Pthreads Programming Assignment
Team_56

Team members:

- 108020022 周昱宏
- 108071003 李彥璋

Contribution:

工作項目	負責人
Trace code	周昱宏、李彥璋
Implementation	周昱宏
Report	周昱宏

Pthreads Programming Assignment

Implementation

1. TSQueue (ts_queue.hpp)：首先是基層queue的實作，包含enqueue、dequeue、get_size的method，方便Input、Worker及Writer queue使用。

   (1)建構子：初始化ts_queue每一個的member，較需注意這邊tail設成-1因為在enqueue method裡會先將tail+1用來記住enqueue進來的元素，因此初始設成-1方便一開始enqueue時利用buffer[0]來存；mutex、cond_enqueue、cond_dequeue變數利用pthread library的function來初始化。

   (2)解構子：刪除buffer空間，並利用pthread library的function來destroy mutex、cond_enqueue、cond_dequeue變數。

   如下圖。

```
54  template <class T>
55  TSQueue<T>::TSQueue(int buffer_size) : buffer_size(buffer_size)
56  {
57      // TODO: implements TSQueue constructor
58      buffer = new T[buffer_size];
59      size = 0;
60      head = 0;
61      tail = -1;
62      pthread_mutex_init(&mutex, NULL);
63      pthread_cond_init(&cond_enqueue, NULL);
64      pthread_cond_init(&cond_dequeue, NULL);
65  }
66
67  template <class T>
68  TSQueue<T>::~TSQueue()
69  {
70      // TODO: implenents TSQueue destructor
71      delete buffer;
72      pthread_mutex_destroy(&mutex);
73      pthread_cond_destroy(&cond_enqueue);
74      pthread_cond_destroy(&cond_dequeue);
75  }
```

(3)enqueue method：

[82]利用pthread library的pthread_mutex_lock函式進行上鎖，防止其他thread同時執行下方區段。

[84-85]當queue size滿的時候，利用pthread library的pthread_cond_wait函式block住，不能enqueue。

[87-89]執行到這裡代表queue size還沒滿，可以放入元素。先將tail+1並mod buffer 容量，並記住enqueue的元素，最後size+1。

[91-92]利用pthread library的pthread_cond_signal函式，通知可以dequeue，並用 pthread_mutex_unlock函式解鎖，讓其他thread可以繼續執行。

如下圖。

```
77  template <class T>
78  void TSQueue<T>::enqueue(T item)
79  {
80      // TODO: enqueues an element to the end of the queue
81
82      pthread_mutex_lock(&mutex);
83
84      while (size == buffer_size)
85          pthread_cond_wait(&cond_enqueue, &mutex);
86
87      tail = (tail + 1) % buffer_size;
88      buffer[tail] = item;
89      size++;
90
91      pthread_cond_signal(&cond_dequeue);
92      pthread_mutex_unlock(&mutex);
93  }
```

(3)dequeue method：

[99]利用pthread library的pthread_mutex_lock函式進行上鎖，防止其他thread同時執行下方區段。

[101-102]當queue size空的時候，利用pthread library的pthread_cond_wait函式block住，不能dequeue。

[104-106]執行到這裡代表queue size不為空，可以移除元素。先記住欲移除元素，並將head+1並mod buffer容量，最後size-1。

[108-109]利用pthread library的pthread_cond_signal函式，通知可以enqueue，並用pthread_mutex_unlock函式解鎖，讓其他thread可以繼續執行。

[111]return 欲移除元素。

(4)get_size method：return size。

如下圖。

```
template <class T>
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 dequeue_one = buffer[head];
    head = (head + 1) % buffer_size;
    size--;

    pthread_cond_signal(&cond_enqueue);
    pthread_mutex_unlock(&mutex);

    return dequeue_one;
}
```

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

Producer (producer.hpp)：實現producer裡的兩個method，start以及process函式，寫法參照reader.hpp。

(1)start method：因為producer也是一個thread(繼承)，因此利用pthread library的pthread_create函式建立新的thread(producer的member t記住)，並以自己為參數開始執行其process method。

