

# *Exercises week 7 - Multi-threading I*

Klaas Isaac Bijlsma  
s2394480

David Vroom  
s2309939

January 18, 2018

## **Exercise 49**

*Learn to apply basic multi-threading*

We used the following code.

main.ih

```
1 | #include <string>
2 | #include <thread>
3 | #include <chrono>
4 | #include <vector>
5 | #include <algorithm>
6 | #include <iostream>
7 | #include <iomanip>
8 |
9 | using namespace std;
10 | using namespace chrono;
11 |
12 | void waiting(bool &ready);
```

waiting.cc

```
1 | #include "main.ih"
2 |
3 | void waiting(bool &ready)
4 | {
```

```

5   while (!ready)
6   {
7       cerr << '.';
8       this_thread::sleep_for(seconds(1));
9   }
10  cerr << '\n';
11 }

```

# main.cc

```

1  #include "main.ih"
2
3  int main(int argc, char **argv)
4  {
5      size_t nPrimes = stoull(argv[1]);
6      bool ready = false;
7
8      thread wait(waiting, ref(ready));
9      auto startTime = system_clock::to_time_t(system_clock::now());
10
11     vector<size_t> vec{2};
12     size_t next = 3;
13
14     while (vec.size() != nPrimes)
15     {
16         auto iter =
17             find_if(vec.begin(), vec.end(),
18                 [=](size_t prime)
19                 {
20                     return next % prime == 0;
21                 }
22             ); // Eratosthenes sieve
23         if (iter == vec.end())
24             vec.push_back(next); // next is prime number
25         ++next;
26     }
27
28     auto endTime = system_clock::to_time_t(system_clock::now());
29     ready = true; // Notify waiting thread that computation finished
30     wait.join();

```

```

31
32     for (size_t elem: vec)
33         cout << elem << ' ';
34     cout << '\n';
35
36     cout << put_time(localtime(&startTime), "Starting time: %c") << '\n'
37         << put_time(localtime(&endTime), "Ending time: %c") << '\n'
38         << "Computation of " << nPrimes << " primes took "
39         << endTime - startTime << " seconds\n";
40 }

```

## Exercise 50

*Learn to perform time conversions*

We used the following code.

main.cc

```
1 #include <iostream>
2 #include <chrono>
3
4 using namespace std;
5 using namespace chrono;
6
7 int main()
8 {
9     cout << "Hours: ";
10    int nHours;
11    cin >> nHours;
12
13    cout << "is equal to "
14          << minutes(hours(nHours)).count()
15          << " minutes\n";
16
17    cout << "Seconds: ";
18    int nSec;
19    cin >> nSec;
20
21    cout << "is equal to "
22          << seconds(nSec).count() / seconds(minutes(1)).count()
23          << " minutes\n";
24 }
```

## Exercise 51

*Learn to use the chrono/clock facilities*

We used the following code.

main.cc

```
1 #include <iostream>
2 #include <chrono>
3 #include <iomanip>
4 #include <string.h>
5
6 using namespace std;
7 using namespace chrono;
8
9 int main(int argc, char **argv)
10 {
11     // get the current time
12     time_point<system_clock> timePoint{system_clock::now()};
13
14     // convert it to a std::time_t:
15     time_t time = system_clock::to_time_t(timePoint);
16
17     // display the time:
18     cout << left << setw(14) << "Current time:"
19          << put_time(localtime(&time), "%c") << '\n';
20
21     // display the gmtime
22     cout << left << setw(14) << "Gmtime:"
23          << put_time(gmtime(&time), "%c") << '\n';
24
25     string arg = argv[1];
26     char suffix = arg.back();
27     int count = stoi(arg);
28
29     // add or subtract specified time to now
30     if (suffix == 's')
31         timePoint += seconds(count);
32     else if (suffix == 'm')
33         timePoint += minutes(count);
```

```

34 |     else if (suffix == 'h')
35 |         timePoint += hours(count);
36 |
37 |         // convert it to a std::time_t:
38 |         time_t newTime = system_clock::to_time_t(timePoint);
39 |
40 |         // display the time:
41 |         cout << left << setw(14) << "New time:"
42 |             << put_time(localtime(&newTime), "%c") << '\n';
43 |     }

```

## Exercise 52

*Learn to define a thread with objects that aren't functors*

We used the following code.

handler/handler.ih

```
1 #include "handler.h"
2 #include <iostream>
3
4 using namespace std;
```

handler/handler.h

```
1 #ifndef INCLUDED_HANDLER_H
2 #define INCLUDED_HANDLER_H
3
4 #include <ostream>
5 #include <string>
6 #include <mutex>
7
8 class Handler
9 {
10     public:
11         void shift(std::ostream &out, std::string const &text,
12                  std::mutex &mut) const;
13 };
14
15 #endif
```

handler/shift.cc

```
1 #include "handler.ih"
2
3 void Handler::shift(ostream &out, string const &text, mutex &mut) const
4 {
5     lock_guard<mutex> lg(mut);
6 }
```

```

7      string str(text);
8      out << str << '\n';
9
10     for (size_t idx = 1; idx != str.size(); ++idx)
11     {
12         char first = str[0];
13         str.erase(0,1);
14         str.push_back(first);
15         out << str << '\n';
16     }
17 }

