quarks Reference Manual

1

Generated by Doxygen 1.5.3

Thu Jun 10 18:17:34 2010

Contents

Chapter 1

quarks Main Page

1.1 Project description

Each normal project that contains code has its own top-level namespace that is documented separately. The "boilerplate" project contains not code but code templates used to start new files with the right copyright info, etc.

This is an unusual project in several ways. It exists only to hold bits and pieces found useful in other projects. Most of the packages are too small to have their own libraries and many will offer only source code. The packages are named after broad categories rather than particular functions.

For that reason the packages list only includes the mergedlib package, which compiles .cc files gathered from the other packages and combines them into one large library. This package generates include files called testdefs.hh and testcalls.cc which are used by testQuarks.cc to call all the package unit-test files (which if they exist are named as testPackagename.cc). testQuarks is included in the merged library.

For these reasons the projects other than mergedlib are not independently buildable and should have empty constituents.mk files. The makefiles for mergedlib assume that ordinary compilable code is in packagedir/src and that unit-test code, including testPackagename.cc, is in packagedir/test. Each package's test driver should return void and take no arguments.

1.2 Project history

1.2.1 2010/02/12

Creation of the project.

1.2.2 2010/03/29

Reworked the logging classes in package "service" to make them thread-safe; added the class LogMessage. Added the package "concurrency" and the class CriticalSection. Added the package "datastructures" and the classes BitSet and BitPq.

1.2.3 2010/04/02

Added the "io" package with its first members readAll() and writeAll(). Made PtyLogger use writeAll() for greater efficiency, slightly modifying the LoggerImpl interface.

1.2.4 2010/06/10

Added some RCE-specific components to the "concurrency" package. These include low-level synchronization primitives plus a CircularFifo template; the FIFO is actually implemented with the lock-free LlScFifo which uses lwarx/stwc for synchronization. There is also NoInterruptFifo which uses critical sections created by turning off interrupts; this is intended as an alternate means of implementing CircularFifo. Class InterruptGuard uses IIRA to allow easy definition of blocks of code to be run with interrupts off, automatically turning them back on when the block is exited.

Added the TimerGuard template to the "service" package (also for RCE only). This uses IIRA to calculate the number of PowerPC time base ticks needed to execute a code block and passes the result to a user-specified object.

Chapter 2

quarks Namespace Index

2.1 quarks Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

quarks Hierarchical Index

3.1 quarks Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

quarks Class Index

4.1 quarks Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

quarks File Index

5.1 quarks File List

Here is a list of all files with brief descriptions:

Chapter 6

quarks Namespace Documentation

6.1 anonymous_namespace{LogMessage.cc} Namespace Reference

Functions

- void addProlog (LogMessage &msg, Logger::Severity sev)

6.1.1 Function Documentation

6.1.1.1 void anonymous_namespace{LogMessage.cc}::addProlog (LogMessage & *msg*, Logger::Severity *sev*)

Definition at line 54 of file LogMessage.cc.

References quarks::service::LogMessage::add().

Referenced by quarks::service::LogMessage::LogMessage().

6.2 anonymous_namespace{PtyLogger.cc} Namespace Reference

Enumerations

- enum { Stderr = 2 }

6.2.1 Enumeration Type Documentation

6.2.1.1 anonymous enum

Enumerator:

Stderr

Definition at line 52 of file PtyLogger.cc.

6.3 anonymous_namespace{testCircularFifo.cc} Namespace Reference

Classes

- struct Node

  The values placed in the queue.

- struct Thread

  The abstract base class for producers and consumers. Non-copyable.

- struct Completion

  Used to perform cleanup after a thread's run() member exits.

- struct Producer

  Enqueues a fixed number of Nodes, yielding after each enqueue, then exits. Each node contains the producer ID and a monotonically increasing sequence number.

- struct ProdRecord
- struct Consumer

Typedefs

- typedef map< int, ProdRecord > RecordMap

Functions

- bool operator== (const Node &lhs, const Node &rhs)

6.3.1 Typedef Documentation

6.3.1.1 typedef map<int, ProdRecord> anonymous_namespace{testCircularFifo.cc}::RecordMap

A set of ProdRecords indexed by Producer ID.

Definition at line 219 of file testCircularFifo.cc.

6.3.2 Function Documentation

6.3.2.1 bool anonymous_namespace{testCircularFifo.cc}::operator== (const Node & *lhs*, const Node & *rhs*)

Definition at line 74 of file testCircularFifo.cc.

References anonymous_namespace{testCircularFifo.cc}::Node::a, and anonymous_namespace{testCircularFifo.cc}::Node::b.

6.4 quarks Namespace Reference

6.4.1 Detailed Description

The top-level namespace for the "quarks" project.

Namespaces

- namespace concurrency

  Tools for managing threads, tasks or whatever.

- namespace datastructures

  Useful non-concurrent data structures.

- namespace io

  General I/O utilities.

- namespace service

  Provides services relating to the run-time environment such as the OS, external servers, etc.

6.5 quarks::concurrency Namespace Reference

6.5.1 Detailed Description

Tools for managing threads, tasks or whatever.

Classes

- class CriticalSection

  Take a Semaphore on construction, give it back on destruction. Non-copyable.

- class NoInterruptFifo

  A concurrent FIFO based on a circular array synchronized by locked critical sections. Locking is achieved by disabling interrupts within the critical sections.

- class CircularFifo

  A generic wrapper for a concurrent FIFO implemented using a circular buffer with a fixed number of slots. T The type of the items to be queued.

- class LlScFifo

  A lock-free concurrent FIFO based on a circular array synchronized by the atomic operations Load and Lock (LL) and Store Conditional (SC).

- struct IsA32BitType
- struct IsA32BitType< true >
- class InterruptGuard

  The constructor saves the state of the interrupt enables and disables interrupts. The destructor restores the prior state of the interrupt enables.

Functions

- template<typename T>
  T LL (const T &x) __attribute__((always_inline))

  Atomic Load and Lock. Read a value from a given address and reserve (lock) that location.

- template<typename T>
  bool SC (T &x, T newval) __attribute__((always_inline))

  Atomic Store Conditional. Place a new value in the location previously reserved via LL() UNLESS the reservation has been cancelled. Cancel the reservation.

- template<typename T>
  bool CAS (T &var, T oldVal, T newVal) __attribute__((always_inline))

  Atomic Compare And Swap. If the value at the given address has the given value then change it to the given new value and return true; else leave the value unchanged and return false.

- template<typename T>
  T FetchAndAdd (T *x, T v) __attribute__((always_inline))

  Atomic Fetch And Add. Add a given amount to a variable, returning the old value.

- template<typename T>
  T FetchAndSub (T *x, T v) __attribute__((always_inline))

  Atomic Fetch And Sub. Subtract a given amount to a variable, returning the old value.

- void yield () __attribute__((always_inline))

  Stop running the current task and place it in the ready chain after all other tasks with the same priority.

- unsigned getMsr () __attribute__((always_inline))

  Get the contents of the Machine Status Register.

- unsigned setMsr (unsigned newMsr) __attribute__((always_inline))

  Set the contents of the Machine Status Register.

- unsigned setMsrMasked (unsigned newMsr, unsigned mask) __attribute__((always_inline))

  Write new values to the MSR bits specified by a 32-bit mask. newMSR = (oldMSR & ~mask) | (newMSR & mask).

6.5.2 Function Documentation

6.5.2.1 template<typename T> bool quarks::concurrency::CAS (T & *var*, T *oldVal*, T *newVal*) [inline]

Atomic Compare And Swap. If the value at the given address has the given value then change it to the given new value and return true; else leave the value unchanged and return false.

Parameters:

← ***var*** A reference to an aligned 32-bit value.

← ***oldVal*** The swap takes place if *addr == oldVal.

← ***newVal*** *addr = newVal if the swap takes place.

See also:

LL(), SC()

Warning:

Uniprocessor only (processors don't share RAM).

This version of CAS doesn't suffer from the ABA problem because the test for equality is considered to fail if the reservation has been lost, no matter the actual value at the address.

Definition at line 120 of file rce405/syncops.hh.

References LL(), and SC().

6.5.2.2 template<typename T> T quarks::concurrency::FetchAndAdd (T * *x*, T *v*) [inline]

Atomic Fetch And Add. Add a given amount to a variable, returning the old value.

Parameters:

← ***x*** The address of an aligned 32-bit integer value.

← ***v*** The value to add to ∗x.

Returns:

The value of ∗x before the addition.

See also:

LL(), SC()

Warning:

Uniprocessor only (processors don't share RAM).

This operation can't fail like SC() or CAS(); instead it keeps looping until it can get undisturbed access to the variable long enough to increment it. Since we're running on a uniprocessor system the only disturbances are using up our time quantum (or yielding) or getting interrupted by a higher priority thread (or an ISR).

Definition at line 143 of file rce405/syncops.hh.

References LL(), and SC().

6.5.2.3 template<typename T> T quarks::concurrency::FetchAndSub (T ∗ *x*, T *v*) [inline]

Atomic Fetch And Sub. Subtract a given amount to a variable, returning the old value.

See also:

FetchAndSub()

FetchAndAdd(unsigned ∗x, -n) wouldn't work since ∗x and -n would not have the same type.

Definition at line 161 of file rce405/syncops.hh.

References LL(), and SC().

6.5.2.4 unsigned quarks::concurrency::getMsr () [inline]

Get the contents of the Machine Status Register.

Definition at line 177 of file rce405/syncops.hh.

Referenced by setMsr(), and setMsrMasked().

6.5.2.5 template<typename T> T quarks::concurrency::LL (const T & *x*) [inline]

Atomic Load and Lock. Read a value from a given address and reserve (lock) that location.

Parameters:

← ***x*** A reference to an aligned value of a 32-bit data type.

Returns:

The value of the word found at the address.

The PPC405 will set an internal "reserved" bit that will be cleared by ANY SC operation, no matter what address the SC operation is given. This means that an LL/SC pair using one address may cause a false indication of change for a pair using a different address. To reduce that interference keep each SC as close as possible to it's matching LL.

See also:

XiLinx PowerPC Processor Reference Guide (UG011) V1.2 pp. 127ff, pp. 559ff

Warning:

Uniprocessor only (processors don't share RAM).

Definition at line 70 of file rce405/syncops.hh.

Referenced by CAS(), quarks::concurrency::LlScFifo< T >::dequeue(), quarks::concurrency::LlScFifo< T >::enqueue(), FetchAndAdd(), and FetchAndSub().

6.5.2.6 template<typename T> bool quarks::concurrency::SC (T & *x*, T *newval*) [inline]

Atomic Store Conditional. Place a new value in the location previously reserved via LL() UNLESS the reservation has been cancelled. Cancel the reservation.

Parameters:

← ***x*** The same variable used with LL().

Returns:

true if the store was performed, else false.

See also:

LL()

Warning:

Uniprocessor only (processors don't share RAM).

Definition at line 89 of file rce405/syncops.hh.

Referenced by CAS(), quarks::concurrency::LlScFifo< T >::dequeue(), quarks::concurrency::LlScFifo< T >::enqueue(), FetchAndAdd(), and FetchAndSub().

6.5.2.7 unsigned quarks::concurrency::setMsr (unsigned *newMsr*) [inline]

Set the contents of the Machine Status Register.

Returns:

The old contents.

This inline function is used to begin and end critical sections wherein all interrupts are disabled.

1. The compiler can't be allowed to move instructions across the critical section boundary for optimization. Therefore we use "asm volatile" and not just "asm".

2. Any memory values cached in registers must be written back to memory before the critical section is entered; we use the "memory" constraint to let the compiler know.
3. The processor can't be allowed to reorder pipelined load and store operations across the boundary so we precede "mtmsr" with "eieio" ("sync" would also have worked).
4. We want the changes made to the MSR to take effect right away. At least some MSR changes take effect only after an unspecified delay (pipeline drain?). To be safe we use "isync" after the "mtmsr" which discards all prefetched instructions in the pipeline.

