Intel Thread Building Blocks, Part V

SPD course 2013-14

Massimo Coppola

21/05/2014

Synchronization mechanisms

- Low level mechanism to control low-level concurrent access to data structures
- Use with great care
  - Can cause software lockout
- Mutexes
  - data structures that allow adding generick locking mechanisms to any data structures
- Atomic
  - template that add very simple, low overhead, hw-supported atomic behaviour to a few machine types available in the language
- PPL Compatibility
  - 2 constructs added for compatibility with Microsoft Parallel Pattern Library
- C++11 syncronizations
  - Supports a subset of the N3000 draft of the C++11 standard
  - will change in future implementations of TBB

atomic objects

- template<typename T> atomic;
- Generate special machine instructions to ensure that operating on a variable in memory is performed atomically
- atomics within the C++11 standard (TBB goes beyond it)
- Integral type, enum type, pointer type
- Template supports atomic read, write, increment, decrement, fetch&add, fetch&store, compare&swap operations
- Arithmetic
  - Pointer arithmetic is T is a pointer
  - not allowed if T is enum, bool or void*

atomic objects

- Copy constructor is never atomic
  - It is compiler generated
  - Need to default construct, then assign

```
atomic<T> y(x);        // Not atomic
atomic<T> z; z=x;      // Atomic assignment
```

  - C+11 uses the constexpr mechanism for this

- atomic <T*> defines the dereferencing of data as
  - T* operator->() const;

