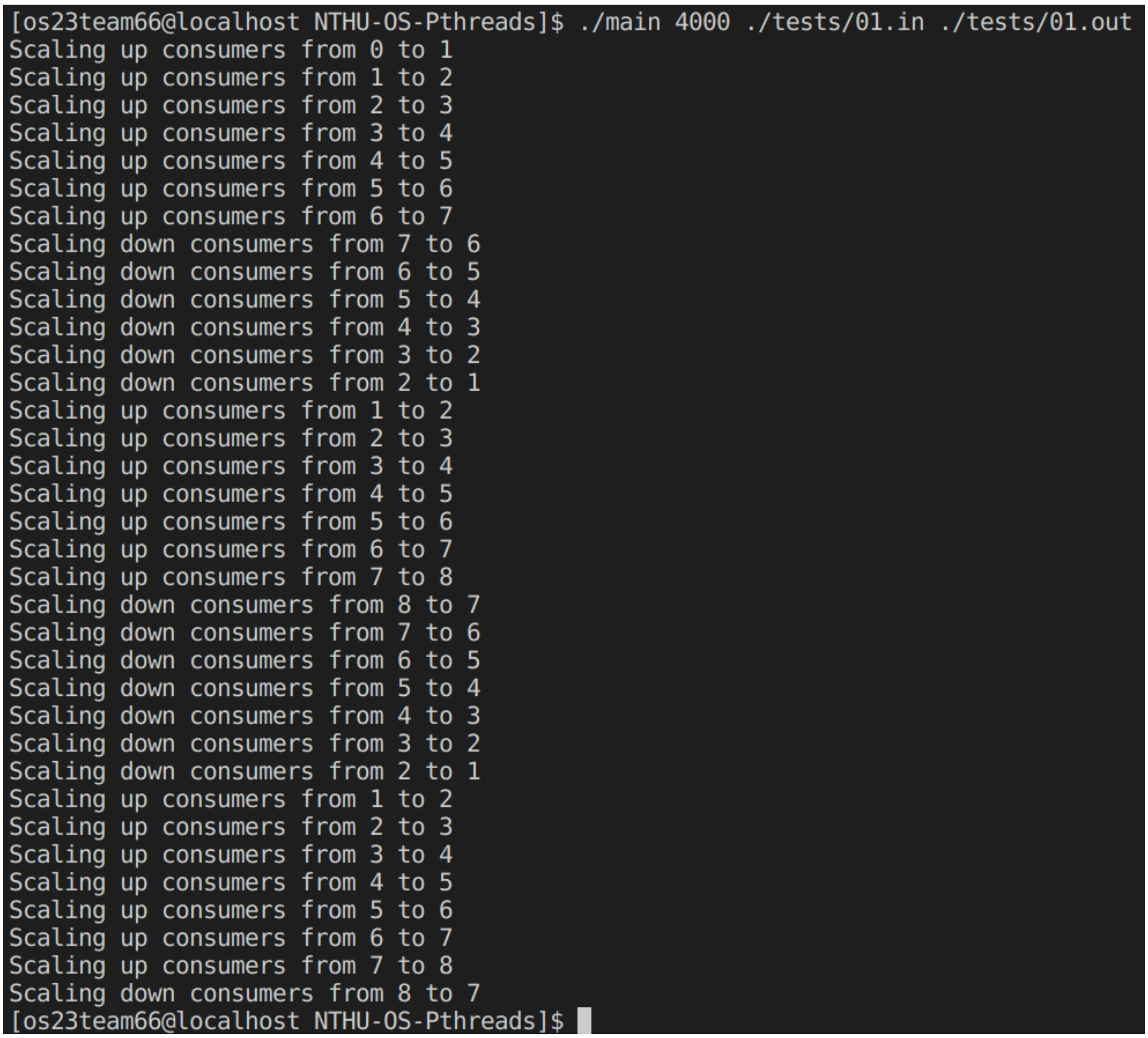
- low / high = 30/80(test01)



```
[os23team66@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 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 down consumers from 3 to 2
Scaling down consumers from 2 to 1
[os23team66@localhost NTHU-OS-Pthreads]$ 
```