Definition at line 206 of file rce405/syncops.hh.

References getMsr().

Referenced by setMsrMasked().

6.5.2.8 unsigned quarks::concurrency::setMsrMasked (unsigned *newMsr*, unsigned *mask*) [inline]

Write new values to the MSR bits specified by a 32-bit mask. newMSR = (oldMSR & ∼mask) | (newMSR & mask).

Parameters:

← ***newMsr*** A value containing the desired content of the masked bits.

← ***mask*** A 32-bit mask with a 1 bit for each MSR bit to be changed.

Returns:

The old MSR contents.

Definition at line 223 of file rce405/syncops.hh.

References getMsr(), and setMsr().

Referenced by quarks::concurrency::InterruptGuard::∼InterruptGuard().

6.5.2.9 void quarks::concurrency::yield () [inline]

Stop running the current task and place it in the ready chain after all other tasks with the same priority.

Definition at line 173 of file rce405/syncops.hh.

Referenced by anonymous_namespace{testCircularFifo.cc}::Thread::dequeueWait(), anonymous_namespace{testCircularFifo.cc}::Thread::enqueueWait(), and anonymous_namespace{testCircularFifo.cc}::Thread::join().

6.6 quarks::datastructures Namespace Reference

6.6.1 Detailed Description

Useful non-concurrent data structures.

Classes

- class BitSet

  A classic powerset stored in an unsigned integer.

- class BitPq

  Yields a sequence of set bit positions in order from lowest to highest.

Functions

- BitSet operator~ (const BitSet &b)

  Produce a new Bitset with the same members as b.complement().

- BitSet operator| (const BitSet &a, const BitSet &b)

  Produce a new Bitset that's the union of a and b.

- BitSet operator & (const BitSet &a, const BitSet &b)

  Produce a new Bitset that's the intersection of a and b.

- BitSet operator^ (const BitSet &a, const BitSet &b)

  Produce a new Bitset that's the symmetric difference of a and b.

- BitSet operator- (const BitSet &a, const BitSet &b)

  Produce a new Bitset that's the difference of a and b.

Variables

- static const unsigned deBruijn = 0x077CB531U
- static const int deBruijnUnscramble [32]

6.6.2 Function Documentation

6.6.2.1 BitSet quarks::datastructures::operator & (const BitSet & *a*, const BitSet & *b*) [inline]

Produce a new Bitset that's the intersection of a and b.

Definition at line 132 of file BitSet.hh.

6.6.2.2 BitSet quarks::datastructures::operator- (const BitSet & *a*, const BitSet & *b*) [inline]

Produce a new Bitset that's the difference of a and b.

Definition at line 146 of file BitSet.hh.

6.6.2.3 BitSet quarks::datastructures::operator^ (const BitSet & *a*, const BitSet & *b*) [inline]

Produce a new Bitset that's the symmetric difference of a and b.

Definition at line 139 of file BitSet.hh.

6.6.2.4 BitSet quarks::datastructures::operator| (const BitSet & *a*, const BitSet & *b*) [inline]

Produce a new Bitset that's the union of a and b.

Definition at line 125 of file BitSet.hh.

6.6.2.5 BitSet quarks::datastructures::operator~ (const BitSet & *b*) [inline]

Produce a new Bitset with the same members as b.complement().

Definition at line 122 of file BitSet.hh.

References quarks::datastructures::BitSet::complement().

6.6.3 Variable Documentation

6.6.3.1 const unsigned quarks::datastructures::deBruijn = 0x077CB531U [static]

Definition at line 166 of file BitSet.hh.

Referenced by quarks::datastructures::BitPq::newtop().

6.6.3.2 const int quarks::datastructures::deBruijnUnscramble[32] [static]

Initial value:

```
{
  0, 1, 28, 2, 29, 14, 24, 3, 30, 22, 20, 15, 25, 17, 4, 8,
  31, 27, 13, 23, 21, 19, 16, 7, 26, 12, 18, 6, 11, 5, 10, 9
}
```

Definition at line 171 of file BitSet.hh.

Referenced by quarks::datastructures::BitPq::newtop().

6.7 quarks::io Namespace Reference

6.7.1 Detailed Description

General I/O utilities.

Functions

- std::ptrdiff_t readAll (int fd, void *buffer, std::size_t nbytes)

  Attempt to read all the bytes requested before returning.

- std::ptrdiff_t writeAll (int fd, const char *buffer, std::size_t nbytes)

  Attempt to write all the bytes requested before returning.

- ptrdiff_t readAll (int fd, void *buffer, size_t nbytes)
- ptrdiff_t writeAll (int fd, const char *buffer, size_t nbytes)

6.7.2 Function Documentation

6.7.2.1 ptrdiff_t quarks::io::readAll (int *fd*, void * *buffer*, size_t *nbytes*)

Definition at line 49 of file rwall.cc.

6.7.2.2 std::ptrdiff_t quarks::io::readAll (int *fd*, void * *buffer*, std::size_t *nbytes*)

Attempt to read all the bytes requested before returning.

Parameters:

← ***fd*** The file descriptor from which to read.

← ***buffer*** The buffer into which to read.

← ***nbytes*** The number of bytes to read.

Returns:

The number of bytes transferred, 0 if EOF or -1 if there was an error.

The POSIX read() function may not transfer all the bytes requested even if it encounters no read error or end-of-file condition. For some devices, e.g., pipes it may simply take what's available in a system buffer and return. This function will call read() repeatedly in an attempt to transfer all the bytes requested.

6.7.2.3 ptrdiff_t quarks::io::writeAll (int *fd*, const char * *buffer*, size_t *nbytes*)

Definition at line 71 of file rwall.cc.

Referenced by quarks::service::PtyLogger::send().

6.7.2.4 std::ptrdiff_t quarks::io::writeAll (int *fd*, const char * *buffer*, std::size_t *nbytes*)

Attempt to write all the bytes requested before returning.

Parameters:

← ***fd*** The file descriptor to which to write.

← ***buffer*** The buffer from which to write.

← ***nbytes*** The number of bytes to write.

Returns:

The number of bytes transferred or -1 if there was an error.

The POSIX write() function may not transfer all the bytes requested even if encounters no write error. For some devices, e.g., pipes it may simply fill the space remaining in a system buffer and return. This function will call write() repeatedly in an attempt to transfer all the bytes requested.

6.8 quarks::service Namespace Reference

6.8.1 Detailed Description

Provides services relating to the run-time environment such as the OS, external servers, etc.

Classes

- struct InitLogging

  Set up logging in the constructor and tear it down in the destructor.

- class Logger

  Handle log messages from applications.

- class LoggerImpl

  Abstract base class for logging implementations.

- class LogMessage

  Assembles a log message in a buffer.

- class PtyLogger

  Write log messages to /dev/pty0. If the PTY can't be opened then log messages will go to fd 1 (stdout).

- class TimerGuard

  Time a block of code between the exiting of the TimeGuard constructor and the start of the TimeGuard destructor. Always in-line.

Chapter 7

quarks Class Documentation

7.1 quarks::datastructures::BitPq Class Reference

`#include <BitSet.hh>`

7.1.1 Detailed Description

Yields a sequence of set bit positions in order from lowest to highest.

A bit position occurs at most once. Typical use: for (BitPq pq(BitSet); !pq.empty(); pq.pop()) {... pq.top() ...}

Definition at line 182 of file BitSet.hh.

Public Member Functions

- BitPq (const BitSet &set)

  Set the initial queue contents from a BitSet.

- bool empty () const

  Is the queue empty?

- int top () const

  Return the top-most (lowest) bit position in the queue.

- void push (int pos)

  Put a bit position in is proper place in the queue (if not already there).

- void pop ()

  Remove the top bit position.

- int size () const

  How many bit positions are in the queue?

Private Member Functions

- void newtop ()

  Find the lowest numbered bit position now present in the queue.

Private Attributes

- unsigned _bits
- int _top

7.1.2 Constructor & Destructor Documentation

7.1.2.1 quarks::datastructures::BitPq::BitPq (const BitSet & *set*) [inline, explicit]

Set the initial queue contents from a BitSet.

Definition at line 188 of file BitSet.hh.

References newtop().

7.1.3 Member Function Documentation

7.1.3.1 bool quarks::datastructures::BitPq::empty () const [inline]

Is the queue empty?

Definition at line 190 of file BitSet.hh.

References _top.

Referenced by pop().

7.1.3.2 int quarks::datastructures::BitPq::top () const [inline]

Return the top-most (lowest) bit position in the queue.

Definition at line 192 of file BitSet.hh.

References _top.

7.1.3.3 void quarks::datastructures::BitPq::push (int *pos*) [inline]

Put a bit position in is proper place in the queue (if not already there).

Definition at line 194 of file BitSet.hh.

References _bits, and newtop().

7.1.3.4 void quarks::datastructures::BitPq::pop () [inline]

Remove the top bit position.

Definition at line 196 of file BitSet.hh.

References _bits, _top, empty(), and newtop().

7.1.3.5 int quarks::datastructures::BitPq::size () const [inline]

How many bit positions are in the queue?

Definition at line 203 of file BitSet.hh.

References _bits.

7.1.3.6 void quarks::datastructures::BitPq::newtop () [inline, private]

Find the lowest numbered bit position now present in the queue.

Definition at line 207 of file BitSet.hh.

References _bits, _top, quarks::datastructures::deBruijn, and quarks::datastructures::deBruijnUnscramble.

Referenced by BitPq(), pop(), and push().

7.1.4 Member Data Documentation

7.1.4.1 unsigned quarks::datastructures::BitPq::_bits [private]

Definition at line 183 of file BitSet.hh.

Referenced by newtop(), pop(), push(), and size().

7.1.4.2 int quarks::datastructures::BitPq::_top [private]

Definition at line 184 of file BitSet.hh.

Referenced by empty(), newtop(), pop(), and top().

The documentation for this class was generated from the following file:

- BitSet.hh

7.2 quarks::datastructures::BitSet Class Reference

`#include <BitSet.hh>`

7.2.1 Detailed Description

A classic powerset stored in an unsigned integer.

std::bitset<> is a bit too general and, as its implemeentation is hidden, is hard to make a nice bit-queue with.

Definition at line 48 of file BitSet.hh.

Public Member Functions

- BitSet (unsigned value=0)

  Specify the set members directly using an unsigned int.

- template<typename T>
  BitSet (T begin, T end)

  Specify the set members via a range of bit positions.

- unsigned bits () const

  Yield the bitset represented as an unsigned int.

- void set (int i)

  Set a single bit.

- void set ()

  Set all bits.

- void reset (int i)

  Clear one bit.

- void reset ()

  Clear all bits.

- bool has (int i) const

  Is bit i set?.

- void operator|= (const BitSet &other)

  Set union.

- void operator &= (const BitSet &other)

  Set intersection.

- void operator-= (const BitSet &other)

  Set difference.

- void operator^= (const BitSet &other)

Symmetric set difference.

- void complement ()
  Flip all the bits.
- bool empty () const
  Are all bits clear?
- int size () const
  Count the number of set bits.
- void swap (BitSet &other)
  Exchange the members of this set with those of another.
- bool isSubsetOf (const BitSet &other) const
  Are all members of this set also members of another?
- bool isSupersetOf (const BitSet &other) const
  Are all members of another set also members of this one?
- bool operator== (const BitSet &other) const
  Does this set have exactly the same members as another?
- bool operator!= (const BitSet &other) const

Private Attributes

- unsigned _bits

7.2.2 Constructor & Destructor Documentation

7.2.2.1 quarks::datastructures::BitSet::BitSet (unsigned *value* = 0) [inline, explicit]

Specify the set members directly using an unsigned int.

