Pthreads Report

Team21

葉宥忻 109062301

陳禹勳 109062134

contribution: 一起

Implementation

int main(int argc, char** argv)

```
17  int main(int argc, char** argv) {
18      assert(argc == 4);
19
20      int n = atoi(argv[1]);
21      std::string input_file_name(argv[2]);
22      std::string output_file_name(argv[3]);
```

argv[1]為item數量，argv[2]和argv[3]分別為input和output的file name

```
25      TSQueue<Item*>* input_queue = new TSQueue<Item*>(READER_QUEUE_SIZE);
26      TSQueue<Item*>* worker_queue = new TSQueue<Item*>(WORKER_QUEUE_SIZE);
27      TSQueue<Item*>* output_queue = new TSQueue<Item*>(WRITER_QUEUE_SIZE);
28
29      /* Create */
30      Transformer* transformer = new Transformer;
31      Reader* reader = new Reader(n, input_file_name, input_queue);
32      Producer* producer1 = new Producer(input_queue, worker_queue, transformer);
33      Producer* producer2 = new Producer(input_queue, worker_queue, transformer);
34      Producer* producer3 = new Producer(input_queue, worker_queue, transformer);
35      Producer* producer4 = new Producer(input_queue, worker_queue, transformer);
36      ConsumerController* consumer_controller = new ConsumerController(worker_queue, output_queue, transformer,
37                                                CONSUMER_CONTROLLER_CHECK_PERIOD,
38                                                CONSUMER_CONTROLLER_LOW_THRESHOLD_PERCENTAGE * WORKER_QUEUE_SIZE / 100,
39                                                CONSUMER_CONTROLLER_HIGH_THRESHOLD_PERCENTAGE * WORKER_QUEUE_SIZE / 100);
40      Writer* writer = new Writer(n, output_file_name, output_queue);
```

Create一開始需要的queue、transformer、producer、reader、writer等

```
42      /* start */
43      reader->start();
44      producer1->start();
45      producer2->start();
46      producer3->start();
47      producer4->start();
48      consumer_controller->start();
49      writer->start();
50
51      /* wait for finish */
52      reader->join();
53      writer->join();
```

start各自的thread，並等待reader和writer結束

```
55      /* delete */
56      delete input_queue;
57      delete worker_queue;
58      delete output_queue;
59      delete transformer;
60      delete reader;
61      delete producer1;
62      delete producer2;
63      delete producer3;
64      delete producer4;
65      delete consumer_controller;
66      delete writer;
```

TSQueue<T>::TSQueue()

```cpp
template <class T>
TSQueue<T>::TSQueue(int buffer_size) : buffer_size(buffer_size) {
    // TODO: implements TSQueue constructor
    pthread_mutex_init(&mutex, NULL);
    pthread_cond_init(&cond_enqueue, NULL);
    pthread_cond_init(&cond_dequeue, NULL);
    buffer = new T [buffer_size];
    size = 0;
    head = 0;
    tail = 0;
}
```

init mutex和condition variable，create一個buffer，初始化size、head、tail

TSQueue<T>::~TSQueue()

```cpp
template <class T>
TSQueue<T>::~TSQueue() {
    // TODO: implenents TSQueue destructor
    pthread_mutex_destroy(&mutex);
    pthread_cond_destroy(&cond_enqueue);
    pthread_cond_destroy(&cond_dequeue);
    delete buffer;
}
```

delete mutex、condition variable、buffer

void TSQueue<T>::enqueue(T item)

```
73  void TSQueue<T>::enqueue(T item) {
74      // TODO: enqueues an element to the end of the queue
75
76      pthread_mutex_lock(&mutex);
77      while (size == buffer_size) {
78          pthread_cond_wait(&cond_enqueue, &mutex);
79      }
80
81      buffer[tail] = item;
82      tail = (tail + 1) % buffer_size;
83      size = size + 1;
84
85      pthread_cond_signal(&cond_dequeue);
86      pthread_mutex_unlock(&mutex);      You, 1 second ago
87  }
```

