Operating System

Pthreads Programming Assignment:

Producer-Consumer Problem

Team 73

Experiment & Implementation: 1121036s 郭逸洪

Contents

Part I - Implementation

1. ts_queue.hpp

TSQueue建構式中初始化queue所需的buffer_size、buffer、size、head和tail屬性與mutex和conditional variable，解構式則將配置的buffer移除、清理mutex和conditional variable。

```
template <class T>
TSQueue<T>::TSQueue(int buffer_size) : buffer_size(buffer_size), buffer(new
T[buffer_size]), size(0), head(0), tail(buffer_size - 1)
{
    pthread_mutex_init(&mutex, nullptr);
    pthread_cond_init(&cond_enqueue, nullptr);
    pthread_cond_init(&cond_dequeue, nullptr);
}

template <class T>
TSQueue<T>::~TSQueue()
{
    delete[] buffer;
    pthread_cond_destroy(&cond_enqueue);
    pthread_cond_destroy(&cond_dequeue);
    pthread_mutex_destroy(&mutex);
}
```

TSQueue enqueue將item放入自身的buffer中，因為TSQueue的操作必須是thread safe的，故對queue相關屬性的修改，都屬於critical section，enqueue必須整個由mutex lock保護，取得鎖後，判斷queue是否已滿，如果已滿，使用conditional variable cond_enqueue等待至queue有空位時，若有空位，將item放入buffer、維護head、tail和size屬性，並通知conditional variable cond_dequeue表示有item在queue中，可以離開pthread_cond_wait。

TSQueue dequeue的操作和enqueue十分類似，先檢查queue是否有item，如果沒有，使用cond_dequeue等待至queue有item時，否則將item從buffer中取出、維護head、tail和size屬性，並通知cond_enqueue表示queue有空位，可以離開pthread_cond_wait，除此之外，由於要在critical section之外return取得的item，在一開始先宣告res變數以儲存critical section中拿到的結果。

TSQueue get_size直接回傳size屬性即可，但TSQueue可能被多個執行緒同時操作，故此結果只是暫態[1]，只適合用於log或粗略評估，比如ConsumerController使用get_size估計worker queue的大小，不適合用於控制程式，因此調整get_size方法的註解。

```
// A thread safe queue that allows multi-thread access.
template <class T>
class TSQueue
{
public:
    // Return the number of elements in the queue. The result is useful only
when this queue not undergoing concurrent updates
    // in other threads. Otherwise the result is just a transient state that
may be adequate for monitoring or estimation purposes,
    // but not for program control.
    int get_size();
    // ......
};

template <class T>
void TSQueue<T>::enqueue(T item)
{
    pthread_mutex_lock(&mutex);
    while (size == buffer_size) // the queue is full, wait until dequeue
    {
        pthread_cond_wait(&cond_enqueue, &mutex);
    }
    buffer[head] = item;
    head = (head + 1) % buffer_size;
    ++size;
    pthread_cond_signal(&cond_dequeue); // we have item(s) in queue, notify
dequeue
    pthread_mutex_unlock(&mutex);
}

template <class T>
T TSQueue<T>::dequeue()
{
    T res;
```

[1] 註解部分的文字說明參考 https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentHashMap.html

```
    pthread_mutex_lock(&mutex);
    while (!size) // the queue is empty, wait until enqueue
    {
        pthread_cond_wait(&cond_dequeue, &mutex);
    }
    tail = (tail + 1) % buffer_size;
    res = buffer[tail];
    --size;
    pthread_cond_signal(&cond_enqueue); // we have slot(s), notify enqueue
    pthread_mutex_unlock(&mutex);
    return res;
}

template <class T>
int TSQueue<T>::get_size()
{
    return size;
}
```

2. writer.hpp

Writer::start將指向自身pthread_t t的指標、指向static方法Writer::process的指標、指向自身實例的指標傳入pthread_create，檢查pthread_create回傳結果為0（成功）。

static方法Writer::process則將傳入的參數轉回對writer實例的指標，對此實例進行操作，在迴圈中判斷該writer實例剩餘需處理的資料行數，如果還有需要處理的資料，直接從該writer的output_queue中取出item並輸出到檔案中，並刪除item，此處直接調用dequeue取得item，因為reader和writer都根據資料行數進行同樣次數的操作，不需要擔心writer對output_queue進行dequeue會永遠卡住，當所有操作結束後，回傳nullptr以滿足回傳型別的定義。

```
void Writer::start()
{
    // pass this, so the thread can access the same Writer instance
    assert(!pthread_create(&t, nullptr, &Writer::process, static_cast<void
*>(this)));
}

void *Writer::process(void *arg)
```

```
{
    // this is a static method, we have to specify the Writer instance
    Writer *writer = static_cast<Writer *>(arg);
    while (writer->expected_lines--) // end after writing all lines
    {
        Item *item = writer->output_queue->dequeue();
        writer->ofs << *item;
        delete item;
    }
    return nullptr;
}
```

3. producer.hpp

Producer::start和Writer::start實作類似，在此不多贅述。Producer::process取得對應的producer實例後，在無窮迴圈中，從該producer實例的input_queue取出item指標，傳入transformer進行計算，並將計算的結果寫回item的val，完成後將該item的指標放入該producer實例的worker_queue當中，因後續的writer在完成寫檔操作後會釋放item配置的heap，此處不需要刪除item，最後回傳nullptr以滿足回傳型別的定義。

```
void Producer::start()
{
    assert(!pthread_create(&t, nullptr, &Producer::process, static_cast<void
*>(this)));
}

void *Producer::process(void *arg)
{
    Producer *producer = static_cast<Producer *>(arg);
    while (true)
    {
        Item *item = producer->input_queue->dequeue();
        item->val = producer->transformer->producer_transform(item->opcode,
item->val);
        producer->worker_queue->enqueue(item);
    }
    return nullptr;
}
```

