Cover Page

Team61

Member List

> 108070039 羅稚涵
> 108070038 簡志宇

Member Contribution

> Trace code 簡志宇、羅稚涵
> Implementation 簡志宇
> Report 羅稚涵
> Experiment 羅稚涵

Implementation

- 1. TSQueue
  - TSQueue()
    首先要初始化mutex，接著是lock mutex讓其他人無法使用，然後初始化condition，兩種情況分別是enqueu與dequeue，最後再把mutex給unlock。

```
template <class T>
TSQueue<T>::TSQueue(int buffer_size) : buffer_size(buffer_size) {
  pthread_mutex_init(&mutex, NULL);

  pthread_mutex_lock(&mutex);
  buffer = new T[buffer_size]();
  pthread_cond_init(&cond_enqueue, NULL);
  pthread_cond_init(&cond_dequeue, NULL);
  size = 0;
  head = 0;
  tail = -1;
  pthread_mutex_unlock(&mutex);
}
```

  - TSQueue<T>::enqueue()
    首先先lock mutex，接著如果buffer是在客滿的狀態，那就先等待，直到有空位後，找到現在queue的尾巴，將要進入queue的item放到尾巴，然後size增加1，最後要signal dequeue該做事並且unlock mutex。

```
template <class T>
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);
  }

  tail = (tail+1) % buffer_size;
  buffer[tail] = item;
  size++;

  pthread_cond_signal(&cond_dequeue);
  pthread_mutex_unlock(&mutex);
}
```

  - TSQueue<T>::dequeue()
    首先先lock mutex，接著如果size內沒有東西，那就先等待，直到有東西進來後，將buffer head設為toRemove，讓head移至下一位，然後size減1，最後要signal enqueue該做事，unlock mutex，回傳toRemove。

```
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 toRemove = buffer[head];
  head = (head+1) % buffer_size;
  size--;

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

  return toRemove;
}
```

  - TSQueue<T>::get_size()
    先將mutex設為lock，就可得知現在狀態下的size，並將其設為toReturn，然後將mutex設回unlock，最後回傳toReturn。

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

```
    int toReturn;

    pthread_mutex_lock(&mutex);
    toReturn = size;
    pthread_mutex_unlock(&mutex);

    return toReturn;
  }
```

- 2. Producer

  在一個無限迴圈裡，如果producer的input_queue裡面有東西，就會將input_queue head的東西取出，並將取出的東西設為toBeTransformed。將toBeTransformed進行transform，最後將其加入worker_queue裡。

```
void* Producer::process(void* arg) {
  Producer *producer = (Producer*)arg;

  while(1){
    if(producer->input_queue->get_size() > 0) {

      Item *toBeTransformed = producer->input_queue->dequeue();
      unsigned long long int transformedValue = producer->transformer->producer_transform(toBeTransformed->opcode, toBeTransformed

      Item *transformedItem = new Item(toBeTransformed->key, transformedValue, toBeTransformed->opcode);
      producer->worker_queue->enqueue(transformedItem);
      delete toBeTransformed;
    }
  }
  return nullptr;
}
```

- 3. Consumer

  在consumer如果consumer的worker_queue裡面有東西，就會將worker_queue head的東西取出，並將取出的東西設為toBeTransformed。將toBeTransformed進行transform，最後就將其加入output_queue裡。

```
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);

    if(consumer->worker_queue->get_size() > 0) {
      Item *toBeTransformed = consumer->worker_queue->dequeue();
      unsigned long long int transformedValue = consumer->transformer->consumer_transform(toBeTransformed->opcode, toBeTransformed

      Item *transformedItem = new Item(toBeTransformed->key, transformedValue, toBeTransformed->opcode);
      consumer->output_queue->enqueue(transformedItem);
      delete toBeTransformed;
    }
    pthread_setcancelstate(PTHREAD_CANCEL_ENABLE, nullptr);
  }
  delete consumer;

  return nullptr;
}
```

- 4. ConsumerContoller

  在無窮迴圈下，如果worker_queue的size小於low_threshold且consumer的size是大於1，會將最尾端的consumer thread給cancel掉，並讓它移出consumer controller，如果worker_queue的size大於high_threshold，會加入新的consumer thread，並讓其開始，並讓它加入consumer controller。

```
void* ConsumerController::process(void* arg) {
  ConsumerController *controller = (ConsumerController*)arg;
  int i = 0;
  while(1){
    sleep(controller->check_period/1000000 + controller->check_period%1000000);

    if(controller->worker_queue->get_size() < controller->low_threshold && controller->consumers.size() > 1)
      Consumer *toPop = controller->consumers[controller->consumers.size()-1];
      toPop->cancel();
```

```
      controller->consumers.pop_back();
      std::cout << "Scaling down consumers from " << controller->consumers.size()+1 << " to " << controller->consumers.size() << '
    }
    else if(controller->worker_queue->get_size() > controller->high_threshold){

      Consumer *new_consumer = new Consumer(controller->worker_queue, controller->writer_queue, controller->transformer);
      new_consumer->start();
      controller->consumers.push_back(new_consumer);
      std::cout << "Scaling up consumers from " << controller->consumers.size() << " to " << controller->consumers.size()+1 << '\r
    }
  }
  return nullptr;
}
```