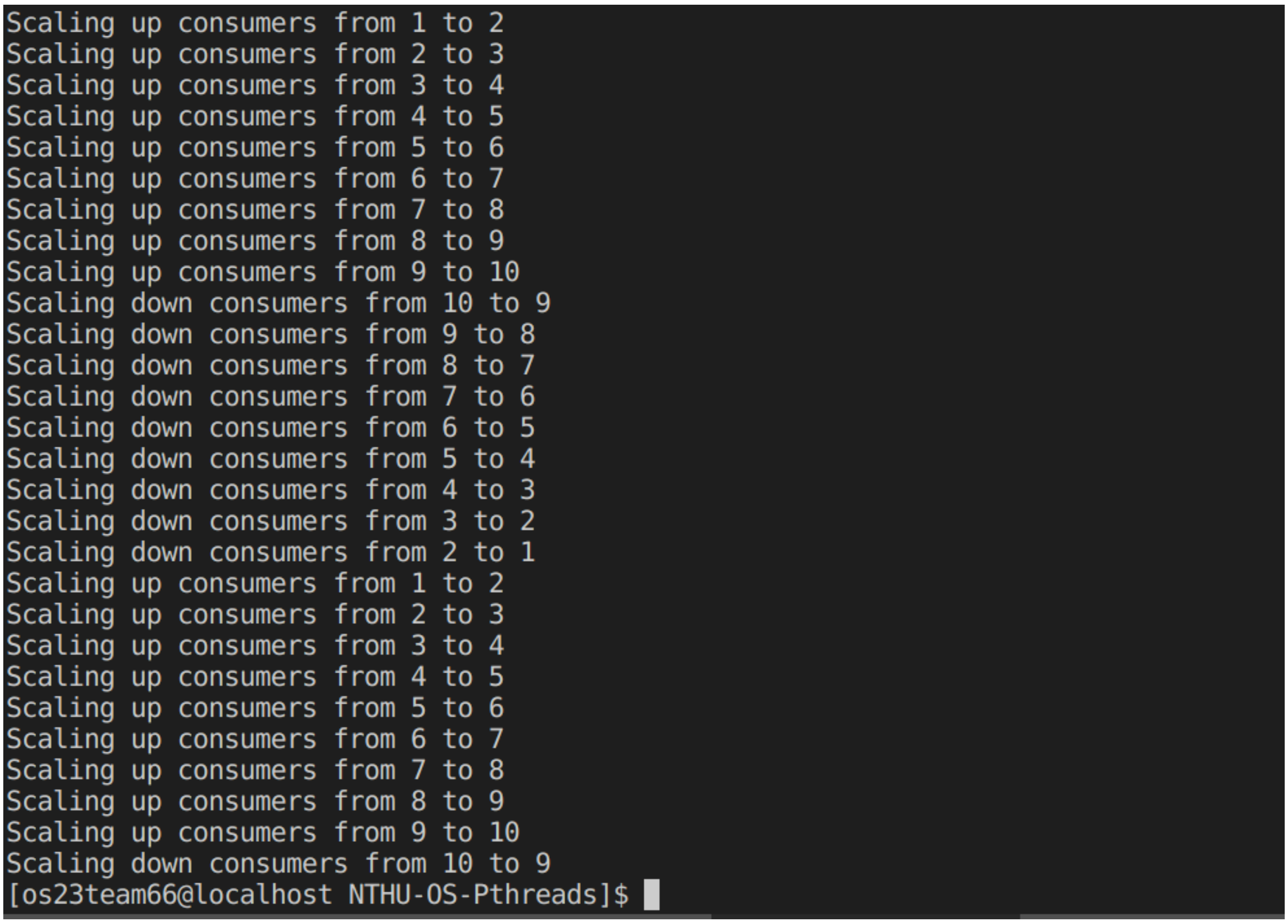
We can see that for lower low_threshold, it will be harder to scale down. Hence the maximum number of consumers will become higher. There is also something strange here. For low = 0, there should not be any scaling down.

- low / high = 20/90(test01)



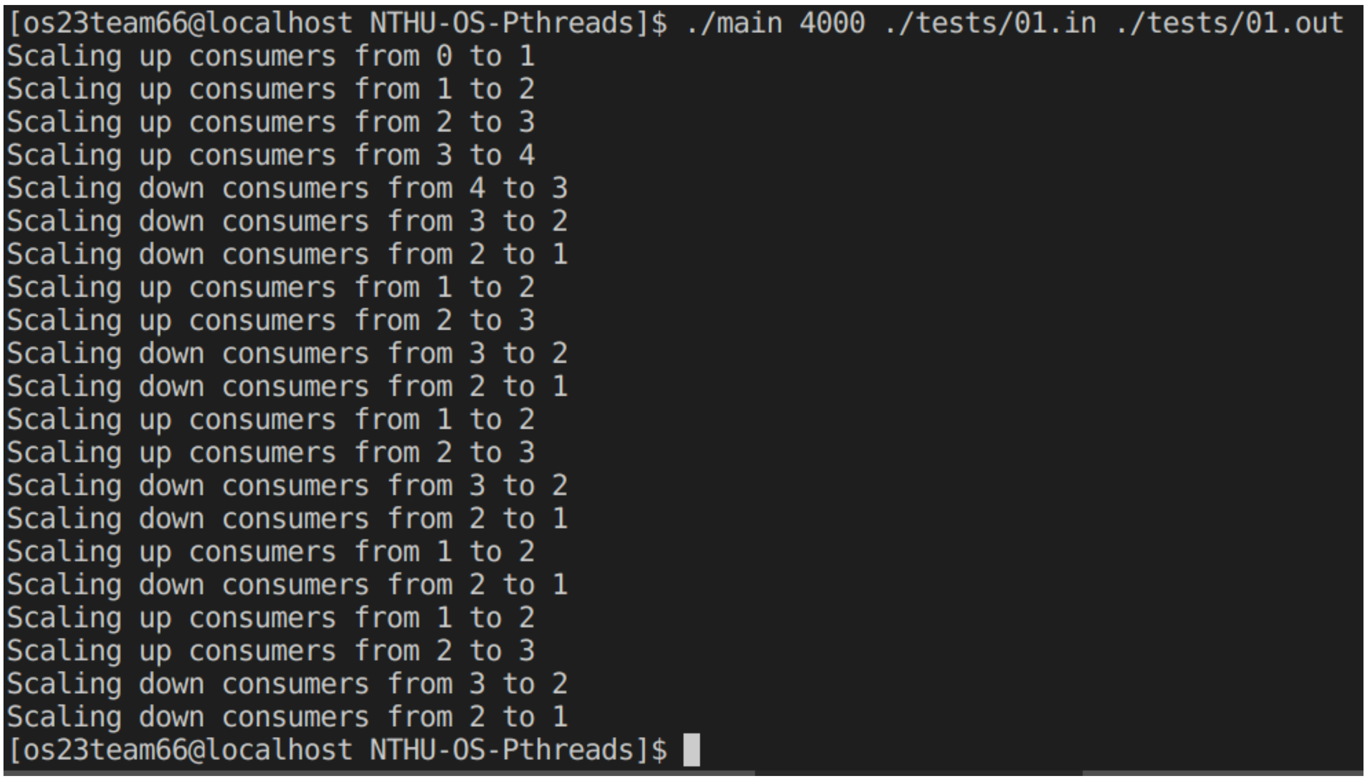
```
[os23team66@localhost NTHU-OS-Pthreads]$ ./main 4000 ./tests/01.in ./tests/01.out
Scaling up consumers from 0 to 1
Scaling up consumers from 1 to 2
Scaling up consumers from 2 to 3
Scaling up consumers from 3 to 4
Scaling up consumers from 4 to 5
Scaling up consumers from 5 to 6
Scaling up consumers from 6 to 7
Scaling 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 up consumers from 7 to 8
Scaling down consumers from 8 to 7
[os23team66@localhost NTHU-OS-Pthreads]$
```