4. consumer.hpp

Consumer::start和Writer::start實作類似，在此不多贅述。Consumer::cancel將is_cancel旗標設置為true，以結束Consumer::process中的無窮迴圈，最後調用父類別的方法Thread::cancel（其中使用pthread_cancel），並回傳其結果，pthread_cancel會發出取消請求，若執行緒的cancel state為disable，會將取消請求放在queue中直到cancel state被enable，而因為這邊將cancel type設為deferred，該執行緒直到下一個取消點才會被結束[2]，而取消點列表可參考此處，其中包含之後ConsumerController會對Consumer調用的pthread_join。

Consumer::process中新增的部分則和Producer::process十分類似，從對應的consumer實例的worker_queue中取得item指標，進行計算後，更新該item的val，並將item指標放入output_queue，由於在進入上述操作的區塊前，已將pthread的cancel state設為disable，故上述區塊的操作並不會被pthread_cancel取消，而會在離開該區塊後，將pthread的cancel state重新enable後，該執行緒才會處理取消的請求，在看到is_cancel旗標為true後，離開while迴圈回傳nullptr結束執行緒。

```
void Consumer::start()
{
    assert(!pthread_create(&t, nullptr, &Consumer::process, static_cast<void
*>(this)));
}

int Consumer::cancel()
{
    is_cancel = true;
    return Thread::cancel();
}

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

    pthread_setcanceltype(PTHREAD_CANCEL_DEFERRED, nullptr);

    while (!consumer->is_cancel)
```

[2] https://man7.org/linux/man-pages/man3/pthread_cancel.3.html

```
    {
        pthread_setcancelstate(PTHREAD_CANCEL_DISABLE, nullptr);

        Item *item = consumer->worker_queue->dequeue();
        item->val = consumer->transformer->consumer_transform(item->opcode,
item->val);
        consumer->output_queue->enqueue(item);

        pthread_setcancelstate(PTHREAD_CANCEL_ENABLE, nullptr);
    }

    delete consumer;

    return nullptr;
}
```

5. consumer_controller.hpp

ConsumerController::start和Writer::start實作類似，在此不多贅述。

ConsumerController::process將傳入的參數轉回對ConsumerController實例的指標，以下都對此實例進行操作，在無窮迴圈中，先將暫態的worker_queue大小存在wn變數中，判斷wn大於上限值，新增並啟動一個consumer，如果wn小於下限值且目前有1個以上的consumer，對最後一個consumer調用cancel及join，確保最後一個consumer順利結束後，移除該consumer，進行上述判斷後，用usleep使ConsumerController實例等待至少一個周期的時間[3]再進行下一次檢查。

```
void ConsumerController::start()
{
    assert(!pthread_create(&t, nullptr, &ConsumerController::process,
static_cast<void *>(this)));
}

void *ConsumerController::process(void *arg)
{
    ConsumerController *cc = static_cast<ConsumerController *>(arg);
    while (true)
    {
        // store transient state of worker queue
```

[3] https://man7.org/linux/man-pages/man3/usleep.3.html

```
        int wn = cc->worker_queue->get_size();
        // The variable cn is just for readability. Since the number of
consumers is not going to be changed by anyone
        // except this one and only instance of ConsumerController.
        int cn = cc->consumers.size();
        if (wn > cc->high_threshold)
        {
            cc->consumers.push_back(new Consumer(cc->worker_queue,
cc->writer_queue, cc->transformer));
            cc->consumers.back()->start();
            std::cout << "Scaling up consumers from " << cn << " to " << (cn +
1) << std::endl;
        }
        // make sure there is at least one Consumer until the program ends
        else if (wn < cc->low_threshold && cn > 1)
        {
            cc->consumers.back()->cancel();
            cc->consumers.back()->join(); // wait until finish
            cc->consumers.pop_back();
            std::cout << "Scaling down consumers from " << cn << " to " << (cn
- 1) << std::endl;
        }
        usleep(cc->check_period); // the timing is not very accurate, but
acceptable.
    }
    return nullptr;
}
```

6. main.cpp

新增PRODUCER_NUM定義，在main方法開頭設定各個參數，這些參數可由外部帶入，以覆蓋參數的預設值，如此一來，實驗時就不需要重新編譯執行檔。接下來計算consumer的上下限數量，根據作業的需求初始化reader queue（input queue）、worker queue和writer queue（output queue）、Transformer等物件後，啟動各個執行緒，並等待reader和writer處理完所有資料，移除配置的資源，並結束程式。

```
// ......
#define PRODUCER_NUM 4
```

```
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]);

    int rq_sz = READER_QUEUE_SIZE;
    int wq_sz = WORKER_QUEUE_SIZE;
    int wrq_sz = WRITER_QUEUE_SIZE;
    int cc_lo_p = CONSUMER_CONTROLLER_LOW_THRESHOLD_PERCENTAGE;
    int cc_hi_p = CONSUMER_CONTROLLER_HIGH_THRESHOLD_PERCENTAGE;
    int cc_period = CONSUMER_CONTROLLER_CHECK_PERIOD;
    int prod_num = PRODUCER_NUM;
    if (argc >= 11)
    {
        rq_sz = atoi(argv[4]);
        wq_sz = atoi(argv[5]);
        wrq_sz = atoi(argv[6]);
        cc_lo_p = atoi(argv[7]);
        cc_hi_p = atoi(argv[8]);
        cc_period = atoi(argv[9]);
        prod_num = atoi(argv[10]);
    }
    assert(rq_sz >= 1);
    assert(wq_sz >= 1);
    assert(wrq_sz >= 1);
    assert(cc_lo_p >= 0 && cc_lo_p <= 100);
    assert(cc_hi_p >= 0 && cc_hi_p <= 100);
    assert(cc_period >= 1);
    assert(prod_num >= 1);

    int consumer_lo_num = wq_sz * cc_lo_p / 100;
    int consumer_hi_num = wq_sz * cc_hi_p / 100;

    TSQueue<Item *> reader_q = TSQueue<Item *>(rq_sz);
    TSQueue<Item *> worker_q = TSQueue<Item *>(wq_sz);
    TSQueue<Item *> writer_q = TSQueue<Item *>(wrq_sz);
    Transformer transformer = Transformer();

    Reader *reader = new Reader(n, input_file_name, &reader_q);
```

```
    Writer *writer = new Writer(n, output_file_name, &writer_q);
    ConsumerController cc = ConsumerController(&worker_q, &writer_q,
&transformer, cc_period, consumer_lo_num, consumer_hi_num);
    std::vector<Producer> producers(prod_num, Producer(&reader_q, &worker_q,
&transformer));
    reader->start();
    writer->start();
    cc.start();
    for (auto &producer : producers)
    {
        producer.start();
    }

    // finish after reader and writer process every line
    reader->join();
    writer->join();
    delete writer;
    delete reader;
    return 0;
}
```