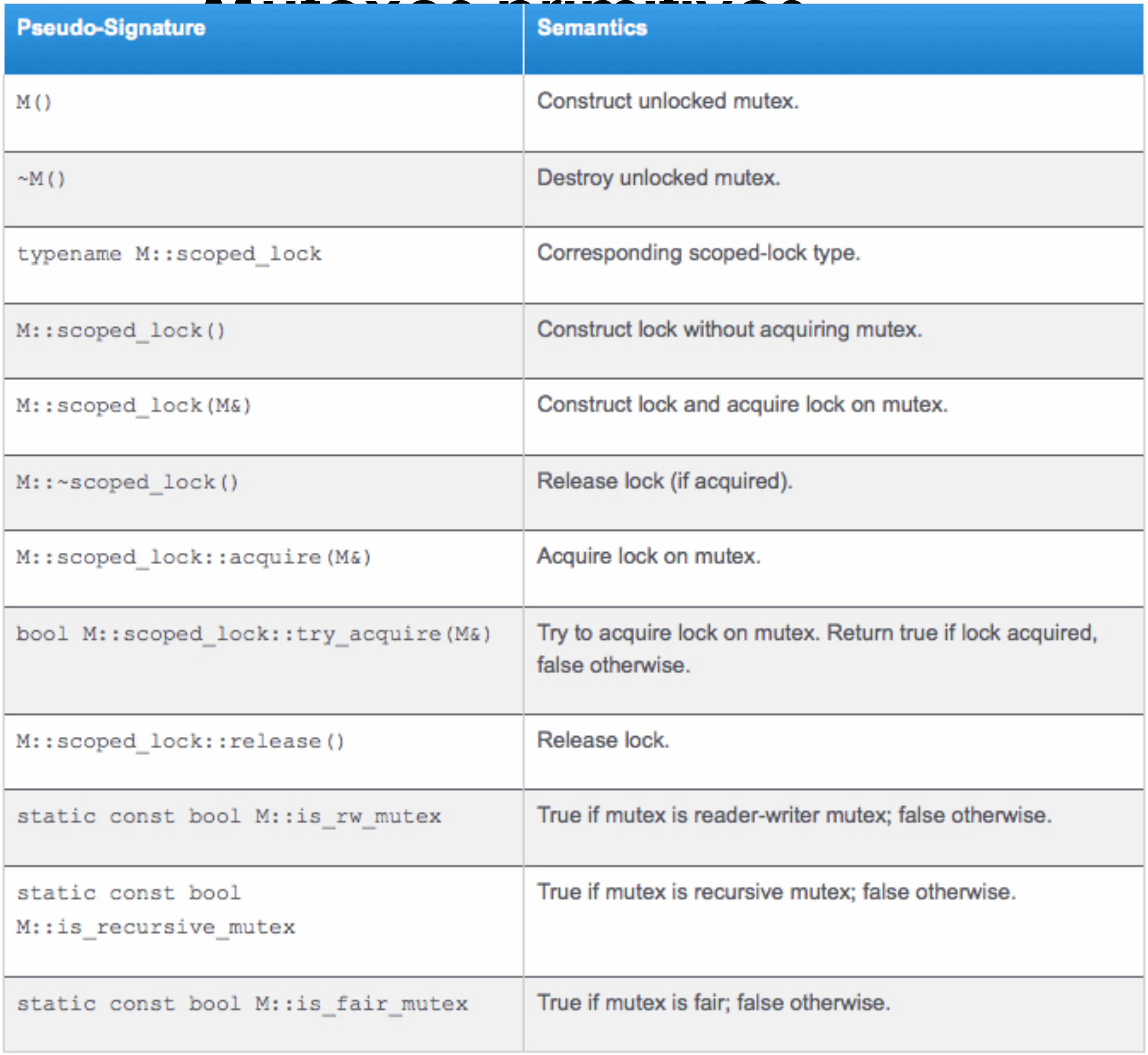
Atomic methods

- value_type fetch_and_add(value_type addend)
  - Add atomically
- value_type fetch_and_increment()
- value_type fetch_and_decrement()
  - Increment/decrement atomically
- value_type compare_and_swap(value_type new_value, value_type comparand)
  - If the atomic has value “comparand” set it to “new_value”
- value_type fetch_and_store(value_type new_value)

Mutexes

- Classes to build *lock objects*
- The new lock object will generally
  - Wait according to specific semantics for locking
  - Lock the object
  - Release lock when destroyed
- Several characteristics of mutexes
  - Scalable
  - Fair
  - Recursive
  - Yield / Block
- Check implementations in the docs:
  - mutex, recursive_mutex, spin_mutex, queueing_mutex, spin_rw_mutex, queueing_rw_mutex, null_mutex, null_rw_mutex
  - Specific reader/writer locks
  - Upgrade/downgrade operation to change r/w role

Mutexes primitives



Pseudo-Signature	Semantics
`M()`	Construct unlocked mutex.
`~M()`	Destroy unlocked mutex.
`typename M::scoped_lock`	Corresponding scoped-lock type.
`M::scoped_lock()`	Construct lock without acquiring mutex.
`M::scoped_lock(M&)`	Construct lock and acquire lock on mutex.
`M::~scoped_lock()`	Release lock (if acquired).
`M::scoped_lock::acquire(M&)`	Acquire lock on mutex.
`bool M::scoped_lock::try_acquire(M&)`	Try to acquire lock on mutex. Return true if lock acquired, false otherwise.
`M::scoped_lock::release()`	Release lock.
`static const bool M::is_rw_mutex`	True if mutex is reader-writer mutex; false otherwise.
`static const bool M::is_recursive_mutex`	True if mutex is recursive mutex; false otherwise.
`static const bool M::is_fair_mutex`	True if mutex is fair; false otherwise.

Types of mutexes

	Scalable	Fair	Reentrant	Long Wait	Size
`mutex`	OS dependent	OS dependent	No	Blocks	>=3 words
`recursive_mutex`	OS dependent	OS dependent	Yes	Blocks	>=3 words
`spin_mutex`	No	No	No	Yields	1 byte
`speculative_spin_mutex`	No	No	No	Yields	2 cache lines
`queuing_mutex`	Yes	Yes	No	Yields	1 word
`spin_rw_mutex`	No	No	No	Yields	1 word
`queuing_rw_mutex`	Yes	Yes	No	Yields	1 word
`null_mutex`	-	Yes	Yes	-	empty
`null_rw_mutex`	-	Yes	Yes	-	empty