如下圖。

```
38  void Producer::start()
39  {
40      // TODO: starts a Producer thread
41      pthread_create(&t, 0, Producer::process, (void *)this);
42  }
```

(2)process method：

[47]將傳遞過來的參數轉型成Producer*。

[49-57]重複執行以下事情(若input_queue的size不為空才執行)，將input_queue dequeue的item記住，並將其val透過transformer的producer_transform函式進行更新，最後將此item放入至worker_queue。

如下圖。

```
44  void *Producer::process(void *arg)
45  {
46      // TODO: implements the Producer's work
47      Producer *producer = (Producer *)arg;
48
49      while (1)
50      {
51          if (producer->input_queue->get_size() > 0)
52          {
53              Item *dequeue_one = producer->input_queue->dequeue();
54              dequeue_one->val = producer->transformer->producer_transform(dequeue_one->opcode, dequeue_one->val);
55              producer->worker_queue->enqueue(dequeue_one);
56          }
57      }
58
59      return nullptr;
60  }
```

Consumer (consumer.hpp)：實現consumer裡的三個method，start、process以及cancel函式，寫法參照reader.hpp。

(1)start method：因為consumer也是一個thread(繼承)，因此利用pthread library的pthread_create函式建立新的thread(consumer的member t記住)，並以自己為參數開始執行其process method。

如下圖。

```
44  void Consumer::start()
45  {
46      // TODO: starts a Consumer thread
47
48      pthread_create(&t, 0, Consumer::process, (void *)this);
49  }
```

(2)process method：

[61]將傳遞過來的參數轉型成Consumer*。

[63、67、77] 67-77程式不是隨時都能執行因此加入這三行。

[70-75]若此consumer沒有被cancel掉，重複執行以下事情(若worker_queue的size不為空才執行)，將worker_queue dequeue的item記住，並將其val透過transformer的consumer_transform函式進行更新，最後將此item放入至output_queue。

如下圖。

```
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
        if (consumer->worker_queue->get_size() > 0)
        {
            Item *dequeue_one = consumer->worker_queue->dequeue();
            dequeue_one->val = consumer->transformer->consumer_transform(dequeue_one->opcode, dequeue_one->val);
            consumer->output_queue->enqueue(dequeue_one);
        }

        pthread_setcancelstate(PTHREAD_CANCEL_ENABLE, nullptr);
    }

    delete consumer;

    return nullptr;
```

(3) cancel method：利用pthread library的pthread_cancel函式取消結束自己的thread，並將is_cancel設成false，使在process method裡跳出迴圈。

如下圖。

```
int Consumer::cancel()
{
    // TODO: cancels the consumer thread
    is_cancel = true;
    pthread_cancel(this->t);
    return 0;
}
```

ConsumerController (consumer_controller.hpp)：實現consumer_controller裡的兩個method，start以及process，寫法參照reader.hpp。

(1)start method：因為consumer_controller也是一個thread(繼承)，因此利用pthread library的pthread_create函式建立新的thread(consumer_controller的member t記住)，並以自己為參數開始執行其process method。

如下圖。

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

(2)process method：

[61]將傳遞過來的參數轉型成Consumer_controller*。

[81、101、108]101-108程式不是隨時都能執行因此加入這三行。

[80-88]創一個變數period，在進行無窮迴圈時每次皆+1，如果其與check_period相等，代表要判斷consumer的增減了，將period歸零，並利用size記住work_queue的size。

[89-96]如果size大於high_threshold，代表要新增一個consumer，將此consumer放入維護的consumer vector裡，並印出consumer的size變化，最後將此consumer start。

[97-108]如果size小於low_threshold，代表要刪除最新的consumer(先判定共有幾個consumer，若剩一個則continue不需執行以下程式)，利用vector的back method取得最新的consumer把他cancel掉，並將consumers pop_back掉，最後印出consumer的size變化。