7. thread.hpp

為了統一檢查執行緒的join和cancel結果，修改父類別Thread的join和cancel方法，如果join和cancel的結果不為0，印出錯誤訊息。

```
int Thread::join()
{
    int res = pthread_join(t, 0);
    if (res)
    {
        std::cerr << "pthread_join error: " << res << std::endl;
    }
    return res;
}

int Thread::cancel()
{
    int res = pthread_cancel(t);
    if (res)
    {
        std::cerr << "pthread_cancel error: " << res << std::endl;
```

```
    }
    return res;
}
```

8. build_pthread

為了方便編譯和除錯，新增以下腳本幫助自動化。

```
#!/bin/sh
if [ "$#" -lt 1 ]; then
  echo "Usage: $0 <spec_number> [cxxflags_extra]"
  exit 1
fi

spec_number=$1
cxxflags_extra="-static -std=c++11 -O3 "
if [ "$#" -ge 2 ]; then
  cxxflags_extra+=$2
fi

# 0. clean
make clean

# 1. Generate transformer.cpp before rebuild
spec_file="./tests/${spec_number}_spec.json"
# ./scripts/auto_gen_transformer --input ./tests/00_spec.json --output
transformer.cpp
# ./scripts/auto_gen_transformer --input ./tests/01_spec.json --output
transformer.cpp
./scripts/auto_gen_transformer --input "$spec_file" --output transformer.cpp

# 2. Build
scl enable devtoolset-8 "make CXXFLAGS='${cxxflags_extra}'"
```

9. run_pthread

為了方便實驗進行與驗證，新增以下腳本幫助自動化。

```
#!/bin/sh
if [ "$#" -lt 3 ]; then
  echo "Usage: $0 <spec_number> <test_case_lines> <verify_flag>
[reader_q_size] [worker_q_size] [writer_q_size] [cc_lo_p] [cc_hi_p]
```

```
[cc_period] [producer_num]"
  exit 1
fi

spec_number=$1
test_case_lines=$2
verify_flag=$3
input_file="./tests/${spec_number}.in"
output_file="./tests/${spec_number}.out"
answer_file="./tests/${spec_number}.ans"

# 3. Run
if [ "$#" -ge 10 ]; then
  ./main "$test_case_lines" "$input_file" "$output_file" "$4" "$5" "$6" "$7"
"$8" "$9" "${10}"
else
  # ./main 200 ./tests/00.in ./tests/00.out
  # ./main 4000 ./tests/01.in ./tests/01.out
  ./main "$test_case_lines" "$input_file" "$output_file"
fi

# 4. Test
if [ "$verify_flag" -ne "0" ]; then
  # ./scripts/verify --output ./tests/00.out --answer ./tests/00.ans
  # ./scripts/verify --output ./tests/01.out --answer ./tests/01.ans
  ./scripts/verify --output "$output_file" --answer "$answer_file"
fi
```

Part II - Experiments

1. Experimental environment

所有實驗都在學校提供的伺服器上運行，使用lscpu指令顯示的CPU規格如下：

```
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                96
On-line CPU(s) list:   0-95
Thread(s) per core:    1
Core(s) per socket:    24
Socket(s):             4
NUMA node(s):          1
Vendor ID:             AuthenticAMD
CPU family:            23
Model:                 49
Model name:            AMD EPYC 7352 24-Core Processor
Stepping:              0
CPU MHz:               2295.686
BogoMIPS:              4591.37
Virtualization:        AMD-V
Hypervisor vendor:     KVM
Virtualization type:   full
L1d cache:             64K
L1i cache:             64K
L2 cache:              512K
L3 cache:              16384K
NUMA node0 CPU(s):     0-95
Flags:                 fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge
mca cmov pat pse36 clflush mmx fxsr sse sse2 ht syscall nx mmxext fxsr_opt
pdpe1gb rdtscp lm art rep_good nopl extd_apicid eagerfpu pni pclmulqdq ssse3
fma cx16 sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx
f16c rdrand hypervisor lahf_lm cmp_legacy svm cr8_legacy abm sse4a misalignsse
3dnowprefetch osvw perfctr_core ssbd rsb_ctxsw ibrs ibpb stibp vmmcall
fsgsbase tsc_adjust bmi1 avx2 smep bmi2 rdseed adx smap clflushopt clwb sha_ni
xsaveopt xsavec xgetbv1 retpoline_amd clzero xsaveerptr arat npt nrip_save
umip spec_ctrl intel_stibp arch_capabilities
```

