Pthreads_Report_58

Team Members & Contributions

- 108062119 鄭幃謙
- 108062133 劉廷宜

Work	Contributions
implementation	鄭幃謙、劉廷宜
debug	鄭幃謙
report	劉廷宜

Implementation

1. TSQueue

- **TSQueue::TSQueue(int buffer_size)**
  初始化class TSQueue中的private members
  buffer是由array實作、配置buffer_size大小

```
size = 0;
head = 0;
tail = 0;
pthread_mutex_init(&mutex, nullptr);
pthread_cond_init(&cond_enqueue, nullptr);
pthread_cond_init(&cond_dequeue, nullptr);
buffer = new T[buffer_size];
```

- **TSQueue::~TSQueue()**
  將配置的東西delete掉

```
delete[] buffer;
pthread_mutex_destroy(&mutex);
pthread_cond_destroy(&cond_enqueue);
pthread_cond_destroy(&cond_dequeue);
```

- **void TSQueue::enqueue(T item)**
  先取得mutex避免其他thread同時進行
  若size == buffer_size
  代表現在buffer是滿的、無法enqueue
  呼叫pthread_cond_wait(&cond_dequeue, &mutex)
  將剛剛取得的mutex釋出並等待cond_dequeue發生

若tail == buffer_size
避免覆蓋到現有的Item，將tail改為0（circular的概念）
將item放入buffer[tail]
更新tail跟size
呼叫pthread_cond_signal(&cond_enqueue)並釋放mutex

```
pthread_mutex_lock(&mutex);
while (size == buffer_size)
    pthread_cond_wait(&cond_dequeue, &mutex);
if (tail == buffer_size) tail = 0;
buffer[tail++] = item;
size++;
pthread_cond_signal(&cond_enqueue);
pthread_mutex_unlock(&mutex);
```

- **T TSQueue::dequeue()**
  先取得mutex避免其他thread同時進行
  若size == 0
  代表現在buffer是空的、無法dequeue
  呼叫pthread_cond_wait(&cond_enqueue, &mutex)
  將剛剛取得的mutex釋出並等待cond_enqueue發生
  若head == buffer_size
  避免覆蓋到現有的Item，將head改為0（circular的概念）
  更新head跟size
  呼叫pthread_cond_signal(&cond_dequeue)並釋放mutex
  return buffer[head]（還沒更新之前的head）

```
pthread_mutex_lock(&mutex);
while (size == 0)
    pthread_cond_wait(&cond_enqueue, &mutex);
if (head == buffer_size) head = 0;
T el = buffer[head];
buffer[head++] = 0;
size--;
pthread_cond_signal(&cond_dequeue);
pthread_mutex_unlock(&mutex);
return el;
```

- **int TSQueue::get_size()**
  回傳size

```
return size;
```

2. Producer

- **void Producer::start()**
  使用pthread_create來create一個Producer thread

```
pthread_create(&t, 0, Producer::process, (void*)this);
```

- **void* Producer::process(void* arg)**
  從input_queue拿出一個Item
  用producer_transform算出新的value
  將新的value創出新的Item後放入worker_queue

```
Producer* producer = (Producer*)arg;
while (1) {
    Item* item = producer->input_queue->dequeue();
    unsigned long long newValue = producer->transformer-
>producer_transform(item->opcode, item->val);
    Item* newItem = new Item(item->key, newValue, item->opcode);
    producer->worker_queue->enqueue(newItem);
}
delete producer;
return nullptr;
```

3. Consumer

- **void Consumer::start()**
  使用pthread_create來create一個Consumer thread

```
pthread_create(&t, 0, Consumer::process, (void*)this);
```

- **int Consumer::cancel()**
  將is_cancel設為true
  在Consumer::process()中就會跳出迴圈
  Consumer thread就會被delete

```
this->is_cancel = true;
return 0;
```

- **void* Consumer::process(void* arg)**
  從worker_queue拿出一個Item
  用consumer_transform算出新的value
  將新的value創出新的Item後放入output_queue

```

```

```
Item* item = consumer->worker_queue->dequeue();
unsigned long long newValue = consumer->transformer-
>consumer_transform(item->opcode, item->val);
Item* newItem = new Item(item->key, newValue, item->opcode);
consumer->output_queue->enqueue(newItem);
```

4. ConsumerController

- **void ConsumerController::start()**
  使用pthread_create來create一個ConsumerController thread

```
pthread_create(&t, 0, ConsumerController::process, (void*)this);
```

- **void* ConsumerController::process(void* arg)**
  用clock()算時間，每一個period檢查一次：
  若worker_queue的size超過high_threshold，create一個新的consumer
  呼叫consumer->start()，將新的consumer放進consumers裡面
  若worker_queue的size小於low_threshold
  cancel掉最新的consumer並將其從consumers移除

```
while (1) {
    if (clock() % consumerController->check_period == 0) {
        if (consumerController->worker_queue->get_size() >
consumerController->high_threshold) {
            Consumer* newConsumer = new Consumer(consumerController-
>worker_queue, consumerController->writer_queue, consumerController-
>transformer);
            newConsumer->start();
            consumerController->consumers.push_back(newConsumer);
            std::cout << "Scaling up consumers from " <<
consumerController->consumers.size() - 1 << " to " <<
consumerController->consumers.size() << std::endl;
        } else if (consumerController->worker_queue->get_size() <
consumerController->low_threshold) {
            if (consumerController->consumers.size() > 1) {
                consumerController->consumers.back()->cancel();
                consumerController->consumers.pop_back();
                std::cout << "Scaling down consumers from " <<
consumerController->consumers.size() + 1 << " to " <<
consumerController->consumers.size() << std::endl;
            }
        }
    }
}
```

