rules.sonarsource.com

C++ static code analysis: A call to "wait()" on a "std::condition_variable" should have a condition

2-3 minutes

A `condition variable` 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`.

Waiting for a `condition variable` without a *condition* can lead to spurious wake-ups or to wait forever.

Noncompliant Code Example

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

```
#include <condition_variable>

std::mutex mutex;
std::condition_variable condVar;

void consumer() {
  std::cout << "Waiting for work" << std::endl;
  std::unique_lock<std::mutex> lck(mutex);
  condVar.wait(lck); // noncompliant: can wait
forever as the order between t1 and t2 is not
guaranteed
  std::cout << "Doing some work" << std::endl;
}

void producer() {
  std::cout << "Work submited" << std::endl;
  condVar.notify_one(); // this can be executed
before or after the wait in consumer, no
guarantee
}

int main() {
  std::thread t1(consumer);
  std::thread t2(producer);
  t1.join();
  t2.join();
}
```

Compliant Solution

```
#include <iostream>
#include <thread>
#include <condition_variable>

std::mutex mutex;
std::condition_variable condVar;

bool pendingWork{false};

void consumer() {
  std::cout << "Waiting for work" << std::endl;
  std::unique_lock<std::mutex> lck(mutex);
  condVar.wait(lck, []{ return pendingWork; }); //
compliant: if this is called after producer in t2, the
call will not block thanks to the condition
  std::cout << "Doing some work" << std::endl;
}
```

```
void producer() {
  {
    std::lock_guard<std::mutex> lck(mutex);
    pendingWork = true;
  }
  std::cout << "Work submitted" << std::endl;
  condVar.notify_one();
}

int main(){
  std::thread t1(consumer);
  std::thread t2(producer);

  t1.join();
  t2.join();
}
```

See

- The traps of condition variables
- C++ Core Guidelines - CP.42 - Don’t wait without a condition