使用hostnamectl指令顯示的OS規格如下：

```
Virtualization: kvm
Operating System: CentOS Linux 7 (Core)
CPE OS Name: cpe:/o:centos:centos:7
Kernel: Linux 3.10.0-1160.80.1.el7.x86_64
Architecture: x86-64
```

2. Single variable experiments

以下表格是各個參數的預設值，本區塊的實驗每次只會有一項參數與預設值不同，並且每一項實驗都使用4000行的tests/01.in和tests/01_spec.json，原始實驗數據附在報告的最後一部分。

參數名稱(簡稱)	預設值
READER_QUEUE_SIZE(reader_q_size)	200
WORKER_QUEUE_SIZE(worker_q_size)	200
WRITER_QUEUE_SIZE(writer_q_size)	4000
CONSUMER_CONTROLLER_LOW_THRESHOLD_PERCENTAGE(cc_lo_p)	20
CONSUMER_CONTROLLER_HIGH_THRESHOLD_PERCENTAGE(cc_hi_p)	80
CONSUMER_CONTROLLER_CHECK_PERIOD(cc_period)	1000000
PRODUCER_NUM(producer_num)	4

a. READER_QUEUE_SIZE

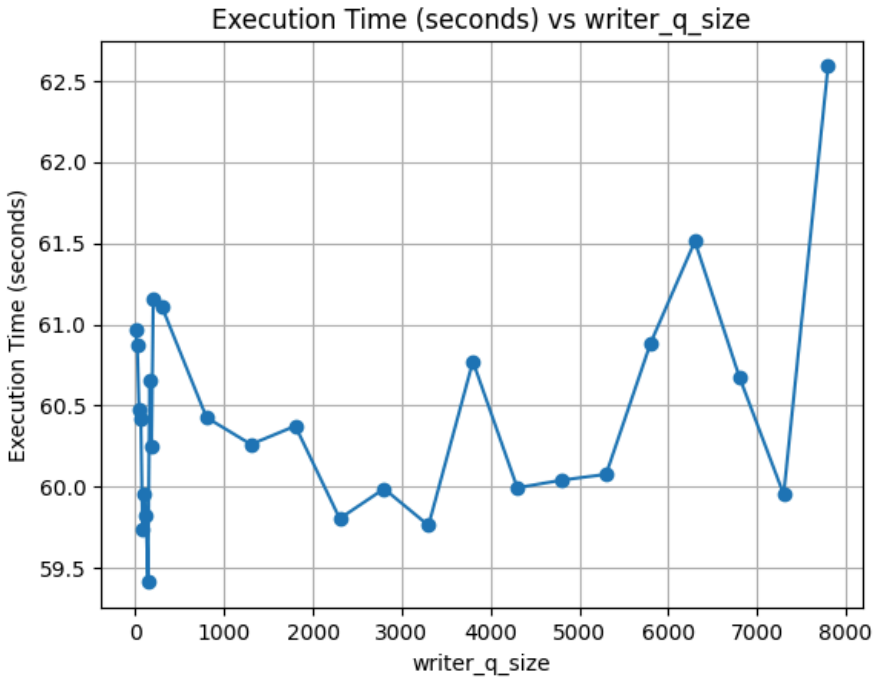
READER_QUEUE_SIZE預設為200，但下圖的實驗結果顯示READER_QUEUE_SIZE從1到4001，程式的執行時間都在59至64秒之間，且沒有明顯的變化趨勢，顯然READER_QUEUE_SIZE在其他參數都不變的狀況下，並非程式的效能瓶頸。

b. WORKER_QUEUE_SIZE

下圖實驗結果顯示，worker queue的容量和執行時間之間沒有明顯的關聯，可能和ConsumerController使用worker queue的容量百分比做為門檻，以調節consumer的數量有關，當worker queue的容量較小時，容易超過80%或低於20%的容量門檻，而調節consumer數量，而worker queue的容量增加時，就不容易碰觸到門檻值，對consumer數量的調節粒度變粗，可能因此導致不穩定的執行時間。

