C++ Programming

Exercise Sheet 10

Secure Software Engineering Group
Philipp Schubert
philipp.schubert@upb.de

July 02, 2021

Solutions to this sheet are due on 09.07.2021 at 16:00. Please hand-in a digital version of your answers via PANDA at https://panda.uni-paderborn.de/course/view.php?id=22691.
Note: If you copy text or code elements from other sources, clearly mark those elements and state the source. Copying solutions from other students is prohibited.

This exercise sheet will help you to familiarize yourself with threads and asynchronous calls. Those concepts are important when you need to speed up bottle-neck parts of your programs. In addition, those concepts allow you to make use of modern multi-core processors. At last, you will make use of the C++ library *Armadillo* to solve some simple mathematical problems. You can achieve 16 points in total.

Caution: If you are working on Linux/Unix/Mac you have to use the additional compiler flag `-pthread` at the end of your compile command that tells the compiler to use the POSIX thread model.

Exercise 1.
Consider the following code:

```
#include <iostream>
#include <thread>
#include <vector>

unsigned global_counter = 0;

void increment_global_counter() { ++global_counter; }

int main() {
  std::cout << global_counter << '\n';
  return 0;
}
```

a) Create a **std::vector** of **std::thread**s in **main()**. Start 10000 threads, each of which calls the **increment_global_counter()** function. After having created (=started) the threads, join them (or use **std::jthread**–https://en.cppreference.com/w/cpp/thread/jthread). (2 P.)

b) What happens if you forget to join the threads after having created them? (1 P.)

c) Compile this program and run it a few times. Does this program always print 10000? If not, what is the problem with this code? (2 P.)

d) Fix the issue! (2 P.)

Exercise 2.

Next, consider this piece of code:

```
#include <chrono>
#include <functional>
#include <future>
#include <iostream>
#include <thread>

unsigned factorial(std::future<unsigned> f) {
  unsigned result = 1;
  unsigned n = f.get();
  for (unsigned i = n; i > 0; --i) {
    result *= i;
  }
  return result;
}

int main() {
  std::promise<unsigned> p;
  std::future<unsigned> f = p.get_future();
  std::future<unsigned> fu = std::async(std::launch::async, factorial, std::move(f));
  std::this_thread::sleep_for(std::chrono::seconds(10));
  p.set_value(4);
  unsigned result = fu.get();
  std::cout << result << '\n';
  return 0;
}
```

a) Explain what happens in the above code (in a few sentences)! (2 P.)

b) What is the use of **std::promise**? And why is it useful? What happens if you break your **std::promise**? (2 P.)

c) At which point does the actual computation of the **factorial()** function start? (1 P.)

Exercise 3.

Real-world applications are usually not written from scratch, but rather use libraries to solve a given task. In this exercise, you will install and use the external C++ library *Armadillo*. The Armadillo library can be used to solve linear algebra problems and scientific computations.

a) Download and install the C++ Armadillo maths library from `http://arma.sourceforge.net/`. Use the latest stable version of the library. You will need to build/compile the library yourself from source; consult the instructions of the *README* file for building and installing the library on your system. (1 P.)

b) Write a small program that uses the Armadillo library to solve the following computations:

i Compute the inverse of matrix A. (1 P.)

$$A = \begin{pmatrix} 1 & 4 & -3 & 6 \\ 3 & 10 & -9 & 15 \\ -2 & -6 & 9 & -5 \\ 2 & 8 & -3 & 15 \end{pmatrix}$$

ii Compute the eigenvalues of matrix B. (1 P.)

$$B = \begin{pmatrix} 0 & 1 & -2 \\ -1 & 2 & -2 \\ 0 & 0 & -1 \end{pmatrix}$$

iii Compute matrix C. (1 P.)

$$C = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}^3 \cdot \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

Use the library's documentation at `http://arma.sourceforge.net/docs.html` to familiarize yourself with the data structures and functionalities provided by Armadillo.