- low / high = 20/80(test01)



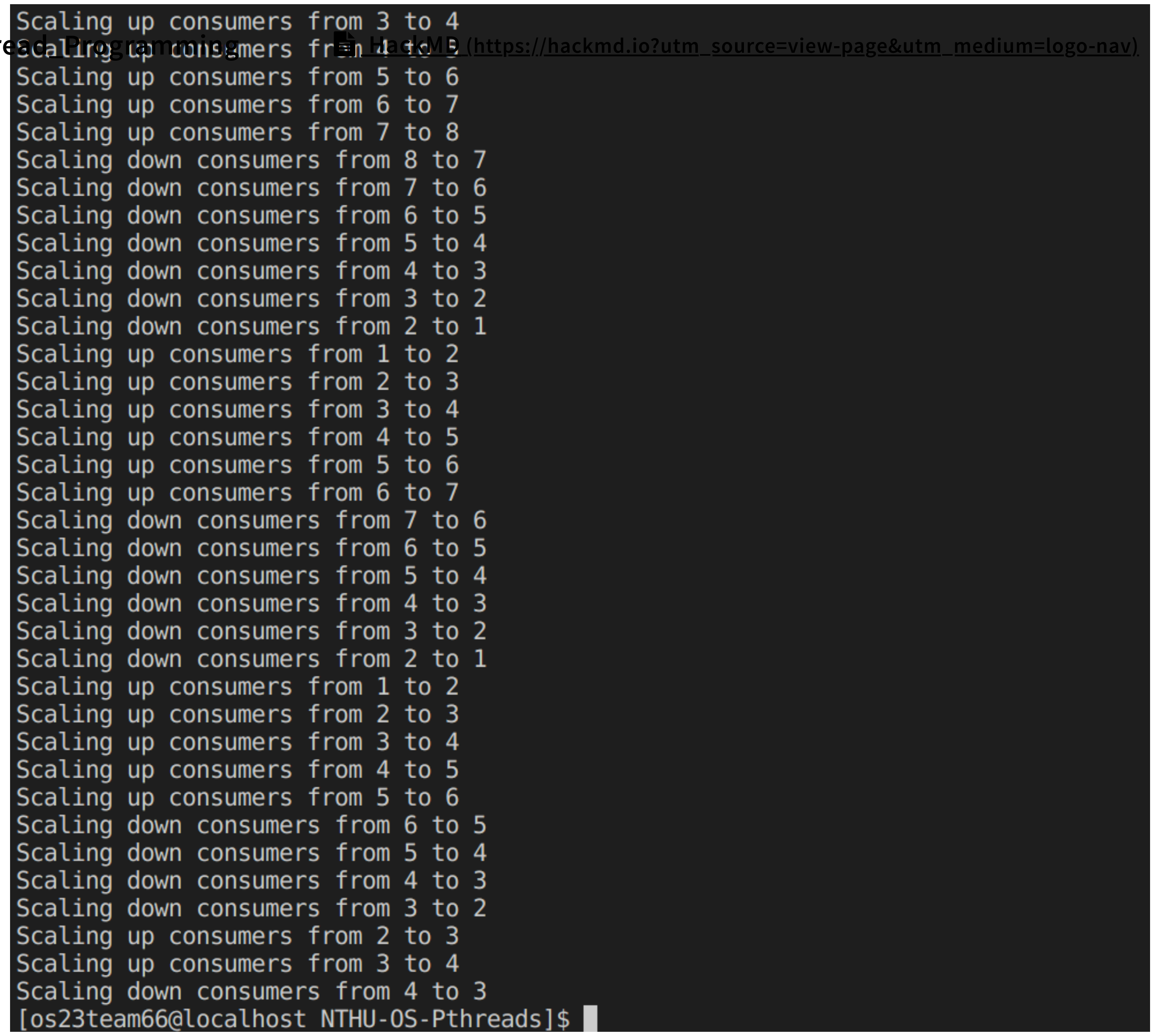
```
Scaling up consumers from 1 to 2
Scaling up consumers from 2 to 3
Scaling up consumers from 3 to 4
Scaling up consumers from 4 to 5
Scaling up consumers from 5 to 6
Scaling up consumers from 6 to 7
Scaling up consumers from 7 to 8
Scaling up consumers from 8 to 9
Scaling up consumers from 9 to 10
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
[os23team66@localhost NTHU-OS-Pthreads]$ 
```