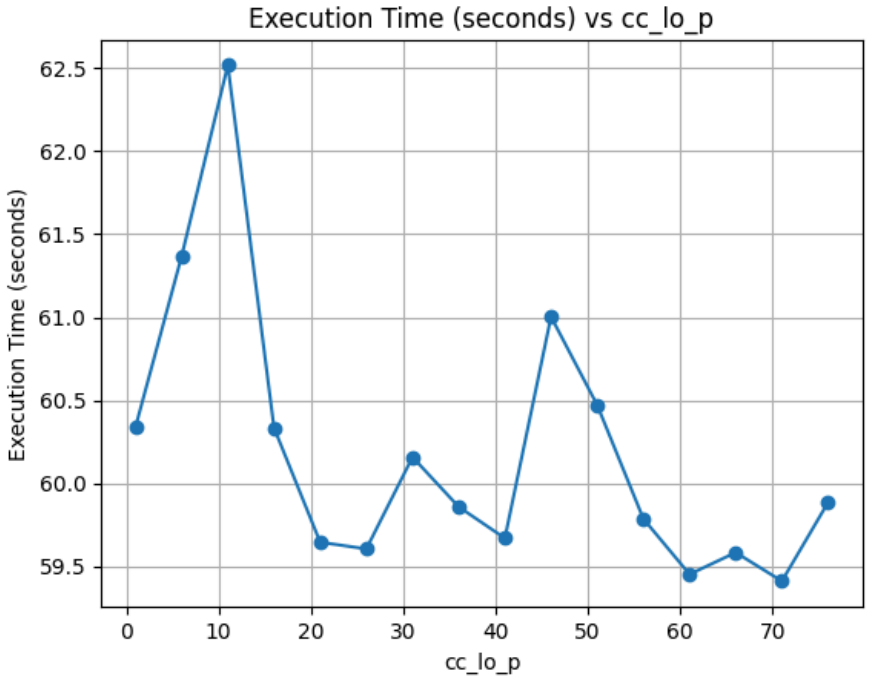
c. WRITER_QUEUE_SIZE

和READER_QUEUE_SIZE的單一變數實驗類似，WRITER_QUEUE_SIZE的變化對程式執行時間沒有顯著影響，WRITER_QUEUE_SIZE從1至7801，程式執行時間都落在59至63秒之間，WRITER_QUEUE_SIZE在其他參數都不變的狀況下，也非效能瓶頸。



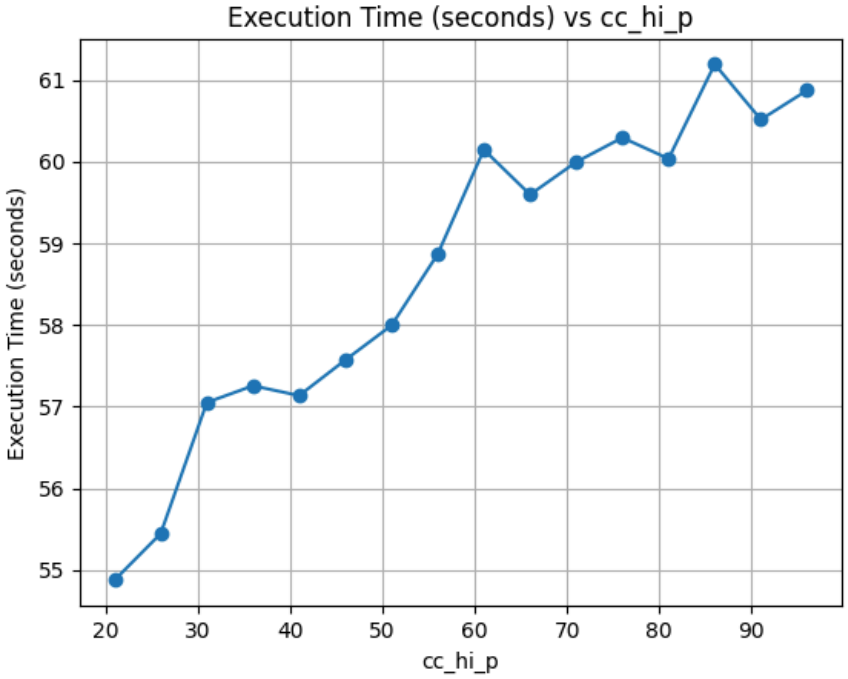
d. CONSUMER_CONTROLLER_LOW_THRESHOLD_PERCENTAGE

下圖實驗結果顯示，cc_lo_p從1至76（不超過固定的cc_hi_p 80），程式執行時間都在59至63秒之間，故cc_lo_p在其他參數不變的狀況下，也非效能瓶頸。



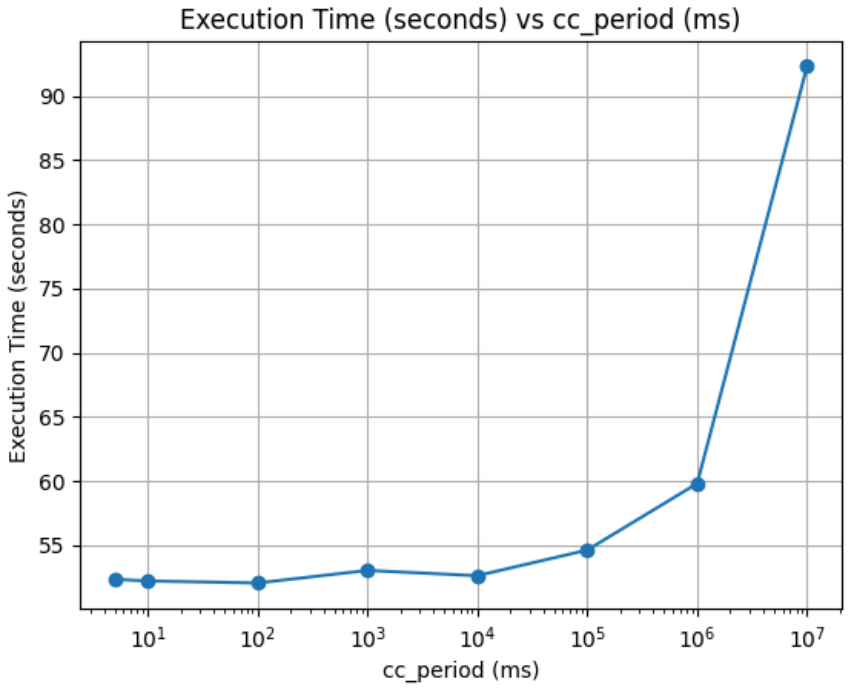
e. CONSUMER_CONTROLLER_HIGH_THRESHOLD_PERCENTAGE