Definition at line 53 of file BitSet.hh.

7.2.2.2 template<typename T> quarks::datastructures::BitSet::BitSet (T *begin*, T *end*) [inline]

Specify the set members via a range of bit positions.

Parameters:

↔ ***begin*** An iterator which refers to an int in the range 0-31.
↔ ***end*** An iterator which refers to an int in the range 0-31.

If dereferencing all the iterators in the range begin to end yields 2, 4, 6 then BitSet(begin, end) yields the same result as BitSet((1U<<2) | (1U<<4) | (1U<<6)).

Definition at line 63 of file BitSet.hh.

References _bits.

7.2.3 Member Function Documentation

7.2.3.1 unsigned quarks::datastructures::BitSet::bits () const [inline]

Yield the bitset represented as an unsigned int.

Definition at line 72 of file BitSet.hh.

References _bits.

7.2.3.2 void quarks::datastructures::BitSet::set (int *i*) [inline]

Set a single bit.

Definition at line 75 of file BitSet.hh.

References _bits.

7.2.3.3 void quarks::datastructures::BitSet::set () [inline]

Set all bits.

Definition at line 77 of file BitSet.hh.

References _bits.

7.2.3.4 void quarks::datastructures::BitSet::reset (int *i*) [inline]

Clear one bit.

Definition at line 79 of file BitSet.hh.

References _bits.

7.2.3.5 void quarks::datastructures::BitSet::reset () [inline]

Clear all bits.

Definition at line 81 of file BitSet.hh.

References _bits.

7.2.3.6 bool quarks::datastructures::BitSet::has (int *i*) const [inline]

Is bit i set?.

Definition at line 83 of file BitSet.hh.

References _bits.

7.2.3.7 void quarks::datastructures::BitSet::operator|= (const BitSet & *other*) [inline]

Set union.

Definition at line 85 of file BitSet.hh.

References _bits.

7.2.3.8 void quarks::datastructures::BitSet::operator &= (const BitSet & *other*) [inline]

Set intersection.

Definition at line 87 of file BitSet.hh.

References _bits.

7.2.3.9 void quarks::datastructures::BitSet::operator-= (const BitSet & *other*) [inline]

Set difference.

Definition at line 89 of file BitSet.hh.

References _bits.

7.2.3.10 void quarks::datastructures::BitSet::operator^= (const BitSet & *other*) [inline]

Symmetric set difference.

Definition at line 91 of file BitSet.hh.

References _bits.

7.2.3.11 void quarks::datastructures::BitSet::complement () [inline]

Flip all the bits.

Definition at line 93 of file BitSet.hh.

References _bits.

Referenced by quarks::datastructures::operator~().

7.2.3.12 bool quarks::datastructures::BitSet::empty () const [inline]

Are all bits clear?

Definition at line 95 of file BitSet.hh.

References _bits.

7.2.3.13 int quarks::datastructures::BitSet::size () const [inline]

Count the number of set bits.

Note:

size() is relatively expensive., O(# of set bits) rather than O(1).

Definition at line 98 of file BitSet.hh.

References _bits.

7.2.3.14 void quarks::datastructures::BitSet::swap (BitSet & *other*) [inline]

Exchange the members of this set with those of another.

Definition at line 110 of file BitSet.hh.

References _bits.

7.2.3.15 bool quarks::datastructures::BitSet::isSubsetOf (const BitSet & *other*) const [inline]

Are all members of this set also members of another?

Definition at line 112 of file BitSet.hh.

References _bits.

7.2.3.16 bool quarks::datastructures::BitSet::isSupersetOf (const BitSet & *other*) const [inline]

Are all members of another set also members of this one?

Definition at line 114 of file BitSet.hh.

References _bits.

7.2.3.17 bool quarks::datastructures::BitSet::operator== (const BitSet & *other*) const [inline]

Does this set have exactly the same members as another?

Definition at line 116 of file BitSet.hh.

References _bits.

7.2.3.18 bool quarks::datastructures::BitSet::operator!= (const BitSet & *other*) const [inline]

Definition at line 118 of file BitSet.hh.

References _bits.

7.2.4 Member Data Documentation

7.2.4.1 unsigned quarks::datastructures::BitSet::_bits [private]

Definition at line 49 of file BitSet.hh.

Referenced by bits(), BitSet(), complement(), empty(), has(), isSubsetOf(), isSupersetOf(), operator &=(), operator!=(), operator-=(), operator==(), operator^=(), operator|=(), reset(), set(), size(), and swap().

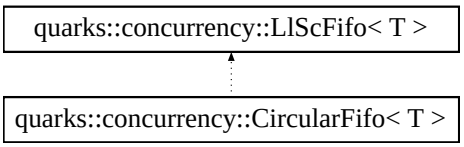
The documentation for this class was generated from the following file:

- BitSet.hh

7.3 quarks::concurrency::CircularFifo< T > Class Template Reference

`#include <CircularFifo.hh>`

Inheritance diagram for quarks::concurrency::CircularFifo< T >::



7.3.1 Detailed Description

template<typename T> class quarks::concurrency::CircularFifo< T >

A generic wrapper for a concurrent FIFO implemented using a circular buffer with a fixed number of slots. T The type of the items to be queued.

Definition at line 49 of file rce405/CircularFifo.hh.

Public Member Functions

- CircularFifo (unsigned maxElements)
- bool enqueue (const T &x)

  Copy in a new value, if possible.

- bool dequeue (T &x)

  Copy out a queued value, if any.

- bool empty () const

  Detect whether the queue is empty. The queue state may change before you have a chance to call another operation.

- bool full () const

  Detect whether the queue is full. The queue state may change before you have a chance to call another operation.

7.3.2 Constructor & Destructor Documentation

7.3.2.1 template<typename T> quarks::concurrency::CircularFifo< T >::CircularFifo (unsigned *maxElements*) [inline]

Allocate the FIFO slots.

Definition at line 52 of file rce405/CircularFifo.hh.

7.3.3 Member Function Documentation

7.3.3.1 template<typename T> bool quarks::concurrency::CircularFifo< T >::enqueue (const T & *x*) [inline]

Copy in a new value, if possible.

Returns:

Was a value enqueued?

Reimplemented from quarks::concurrency::LlScFifo< T >.

Definition at line 56 of file rce405/CircularFifo.hh.

Referenced by anonymous_namespace{testCircularFifo.cc}::Thread::enqueueWait(), and testCircularFifo().

7.3.3.2 template<typename T> bool quarks::concurrency::CircularFifo< T >::dequeue (T & *x*) [inline]

Copy out a queued value, if any.

Returns:

Was a value dequeued?

Reimplemented from quarks::concurrency::LlScFifo< T >.

Definition at line 60 of file rce405/CircularFifo.hh.

Referenced by anonymous_namespace{testCircularFifo.cc}::Thread::dequeueWait(), and testCircularFifo().

7.3.3.3 template<typename T> bool quarks::concurrency::CircularFifo< T >::empty () const [inline]

Detect whether the queue is empty. The queue state may change before you have a chance to call another operation.

Reimplemented from quarks::concurrency::LlScFifo< T >.

Definition at line 64 of file rce405/CircularFifo.hh.

Referenced by testCircularFifo().

7.3.3.4 template<typename T> bool quarks::concurrency::CircularFifo< T >::full () const [inline]

Detect whether the queue is full. The queue state may change before you have a chance to call another operation.

Reimplemented from quarks::concurrency::LlScFifo< T >.

Definition at line 68 of file rce405/CircularFifo.hh.

The documentation for this class was generated from the following file:

- rce405/CircularFifo.hh

7.4 anonymous_namespace{testCircularFifo.cc}::Completion Struct Reference

7.4.1 Detailed Description

Used to perform cleanup after a thread's run() member exits.

Definition at line 138 of file testCircularFifo.cc.

Public Member Functions

- Completion (Thread *thr)
  Save the Thread's address.
- ~Completion ()
  Set our Thread's done flag and delete the OS thread.

Public Attributes

- Thread * _thr
  Pointer to the Thread in question.

7.4.2 Constructor & Destructor Documentation

7.4.2.1 anonymous_namespace{testCircularFifo.cc}::Completion::Completion (Thread * *thr*) [inline]

Save the Thread's address.

Definition at line 142 of file testCircularFifo.cc.

7.4.2.2 anonymous_namespace{testCircularFifo.cc}::Completion::~Completion () [inline]

Set our Thread's done flag and delete the OS thread.

Definition at line 145 of file testCircularFifo.cc.

7.4.3 Member Data Documentation

7.4.3.1 Thread* anonymous_namespace{testCircularFifo.cc}::Completion::_thr

Pointer to the Thread in question.

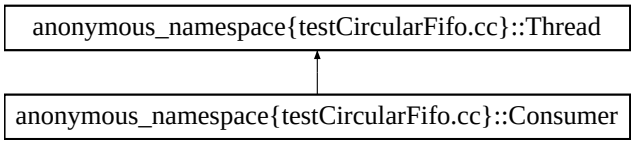
Definition at line 139 of file testCircularFifo.cc.

The documentation for this struct was generated from the following file:

- testCircularFifo.cc

7.5 anonymous_namespace{testCircularFifo.cc}::Consumer Struct Reference

Inheritance diagram for anonymous_namespace{testCircularFifo.cc}::Consumer::



7.5.1 Detailed Description

Receives Nodes and updates a ProdRecord each time.

Definition at line 222 of file testCircularFifo.cc.

Public Member Functions

- Consumer (int id, CircularFifo< Node > &fifo, const set< int > &prodIds)
- int id () const
- void stop ()
- void run ()

  Override this with the implementation of the thread's code.

Public Attributes

- RecordMap records
- int nrecv

Private Attributes

- int _id
- CircularFifo< Node > * _fifo
- volatile bool _go

7.5.2 Constructor & Destructor Documentation

7.5.2.1 anonymous_namespace{testCircularFifo.cc}::Consumer::Consumer (int *id*, CircularFifo< Node > & *fifo*, const set< int > & *prodIds*) [inline]

Create a new ProdRecord for each possible Producer.

Definition at line 224 of file testCircularFifo.cc.

7.5.3 Member Function Documentation

7.5.3.1 int anonymous_namespace{testCircularFifo.cc}::Consumer::id () const [inline]

Definition at line 236 of file testCircularFifo.cc.

7.5.3.2 void anonymous_namespace{testCircularFifo.cc}::Consumer::stop () [inline]

Definition at line 238 of file testCircularFifo.cc.

7.5.3.3 void anonymous_namespace{testCircularFifo.cc}::Consumer::run () [inline, virtual]

Override this with the implementation of the thread's code.

Implements anonymous_namespace{testCircularFifo.cc}::Thread.

Definition at line 240 of file testCircularFifo.cc.

References anonymous_namespace{testCircularFifo.cc}::Node::a, anonymous_namespace{testCircularFifo.cc}::Node::b, anonymous_namespace{testCircularFifo.cc}::ProdRecord::mark(), and anonymous_namespace{testCircularFifo.cc}::Thread::yield().

7.5.4 Member Data Documentation

7.5.4.1 RecordMap anonymous_namespace{testCircularFifo.cc}::Consumer::records

Definition at line 259 of file testCircularFifo.cc.

7.5.4.2 int anonymous_namespace{testCircularFifo.cc}::Consumer::nrecv

Definition at line 260 of file testCircularFifo.cc.

7.5.4.3 int anonymous_namespace{testCircularFifo.cc}::Consumer::_id [private]

Definition at line 262 of file testCircularFifo.cc.

7.5.4.4 CircularFifo<Node>* anonymous_namespace{testCircularFifo.cc}::Consumer::_fifo [private]

Definition at line 263 of file testCircularFifo.cc.