- low / high = 20/70(test01)



```
[os23team66@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 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 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
[os23team66@localhost NTHU-OS-Pthreads]$ 
```

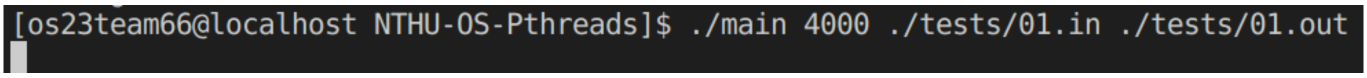
- low / hign = 20/40(test01)



```
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
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 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 down consumers from 4 to 3
[os23team66@localhost NTHU-OS-Pthreads]$
```

We can see that the speed of scaling up become faster as the high threshold get lower. Also, we should have observed that the maximum number of consumers will become higher, while this is not obvious.

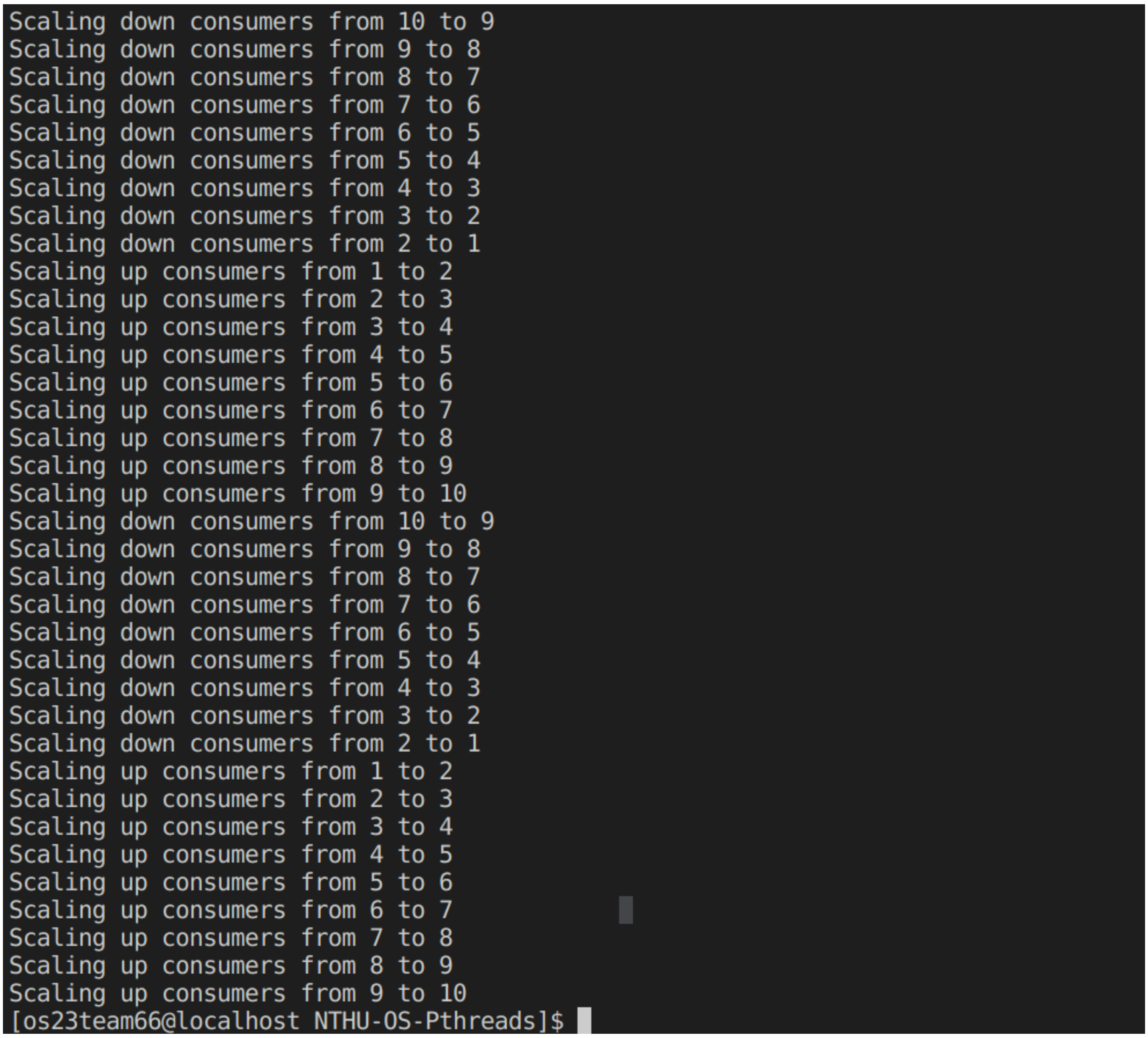
- low / high = 80/20(should be an error)



```
[os23team66@localhost NTHU-OS-Pthreads]$ ./main 4000 ./tests/01.in ./tests/01.out
```

