Threads and Concurrency with C++

Threading and concurrency control vary widely with different operating systems. To make threads programming easier and portable, Ice provides a simple thread abstraction layer that allows you to write portable source code regardless of the underlying platform.

This section looks at the threading and concurrency control mechanisms in Ice for C++. It explains the threading abstractions provided by Ice: mutexes, monitors, and threads. Using these APIs allows you to make your code thread safe and to create threads of your own without having to use non-portable APIs that differ in syntax or semantics across different platforms: Ice not only provides a portable API but also guarantees that the semantics of the various functions are the same across different platforms. This makes it easier to create thread-safe applications and lets you move your code between platforms with simple recompilation.

> This section assumes you are familiar with light-weight threads and concurrency control. See [1] for an excellent treatment of programming with threads. Also see The Ice Threading Model, which provides a language-neutral introduction to the Ice threading model.

Library Overview

The Ice threading library provides the following thread-related abstractions:

- mutexes
- recursive mutexes
- monitors
- a thread abstraction that allows you to create, control, and destroy threads

The synchronization primitives permit you to implement concurrency control at different levels of granularity. In addition, the thread abstraction allows you to, for example, create a separate thread that can respond to GUI or other asynchronous events. All of the threading APIs are part of the `IceUtil` namespace.

Topics

References

1. Butenhof, D. 1996. Programming with POSIX Threads. Reading, MA: Addison-Wesley.