76行：避免和其他thread同時access share memory，例如buffer、size、tail等

77-79行：如果buffer現在已經滿了，讓這個thread wait cond_enqueue這個condition variable，等待其他thread執行dequeue後呼叫signal，cond_wait要將mutex也傳進去進行release，這樣其他人才能拿得到mutex

81-83行：新增item到buffer，更新tail和size

85-86行：signal cond_dequeue並release mutex

void TSQueue<T>::dequeue()

```
90  T TSQueue<T>::dequeue() {
91      // TODO: dequeues the first element of the queue
92
93      pthread_mutex_lock(&mutex);
94      while (size == 0) {
95          pthread_cond_wait(&cond_dequeue, &mutex);
96      }
97
98      T dequeued_element = buffer[head];
99      head = (head + 1) % buffer_size;
100     size = size - 1;
101
102     pthread_cond_signal(&cond_enqueue);
103     pthread_mutex_unlock(&mutex);
104
105     return dequeued_element;
106 }
```

93行：避免和其他thread同時access share memory，例如buffer、size、tail等

94-96行：如果buffer現在已經是空的，讓這個thread wait cond_dequeue這個condition variable，等待其他thread執行enqueue後呼叫signal，cond_wait要將mutex也傳進去進行release，這樣其他人才能拿得到mutex

98-100行：取出item，更新head和size

102-103行：signal cond_enqueue並release mutex

105行：return取出的item

int TSQueue<T>::get_size()

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

    pthread_mutex_lock(&mutex);
    int curr_size = size;
    pthread_mutex_unlock(&mutex);

    return curr_size;
}
```

因為size也是share memory，也要用mutex包起來

void Producer::start()

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

```cpp
class Thread {
public:
    // to start a new pthread work
    virtual void start() = 0;

    // to wait for the pthread work to complete
    virtual int join();

    // to cancel the pthread work
    virtual int cancel();
protected:
    pthread_t t;
};
```

pthread_create create一個thread給Producer::process，並把Producer指標當作arg傳給Producer::process，t為create好的pthread，這個變數在繼承的Thread class中

void* Producer::process(void* arg)

```
40  void* Producer::process(void* arg) {
41      // TODO: implements the Producer's work
42      Producer* producer = (Producer*)arg;
43
44      while (true) {
45          Item* item = producer->input_queue->dequeue();
46          item->val = producer->transformer->producer_transform(item->opcode, item->val);
47          producer->worker_queue->enqueue(item);
48      }
49
50      return nullptr;
51  }
```

重複從input_queue中dequeue item，並把item value透過Transformer::producer_transform轉換後，enqueue item到worker queue

void ConsumerController::start()

```
69  void ConsumerController::start() {
70      // TODO: starts a ConsumerController thread
71      pthread_create(&t, 0, ConsumerController::process, (void*)this);
72  }
```

pthread_create create一個thread給ConsumerController::process，並把ConsumerController指標當作arg傳給ConsumerController::process

void ConsumerController::process(void* arg)

```
74  void* ConsumerController::process(void* arg) {
75      // TODO: implements the ConsumerController's work
76      ConsumerController* consumer_controller = (ConsumerController*)arg;
77
78      while (true) {
79          int curr_size = consumer_controller->worker_queue->get_size();
80
81          if (curr_size < consumer_controller->low_threshold) {···
90          else if (curr_size > consumer_controller->high_threshold) {···
97
98          // Check periodically in microsecond (us)
99          usleep(consumer_controller->check_period);
100     }
101
102     return nullptr;
103 }
```

```
81      if (curr_size < consumer_controller->low_threshold) {
82          if (consumer_controller->consumers.size() > 1) {
83              std::cout << "Scaling down consumers from " << consumer_controller->consumers.size() << " to " <<
                consumer_controller->consumers.size()-1 << std::endl;
84
85              Consumer* to_delete = consumer_controller->consumers.back();
86              consumer_controller->consumers.pop_back();
87              to_delete->cancel();
88          }
89      }
```