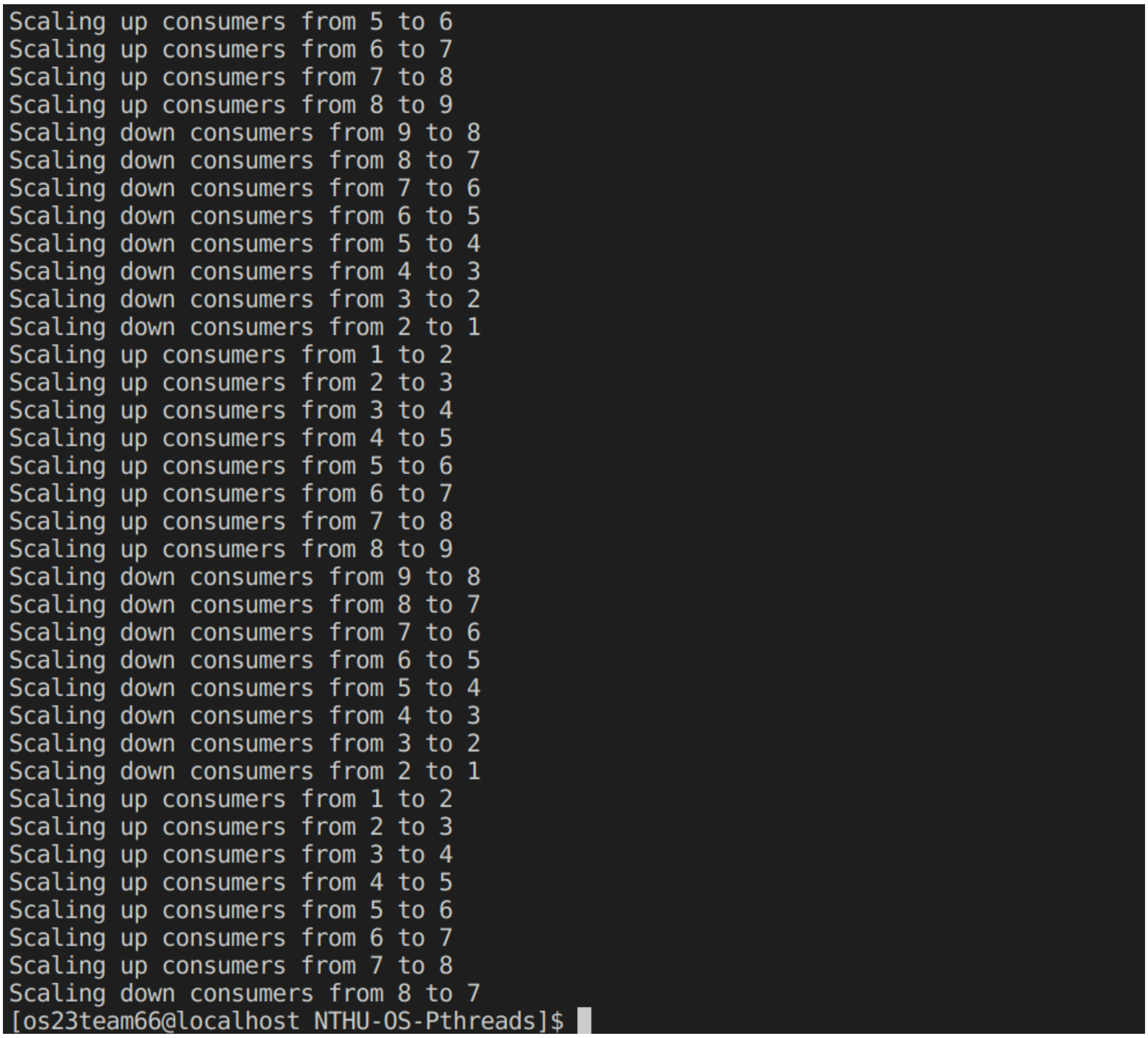
Different values of WORKER_QUEUE_SIZE.

