C++ Programming

Exercise Sheet 11
Software engineering, EIM-I
Philipp Schubert
January 12, 2018

Solutions to this sheet are due on 19.01.2018 til 14:00. Please hand in a digital version of your answers via e-mail. The e-mails subject has to contain cppp. Do zip-compress your solutions. For questions please send mail or speak to me during the exercises.

Note: If you copy text elements / code elements from other sources, clearly mark those elements and state the source. Copying solutions from other students is prohibited. All of your files that belong to your solution have to be contained in a single .zip file that is named according to the following naming scheme: <name>_<surname>_solution<XX>.zip. Replace <name> and <surname> with your actual name and replace <XX> with the sheet number the solutions belong to. You can look up your results using this link https://docs.google.com/spreadsheets/d/1V8rKtimsQS6thKGkTh6CCh1vLwulBIA3RvKA1ZSH2M/edit?usp=sharing

During this exercise sheet you have to deal with threads and asynchronous calls. These concepts are really important when trying to speed up bottle-neck parts of your programs and making use of modern multi-core processors which can be virtually found in every modern machine.

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 which specifies the POSIX thread model.

Exercise 1.
Consider the following code:

```cpp
#include <iostream>
#include <thread>
#include <vector>
using namespace std;

unsigned global_counter = 0;

void increment_g_counter() { ++global_counter; }

int main() {
    cout << global_counter << "\n";
    return 0;
}
```
a) Inside `main()` create a vector of threads. Start 10000 threads, each of which should call the `increment_g_counter()` function. After having created (=started) the threads, join them. (4 P.)

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

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

d) How can you fix this issue? (1 P.)

e) Fix it! (3 P.)

Exercise 2.
Next, consider this piece of code:

```cpp
#include <iostream>
#include <thread>
#include <future>
#include <functional>
#include <chrono>
using namespace std;

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

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

a) Explain in a few sentences what is going on in the above code! (2 P.)

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

c) When does the actual computation of the `factorial()` function start? (1 P.)
Exercise 3.

This is an optional exercise: Remember our integral function, that is able to integrate a function $f : \mathbb{R} \to \mathbb{R}$. The function is depicted in the following.

```cpp
long double integrate(const long double from,
const long double to,
const size_t iterations,
function<
    long double(long double)>
f) {
    long double integral_val = 0.0;
    long double x = from;
    const long double step_with = abs(from - to) / static_cast<long double>(iterations);
    for (size_t n = 0; n < iterations; ++n) {
        integral_val += f(x);
        x += step_with;
    }
    return integral_val / iterations;
}
```

a) Implement a new function `parallel_integrate()` that calculates the desired integral in parallel by using calls to `std::async()`. (0 P.)