7.5.4.5 volatile bool anonymous_namespace{testCircularFifo.cc}::Consumer::_go [private]

Definition at line 264 of file testCircularFifo.cc.

The documentation for this struct was generated from the following file:

- testCircularFifo.cc

7.6 quarks::concurrency::CriticalSection Class Reference

`#include <CriticalSection.hh>`

7.6.1 Detailed Description

Take a Semaphore on construction, give it back on destruction. Non-copyable.

Creates a critical section using the Resource Allocation Is Initialization strategy. A block of code that needs to execute while holding a Semaphore constructs a CriticalSection object with the Semaphore, which is immediately taken. The Semaphore is given back when the CriticalSection object is destroyed so you don't have to remember to give the Semaphore back yourself. This technique automatically handles nested critical sections correctly since the C++ destructors are called in the order opposite that of the correspondin constructs (the Last Shall Be First rule). Examples:

```
void Foo::func() {
...
CriticalSection cs(globalSem);
// The remaining function body is a critical section.
...
}

void Spam::sausage() {
...
Semaphore lock(Semaphore::Green);
startChildTask(lock);
...
{ CriticalSection cs(lock);
// This block is a critical section.
...
}
}
```

Definition at line 79 of file CriticalSection.hh.

Public Member Functions

- CriticalSection (RceSvc::Semaphore &lock)
- ~CriticalSection ()

Private Attributes

- RceSvc::Semaphore & _lock

7.6.2 Constructor & Destructor Documentation

7.6.2.1 quarks::concurrency::CriticalSection::CriticalSection (RceSvc::Semaphore & *lock*) [inline]

Definition at line 85 of file CriticalSection.hh.

References _lock.

7.6.2.2 quarks::concurrency::CriticalSection::~CriticalSection () [inline]

Definition at line 87 of file CriticalSection.hh.

References _lock.

7.6.3 Member Data Documentation

7.6.3.1 RceSvc::Semaphore& quarks::concurrency::CriticalSection::_lock [private]

Definition at line 81 of file CriticalSection.hh.

Referenced by CriticalSection(), and ~CriticalSection().

The documentation for this class was generated from the following file:

- CriticalSection.hh

7.7 quarks::service::InitLogging Struct Reference

`#include <InitLogging.hh>`

7.7.1 Detailed Description

Set up logging in the constructor and tear it down in the destructor.

Definition at line 48 of file InitLogging.hh.

Public Member Functions

- InitLogging (Logger::Severity sev, LoggerImpl ∗impl)
- ∼InitLogging ()

7.7.2 Constructor & Destructor Documentation

7.7.2.1 quarks::service::InitLogging::InitLogging (Logger::Severity *sev*, LoggerImpl ∗ *impl*) [inline]

Definition at line 49 of file InitLogging.hh.

References quarks::service::Logger::initLogging().

7.7.2.2 quarks::service::InitLogging::∼InitLogging () [inline]

Definition at line 51 of file InitLogging.hh.

References quarks::service::Logger::stopLogging().

The documentation for this struct was generated from the following file:

- InitLogging.hh

7.8 quarks::concurrency::InterruptGuard Class Reference

`#include <syncops.hh>`

7.8.1 Detailed Description

The constructor saves the state of the interrupt enables and disables interrupts. The destructor restores the prior state of the interrupt enables.

Definition at line 230 of file rce405/syncops.hh.

Public Member Functions

- InterruptGuard () __attribute__((always_inline))
- ~InterruptGuard () __attribute__((always_inline))

Private Types

- enum { INTERRUPT_MASK = 0x00028200 }

Private Attributes

- unsigned oldMsr

7.8.2 Member Enumeration Documentation

7.8.2.1 anonymous enum [private]

Enumerator:

INTERRUPT_MASK

Definition at line 231 of file rce405/syncops.hh.

7.8.3 Constructor & Destructor Documentation

7.8.3.1 quarks::concurrency::InterruptGuard::InterruptGuard () [inline]

Definition at line 238 of file rce405/syncops.hh.

7.8.3.2 quarks::concurrency::InterruptGuard::~InterruptGuard () [inline]

Definition at line 240 of file rce405/syncops.hh.

References INTERRUPT_MASK, oldMsr, and quarks::concurrency::setMsrMasked().

7.8.4 Member Data Documentation

7.8.4.1 unsigned quarks::concurrency::InterruptGuard::oldMsr [private]

Definition at line 232 of file rce405/syncops.hh.

Referenced by ~InterruptGuard().

The documentation for this class was generated from the following file:

- rce405/syncops.hh

7.9 quarks::concurrency::IsA32BitType< B > Struct Template Reference

`#include <syncops.hh>`

7.9.1 Detailed Description

template<bool B> struct quarks::concurrency::IsA32BitType< B >

Definition at line 45 of file rce405/syncops.hh.

The documentation for this struct was generated from the following file:

- rce405/syncops.hh

7.10 quarks::concurrency::IsA32BitType< true > Struct Template Reference

`#include <syncops.hh>`

7.10.1 Detailed Description

template<> struct quarks::concurrency::IsA32BitType< true >

Definition at line 47 of file rce405/syncops.hh.

Public Member Functions

- void yes () __attribute__((always_inline))

7.10.2 Member Function Documentation

7.10.2.1 void quarks::concurrency::IsA32BitType< true >::yes ()

Definition at line 51 of file rce405/syncops.hh.

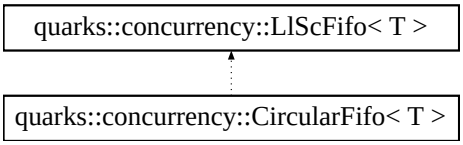
The documentation for this struct was generated from the following file:

- rce405/syncops.hh

7.11 quarks::concurrency::LlScFifo< T > Class Template Reference

`#include <LlScFifo.hh>`

Inheritance diagram for quarks::concurrency::LlScFifo< T >::



7.11.1 Detailed Description

template<typename T> class quarks::concurrency::LlScFifo< T >

A lock-free concurrent FIFO based on a circular array synchronized by the atomic operations Load and Lock (LL) and Store Conditional (SC).

This implementation is based on the one presented by Claude Evequoz in his paper "Non-Blocking Concurrent FIFO With Single Word Synchronization Primitives", in ICPP '08: Proceedings of the 2008 37th International Conference on Parallel Processing, published September 2008 by the IEEE Computer Society.

The constructor creates two arrays. One contains the state of each FIFO slot, the other contains the stored value for each slot. Each slot may be in the state FREE, ALLOCATED or COPYOUT. The value of a free slot is T() and initially all slots are free. Values are copied into and out of slots by assignment. The COPYOUT state is used to prevent a slot from being reallocated before the slot value can be copied out; once that is done the slot value is set to T() and the state is set to FREE.

If you're using this class from an interrupt service routine you have to ensure that T() and T &operator=(const T&) perform only operations permissable at interrupt level.

Depending on what the consumers do after each dequeue operation (e.g., yield()) some consumers may starve if the FIFO has fewer than one slot per consumer.

T The item data type. It must have a default constructor and an assignment operation.

Definition at line 81 of file LlScFifo.hh.

Public Member Functions

- LlScFifo (unsigned maxElements)

  head = tail = 0, set all slots to T().

- void dump (const char *const label)
- ~LlScFifo ()

  Destroy the items for any Allocated slots.

- bool enqueue (const T &newValue)

  Copy a value into the next available slot, if any.

- bool dequeue (T &oldValue)

Copy the next value, if any, out to oldValue.

- bool empty () const

  Inlineable test for an empty FIFO. Doesn't guarantee the success or failure of a subsequent attempt to enqueue or dequeue.

- bool full () const

  Inlineable test for a full FIFO. Doesn't guarantee the success or failure of a subsequent attempt to enqueue or dequeue.

Private Member Functions

- bool _empty (unsigned oldHead)

  A convenience function used by dequeue().

- bool _full (unsigned newTail)

  A convenience function used by enqueue().

Private Attributes

- const unsigned nslots

  The total number of elements in state[] and in value[].

- unsigned head

  The index just before that of the first allocated FIFO slot (modulo nslots).

- unsigned tail

  The index just before that of the next unused FIFO slot (modulo nslots).

- unsigned *const state

  The slot states.

- T *const value

  The slot values.

- T defaultValue

  Empty slots have this value.

Static Private Attributes

- static const unsigned FREE = 0
- static const unsigned ALLOCATED = 1
- static const unsigned COPYOUT = 2
- static const char * name [3] = {"FREE", "ALLOCATED", "COPYOUT"}

7.11.2 Constructor & Destructor Documentation

7.11.2.1 template<typename T> quarks::concurrency::LlScFifo< T >::LlScFifo (unsigned *maxElements*) [inline]

head = tail = 0, set all slots to T().

Parameters:

← ***maxElements*** The maximum number of elements that can be queued.

Definition at line 104 of file LlScFifo.hh.

References quarks::concurrency::LlScFifo< T >::FREE, and quarks::concurrency::LlScFifo< T >::nslots.

7.11.2.2 template<typename T> quarks::concurrency::LlScFifo< T >::~LlScFifo () [inline]

Destroy the items for any Allocated slots.

Definition at line 123 of file LlScFifo.hh.

References quarks::concurrency::LlScFifo< T >::state, and quarks::concurrency::LlScFifo< T >::value.

7.11.3 Member Function Documentation

7.11.3.1 template<typename T> bool quarks::concurrency::LlScFifo< T >::_empty (unsigned *oldHead*) [inline, private]

A convenience function used by dequeue().

Definition at line 96 of file LlScFifo.hh.

References quarks::concurrency::LlScFifo< T >::tail.

7.11.3.2 template<typename T> bool quarks::concurrency::LlScFifo< T >::_full (unsigned *newTail*) [inline, private]

A convenience function used by enqueue().

Definition at line 99 of file LlScFifo.hh.

References quarks::concurrency::LlScFifo< T >::head.

7.11.3.3 template<typename T> void quarks::concurrency::LlScFifo< T >::dump (const char *const *label*) [inline]

Definition at line 115 of file LlScFifo.hh.

References quarks::concurrency::LlScFifo< T >::name.

7.11.3.4 template<typename T> bool quarks::concurrency::LlScFifo< T >::enqueue (const T & *newValue*) [inline]

Copy a value into the next available slot, if any.

Parameters:

← ***newValue*** The enqueued value is copied from this (if possible).

Returns:

Was a value enqueued?

Reimplemented in quarks::concurrency::CircularFifo< T >, and quarks::concurrency::CircularFifo< anonymous_namespace{testCircularFifo.cc}::Node >.

Definition at line 131 of file LlScFifo.hh.

References quarks::concurrency::LlScFifo< T >::ALLOCATED, quarks::concurrency::LlScFifo< T >::FREE, quarks::concurrency::LL(), quarks::concurrency::SC(), quarks::concurrency::LlScFifo< T >::tail, and quarks::concurrency::LlScFifo< T >::value.

7.11.3.5 template<typename T> bool quarks::concurrency::LlScFifo< T >::dequeue (T & *oldValue*) [inline]

Copy the next value, if any, out to oldValue.

Parameters:

→ ***oldValue*** The dequeued value (if any) is copied to this.

Returns:

Was a value dequeued?

Reimplemented in quarks::concurrency::CircularFifo< T >, and quarks::concurrency::CircularFifo< anonymous_namespace{testCircularFifo.cc}::Node >.

Definition at line 161 of file LlScFifo.hh.

References quarks::concurrency::LlScFifo< T >::COPYOUT, quarks::concurrency::LlScFifo< T >::defaultValue, quarks::concurrency::LlScFifo< T >::FREE, quarks::concurrency::LlScFifo< T >::head, quarks::concurrency::LL(), quarks::concurrency::SC(), quarks::concurrency::LlScFifo< T >::state, and quarks::concurrency::LlScFifo< T >::value.