- Worker queue size = 200(test01)



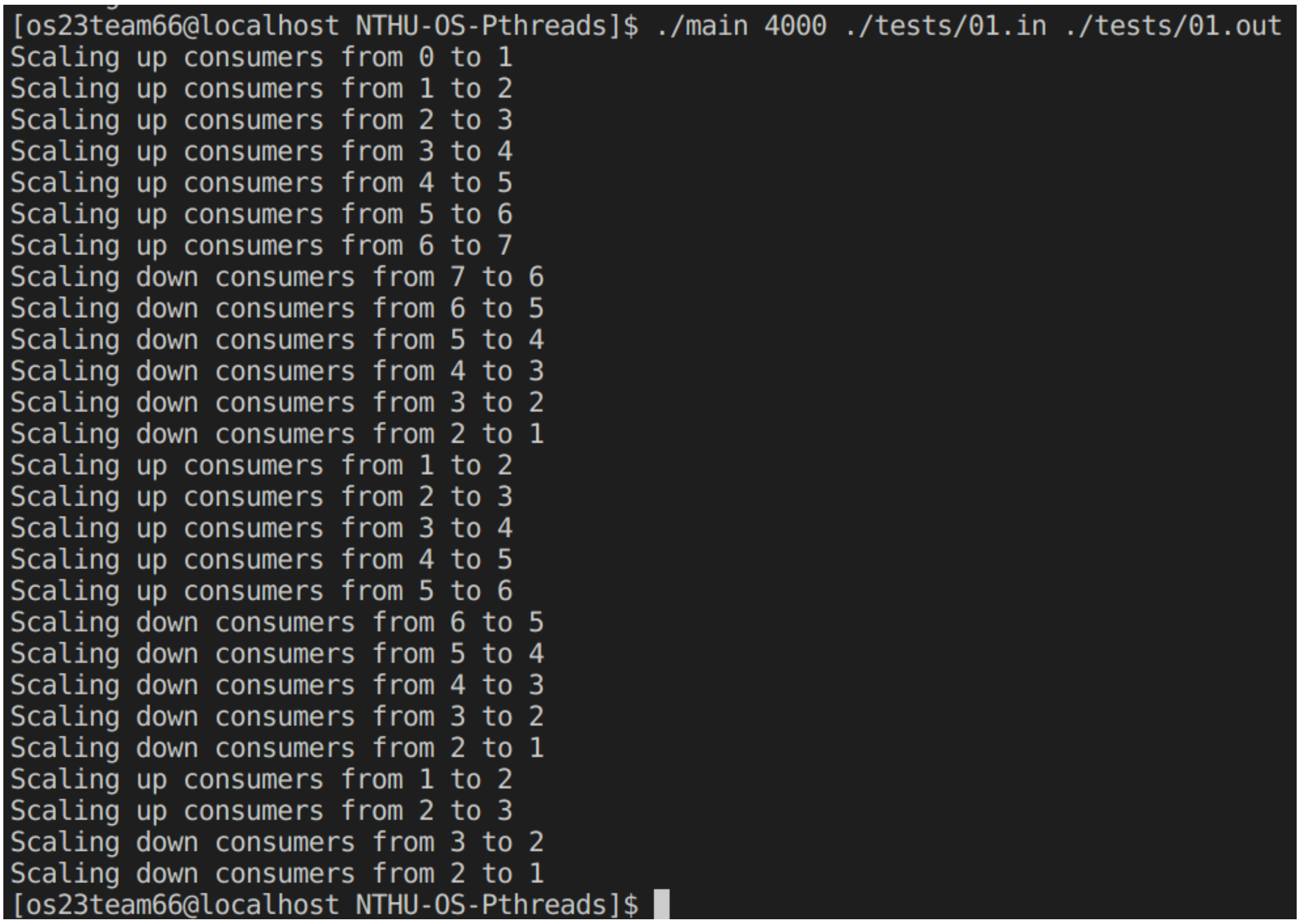
```
Scaling down consumers from 10 to 9
Scaling down consumers from 9 to 8
Scaling down consumers from 8 to 7
Scaling down consumers from 7 to 6
Scaling down consumers from 6 to 5
Scaling down consumers from 5 to 4
Scaling down consumers from 4 to 3
Scaling down consumers from 3 to 2
Scaling down consumers 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
[os23team66@localhost NTHU-OS-Pthreads]$
```

- Worker queue size = 400(test01)



