chap4.1.2

threadsafe_queue.hpp
FIFO

lock_guard
- you have to destroy it, can't unlock halfway

unique_lock
- more flexible to deal w/

```
#include <mutex>
#include <condition_variable>
#include <queue>
#include <memory>

template<typename T>
class threadsafe_queue
{
private:
   mutable std::mutex mut;
   std::queue<T> data_queue;
   std::condition_variable data_cond;
public:
   threadsafe_queue()
   {}
   threadsafe_queue(threadsafe_queue const& other)
   {
      std::lock_guard lk(other.mut);
      data_queue=other.data_queue;
   }

   void push(T new_value)
   {
      std::lock_guard lk(mut);
      data_queue.push(new_value);
      data_cond.notify_one();
   }
```

notify thread that is waiting for it

// notify there is s.t in the queque

* Consummer needs to be wait if there is something in the queque
if ready, you might have it right away

```
   void wait_and_pop(T& value)
   {
      std::unique_lock lk(mut);
      data_cond.wait(lk,[this]{return !data_queue.empty();});
      value=data_queue.front();
      data_queue.pop();
   }

   std::shared_ptr<T> wait_and_pop()
   {
      std::unique_lock lk(mut);
      data_cond.wait(lk,[this]{return !data_queue.empty();});
      std::shared_ptr<T> res(std::make_shared<T>(data_queue.front()));
      data_queue.pop();
      return res;
```

func is passed as argument

lamda expression

if true, don't need to wait
not empty

if false, unlock lk
empty

wait until data_queque not empty

wait() in cond_var
while (!_p()){
wait(_lock);

[illegible]
{

```
    }

    bool try_pop(T& value)
    {
        std::lock_guard lk(mut);
        if(data_queue.empty)
            return false;
        value=data_queue.front();
        data_queue.pop();
        return true; // Added by Craig
    }

    std::shared_ptr<T> try_pop()
    {
        std::lock_guard lk(mut);
        if(data_queue.empty())
            return std::shared_ptr<T>();
        auto res(std::make_shared<T>(data_queue.front()));
        data_queue.pop();
        return res;
    }

    bool empty() const
    {
        std::lock_guard lk(mut);
        return data_queue.empty();
    }
};

//int main()
//{}
```

CraigMain.cpp

```
// Adapted by Craig Scratchley from Listing 4.1
#include <thread>
#include <iostream>
#include "threadsafe_queue.hpp"

bool more_data_to_prepare()
{
    return true; // false;
}

struct data_chunk // empty constructor
{};

data_chunk prepare_data()
{
    return data_chunk();
}

void process(data_chunk&)
{}

bool is_last_chunk(data_chunk&)
{
    return false; // true;
}

threadsafe_queue<data_chunk> ts_queue; // renamed from data_queue
```

→ check on debug vars

there are 248 "empty constructors" in a queque

```
void data_preparation_thread()
{
    while(more_data_to_prepare())
    {
        data_chunk const data=prepare_data();
        ts_queue.push(data);
    }
}
```

in push()

by notify_one()

wake up when need to check cond.

↳ save energy

```
void data_processing_thread()
{
    while(true)
    {
        data_chunk data=*ts_queue.wait_and_pop();
        process(data);
        if(is_last_chunk(data))
            break;
    }
}

int main()
{
    std::thread t2(data_processing_thread); // re-ordered thread creation
    std::cout << "Between thread creation" << std::endl; // delay next thread creation
    std::thread t1(data_preparation_thread);

    t1.join();
    t2.join();
}
```

Debugger console:

```
(gdb) set schedueler_locking step
(gdb) show schedueler_locking
```

} locking step mode

→ helps show details of thread in console output when debugging