7.11.3.6 template<typename T> bool quarks::concurrency::LlScFifo< T >::empty () const [inline]

Inlineable test for an empty FIFO. Doesn't guarantee the success or failure of a subsequent attempt to enqueue or dequeue.

Reimplemented in quarks::concurrency::CircularFifo< T >, and quarks::concurrency::CircularFifo< anonymous_namespace{testCircularFifo.cc}::Node >.

Definition at line 190 of file LlScFifo.hh.

References quarks::concurrency::LlScFifo< T >::head, and quarks::concurrency::LlScFifo< T >::tail.

7.11.3.7 template<typename T> bool quarks::concurrency::LlScFifo< T >::full () const [inline]

Inlineable test for a full FIFO. Doesn't guarantee the success or failure of a subsequent attempt to enqueue or dequeue.

Reimplemented in quarks::concurrency::CircularFifo< T >, and quarks::concurrency::CircularFifo< anonymous_namespace{testCircularFifo.cc}::Node >.

Definition at line 195 of file LlScFifo.hh.

References quarks::concurrency::LlScFifo< T >::head, and quarks::concurrency::LlScFifo< T >::tail.

7.11.4 Member Data Documentation

7.11.4.1 template<typename T> const unsigned quarks::concurrency::LlScFifo< T >::FREE = 0 [static, private]

Definition at line 83 of file LlScFifo.hh.

Referenced by quarks::concurrency::LlScFifo< T >::dequeue(), quarks::concurrency::LlScFifo< T >::enqueue(), and quarks::concurrency::LlScFifo< T >::LlScFifo().

7.11.4.2 template<typename T> const unsigned quarks::concurrency::LlScFifo< T >::ALLOCATED = 1 [static, private]

Definition at line 84 of file LlScFifo.hh.

Referenced by quarks::concurrency::LlScFifo< T >::enqueue().

7.11.4.3 template<typename T> const unsigned quarks::concurrency::LlScFifo< T >::COPYOUT = 2 [static, private]

Definition at line 85 of file LlScFifo.hh.

Referenced by quarks::concurrency::LlScFifo< T >::dequeue().

7.11.4.4 template<typename T> const char * quarks::concurrency::LlScFifo< T >::name = {"FREE", "ALLOCATED", "COPYOUT"} [inline, static, private]

Definition at line 86 of file LlScFifo.hh.

Referenced by quarks::concurrency::LlScFifo< T >::dump().

7.11.4.5 template<typename T> const unsigned quarks::concurrency::LlScFifo< T >::nslots [private]

The total number of elements in state[] and in value[].

Definition at line 88 of file LlScFifo.hh.

Referenced by quarks::concurrency::LlScFifo< T >::LlScFifo().

7.11.4.6 template<typename T> unsigned quarks::concurrency::LlScFifo< T >::head [private]

The index just before that of the first allocated FIFO slot (modulo nslots).

Definition at line 89 of file LlScFifo.hh.

Referenced by quarks::concurrency::LlScFifo< T >::_full(), quarks::concurrency::LlScFifo< T >::dequeue(), quarks::concurrency::LlScFifo< T >::empty(), and quarks::concurrency::LlScFifo< T >::full().

7.11.4.7 template<typename T> unsigned quarks::concurrency::LlScFifo< T >::tail [private]

The index just before that of the next unused FIFO slot (modulo nslots).

Definition at line 90 of file LlScFifo.hh.

Referenced by quarks::concurrency::LlScFifo< T >::_empty(), quarks::concurrency::LlScFifo< T >::empty(), quarks::concurrency::LlScFifo< T >::enqueue(), and quarks::concurrency::LlScFifo< T >::full().

7.11.4.8 template<typename T> unsigned* const quarks::concurrency::LlScFifo< T >::state [private]

The slot states.

Definition at line 91 of file LlScFifo.hh.

Referenced by quarks::concurrency::LlScFifo< T >::dequeue(), and quarks::concurrency::LlScFifo< T >::~LlScFifo().

7.11.4.9 template<typename T> T* const quarks::concurrency::LlScFifo< T >::value [private]

The slot values.

Definition at line 92 of file LlScFifo.hh.

Referenced by quarks::concurrency::LlScFifo< T >::dequeue(), quarks::concurrency::LlScFifo< T >::enqueue(), and quarks::concurrency::LlScFifo< T >::~LlScFifo().

7.11.4.10 template<typename T> T quarks::concurrency::LlScFifo< T >::defaultValue [private]

Empty slots have this value.

Definition at line 93 of file LlScFifo.hh.

Referenced by quarks::concurrency::LlScFifo< T >::dequeue().

The documentation for this class was generated from the following file:

- LlScFifo.hh

7.12 quarks::service::Logger Class Reference

`#include <Logger.hh>`

7.12.1 Detailed Description

Handle log messages from applications.

All logging is delegated to a single instance of LoggerImpl which is set using the InitLogging class. Instances of Logger have no state of their own so you can just create them as you need them, e.g.:

```
Logger().info("My %s has no nose.", pet);

LogMessage(Logger::Info).add("How does ").add("he smell?");

{ LogMessage msg(Logger::Info);
  msg.add("Awful!");
}
```

Messages with severity less than the threshold set with InitLogging are never formatted or sent. Your formats need not end with a newline because one is always appended to the end of a message.

The member functions log(), debug(), info(), etc., each create and send a single message. The begin() member function creates a message to which you can add to piecemeal. The message is sent when the LogMessage object is destroyed. In the second example above that happens at the semicolon. In the third example it happens at the right brace.

To initialize logging allocate with new an instance of a class derived from LoggerImpl and pass the pointer to the constructor of InitLogging along with the minimum severity you want to appear in the final log. Logging will be cleanly shut down once the InitLogging object is destroyed, so normally you'd construct it in your main function.

```
#include "quarks/service/InitLogging.hh"
using quarks::service::InitLogging;

#include "quarks/service/Logger.hh"
using quarks::service::Logger;

#include "quarks/service/PtyLogger.hh"
using quarks::service::PtyLogger;

InitLogging il(Logger::Info, new PtyLogger());
```

If you need to skip some code based on the logging level then use aboveThreshold():

```
if (Logger().aboveThreshold(Logger::Debug)) {
   Debugging code ...
}
```

This class' design is a variation on the Flyweight pattern in which all instances share the same state. In Python programming this is called the Borg pattern.

Definition at line 108 of file Logger.hh.

Public Types

- enum Severity {

Debug, Info, Warning, Error,

Fatal }

The severity levels for log messages.

Public Member Functions

- Severity threshold (Severity sev) const

  Return the severity threshold.

- bool overThreshold (Severity sev) const

  Is the given severity level over the threshold?

- bool needTimestamps () const

  Do we need to add a timestamp to each message?

- void logString (Severity, const char *msg)

  Send a message that's a preformatted string.

- void log (Severity, const char *format,...)

  Log with an adjustable severity.

- void vlog (Severity, const char *format, std::va_list ap)
- void debug (const char *format,...)

  Log with severity Debug.

- void info (const char *format,...)

  Log with severity Info.

- void warning (const char *format,...)

  Log with severity Warning.

- void error (const char *format,...)

  Log with severity Error.

- void fatal (const char *format,...)

  Log with severity Fatal.

Static Public Member Functions

- static void initLogging (Severity thresh, LoggerImpl *imp)

  Mandatory setup for logging.

- static void stopLogging ()

  Free the resources allocated by initLogging().

Private Member Functions

- LoggerImpl & currentLogger () const
- void send (const char *msg, size_t nbytes)

Static Private Attributes

- static Severity _threshold
- static std::tr1::shared_ptr< LoggerImpl > _currentLogger
- static RceSvc::Semaphore _lock

Friends

- class LogMessage

7.12.2 Member Enumeration Documentation

7.12.2.1 enum quarks::service::Logger::Severity

The severity levels for log messages.

Enumerator:

Debug

Info

Warning

Error

Fatal

Definition at line 113 of file Logger.hh.

7.12.3 Member Function Documentation

7.12.3.1 void quarks::service::Logger::initLogging (Severity *thresh*, LoggerImpl * *imp*) [static]

Mandatory setup for logging.

Parameters:

← ***thresh*** The logging severity threshold.

↔ ***imp*** A pointer to a LoggerImpl instance assumed to be allocated by new; ownership is transferred to the logging system.

Must be called at least once before logging anything. Any prior threshold is replaced and any prior LoggerImpl instance is destroyed.

Note:

Normally this function is called using class InitLogging.

Definition at line 63 of file Logger.cc.

References _currentLogger, and _threshold.

Referenced by quarks::service::InitLogging::InitLogging().

7.12.3.2 void quarks::service::Logger::stopLogging () [static]

Free the resources allocated by initLogging().

Definition at line 69 of file Logger.cc.

References _currentLogger.

Referenced by quarks::service::InitLogging::~InitLogging().

7.12.3.3 Severity quarks::service::Logger::threshold (Severity *sev*) const [inline]

Return the severity threshold.

Definition at line 131 of file Logger.hh.

References _threshold.

7.12.3.4 bool quarks::service::Logger::overThreshold (Severity *sev*) const [inline]

Is the given severity level over the threshold?

Definition at line 134 of file Logger.hh.

References _threshold.

7.12.3.5 bool quarks::service::Logger::needTimestamps () const

Do we need to add a timestamp to each message?

A timestamp may be added by the system, depending on the destination of the messages, e.g., Unix syslog.

Definition at line 71 of file Logger.cc.

References currentLogger(), and quarks::service::LoggerImpl::needTimestamps().

7.12.3.6 void quarks::service::Logger::logString (Severity *sev*, const char * *msg*)

Send a message that's a preformatted string.

Definition at line 78 of file Logger.cc.

References LogMessage.

7.12.3.7 void quarks::service::Logger::log (Severity *sev*, const char * *format*, ...)

Log with an adjustable severity.

Definition at line 84 of file Logger.cc.

References LogMessage.

7.12.3.8 void quarks::service::Logger::vlog (Severity, const char * *format*, std::va_list *ap*)

Use this form when you have a va_list already initialized.

7.12.3.9 void quarks::service::Logger::debug (const char * *format*, ...)

Log with severity Debug.

Definition at line 95 of file Logger.cc.

References Debug, and LogMessage.

7.12.3.10 void quarks::service::Logger::info (const char * *format*, ...)

Log with severity Info.

Definition at line 102 of file Logger.cc.

References Info, and LogMessage.

7.12.3.11 void quarks::service::Logger::warning (const char * *format*, ...)

Log with severity Warning.

Definition at line 109 of file Logger.cc.

References LogMessage, and Warning.

7.12.3.12 void quarks::service::Logger::error (const char * *format*, ...)

Log with severity Error.

Definition at line 116 of file Logger.cc.

References Error, and LogMessage.

7.12.3.13 void quarks::service::Logger::fatal (const char * *format*, ...)

Log with severity Fatal.

Definition at line 123 of file Logger.cc.

References Fatal, and LogMessage.

7.12.3.14 LoggerImpl& quarks::service::Logger::currentLogger () const [inline, private]

Definition at line 170 of file Logger.hh.

References _currentLogger.

Referenced by needTimestamps(), and send().

7.12.3.15 void quarks::service::Logger::send (const char * *msg*, size_t *nbytes*) [private]

Definition at line 73 of file Logger.cc.

References _lock, currentLogger(), and quarks::service::LoggerImpl::send().

7.12.4 Friends And Related Function Documentation

7.12.4.1 friend class LogMessage [friend]

Definition at line 168 of file Logger.hh.

Referenced by debug(), error(), fatal(), info(), log(), logString(), and warning().

7.12.5 Member Data Documentation

7.12.5.1 Logger::Severity quarks::service::Logger::_threshold [static, private]

Definition at line 174 of file Logger.hh.