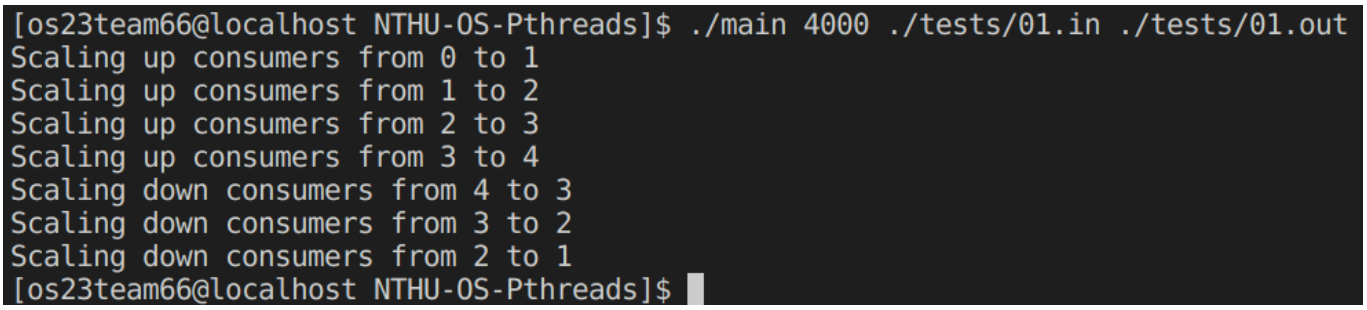
```
Scaling up consumers from 5 to 6
Scaling up consumers from 6 to 7
Scaling up consumers from 7 to 8
Scaling up consumers from 8 to 9
Scaling down consumers from 9 to 8
Scaling down consumers from 8 to 7
Scaling down consumers from 7 to 6
Scaling down consumers from 6 to 5
Scaling down consumers from 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 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 down consumers from 8 to 7
[os23team66@localhost NTHU-OS-Pthreads]$ 
```

- Worker queue size = 800(test01)



```
[os23team66@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 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 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
[os23team66@localhost NTHU-OS-Pthreads]$
```

- Worker queue size = 1600(test01)

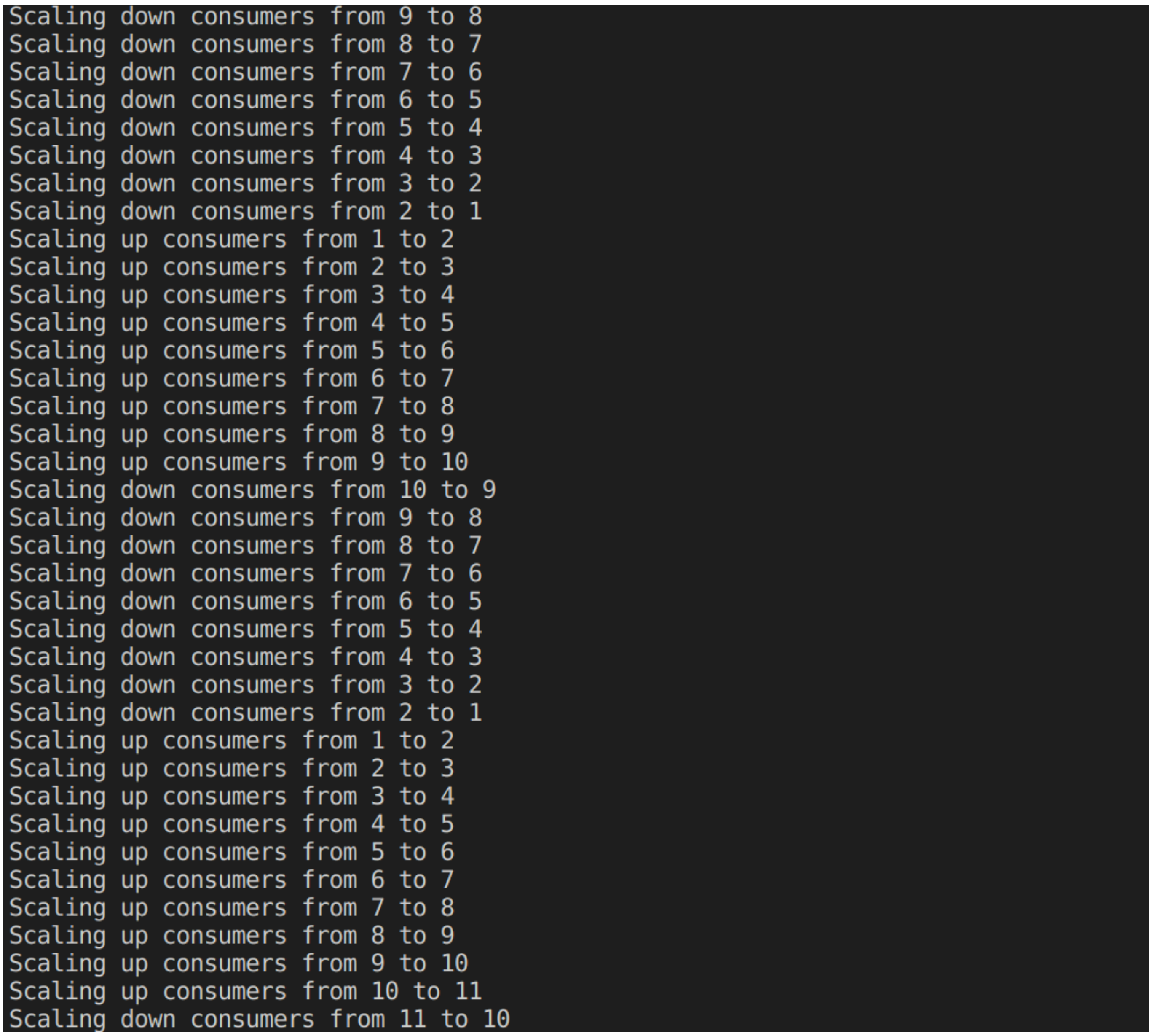


```
[os23team66@localhost NTHU-OS-Pthreads]$ ./main 4000 ./tests/01.in ./tests/01.out
Scaling up consumers from 0 to 1
Scaling up consumers from 1 to 2
Scaling up consumers from 2 to 3
Scaling up consumers from 3 to 4
Scaling down consumers from 4 to 3
Scaling down consumers from 3 to 2
Scaling down consumers from 2 to 1
[os23team66@localhost NTHU-OS-Pthreads]$
```

As worker queue size become larger, it should be harder to reach the high threshold, so it will be harder to do scale up, so the maximum number of consumers is less.

