Condition Variables

Condition variables enable it to synchronize threads.

- Typical use cases
  - Sender – receiver workflow
  - Producer – consumer workflow

- `std::condition_var`
  - Needs the header `<condition_var>`.
  - Can play the role of the sender and of the receiver.

To synchronize threads, tasks are often the better choice.

Condition Variables

- Sender sends a notification.

Member Function	Description
`cv.notify_one()`	Notifies one waiting thread
`cv.notify_all()`	Notifies all waiting threads

- Receiver is waiting for the notification while holding the mutex.

Member Function	Description
`cv.wait(lock, ... )`	Waits for the notification
`cv.wait_for(lock, relTime, ... )`	Waits for the notification for a time period
`cv.wait_until(lock, absTime, ... )`	Waits for the notification until a time point

In order to protect against spurious wakeup and lost wakeup, the wait member function should be used with an additional predicate.

Condition Variables

Thread 1: Sender

- Does its work
- Notifies the receiver

```
// do the work
{
  lock_guard<mutex> lck(mut);
  ready= true;
}
condVar.notify_one();
```

Thread 2: Receiver

- Waits for its notification while holding the lock
  - Gets the lock
  - Checks and continues to sleep
- Does its work
- Releases the lock

```
{
   unique_lock<mutex>lck(mut);
   condVar.wait(lck, []{return ready;});
 // do the work
}
```

conditionVariable.cpp