Referenced by initLogging(), overThreshold(), and threshold().

7.12.5.2 shared_ptr< LoggerImpl > quarks::service::Logger::_currentLogger [static, private]

Definition at line 176 of file Logger.hh.

Referenced by currentLogger(), initLogging(), and stopLogging().

7.12.5.3 Semaphore quarks::service::Logger::_lock [static, private]

Definition at line 178 of file Logger.hh.

Referenced by send().

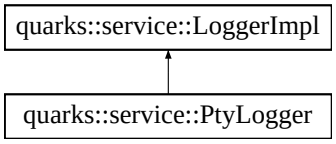
The documentation for this class was generated from the following files:

- Logger.hh
- Logger.cc

7.13 quarks::service::LoggerImpl Class Reference

`#include <LoggerImpl.hh>`

Inheritance diagram for quarks::service::LoggerImpl::



7.13.1 Detailed Description

Abstract base class for logging implementations.

Logger::_currentLogger is a shared_ptr to an instance of a class derived from this base class. A logger implementation actually carries out delivery of messages and need not worry about concurrency or about the severity threshold.

See also:

Logger

Definition at line 54 of file LoggerImpl.hh.

Public Member Functions

- virtual void send (const char *msg, size_t nbytes)=0

  Deliver a log message, already formatted.

- virtual bool needTimestamps () const =0

  Should messages already have timestamps?

- virtual ~LoggerImpl ()

7.13.2 Constructor & Destructor Documentation

7.13.2.1 virtual quarks::service::LoggerImpl::~LoggerImpl () [inline, virtual]

Definition at line 65 of file LoggerImpl.hh.

7.13.3 Member Function Documentation

7.13.3.1 virtual void quarks::service::LoggerImpl::send (const char * *msg*, size_t *nbytes*) [pure virtual]

Deliver a log message, already formatted.

Parameters:

← ***msg*** The message to send. Need not be null terminated.

← ***nbytes*** The number of bytes to send (not counting any terminating null).

Implemented in quarks::service::PtyLogger.

Referenced by quarks::service::Logger::send().

7.13.3.2 virtual bool quarks::service::LoggerImpl::needTimestamps () const [pure virtual]

Should messages already have timestamps?

Implemented in quarks::service::PtyLogger.

Referenced by quarks::service::Logger::needTimestamps().

The documentation for this class was generated from the following file:

- LoggerImpl.hh

7.14 quarks::service::LogMessage Class Reference

`#include <LogMessage.hh>`

7.14.1 Detailed Description

Assembles a log message in a buffer.

Non-copyable.

See also:

Logger

Definition at line 53 of file LogMessage.hh.

Public Member Functions

- LogMessage (Logger::Severity)
  Determine whether the message will be sent.
- ~LogMessage ()
  Send the accumulated message string.
- LogMessage & add (const char *format,...)
  Add with formatting to the message string.
- LogMessage & vadd (const char *format, std::va_list)
  Add with formatting using an already initialized va_list.

Static Public Member Functions

- static size_t initialBufferSize ()
  Return the initial number of bytes allocated for each message.
- static void initialBufferSize (size_t s)
  Set the initial number of bytes allocated for each message.

Private Attributes

- size_t _messageSize
- const bool _willSend
- std::vector< char > _buffer

Static Private Attributes

- static size_t _initialBufferSize = 200

7.14.2 Constructor & Destructor Documentation

7.14.2.1 quarks::service::LogMessage::LogMessage (Logger::Severity *s*)

Determine whether the message will be sent.

Definition at line 83 of file LogMessage.cc.

References _willSend, and anonymous_namespace{LogMessage.cc}::addProlog().

7.14.2.2 quarks::service::LogMessage::~LogMessage ()

Send the accumulated message string.

Definition at line 91 of file LogMessage.cc.

References _buffer, _messageSize, and _willSend.

7.14.3 Member Function Documentation

7.14.3.1 static size_t quarks::service::LogMessage::initialBufferSize () [inline, static]

Return the initial number of bytes allocated for each message.

Definition at line 56 of file LogMessage.hh.

References _initialBufferSize.

7.14.3.2 static void quarks::service::LogMessage::initialBufferSize (size_t *s*) [inline, static]

Set the initial number of bytes allocated for each message.

Definition at line 59 of file LogMessage.hh.

References _initialBufferSize.

7.14.3.3 LogMessage & quarks::service::LogMessage::add (const char * *format*, ...)

Add with formatting to the message string.

Returns:

A reference to this object so that adds may be chained.

Definition at line 124 of file LogMessage.cc.

References vadd().

Referenced by anonymous_namespace{LogMessage.cc}::addProlog(), and testLogging().

7.14.3.4 LogMessage& quarks::service::LogMessage::vadd (const char * *format*, std::va_list)

Add with formatting using an already initialized va_list.

Returns:

A reference to this object so that adds may be chained.

Referenced by add().

7.14.4 Member Data Documentation

7.14.4.1 size_t quarks::service::LogMessage::_initialBufferSize = 200 [static, private]

Definition at line 77 of file LogMessage.hh.

Referenced by initialBufferSize().

7.14.4.2 size_t quarks::service::LogMessage::_messageSize [private]

Definition at line 79 of file LogMessage.hh.

Referenced by ~LogMessage().

7.14.4.3 const bool quarks::service::LogMessage::_willSend [private]

Definition at line 81 of file LogMessage.hh.

Referenced by LogMessage(), and ~LogMessage().

7.14.4.4 std::vector<char> quarks::service::LogMessage::_buffer [private]

Definition at line 83 of file LogMessage.hh.

Referenced by ~LogMessage().

The documentation for this class was generated from the following files:

- LogMessage.hh
- LogMessage.cc

7.15 anonymous_namespace{testCircularFifo.cc}::Node Struct Reference

7.15.1 Detailed Description

The values placed in the queue.

Definition at line 67 of file testCircularFifo.cc.

Public Member Functions

- Node ()
- Node (int x, int y)

Public Attributes

- int a
- int b

7.15.2 Constructor & Destructor Documentation

7.15.2.1 anonymous_namespace{testCircularFifo.cc}::Node::Node () [inline]

Definition at line 70 of file testCircularFifo.cc.

7.15.2.2 anonymous_namespace{testCircularFifo.cc}::Node::Node (int *x*, int *y*) [inline]

Definition at line 71 of file testCircularFifo.cc.

7.15.3 Member Data Documentation

7.15.3.1 int anonymous_namespace{testCircularFifo.cc}::Node::a

Definition at line 68 of file testCircularFifo.cc.

Referenced by anonymous_namespace{testCircularFifo.cc}::operator==(), and anonymous_namespace{testCircularFifo.cc}::Consumer::run().

7.15.3.2 int anonymous_namespace{testCircularFifo.cc}::Node::b

Definition at line 69 of file testCircularFifo.cc.

Referenced by anonymous_namespace{testCircularFifo.cc}::operator==(), and anonymous_namespace{testCircularFifo.cc}::Consumer::run().

The documentation for this struct was generated from the following file:

- testCircularFifo.cc

7.16 quarks::concurrency::NoInterruptFifo< T > Class Template Reference

`#include <NoInterruptFifo.hh>`

7.16.1 Detailed Description

template<typename T> class quarks::concurrency::NoInterruptFifo< T >

A concurrent FIFO based on a circular array synchronized by locked critical sections. Locking is achieved by disabling interrupts within the critical sections.

Unused slots in the array have the value T(). Values are copied in and out of the array using assignment.

T The item data type. It must have a default constructor and an assignment operation both of which can operate while interrupts are disabled.

Definition at line 61 of file NoInterruptFifo.hh.

Public Member Functions

- NoInterruptFifo (unsigned maxElements)

  head = tail = 0, set all slots to T().

- void dump (const char *const label)
- ~NoInterruptFifo ()

  Destroy the items still enqueued in slots.

- bool enqueue (const T &newValue)

  Copy a value into the next available slot, if any.

- bool dequeue (T &oldValue)

  Copy the next value, if any, out to oldValue.

- bool empty ()

  Inlineable test for an empty FIFO. Doesn't guarantee the success or failure of a subsequent attempt to enqueue or dequeue.

- bool full ()

  Inlineable test for a full FIFO. Doesn't guarantee the success or failure of a subsequent attempt to enqueue or dequeue.

Private Attributes

- const unsigned nslots

  The total number of elements in state[].

- unsigned head

  The index just before that of the first allocated FIFO slot (modulo nslots).

- unsigned tail

  The index just before that of the next unused FIFO slot (modulo nslots).

- T *const value

  The slot values.

7.16.2 Constructor & Destructor Documentation

7.16.2.1 template<typename T> quarks::concurrency::NoInterruptFifo< T >::NoInterruptFifo (unsigned *maxElements*) [inline]

head = tail = 0, set all slots to T().

Parameters:

← ***maxElements*** The maximum number of elements that can be queued.

Definition at line 71 of file NoInterruptFifo.hh.

7.16.2.2 template<typename T> quarks::concurrency::NoInterruptFifo< T >::~NoInterruptFifo () [inline]

Destroy the items still enqueued in slots.

Definition at line 84 of file NoInterruptFifo.hh.

References quarks::concurrency::NoInterruptFifo< T >::value.

7.16.3 Member Function Documentation

7.16.3.1 template<typename T> void quarks::concurrency::NoInterruptFifo< T >::dump (const char *const *label*) [inline]

Definition at line 78 of file NoInterruptFifo.hh.

7.16.3.2 template<typename T> bool quarks::concurrency::NoInterruptFifo< T >::enqueue (const T & *newValue*) [inline]

Copy a value into the next available slot, if any.

Parameters:

← ***newValue*** The enqueued value is copied from this (if possible).

Returns:

Was there room for the insertion?

Definition at line 91 of file NoInterruptFifo.hh.

References quarks::concurrency::NoInterruptFifo< T >::tail, and quarks::concurrency::NoInterruptFifo< T >::value.

7.16.3.3 template<typename T> bool quarks::concurrency::NoInterruptFifo< T >::dequeue (T & *oldValue*) [inline]

Copy the next value, if any, out to oldValue.

Parameters:

→ ***oldValue*** The dequeued value (if any) is copied to this.

Returns:

Was the FIFO nonempty?

Definition at line 106 of file NoInterruptFifo.hh.

References quarks::concurrency::NoInterruptFifo< T >::head.

7.16.3.4 template<typename T> bool quarks::concurrency::NoInterruptFifo< T >::empty () [inline]

Inlineable test for an empty FIFO. Doesn't guarantee the success or failure of a subsequent attempt to enqueue or dequeue.

Definition at line 120 of file NoInterruptFifo.hh.

References quarks::concurrency::NoInterruptFifo< T >::head, and quarks::concurrency::NoInterruptFifo< T >::tail.

7.16.3.5 template<typename T> bool quarks::concurrency::NoInterruptFifo< T >::full () [inline]

Inlineable test for a full FIFO. Doesn't guarantee the success or failure of a subsequent attempt to enqueue or dequeue.

Definition at line 125 of file NoInterruptFifo.hh.

References quarks::concurrency::NoInterruptFifo< T >::head, and quarks::concurrency::NoInterruptFifo< T >::tail.

7.16.4 Member Data Documentation

7.16.4.1 template<typename T> const unsigned quarks::concurrency::NoInterruptFifo< T >::nslots [private]

The total number of elements in state[].

Definition at line 63 of file NoInterruptFifo.hh.

7.16.4.2 template<typename T> unsigned quarks::concurrency::NoInterruptFifo< T >::head [private]

The index just before that of the first allocated FIFO slot (modulo nslots).

Definition at line 64 of file NoInterruptFifo.hh.

Referenced by quarks::concurrency::NoInterruptFifo< T >::dequeue(), quarks::concurrency::NoInterruptFifo< T >::empty(), and quarks::concurrency::NoInterruptFifo< T >::full().