What happens if WRITER_QUEUE_SIZE is very small?

- Writer queue size = 1

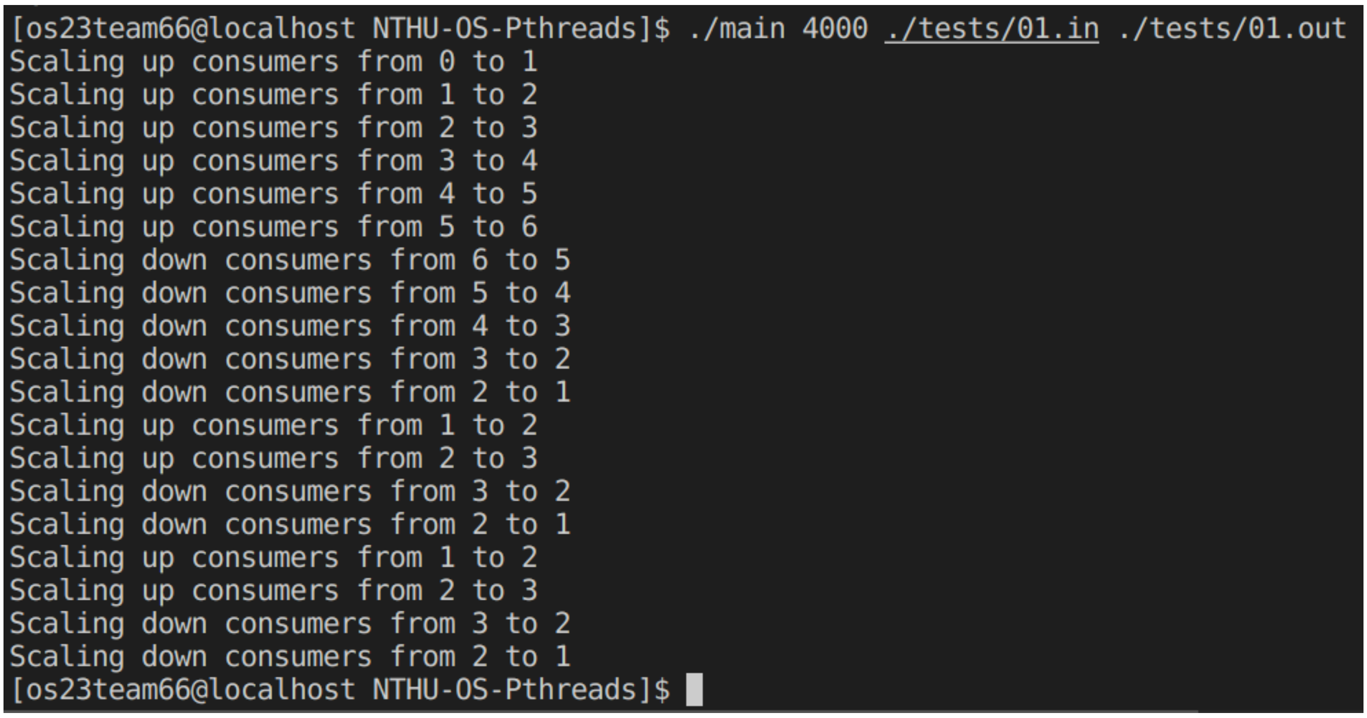


```
Scaling down consumers from 9 to 8
Scaling down consumers from 8 to 7
Scaling down consumers from 7 to 6
Scaling down consumers from 6 to 5
Scaling down consumers from 5 to 4
Scaling down consumers from 4 to 3
Scaling down consumers from 3 to 2
Scaling down consumers from 2 to 1
Scaling up consumers 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 up consumers from 10 to 11
Scaling down consumers from 11 to 10
```

The writer size is too small, so the tasks will be conjested in consumer, causing the number of consumer to rise.

What happens if READER_QUEUE_SIZE is very small?

- Reader queue size = 1



```
[os23team66@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 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
[os23team66@localhost NTHU-OS-Pthreads]$ 
```

The reader size is too small, so the worker queue will increase slowly, causing the comsumer not increase that much.

Difficulties & Feedback

這次的作業很像填空題，Spec上都有很清楚的說要做什麼，只需要模仿原本有的部份的架構寫就好，有點像是把助教挖空的部份填上去而已。我猜這次作業的目的可能不在於每個thread如何實做，比較像是要如何用pthread library來處理同步化。但是因為我們這學期有修UNIX進階程式設計，有兩大章節在教如何使用pthread裡面的各種function，所以同步化對我們來說不算是大問題。