cc_hi_p和程式執行時間正相關，當cc_hi_p的值越高，ConsumerController越不容易創建新的Consumer，故程式執行時間延長。



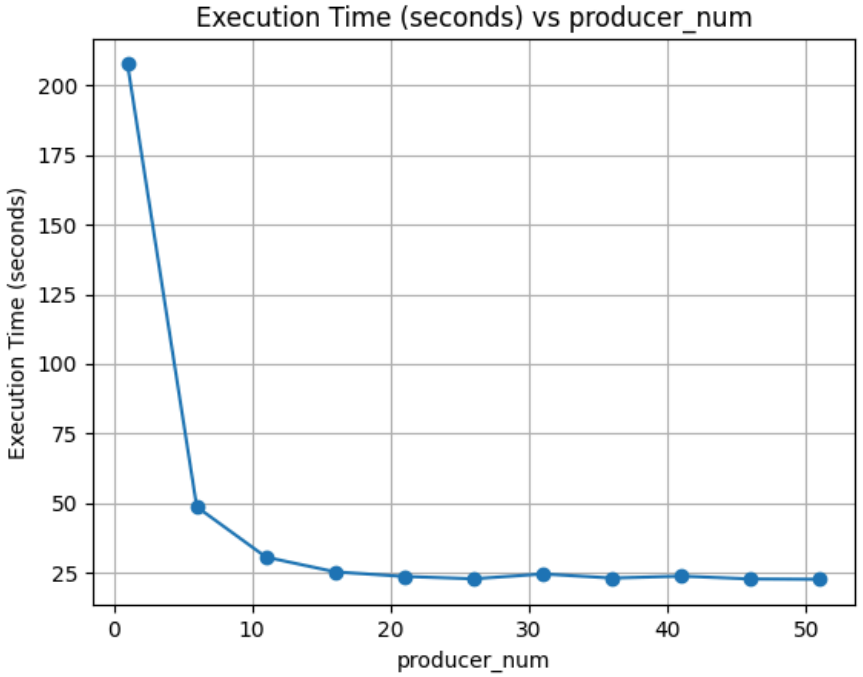
f. CONSUMER_CONTROLLER_CHECK_PERIOD

ConsumerController的檢查週期越長，執行時間越久，推測是ConsumerController對consumer數量的檢查週期過長，導致來不及分配適當數量的consumer。



g. PRODUCER_NUM

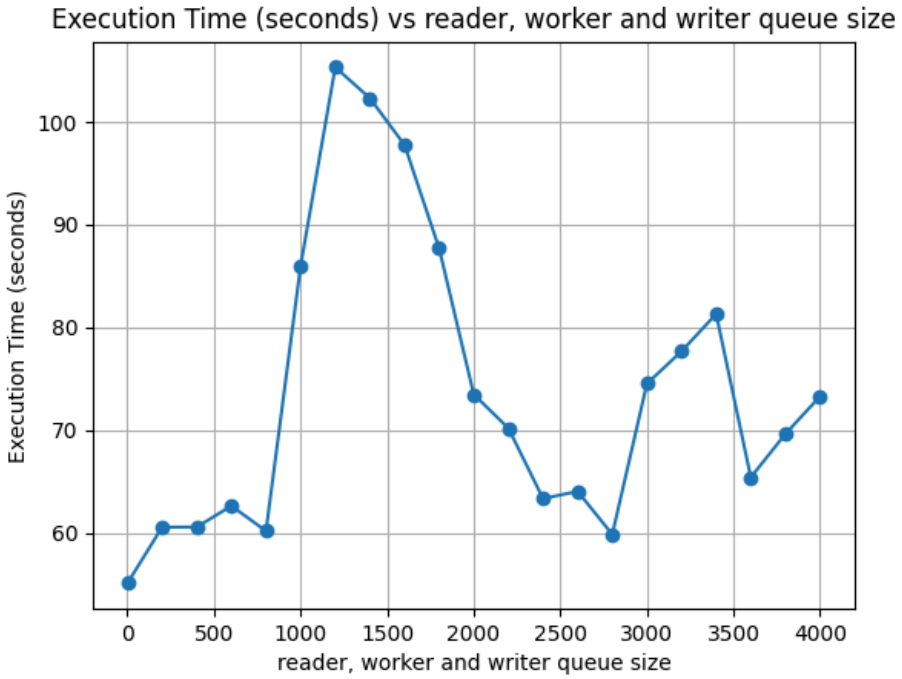
producer數量的增加，能夠顯著加速執行，但producer數量增加到26左右，便沒有顯著的效能提升，推測當producer的數量到21之後，對input_queue的dequeue和worker_queue enqueue的速度就已達上限，繼續增加producer也只會增加對兩個queue的mutex lock的競爭，故無法提升速度。



3. Multivariate experiment

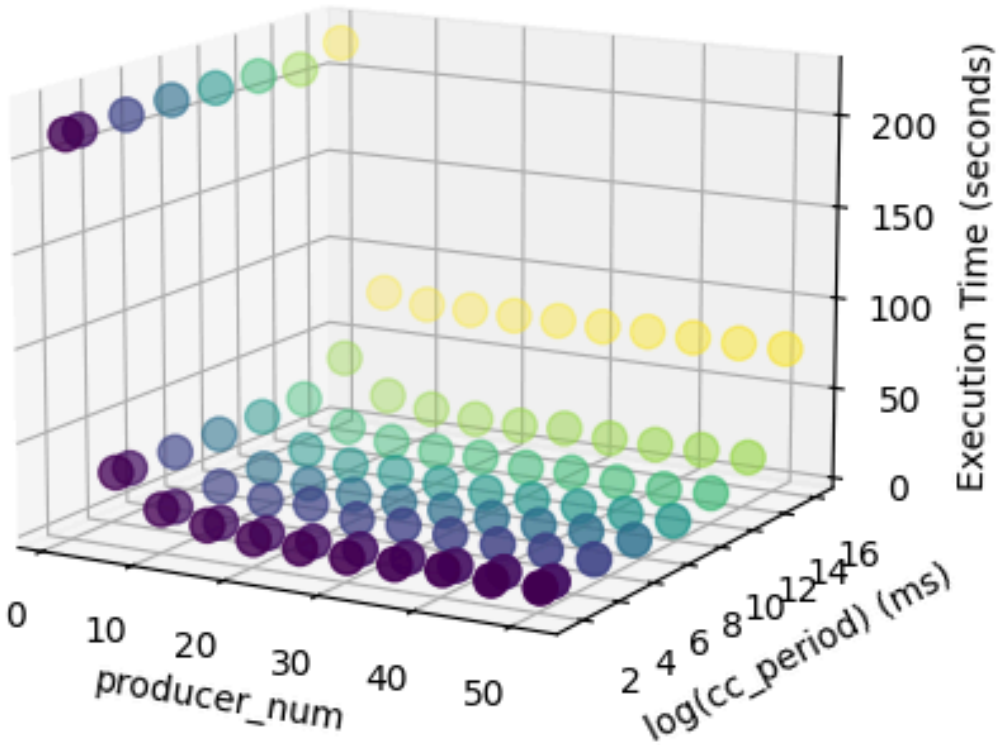
a. Change the size of reader, worker and writer queues