7.16.4.3 template<typename T> unsigned quarks::concurrency::NoInterruptFifo< T >::tail [private]

The index just before that of the next unused FIFO slot (modulo nslots).

Definition at line 65 of file NoInterruptFifo.hh.

Referenced by quarks::concurrency::NoInterruptFifo< T >::empty(), quarks::concurrency::NoInterruptFifo< T >::enqueue(), and quarks::concurrency::NoInterruptFifo< T >::full().

7.16.4.4 template<typename T> T* const quarks::concurrency::NoInterruptFifo< T >::value [private]

The slot values.

Definition at line 66 of file NoInterruptFifo.hh.

Referenced by quarks::concurrency::NoInterruptFifo< T >::enqueue(), and quarks::concurrency::NoInterruptFifo< T >::~NoInterruptFifo().

The documentation for this class was generated from the following file:

- NoInterruptFifo.hh

7.17 anonymous_namespace{testCircularFifo.cc}::ProdRecord Struct Reference

7.17.1 Detailed Description

The record kept by each Consumer for each Producer.

Definition at line 187 of file testCircularFifo.cc.

Public Member Functions

- ProdRecord ()
- void mark (int consId, int prodId, int seqno)

  Update a Producer record after dequeueing a Node.

Public Attributes

- bool seqerr

  Has a sequence error been detected?

- int maxSeq

  The greatest sequence number seen.

- set< int > seqSet

  The set of all sequence numbers seen.

7.17.2 Constructor & Destructor Documentation

7.17.2.1 anonymous_namespace{testCircularFifo.cc}::ProdRecord::ProdRecord () [inline]

Definition at line 192 of file testCircularFifo.cc.

7.17.3 Member Function Documentation

7.17.3.1 void anonymous_namespace{testCircularFifo.cc}::ProdRecord::mark (int *consId*, int *prodId*, int *seqno*) [inline]

Update a Producer record after dequeueing a Node.

Parameters:

← ***consId*** Which consumer received the Node.

← ***prodId*** Which Producer sent the Node.

← ***seqno*** The sequence number

Definition at line 199 of file testCircularFifo.cc.

Referenced by anonymous_namespace{testCircularFifo.cc}::Consumer::run().

7.17.4 Member Data Documentation

7.17.4.1 bool anonymous_namespace{testCircularFifo.cc}::ProdRecord::seqerr

Has a sequence error been detected?

Definition at line 188 of file testCircularFifo.cc.

7.17.4.2 int anonymous_namespace{testCircularFifo.cc}::ProdRecord::maxSeq

The greatest sequence number seen.

Definition at line 189 of file testCircularFifo.cc.

7.17.4.3 set<int> anonymous_namespace{testCircularFifo.cc}::ProdRecord::seqSet

The set of all sequence numbers seen.

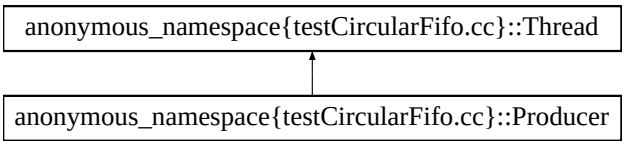
Definition at line 190 of file testCircularFifo.cc.

The documentation for this struct was generated from the following file:

- testCircularFifo.cc

7.18 anonymous_namespace{testCircularFifo.cc}::Producer Struct Reference

Inheritance diagram for anonymous_namespace{testCircularFifo.cc}::Producer::



7.18.1 Detailed Description

Enqueues a fixed number of Nodes, yielding after each enqueue, then exits. Each node contains the producer ID and a monotonically increasing sequence number.

Definition at line 160 of file testCircularFifo.cc.

Public Member Functions

- Producer (int id, CircularFifo< Node > &fifo, const set< int > &seqnos)

  Parameters:

  ← ***seqnos*** *The set of Node sequence numbers to use.*

- int id () const
- void run ()

  Override this with the implementation of the thread's code.

Private Attributes

- const int _id
- CircularFifo< Node > * _fifo
- const set< int > * _seqnos

7.18.2 Constructor & Destructor Documentation

7.18.2.1 anonymous_namespace{testCircularFifo.cc}::Producer::Producer (int *id*, CircularFifo< Node > & *fifo*, const set< int > & *seqnos*) [inline]

Parameters:

← ***seqnos*** The set of Node sequence numbers to use.

Definition at line 162 of file testCircularFifo.cc.

7.18.3 Member Function Documentation

7.18.3.1 int anonymous_namespace{testCircularFifo.cc}::Producer::id () const [inline]

Definition at line 169 of file testCircularFifo.cc.

7.18.3.2 void anonymous_namespace{testCircularFifo.cc}::Producer::run () [inline, virtual]

Override this with the implementation of the thread's code.

Implements anonymous_namespace{testCircularFifo.cc}::Thread.

Definition at line 171 of file testCircularFifo.cc.

References anonymous_namespace{testCircularFifo.cc}::Thread::enqueueWait(), and anonymous_namespace{testCircularFifo.cc}::Thread::yield().

7.18.4 Member Data Documentation

7.18.4.1 const int anonymous_namespace{testCircularFifo.cc}::Producer::_id [private]

Definition at line 180 of file testCircularFifo.cc.

7.18.4.2 CircularFifo<Node>* anonymous_namespace{testCircularFifo.cc}::Producer::_fifo [private]

Definition at line 181 of file testCircularFifo.cc.

7.18.4.3 const set<int>* anonymous_namespace{testCircularFifo.cc}::Producer::_seqnos [private]

Definition at line 182 of file testCircularFifo.cc.

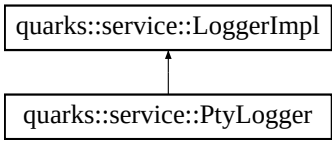
The documentation for this struct was generated from the following file:

- testCircularFifo.cc

7.19 quarks::service::PtyLogger Class Reference

`#include <PtyLogger.hh>`

Inheritance diagram for quarks::service::PtyLogger::



7.19.1 Detailed Description

Write log messages to /dev/pty0. If the PTY can't be opened then log messages will go to fd 1 (stdout).

The shell available for core.x.y.devel opens the telnet session socket as /dev/pty0 so writing to that device will send output to your terminal rather than to the syslog.

Definition at line 55 of file PtyLogger.hh.

Public Member Functions

- PtyLogger ()
- virtual void send (const char *msg, size_t nbytes)

  Deliver a preformatted log message.
- virtual bool needTimestamps () const

  Should the preformatted messages already have timestamps?
- virtual ~PtyLogger ()

Private Attributes

- int _fd

  The file descriptor to which output will go.

7.19.2 Constructor & Destructor Documentation

7.19.2.1 quarks::service::PtyLogger::PtyLogger ()

Definition at line 59 of file PtyLogger.cc.

References _fd, and anonymous_namespace{PtyLogger.cc}::Stderr.

7.19.2.2 quarks::service::PtyLogger::~PtyLogger () [virtual]

Definition at line 89 of file PtyLogger.cc.

References _fd, and anonymous_namespace{PtyLogger.cc}::Stderr.

7.19.3 Member Function Documentation

7.19.3.1 void quarks::service::PtyLogger::send (const char * *msg*, size_t *nbytes*) [virtual]

Deliver a preformatted log message.

Implements quarks::service::LoggerImpl.

Definition at line 71 of file PtyLogger.cc.

References _fd, anonymous_namespace{PtyLogger.cc}::Stderr, and quarks::io::writeAll().

7.19.3.2 bool quarks::service::PtyLogger::needTimestamps () const [virtual]

Should the preformatted messages already have timestamps?

Implements quarks::service::LoggerImpl.

Definition at line 79 of file PtyLogger.cc.

References _fd, and anonymous_namespace{PtyLogger.cc}::Stderr.

7.19.4 Member Data Documentation

7.19.4.1 int quarks::service::PtyLogger::_fd [private]

The file descriptor to which output will go.

Definition at line 57 of file PtyLogger.hh.

Referenced by needTimestamps(), PtyLogger(), send(), and ~PtyLogger().

The documentation for this class was generated from the following files:

- PtyLogger.hh
- PtyLogger.cc

7.20 SayTime Struct Reference

7.20.1 Detailed Description

Definition at line 43 of file rce405/testTiming.cc.

Public Member Functions

- void takeTime (unsigned long long ticks, unsigned rpt, unsigned id, const char ∗comment)

7.20.2 Member Function Documentation

7.20.2.1 void SayTime::takeTime (unsigned long long *ticks*, unsigned *rpt*, unsigned *id*, const char ∗ *comment*) [inline]

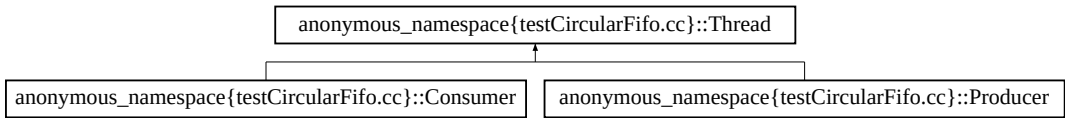
Definition at line 44 of file rce405/testTiming.cc.

The documentation for this struct was generated from the following file:

- rce405/testTiming.cc

7.21 anonymous_namespace{testCircularFifo.cc}::Thread Struct Reference

Inheritance diagram for anonymous_namespace{testCircularFifo.cc}::Thread::



7.21.1 Detailed Description

The abstract base class for producers and consumers. Non-copyable.

Definition at line 78 of file testCircularFifo.cc.

Public Member Functions

- Thread ()
- void start ()

  Create and start a thread/task/whatever as provided by the OS.

- virtual void run ()=0

  Override this with the implementation of the thread's code.

- void yield () const

  Yield the CPU, get rescheduled behind all threads of the same priority.

- void join () const

  Wait for run() to exit.

- void enqueueWait (CircularFifo< Node > *fifo, const Node &d)

  Endless loop of (Try to enqueue, yield if it fails.).

- void dequeueWait (CircularFifo< Node > *fifo, Node &d)
- virtual ~Thread ()

Private Member Functions

- Thread (const Thread &)
- Thread & operator= (const Thread &)

Static Private Member Functions

- static rtems_task jumpoff (Thread *)

  The starting point for all threads.

Private Attributes

- Objects_Id _osid

  RTEMS task ID.

- volatile bool _done

  Has run() exited yet?

Friends

- class Completion

  Used to perform cleanup after run() exits.

7.21.2 Constructor & Destructor Documentation

7.21.2.1 anonymous_namespace{testCircularFifo.cc}::Thread::Thread () [inline]

Definition at line 79 of file testCircularFifo.cc.

7.21.2.2 virtual anonymous_namespace{testCircularFifo.cc}::Thread::~Thread () [inline, virtual]

Definition at line 121 of file testCircularFifo.cc.

7.21.2.3 anonymous_namespace{testCircularFifo.cc}::Thread::Thread (const Thread &) [private]

7.21.3 Member Function Documentation

7.21.3.1 void anonymous_namespace{testCircularFifo.cc}::Thread::start () [inline]

Create and start a thread/task/whatever as provided by the OS.

Definition at line 85 of file testCircularFifo.cc.

7.21.3.2 virtual void anonymous_namespace{testCircularFifo.cc}::Thread::run () [pure virtual]

Override this with the implementation of the thread's code.

Implemented in anonymous_namespace{testCircularFifo.cc}::Producer, and anonymous_namespace{testCircularFifo.cc}::Consumer.

Referenced by jumpoff().

7.21.3.3 void anonymous_namespace{testCircularFifo.cc}::Thread::yield () const [inline]

Yield the CPU, get rescheduled behind all threads of the same priority.

Definition at line 101 of file testCircularFifo.cc.

