Multithreading in C++

Bereczki Norbert Cristian

October 27, 2017

1 Methods

std::shared_future

The class template std::shared_future provides a mechanism to access the result of asynchronous operations, similar to std::future, except that multiple threads are allowed to wait for the same shared state. Unlike std::future, which is only moveable (so only one instance can refer to any particular asynchronous result), std::shared_future is copyable and multiple shared future objects may refer to the same shared state. Access to the same shared state from multiple threads is safe if each thread does it through its own copy of a shared_future object.

std::async

The template function async runs the function f asynchronously (potentially in a separate thread which may be part of a thread pool) and returns a std::future that will eventually hold the result of that function call.

std::mutex (helps lock the pool queue)

The mutex class is a synchronization primitive that can be used to protect shared data from being simultaneously accessed by multiple threads. mutex offers exclusive, non-recursive ownership semantics: A calling thread owns a mutex from the time that it successfully calls either lock or try_lock until it calls unlock. When a thread owns a mutex, all other threads will block (for calls to lock) or receive a false return value (for try_lock) if they attempt to claim ownership of the mutex. A calling thread must not own the mutex prior to calling lock or try_lock.

std::conditional_variable

The condition_variable class is a synchronization primitive that can be used to block a thread, or multiple threads at the same time, until another thread both modifies a shared variable (the condition), and notifies the condition_variable.

2 Bentchmarking

Async and future

1.
500 500 500
one thread -> 2152685310
5 workers -> 2142205059

2.
500 500 500

one thread -> 2153788178
5 workers -> 2171480399

3.
100 100 100
one thread -> 14910051
5 workerds -> 18689751

Threadpool

1.
500 500 500
one thread -> 2134286972
5 workers -> 1166736539

2.
500 500 500
one thread -> 2141809657
5 workerds -> 1174480346

3.
100 100 100
one thread -> 14717043
5 workers -> 8827785