當三個queue的大小一起從1變化到4001，程式執行時間和單一變數的WORKER_QUEUE_SIZE實驗非常類似，可能是因為另外兩個queue的大小對程式執行效能影響不大（參見單一變數實驗READER_QUEUE_SIZE和WRITER_QUEUE_SIZE），所以整體結果由worker queue的大小控制。



b. Change check period and producer number

由下圖可以看見不論cc_period如何變化，當producer_num成長到20左右，繼續增加producer並不會顯著提升程式執行效能，而檢查週期越短，程式執行越快，但對效能的提升微小，只要檢查週期不大於10^5毫秒，都有機會在13秒內執行完程式。


Execution Time (seconds)
200
150
100
50
0
producer_num
0
10
20
30
40
50
log(cc_period) (ms)
2
4
6
8
10
12
14
16

Part III - Feedback

本次作業大幅提升了我對C/C++非同步程式的認識，這也是我第一次不使用Thread pool和BlockingQueue等程式語言內建的library，自行手動管理執行緒，以前工作時使用Java語言，Java原生就有非常多Thread safe的容器和API可使用（位於java.util.concurrent package之下），故撰寫起來輕鬆許多，然而若對非同步程式有錯誤的理解，仍然會經常出錯，當時從Java Concurrency in Practice這本書學到了非常多錯誤和正確的使用案例，應該也要找時間好好讀一下C/C++ concurrency相關的書，以提升對C/C++語言的熟悉度。

Part IV - Experimental Data

1. Single variable experiments

a. READER_QUEUE_SIZE

```
1,60.70144534111023
11,59.69263672828674
21,60.29948878288269
31,59.90729355812073
41,60.23296856880188
51,59.834328413009644
61,59.753705739974976
71,61.2445764541626
81,60.57598686218262
91,59.93975353240967
101,61.17088270187378
111,59.575968503952026
121,60.388336420059204
131,60.748433351516724
141,61.64307618141174
151,59.96585512161255
161,60.145029067993164
171,61.04209327697754
181,59.75463318824768
191,60.79671287536621
201,60.637290477752686
301,60.059826374053955
401,60.25691890716553
501,62.42013072967529
601,60.581968545913696
701,60.35468101501465
801,59.425143241882324
901,60.87365651130676
1001,59.82361960411072
1101,60.14984178543091
1201,61.73697471618652
1301,59.97431778907776
1401,60.609110593795776
1501,59.83401894569397
1601,60.492770195007324
1701,59.56396794319153
1801,59.80483627319336
1901,59.43327474594116
```

```
2001,59.80438756942749
2101,60.59100270271301
2201,61.112802028656006
2301,59.599671363830566
2401,59.708245038986206
2501,61.01647996902466
2601,60.01897668838501
2701,60.11245918273926
2801,60.04547119140625
2901,59.6883487701416
3001,59.95584273338318
3101,63.38290977478027
3201,59.69222545623779
3301,59.62093210220337
3401,61.203227043151855
3501,61.16637396812439
3601,59.95951008796692
3701,61.4580614566803
3801,59.60620403289795
3901,59.870928049087524
4001,59.87256073951721
```

b. WORKER_QUEUE_SIZE

```
1,55.372225761413574
101,56.727333545684814
201,59.69884705543518
301,62.04710340499878
401,61.429200410842896
501,63.09409785270691
601,65.30653381347656
701,64.7049458026886
801,59.96150803565979
901,74.29550504684448
1001,106.63313388824463
1101,106.43474102020264
1201,105.47868371009827
1301,103.4000391960144
1401,100.0694625377655
1501,100.76804399490356
1601,99.12955284118652
1701,93.59546399116516
1801,84.90589761734009
1901,79.27387285232544
2001,66.00341629981995
2101,62.034034967422485
```

```
2201,64.15882015228271
2301,63.241421699523926
2401,61.79388451576233
2501,57.088655948638916
2601,56.619147300720215
2701,55.69988822937012
2801,63.38900017738342
2901,64.61102676391602
3001,83.23207902908325
3101,76.49733543395996
3201,77.7299964427948
3301,92.74411153793335
3401,87.4010214805603
3501,67.25694489479065
3601,65.8187165260315
3701,67.8934736251831
3801,69.67498302459717
3901,71.21392464637756
4001,83.56456708908081
```

