Single Threaded Summation: Protection with Locks

This lesson explains the solution for calculating the sum of a vector problem using locks in C++.

If I protect access to the summation variable with a lock, I will get the answers to two questions.

1. How expensive is the synchronization of a lock without contention?
2. How fast can a lock be in the optimal case?

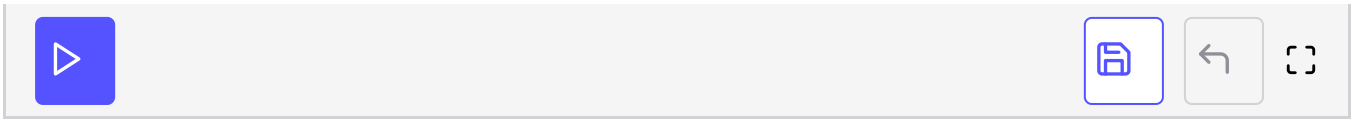
I can draw an interesting conclusion from question 2. If there is contention on a lock, the access time will decrease. That being said, I will only show the application of `std:lock_guard`.

```
// calculateWithLock.cpp

...

std::mutex myMutex;

for (auto i: randValues){
  std::lock_guard<std::mutex> myLockGuard(myMutex);
  sum += i;
}

...
```

Let's see the above fragment of code in action:

```
// calculateWithLoop.cpp

#include <chrono>
#include <iostream>
#include <random>
#include <vector>
#include <mutex>

constexpr long long size = 100000000;

int main(){

  std::cout << std::endl;
```

```
  std::vector<int> randValues;
  randValues.reserve(size);


  // random values
  std::random_device seed;
  std::mt19937 engine(seed());
  std::uniform_int_distribution<> uniformDist(1, 10);
  for (long long i = 0 ; i < size ; ++i)
       randValues.push_back(uniformDist(engine));

  const auto sta = std::chrono::steady_clock::now();

  std::mutex myMutex;
  unsigned long long sum = {};
  for (auto i: randValues){
  std::lock_guard<std::mutex> myLockGuard(myMutex);
  sum += i;
}

  const std::chrono::duration<double> dur =
        std::chrono::steady_clock::now() - sta;

  std::cout << "Time for mySumition " << dur.count()
            << " seconds" << std::endl;
  std::cout << "Result: " << sum << std::endl;

  std::cout << std::endl;

}
```



The execution time is as expected; the access to the protected variable `add` is slower. Using a `std::lock_guard` without contention is about 50 - 150 times slower than using `std::accumulate`.

Let’s finally get to atomics! See you in the next lesson.