

University of
Waterloo



Concurrency in C++

Peter A. Buhr

<http://plg.uwaterloo.ca/~usystem/uC++.html>

1 Introduction

1.1 Why Concurrency

- Processor speed has slowed (stopped).
- Use transistors from Moore's law for parallelism to increase speed.
- But concurrent programming is necessary to utilize parallel hardware.
- Some success in *implicitly* discovering concurrency in sequential programs.
- Fundamental limits to finding parallelism and only for certain kinds of problems.
- Alternatively, programmer *explicitly* thinks about and specifies the concurrency.
- Implicit and explicit approaches are complementary, i.e., can appear together.
- However, the limitations of the implicit approach mandate an explicit approach.

Except as otherwise noted, the content of this presentation is licensed under the Creative Commons Attribution 2.5 License.

1.2 Why C++

- C++ is one of the dominant programming languages.
- Why?
 - based on C, which has a large programmer and code base,
 - as efficient as C in most situations,
 - low-level features, e.g., direct memory access, needed for:
 - * systems programming
 - * memory management
 - * embedded / real-time
 - high-level features, e.g., objects, polymorphism, exception handling, STL, etc., which programmers now demand,
 - allows direct interaction with UNIX/Windows.

1.3 Concurrency in C++

- **C++ has no concurrency!**
- Many different concurrency approaches for C++ have been implemented with only varying degrees of adoption.

- No de facto approach dominating concurrent programming in C++.
 - C has two dominant but incompatible concurrency libraries: pthreads and Win32.
- C++'s lack of concurrency limits its future in the parallel domain.
- Therefore, it is imperative C++ be augmented with concurrency facilities to extend its programming base.
- Interestingly, concurrency **CANNOT** be safely added to **ANY** language via library code.

1.4 High-Level Concurrency

- Want a single consistent high-level powerful concurrency mechanism, but what should it look like?
- In theory, any high-level concurrency paradigm/model can be adapted into C++.
- However, C++ does not support all concurrency approaches equally well, e.g., tuple space, message passing, channels.
- C++ is fundamentally based on a class model using routine call, and its other features leverage this model.

- Any concurrency approach matching the C++ model is better served because its concepts interact consistently with the language.
- Apply “Principle of Least Astonishment” whenever possible.
- Let C++ dictate which concurrency approaches fits best via its design principles.
- For OO language, thread/stack is best associated with class, and mutual-exclusion/synchronization with member routines.

1.5 μ C++ : Advanced Control Flow for C++

- integrate advanced control flow tightly into C++
 - leverage all class features
 - threads are light-weight: M:N thread model versus 1:1
- use mutex objects to contain mutual exclusion and synchronization
 - communication using routine call (versus messages/channels)
 - statically typed
- handle multi-processor environment

1.6 Outline

- coroutines : suspending and resuming (concurrency precursor)
- concurrency : introduce multiple threads
- locking : thread control through synchronization and mutual exclusion
- errors : new concurrency issues
- monitors : high-level thread control
- tasks : active objects
- other concurrent approaches : different concurrency paradigms