Condition Variables

In this lesson, we will see how tasks should be your first choice when using multithreading.

WE'LL COVER THE FOLLOWING

Introduction

Condition variables enable threads to be synchronized via messages.

> They need the `<condition_variable>` header.

One thread acts as a sender, and the other acts as the receiver of the message. The receiver waits for a notification from the sender. Typical use cases for condition variables are sender-receiver or producer-consumer workflows.

A condition variable can be the sender or the receiver of the message.

The following table details the different condition variables and their respective functions.

Methods of Condition Variables

Method	Description

`cv.notify_one()`	Notifies a waiting thread.
`cv.notify_all()`	Notifies all waiting threads.
`cv.wait(lock, ...)`	Waits for the notification while holding a `std::unique_lock`.
`cv.wait_for(lock, relTime, ...)`	Waits for a time duration of the notification while holding a `std::unique_lock`.
`cv.wait_until(lock, absTime, ...)`	Waits until a time point for the notification while holding a `std::unique_lock`.

The subtle difference between `cv.notify_one` and `cv.notify_all` is that `cv.notify_all` will notify all waiting threads. In contrast, `cv.notify_one` will notify only one of the waiting threads. The other threads will remain in the wait state.

Condition variables are a victim to two known phenomena: lost wakeup and spurious wakeup. Let's discuss these issues.

The Issues of Condition Variables

Lost Wakeup

The phenomena of lost wakeup is as follows: the sender sends its notification before the receiver arrives at a wait state. As a result, the notification is lost. The C++ standard describes condition variables as a simultaneous synchronization mechanism: "The condition_variable class is a synchronization primitive that can be used to block a thread, or multiple threads at the same time, ...". The notification is lost, and the receiver is left waiting.

Spurious Wakeup

The phenomena of spurious wakeup is as follows: the receiver wakes up though no notification has occurred. POSIX Threads and the Windows API can

though no notification has occurred. POSIX Threads and the Windows API can both be victims of these two phenomena.

In most use-cases, tasks are the least error-prone way to synchronize threads.

Before we investigate the details of condition variables, let's take a look at an example of these issues in the next lesson.