- 5. Writer

  while迴圈會持續運作直到expected_line等於0，也就是沒有東西可以read的時候會結束，在迴圈內會不停地將writer的output_queue裡的東西取出，最後將item內所含的key, val, opcode輸出給output file。

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

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

- main.cpp

  (1) 設立基本三個queue分別是input_queue、worker_queue、writer_queue。
  (2) 建立reader, writer並初始化，讓它們在thread被terminated可以join。
  (3) 按造2. Process創造出4個process分別是p1, p2, p3, p4，並讓它們start。
  (4) 最後delete掉所有創造過的pointer。

```
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 functio
  TSQueue<Item *> *input_queue;
  TSQueue<Item *> *worker_queue;
  TSQueue<Item *> *writer_queue;

  input_queue = new TSQueue<Item*>(READER_QUEUE_SIZE);
  worker_queue = new TSQueue<Item*>(WORKER_QUEUE_SIZE);
  writer_queue = new TSQueue<Item*>(WRITER_QUEUE_SIZE);

  Transformer *transformer = new Transformer;

  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);

  ConsumerController *controller;
  controller = new ConsumerController(worker_queue, writer_queue, transformer,
                   CONSUMER_CONTROLLER_CHECK_PERIOD,
                   CONSUMER_CONTROLLER_LOW_THRESHOLD_PERCENTAGE,
                   CONSUMER_CONTROLLER_HIGH_THRESHOLD_PERCENTAGE);

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

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

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

  delete p1; delete p2; delete p3; delete p4;

  delete controller; delete writer; delete reader;
```

```
  delete input_queue; delete worker_queue; delete writer_queue;

  return 0;
}
```

Experiment

- Different values of CONSUMER_CONTROLLER_CHECK_PERIOD.

  對照組: CONSUMER_CONTROLLER_CHECK_PERIOD = 1000000
  實驗組: CONSUMER_CONTROLLER_CHECK_PERIOD = 100000
  以’./main 200 ./tests/00.in ./tests/00.out’跑的結果，兩者皆無改變。但是以’./main 4000 ./tests/01.in ./tests/01.out’跑的結果，對照組的scaling up consumer的次數為31與scaling down consumer的次數為20，相較於實驗組的是30與19。

- Different values of CONSUMER_CONTROLLER_LOW_THRESHOLD_PERCENTAGE and CONSUMER_CONTROLLER_HIGH_THRESHOLD_PERCENTAGE.

  對照組: CONSUMER_CONTROLLER_LOW_THRESHOLD = 20
  實驗組: CONSUMER_CONTROLLER_LOW_THRESHOLD = 40
  以’./main 200 ./tests/00.in ./tests/00.out’跑的結果，兩者皆無改變。但是以’./main 4000 ./tests/01.in ./tests/01.out’跑的結果，對照組的scaling up consumer的次數為31與scaling down consumer的次數為20，相較於實驗組的是33與22。

  對照組: CONSUMER_CONTROLLER_HIGH_THRESHOLD = 80
  實驗組: CONSUMER_CONTROLLER_HIGH_THRESHOLD = 90
  以’./main 200 ./tests/00.in ./tests/00.out’跑的結果，兩者皆無改變。但是以’./main 4000 ./tests/01.in ./tests/01.out’跑的結果，對照組的scaling up consumer的次數為31與scaling down consumer的次數為20，相較於實驗組的是31與21。

- Different values of WORKER_QUEUE_SIZE.

  對照組: WORKER_QUEUE_SIZE = 200
  實驗組: WORKER_QUEUE_SIZE = 1000
  以’./main 200 ./tests/00.in ./tests/00.out’跑的結果，兩者皆無改變。以’./main 4000 ./tests/01.in ./tests/01.out’跑的結果，兩者也皆無改變。

- What happens if WRITER_QUEUE_SIZE is very small?

  對照組: WRITER_QUEUE_SIZE = 4000
  實驗組: WRITER_QUEUE_SIZE = 10
  以’./main 200 ./tests/00.in ./tests/00.out’跑的結果，兩者皆無改變。以’./main 4000 ./tests/01.in ./tests/01.out’跑的結果，兩者也皆無改變。

- What happens if READER_QUEUE_SIZE is very small?

對照組: CONSUMER_CONTROLLER_LOW_THRESHOLD = 20
實驗組: CONSUMER_CONTROLLER_LOW_THRESHOLD = 40
以’./main 200 ./tests/00.in ./tests/00.out’跑的結果，兩者皆無改變。但是以’./main 4000 ./tests/01.in ./tests/01.out’跑的結果，對照組的scaling up consumer的次數為31與scaling down consumer的次數為20，相較於實驗組的是31與21。

Difficulties & Feedback

簡志宇

與MP4有一樣的問題，就是時間不夠。

羅稚涵

因為離期末考已經有段時間了，所以還是花了點時間複習mutex與Pthread，這次的作業又跟MP4卡在一起，所以experiments不是做得很完整，應該是要有更多的實驗組的。