5. Writer

- **void Writer::start()**
  使用pthread_create來create一個Writer thread

```
pthread_create(&t, 0, Writer::process, (void*)this);
```

- **void* Writer::process(void* arg)**
  將output_queue的Item輸出

```
while (writer->expected_lines--) {
    Item* item = writer->output_queue->dequeue();
    writer->ofs << item->key << " " << item->val << " " << item-
>opcode << std::endl;
}
```

6. main.cpp

- 宣告所需的東西

```
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);
Transformer* transformer = new Transformer();
Reader* reader = new Reader(n, input_file_name, input_queue);
Writer* writer = new Writer(n, output_file_name, output_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* consumerController = new
ConsumerController(worker_queue, output_queue, transformer,
CONSUMER_CONTROLLER_CHECK_PERIOD,
CONSUMER_CONTROLLER_LOW_THRESHOLD_PERCENTAGE,
CONSUMER_CONTROLLER_HIGH_THRESHOLD_PERCENTAGE);
int lt =
WORKER_QUEUE_SIZE*CONSUMER_CONTROLLER_LOW_THRESHOLD_PERCENTAGE/100;
int ht =
WORKER_QUEUE_SIZE*CONSUMER_CONTROLLER_HIGH_THRESHOLD_PERCENTAGE/100;
```

- start threads until writer thread ends

```
reader->start();
p1->start();
p2->start();
p3->start();
p4->start();
consumerController->start();
```

```
writer->start();
writer->join();
delete writer;
delete reader;
delete p1;
delete p2;
delete p3;
delete p4;
delete consumerController;
```

Experiment

1. Different values of CONSUMER_CONTROLLER_CHECK_PERIOD

- 小：CONSUMER_CONTROLLER_CHECK_PERIOD較小所以檢查worker_queue的頻率高，所以新增或減少consumer的頻率都較高。這樣會造成較短的時間產生相對多的consumer，所以會比較快寫完。
- 大：CONSUMER_CONTROLLER_CHECK_PERIOD較大所以檢查worker_queue的頻率低，對buffer_size的變化不易察覺。新增或取消consumer的頻率降低，scale的數量變少。

2. Different values of CONSUMER_CONTROLLER_LOW_THRESHOLD and CONSUMER_CONTROLLER_HIGH_THRESHOLD

- **CONSUMER_CONTROLLER_LOW_THRESHOLD變小**
  判斷buffer為空的標準降低，consumer減少的頻率下降，留著的consumer數較多。
- **CONSUMER_CONTROLLER_LOW_THRESHOLD變大**
  判斷buffer為空的標準變高，consumer減少的頻率上升，留著的consumer數較少。
- **CONSUMER_CONTROLLER_HIGH_THRESHOLD變小**
  判斷buffer為滿的標準降低，consumer新增的頻率上升，增加的consumer數變多。
- **CONSUMER_CONTROLLER_HIGH_THRESHOLD變大**
  判斷buffer為滿的標準變高，consumer新增的頻率下降，增加的consumer數變少。

3. Different values of WORKER_QUEUE_SIZE

- 小：很容易就滿導致consumer_controller較常增加consumer。
- 大：不容易變滿所以consumer_controller不常增加consumer。

4. What happens if WRITER_QUEUE_SIZE is very small?

若WRITER_QUEUE_SIZE很小，代表writer_queue很容易滿。
consumer要等到writer_queue有空位才能將Item放進去，但因為writer_queue很容易滿所以要consumer等到writer_queue有空位。
如此會造成consumer處理worker_queue中Item的速度連帶變慢，所以worker_queue也較容易變滿。
consumer_controller偵測worker_queue變滿後會增加consumer，所以consumer的數量比較起來較多。

5. What happens if READER_QUEUE_SIZE is very small?

若READER_QUEUE_SIZE很小，代表reader_queue很容易滿。
reader要等到reader_queue有空位才能將Item放進去，但因為reader_queue很容易滿 所以reader放Item進去

reader_queue的速度下降。
如此會造成producer將Item放入worker_queue的速度下降，worker_queue超過CONSUMER_CONTROLLER_HIGH_THRESHOLD的機會下降。
所以consumer_controller不太會增加consumer的數量，consumer的數量比較起來較少。

Difficulties

- 鄭幃謙
  寫main function的時候想了一陣子，其他沒有太多問題．
- 劉廷宜
  在寫TSQueue的時候又把第六章拿出來看
  對lock的機制本來忘記了複習之後又想起來
  本來想用queue實作TSqueue結果發現不行
  之後才改成用array（才發現head、tail、size都要使用）
  還有在main.cpp等待writer thread結束時的實作也是跟隊友討論了一下才寫出來