```
90      else if (curr_size > consumer_controller->high_threshold) {
91          std::cout << "Scaling up consumers from " << consumer_controller->consumers.size() << " to " << consumer_controller->consumers.
            size()+1 << std::endl;
92
93          Consumer* newConsumer = new Consumer(consumer_controller->worker_queue, consumer_controller->writer_queue,
            consumer_controller->transformer);
94          consumer_controller->consumers.push_back(newConsumer);
95          newConsumer->start();
96      }
```

```
98          // Check periodically in microsecond (us)
99          usleep(consumer_controller->check_period);
```

1. 先獲取現在worker queue的item數量
2. 如果數量小於low_threshold，並且running consumer數量大於一個，呼叫Consumer::cancel cancel最後一個consumer
3. 如果數量大於high_threshold，新增一個consumer
4. 每consumer_controller->check_period(microsecond)重複1-3

void Consumer::start()

```
41  void Consumer::start() {
42      // TODO: starts a Consumer thread
43      pthread_create(&t, 0, Consumer::process, (void*)this);
44  }
```

pthread_create create一個thread給Consumer::process，並把Consumer指標當作arg傳給Consumer::process

void Consumer::process(void* arg)

```
52  void* Consumer::process(void* arg) {
53      Consumer* consumer = (Consumer*)arg;
54
55      pthread_setcanceltype(PTHREAD_CANCEL_DEFERRED, nullptr);
56
57      while (!consumer->is_cancel) {
58          pthread_setcancelstate(PTHREAD_CANCEL_DISABLE, nullptr);
59
60          // TODO: implements the Consumer's work
61          Item* item = consumer->worker_queue->dequeue();
62          item->val = consumer->transformer->consumer_transform(item->opcode, item->val);
63          consumer->output_queue->enqueue(item);
64
65          pthread_setcancelstate(PTHREAD_CANCEL_ENABLE, nullptr);
66      }
67
68      delete consumer;
69
70      return nullptr;
71  }
```

先將canceltype設為PTHREAD_CANCEL_DEFERRED，所以當thread接收到cancel請求後，不會立即cancel，會等執行到cancellation point才做cancel

接下來當consumer->is_cancel為false時，重複以下步驟

1. cancelstate設為PTHREAD_CANCEL_DISABLE，此時如果有cancel請求，會先把請求記下來，等到state變回PTHREAD_CANCEL_ENABLE再執行
2. 從worker_queue中dequeue item，並把item value透過Transformer::consumer_transform轉換後，enqueue item到output_queue
3. cancelstate設為PTHREAD_CANCEL_ENABLE，可以處理cancel請求

int Consumer::cancel()

```
46  int Consumer::cancel() {
47      // TODO: cancels the consumer thread
48      is_cancel = true;
49      return pthread_cancel(this->t);
50  }
```

把is_cancel設為true，並呼叫pthread_cancel把這個thread cancel

void Writer::start()

```
41  void Writer::start() {
42      // TODO: starts a Writer thread
43      pthread_create(&t, 0, Writer::process, (void*)this);
44  }
```

pthread_create create一個thread給Writer::process，並把Writer指標當作arg傳給Writer::process

void Writer::process(void* arg)

```
46  void* Writer::process(void* arg) {
47      // TODO: implements the Writer's work
48      Writer* writer = (Writer*)arg;
49
50      while (writer->expected_lines--) {
51          Item *item = writer->output_queue->dequeue();
52          writer->ofs << *item;
53      }
54
55      return nullptr;
56  }
```

```cpp
std::istream& operator>>(std::istream& in, Item& item) {
    in >> item.key >> item.val >> item.opcode;
    return in;
}

std::ostream& operator<<(std::ostream& os, const Item& item) {
    os << item.key << ' ' << item.val << ' ' << item.opcode << '\n';
    return os;
}
```

將預計要output數量的item，依序從output_queue中dequeue，再用std::ofstream output item到output file，item的<<和>> operator定義在item.hpp中

Experiment

以下experiment除了Different value以外其他變數相同

1. Different values of CONSUMER_CONTROLLER_CHECK_PERIOD.

CONSUMER_CONTROLLER_CHECK_PERIOD = 1000000(us)