c. WRITER_QUEUE_SIZE

```
1,60.971158027648926
21,60.869529485702515
41,60.47599005699158
61,60.414796590805054
81,59.735912561416626
101,59.95626974105835
121,59.825368881225586
141,59.412379026412964
161,60.6598334312439
181,60.2471079826355
201,61.15518856048584
301,61.105530977249146
801,60.42672634124756
1301,60.261250734329224
1801,60.37441611289978
2301,59.803945541381836
2801,59.98772692680359
3301,59.761706590652466
3801,60.772329568862915
4301,59.99367332458496
4801,60.04078793525696
5301,60.07665395736694
5801,60.8837251663208
6301,61.51248121261597
```

```
6801,60.6781690120697
7301,59.953192710876465
7801,62.59109044075012
```

d. CONSUMER_CONTROLLER_LOW_THRESHOLD_PERCENTAGE

```
1,60.338173151016235
6,61.3682177066803
11,62.51827621459961
16,60.3336615562439
21,59.6440794467926
26,59.60401773452759
31,60.157126903533936
36,59.857476234436035
41,59.66820168495178
46,61.00286149978638
51,60.47058820724487
56,59.78529667854309
61,59.45094013214111
66,59.582159996032715
71,59.40967130661011
76,59.88393521308899
```

e. CONSUMER_CONTROLLER_HIGH_THRESHOLD_PERCENTAGE

```
21,54.87585186958313
26,55.44567561149597
31,57.0519597530365
36,57.25653028488159
41,57.136635065078735
46,57.57258868217468
51,57.9962899684906
56,58.868574380874634
61,60.14854431152344
66,59.59797239303589
71,59.9972767829895
76,60.2956063747406
81,60.03771901130676
86,61.19504928588867
91,60.517452001571655
96,60.875096559524536
```

f. CONSUMER_CONTROLLER_CHECK_PERIOD

```
5,52.345072507858276
10,52.225080490112305
100,52.0623300075531
```

```
1000,53.04042434692383
10000,52.63102984428406
100000,54.62888550758362
1000000,59.81545376777649
10000000,92.31800246238708
```

g. PRODUCER_NUM

```
1,207.68504333496094
6,48.61853218078613
11,30.556418895721436
16,25.25121521949768
21,23.607808828353882
26,22.767008781433105
31,24.51137113571167
36,23.02871012687683
41,23.730894804000854
46,22.715794563293457
51,22.595319271087646
```

2. Multivariate experiment

a. Change the size of reader, worker and writer queues

```
1,55.104061126708984
201,60.55332636833191
401,60.562265157699585
601,62.626293420791626
801,60.178526401519775
1001,86.01349949836731
1201,105.33485865592957
1401,102.32224678993225
1601,97.77099561691284
1801,87.69971466064453
2001,73.47244620323181
2201,70.18789601325989
2401,63.33400797843933
2601,64.03799200057983
2801,59.842392921447754
3001,74.52493524551392
3201,77.68155217170715
3401,81.30881714820862
3601,65.40086078643799
3801,69.63444137573242
4001,73.29299139976501
```

b. Change check period and producer number

```
5,1,211.37610244750977
5,6,36.14284372329712
5,11,20.122761011123657
5,16,15.37640929222107
5,21,11.857044458389282
5,26,10.820035934448242
5,31,9.946214199066162
5,36,10.29518747329712
5,41,11.27804946899414
5,46,9.753935813903809
5,51,9.769644021987915
10,1,211.39589595794678
10,6,35.50095820426941
10,11,18.530673027038574
10,16,13.777046918869019
10,21,11.884290933609009
10,26,11.389235973358154
10,31,11.075940608978271
10,36,10.426566362380981
10,41,10.245046615600586
10,46,10.851529836654663
10,51,9.907069683074951
100,1,212.06461668014526
100,6,34.09785771369934
100,11,19.078606128692627
100,16,15.47737169265747
100,21,16.843970775604248
100,26,12.545857667922974
100,31,13.11445164680481
100,36,11.854023218154907
100,41,10.531078815460205
100,46,11.158605575561523
100,51,10.371476888656616
1000,1,212.67960476875305
1000,6,34.012110233306885
1000,11,18.557743549346924
1000,16,13.952811002731323
1000,21,11.736845970153809
1000,26,11.30843210220337
1000,31,10.773969650268555
1000,36,10.333141803741455
1000,41,10.11055612564087
1000,46,10.706815719604492
1000,51,9.839981317520142
```

```
10000,1,211.989661693573
10000,6,33.8823139667511
10000,11,18.8542537689209
10000,16,14.052186965942383
10000,21,13.574125051498413
10000,26,11.347289323806763
10000,31,10.059531927108765
10000,36,9.847947597503662
10000,41,9.696576595306396
10000,46,9.483459234237671
10000,51,9.344334125518799
100000,1,211.53939056396484
100000,6,34.20002508163452
100000,11,21.965729236602783
100000,16,17.12404203414917
100000,21,15.207889318466187
100000,26,15.467179536819458
100000,31,13.404388189315796
100000,36,14.336881399154663
100000,41,12.677109956741333
100000,46,12.466734647750854
100000,51,13.949782848358154
1000000,1,208.16437530517578
1000000,6,48.01818895339966
1000000,11,29.907641172409058
1000000,16,25.727740049362183
1000000,21,23.605189561843872
1000000,26,23.343767404556274
1000000,31,25.213424921035767
1000000,36,23.220040798187256
1000000,41,23.231716871261597
1000000,46,23.9090678691864
1000000,51,23.257762908935547
10000000,1,216.2878406047821
10000000,6,76.66684579849243
10000000,11,72.83506226539612
10000000,16,72.87572121620178
10000000,21,72.83477258682251
10000000,26,73.11967539787292
10000000,31,72.92192673683167
10000000,36,73.37932467460632
10000000,41,73.25596904754639
10000000,46,73.35201835632324
10000000,51,73.46554255485535
```