如下圖。

```
75  void *ConsumerController::process(void *arg)
76  {
77      // TODO: implements the ConsumerController's work
78      ConsumerController *consumerController = (ConsumerController *)arg;
79
80      int period = 0;
81      pthread_setcanceltype(PTHREAD_CANCEL_DEFERRED, nullptr);
82      while (1)
83      {
84          ++period;
85          if (period >= consumerController->check_period)
86          {
87              period = 0;
88              int size = consumerController->worker_queue->get_size();
89              if (size > consumerController->high_threshold)
90              {
91                  Consumer *consumer = new Consumer(consumerController->worker_queue, consumerController->writer_queue, consumerController->transformer);
92                  int original_size = consumerController->consumers.size();
93                  consumerController->consumers.push_back(consumer);
94                  std::cout << "Scaling up consumers from " << original_size << " to " << original_size + 1 << '\n';
95                  consumer->start();
96              }
97              if (size < consumerController->low_threshold)
98              {
99                  if (consumerController->consumers.size() <= 1)
100                     continue;
101                 pthread_setcancelstate(PTHREAD_CANCEL_DISABLE, nullptr);
102                 int original_size = consumerController->consumers.size();
103                 Consumer *newest_consumer = consumerController->consumers.back();
104                 consumerController->consumers.pop_back();
105                 std::cout << "Scaling down consumers from " << original_size << " to " << original_size - 1 << '\n';
106                 newest_consumer->cancel();
107
108                 pthread_setcancelstate(PTHREAD_CANCEL_ENABLE, nullptr);
```

Writer (writer.hpp)：實現writer裡的兩個method，start以及process，寫法幾乎參照reader.hpp，除了下述區域更新(將output_queue的dequeue元素印出來)。

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

main.cpp：寫法參照producer_test.cpp，將一開始需要的thread引入，利用reader->join()、writer->join()等待reader及writer thread結束才解構釋放所有的thread。

```
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;
    TSQueue<Item *> *worker_queue;
    TSQueue<Item *> *writer_queue;
    Transformer *transformer;
    ConsumerController *consumerController;

    input_queue = new TSQueue<Item *>;
    worker_queue = new TSQueue<Item *>;
    writer_queue = new TSQueue<Item *>;
    transformer = new Transformer;
    consumerController = new ConsumerController(worker_queue, writer_queue, transformer, CONSUMER_CONTROLLER_CHECK_PERIOD, CON
```

```
    Reader *reader = new Reader(n, input_file_name, input_queue);
    Writer *writer = new Writer(n, output_file_name, writer_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();
    writer->start();
    consumerController->start();

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

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

```
    delete p4;
    delete p3;
    delete p2;
    delete p1;
    delete writer;
    delete reader;
    delete input_queue;
    delete worker_queue;
    delete writer_queue;
    delete consumerController;
    delete transformer;
    return 0;
```

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

1. 將CONSUMER_CONTROLLER_CHECK_PERIOD從1000000改成1000，發現增減consumer的頻率增加，結果如下：

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

變成

```
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
```

2. CONSUMER_CONTROLLER_LOW_THRESHOLD_PERCENTAGE從20改成5，發現要增加consumer變難，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 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
```

變成

```
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
```

CONSUMER_CONTROLLER_HIGH_THRESHOLD_PERCENTAGE同理，若增加則使減少consumer變難。

3. WORKER_QUEUE_SIZE從200改成10，發現consumers裡consumer上升的幅度變小。
4. What happens if WRITER_QUEUE_SIZE is very small?執行時間變相當慢。
5. What happens if READER_QUEUE_SIZE is very small?執行時間變相當慢。

What difficulties did you encounter when implementing this assignment?

1. 要非常了解wait&signal的原理，我一開始在queue的dequeue method實作時我最後是signal dequeue的condition variable，結果一直跑錯，最後發現dequeue完要通知的是enqueue的condition variable。
2. 需要把每一份檔案都看過，就算不是TODO範圍。