```

#### main.ih

```

1 #include <iostream>
2 #include <fstream>
3 #include <thread>
4 #include <mutex>
5 #include "handler/handler.h"
6
7 using namespace std;
8
9 void callShift(Handler const &handlerObj, ostream &out,
10               string const &text, mutex &mut);

```

#### callshift.cc

```

1 #include "main.ih"
2
3 void callShift(Handler const &handlerObj, ostream &out,
4               string const &text, mutex &mut)
5 {
6     handlerObj.shift(out, text, mut);
7 }

```

#### main.cc

```

1 #include "main.ih"

```



```

2 |
3 | int main(int argc, char **argv)
4 | {
5 |     ofstream out(argv[1]);
6 |     cout << "Give text: \n";
7 |     string txt;
8 |     getline(cin, txt);
9 |
10 |     mutex shiftMutex;
11 |
12 |     Handler object;
13 |     thread th(callShift, ref(object), ref(out), ref(txt), ref(shiftMutex));
14 |
15 |     object.shift(out, txt, shiftMutex);
16 |     th.join();
17 | }

```

## Exercise 53

*Learn to design a simple producer/consumer program*

The thread that reads lines from `cin` and pushes them into the queue is the main function itself. A separate thread uses polling to find out if something is available in the queue. If so, it prints this to a file and removes the line from the queue. However, if it finds an empty queue (via the member function `empty`), it may be because all lines from `cin` are read, or the main thread hasn't finished inserting a new line yet. Because of the latter case, the separate thread should not end when it finds an empty queue, but it should just try again some time later. In the former case, it should be informed that it can end. This is accomplished by main setting a datamember `d_finished`, and let the other thread check for its state. So if all lines have been read from `cin` and pushed, main should not just end, because the other thread isn't finished. We used the following code.

storage/storage.ih

```
1 | #include "storage.h"
2 |
3 | using namespace std;
```

storage/storage.h

```
1 | #ifndef INCLUDED_STORAGE_H
2 | #define INCLUDED_STORAGE_H
3 |
4 | #include <queue>
5 | #include <mutex>
6 | #include <string>
7 |
8 | class Storage
9 | {
10 |     std::queue<std::string> d_queue;
11 |     std::mutex d_mutex;
12 |     bool d_finished = false;
13 |
14 |     public:
15 |         void push(std::string &line);
16 |         std::string &front();
```

```

17 |         void pop();
18 |         bool empty() const;
19 |         bool finished() const;
20 |         void setFinished();
21 |     };
22 |
23 | #endif

```

#### storage/empty.cc

```

1 | #include "storage.ih"
2 |
3 | bool Storage::empty() const
4 | {
5 |     return d_queue.empty();
6 | }

```

#### storage/finished.cc

```

1 | #include "storage.ih"
2 |
3 | bool Storage::finished() const
4 | {
5 |     return d_finished;
6 | }

```

#### storage/front.cc

```

1 | #include "storage.ih"
2 |
3 | string &Storage::front()
4 | {
5 |     lock_guard<mutex> lg(d_mutex);
6 |     return d_queue.front();
7 | }

```

storage/pop.cc

```
1 | #include "storage.ih"
2 |
3 | void Storage::pop()
4 | {
5 |     lock_guard<mutex> lg(d_mutex);
6 |     d_queue.pop();
7 | }
```

storage/push.cc

```
1 | #include "storage.ih"
2 |
3 | void Storage::push(string &line)
4 | {
5 |     lock_guard<mutex> lg(d_mutex);
6 |     d_queue.push(line);
7 | }
```

storage/setfinished.cc

```
1 | #include "storage.ih"
2 |
3 | void Storage::setFinished()
4 | {
5 |     d_finished = true;
6 | }
```

main.ih

```
1 | #include <iostream>
2 | #include <thread>
3 | #include <string>
4 | #include <fstream>
5 | #include "storage/storage.h"
6 | #include <chrono>
7 |
8 | using namespace std;
```

```

9 | using namespace chrono;
10 |
11 | void processQ(Storage &storage, string const &fileName);

```

#### processq.cc

```

1 | #include "main.ih"
2 |
3 | void processQ(Storage &storage, string const &fileName)
4 | {
5 |     ofstream file{ fileName };
6 |
7 |     while (!storage.finished())
8 |     {
9 |         this_thread::sleep_for(seconds(1));
10 |         if (!storage.empty())
11 |         {
12 |             file << storage.front() << '\n';
13 |             storage.pop();
14 |         }
15 |     }
16 | }

```

#### main.cc

```

1 | #include "main.ih"
2 |
3 | int main()
4 | {
5 |     Storage storage;
6 |     string fileName = "output.txt";
7 |
8 |     thread thr(processQ, ref(storage), ref(fileName));
9 |
10 |    string line;
11 |    while (getline(cin, line))
12 |        storage.push(line);
13 |
14 |    storage.setFinished();

```

```
15 |     thr.join();  
16 | }
```