```
Scaling up consumers from 0 to 1
Scaling up consumers from 1 to 2
Scaling down consumers from 2 to 1
```

CONSUMER_CONTROLLER_CHECK_PERIOD = 1000(us)

```
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
```

當CONSUMER_CONTROLLER_CHECK_PERIOD越小，ConsumerController會越頻繁的去確認worker queue的大小，所以如果size超過HIGH_THRESHOLD，可能會新增比較多的consuemr，size小於LOW_THRESHOLD減少consumer也是。

2. Different values of CONSUMER_CONTROLLER_LOW_THRESHOLD_PERCENTAGE and CONSUMER_CONTROLLER_HIGH_THRESHOLD_PERCENTAGE.

CONSUMER_CONTROLLER_HIGH_THRESHOLD_PERCENTAGE = 80

```
Scaling up consumers from 0 to 1
Scaling up consumers from 1 to 2
Scaling down consumers from 2 to 1
```

CONSUMER_CONTROLLER_HIGH_THRESHOLD_PERCENTAGE = 40

```
Scaling up consumers from 0 to 1
Scaling up consumers from 1 to 2
Scaling up consumers from 2 to 3
```

如果HIGH_THRESHOLD變低了，HIGH_THRESHOLD比較容易超過，因此consumers可能會新增的比較快，或是增加的次數比較多，但是因為consumer比較多，所以會比較快結束，有可能還沒減少consumer就做完了

CONSUMER_CONTROLLER_HIGH_THRESHOLD_PERCENTAGE = 40

```
Scaling up consumers from 0 to 1
```

如果HIGH_THRESHOLD變高了，HIGH_THRESHOLD比較難超過，因此consumers可能會新增的比較慢，或是增加的次數比較少

==========

CONSUMER_CONTROLLER_LOW_THRESHOLD_PERCENTAGE = 20

```
Scaling up consumers from 0 to 1
Scaling up consumers from 1 to 2
Scaling down consumers from 2 to 1
```

CONSUMER_CONTROLLER_LOW_THRESHOLD_PERCENTAGE = 5

```
Scaling up consumers from 0 to 1
Scaling up consumers from 1 to 2
```

如果LOW_THRESHOLD比較低，會比較難低於LOW_THRESHOLD，所以減少consumer的次數可能會比較少

CONSUMER_CONTROLLER_LOW_THRESHOLD_PERCENTAGE = 5

```
Scaling up consumers from 0 to 1
Scaling up consumers from 1 to 2
Scaling down consumers from 2 to 1
```

如果LOW_THRESHOLD比較高，會比較容易低於LOW_THRESHOLD，所以減少consumer的次數可能會比較多，但是會使執行時間變久

3. Different values of WORKER_QUEUE_SIZE.

WORKER_QUEUE_SIZE = 200

```
Scaling up consumers from 0 to 1
Scaling up consumers from 1 to 2
Scaling down consumers from 2 to 1
```

WORKER_QUEUE_SIZE = 20

```
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
```

因為WORKER_QUEUE_SIZE變小，所以比較快超過HIGH_THRESHOLD，所以增加consumers的速度可能會比較快

4. What happens if WRITER_QUEUE_SIZE is very small?

WRITER_QUEUE_SIZE = 4000

```
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
```

WRITER_QUEUE_SIZE = 1

```
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
```

因為writer queue很小，所以consumer們就需要排隊去enqueue writer queue，dequeue worker queue的速度就會降低，造成worker queue消耗速度變慢，所以更有可能超過HIGH_THRESHOLD去增加consumer

5. What happens if READER_QUEUE_SIZE is very small?

READER_QUEUE_SIZE = 200

```
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
```

READER_QUEUE_SIZE = 1

```
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
```

當reader queue很小，會使得producer增加worker queue的速度變慢，同時讓consumer數量更難增加

Difficulties

因為上課有教過mutexc和condition variable了，所以在實作TSQueue時沒有太多的困難，主要的困難是不太熟悉pthread的functino如何使用，例如create、cancel，cancel還要特別注意不是一發送cancel請求就執行，要透過cancelstate和canceltype來確保在安全的地方cancel