Referenced by anonymous_namespace{testCircularFifo.cc}::Consumer::run(), and anonymous_namespace{testCircularFifo.cc}::Producer::run().

7.21.3.4 void anonymous_namespace{testCircularFifo.cc}::Thread::join () const [inline]

Wait for run() to exit.

Definition at line 104 of file testCircularFifo.cc.

References quarks::concurrency::yield().

7.21.3.5 void anonymous_namespace{testCircularFifo.cc}::Thread::enqueueWait (CircularFifo< Node > * *fifo*, const Node & *d*) [inline]

Endless loop of (Try to enqueue, yield if it fails.).

Definition at line 107 of file testCircularFifo.cc.

References quarks::concurrency::CircularFifo< T >::enqueue(), and quarks::concurrency::yield().

Referenced by anonymous_namespace{testCircularFifo.cc}::Producer::run().

7.21.3.6 void anonymous_namespace{testCircularFifo.cc}::Thread::dequeueWait (CircularFifo< Node > * *fifo*, Node & *d*) [inline]

Endless loop of (Try to dequeue, yield if it fails.)

Definition at line 114 of file testCircularFifo.cc.

References quarks::concurrency::CircularFifo< T >::dequeue(), and quarks::concurrency::yield().

7.21.3.7 Thread& anonymous_namespace{testCircularFifo.cc}::Thread::operator= (const Thread &) [private]

7.21.3.8 rtems_task anonymous_namespace{testCircularFifo.cc}::Thread::jumpoff (Thread * *t*) [static, private]

The starting point for all threads.

Definition at line 151 of file testCircularFifo.cc.

References run().

7.21.4 Friends And Related Function Documentation

7.21.4.1 friend class Completion [friend]

Used to perform cleanup after run() exits.

Definition at line 131 of file testCircularFifo.cc.

7.21.5 Member Data Documentation

7.21.5.1 Objects_Id anonymous_namespace{testCircularFifo.cc}::Thread::_osid [private]

RTEMS task ID.

Definition at line 128 of file testCircularFifo.cc.

7.21.5.2 volatile bool anonymous_namespace{testCircularFifo.cc}::Thread::_done [private]

Has run() exited yet?

Definition at line 129 of file testCircularFifo.cc.

The documentation for this struct was generated from the following file:

- testCircularFifo.cc

7.22 quarks::service::TimerGuard< Accumulator > Class Template Reference

`#include <TimerGuard.hh>`

7.22.1 Detailed Description

template<typename Accumulator> class quarks::service::TimerGuard< Accumulator >

Time a block of code between the exiting of the TimeGuard constructor and the start of the TimeGuard destructor. Always in-line.

The destructor will call the void takeTime(unsigned long long,...) member function of the Accumulator object passed to the constructor. The first argument will be the elapsed time expressed as the number of ticks of whatever system clock is being used. Static member functions are provided which convert these values into seconds or CPU clock cycles. The second argument is the number of repetitions of the desired instructions in the timed region. The third argument is an arbitrary ID number for the section of code. The last argument is a short comment string.

If needed by your system TimerGuard uses barrier instructions in order to clear the CPU pipeline, prefetched instructions and pending data fetch/store operations before and after the code to be timed.

Here's an example of use when the Accumulator object is a classic Singleton which makes histograms. This particular example would probably have to be compiled without optimization, or else the divisions might be done at compile time. The constructor and destructor for TimerGuard have the "always inline" GCC attribute in order to avoid function call overhead.

```
void timeDivUlong()
{
  register unsigned long a = 314159UL; // Avoid arg fetch in timed code.
  register unsigned long b = 271727UL;
  TimerGuard tg(*TimeHist::instance(), 100, TimeHist::UlongDiv, "100 ulong divisions");
  register c = a/b;
  c = b/a;
  c = a/b;
  ...
}
```

Accumulator The type of the object whose takeTime(unsigned long long,...) member function is called by the TimerGuard destructor.

Definition at line 49 of file rce405/TimerGuard.hh.

Public Member Functions

- TimerGuard (Accumulator &accum, unsigned rptc, unsigned id, const char *comment) __attribute__((always_inline))
- ~TimerGuard () __attribute__((always_inline))

Private Attributes

- unsigned long long _ticks
- Accumulator & _accum
- const unsigned _repeatCount

- const unsigned _id
- const char *const _comment

7.22.2 Constructor & Destructor Documentation

7.22.2.1 template<typename Accumulator> quarks::service::TimerGuard< Accumulator >::TimerGuard (Accumulator & *accum*, unsigned *rptc*, unsigned *id*, const char * *comment*) [inline]

Definition at line 64 of file rce405/TimerGuard.hh.

References quarks::service::TimerGuard< Accumulator >::_ticks.

7.22.2.2 template<typename Accumulator> quarks::service::TimerGuard< Accumulator >::~TimerGuard () [inline]

Definition at line 83 of file rce405/TimerGuard.hh.

References quarks::service::TimerGuard< Accumulator >::_accum, and quarks::service::TimerGuard< Accumulator >::_ticks.

7.22.3 Member Data Documentation

7.22.3.1 template<typename Accumulator> unsigned long long quarks::service::TimerGuard< Accumulator >::_ticks [private]

Definition at line 50 of file rce405/TimerGuard.hh.

Referenced by quarks::service::TimerGuard< Accumulator >::TimerGuard(), and quarks::service::TimerGuard< Accumulator >::~TimerGuard().

7.22.3.2 template<typename Accumulator> Accumulator& quarks::service::TimerGuard< Accumulator >::_accum [private]

Definition at line 51 of file rce405/TimerGuard.hh.

Referenced by quarks::service::TimerGuard< Accumulator >::~TimerGuard().

7.22.3.3 template<typename Accumulator> const unsigned quarks::service::TimerGuard< Accumulator >::_repeatCount [private]

Definition at line 52 of file rce405/TimerGuard.hh.

7.22.3.4 template<typename Accumulator> const unsigned quarks::service::TimerGuard< Accumulator >::_id [private]

Definition at line 53 of file rce405/TimerGuard.hh.

7.22.3.5 template<typename Accumulator> const char* const quarks::service::TimerGuard< Accumulator >::_comment [private]

Definition at line 54 of file rce405/TimerGuard.hh.

The documentation for this class was generated from the following file:

- rce405/TimerGuard.hh

Chapter 8

quarks File Documentation

8.1 BitSet.hh File Reference

8.1.1 Detailed Description

Facility:

RCE Petacache

Abstract:

Define and implement class BitSet.

Author:

Sephen Tether <tether@slac.stanford.edu>

Date created:

2010/03/24

Last commit:

$Date: 2010-03-29 14:35:22 -0700 (Mon, 29 Mar 2010) $ by $Author: tether $.

Revision number:

$Revision: 1118 $

Location in repository:

$HeadURL: file:///reg/g/npa/svnrepo/quarks/trunk/datastructures/BitSet.hh $

Credits:

SLAC

Definition in file BitSet.hh.

```
#include <algorithm>
```

Namespaces

- namespace quarks
- namespace quarks::datastructures

Classes

- class quarks::datastructures::BitSet
  A classic powerset stored in an unsigned integer.
- class quarks::datastructures::BitPq
  Yields a sequence of set bit positions in order from lowest to highest.

Defines

- #define QUARKS_DATASTRUCTURES_BITSET_HH

Functions

- BitSet quarks::datastructures::operator~ (const BitSet &b)
  Produce a new Bitset with the same members as b.complement().
- BitSet quarks::datastructures::operator| (const BitSet &a, const BitSet &b)
  Produce a new Bitset that's the union of a and b.
- BitSet quarks::datastructures::operator & (const BitSet &a, const BitSet &b)
  Produce a new Bitset that's the intersection of a and b.
- BitSet quarks::datastructures::operator^ (const BitSet &a, const BitSet &b)
  Produce a new Bitset that's the symmetric difference of a and b.
- BitSet quarks::datastructures::operator- (const BitSet &a, const BitSet &b)
  Produce a new Bitset that's the difference of a and b.

Variables

- static const unsigned quarks::datastructures::deBruijn = 0x077CB531U
- static const int quarks::datastructures::deBruijnUnscramble [32]

8.1.2 Define Documentation

8.1.2.1 #define QUARKS_DATASTRUCTURES_BITSET_HH

Definition at line 37 of file BitSet.hh.

8.2 CircularFifo.hh File Reference

8.2.1 Detailed Description

Facility:

RCE Petacache

Abstract:

Defers the definition of the class template CircularFifo to the appropriate system dependent header.

Author:

Stephen Tether <tether@slac.stanford.edu>

Date created:

2010/05/14

Last commit:

$Date: 2010-05-18 17:54:11 -0700 (Tue, 18 May 2010) $ by $Author: tether $.

Revision number:

$Revision: 1152 $

Location in repository:

$HeadURL: file:///reg/g/npa/svnrepo/quarks/trunk/concurrency/CircularFifo.hh $

Credits:

SLAC

Definition in file CircularFifo.hh.

Defines

- #define QUARKS_CONCURRENCY_CIRCULARFIFO_HH

8.2.2 Define Documentation

8.2.2.1 #define QUARKS_CONCURRENCY_CIRCULARFIFO_HH

Definition at line 38 of file CircularFifo.hh.

8.3 CircularFifo.hh File Reference

8.3.1 Detailed Description

Facility:

RCE Petacache

Abstract:

Define the CircularFifo class template for RCEs with PowerPC processors.

Author:

Stephen Tether <tether@slac.stanford.edu>

Date created:

2010/05/14

Last commit:

$Date: 2010-05-18 17:57:37 -0700 (Tue, 18 May 2010) $ by $Author: tether $.

Revision number:

$Revision: 1153 $

Location in repository:

$HeadURL: file:///reg/g/npa/svnrepo/quarks/trunk/concurrency/rce405/CircularFifo.hh
$

Credits:

SLAC

Definition in file rce405/CircularFifo.hh.

```
#include "quarks/concurrency/rce405/LlScFifo.hh"
```

Namespaces

- namespace quarks
- namespace quarks::concurrency

Classes

- class quarks::concurrency::CircularFifo< T >

  A generic wrapper for a concurrent FIFO implemented using a circular buffer with a fixed number of slots. T The type of the items to be queued.

Defines

- #define QUARKS_CONCURRENCY_RCE405_CIRCULARFIFO_HH

8.3.2 Define Documentation

8.3.2.1 #define QUARKS_CONCURRENCY_RCE405_CIRCULARFIFO_HH

Definition at line 37 of file rce405/CircularFifo.hh.

8.4 CriticalSection.hh File Reference

8.4.1 Detailed Description

Facility:

RCE Petacache

Abstract:

declares class CriticalSection.

Author:

Stephen Tether <tether@slac.stanford.edu>

Date created:

2010/03/26

Last commit:

$Date: 2010-03-29 14:35:22 -0700 (Mon, 29 Mar 2010) $ by $Author: tether $.

Revision number:

$Revision: 1118 $

Location in repository:

$HeadURL: file:///reg/g/npa/svnrepo/quarks/trunk/concurrency/CriticalSection.hh $

Credits:

SLAC

Definition in file CriticalSection.hh.

```
#include "rce/service/Semaphore.hh"
```

Namespaces

- namespace quarks
- namespace quarks::concurrency

Classes

- class quarks::concurrency::CriticalSection

  Take a Semaphore on construction, give it back on destruction. Non-copyable.

Defines

- #define QUARKS_CONCURRENCY_CRITICALSECTION_HH

8.4.2 Define Documentation

8.4.2.1 #define QUARKS_CONCURRENCY_CRITICALSECTION_HH

Definition at line 38 of file CriticalSection.hh.

8.5 docmain.hh File Reference

8.5.1 Detailed Description

Facility:

RCE Petacache

Abstract:

Define for project "quarks" the doxygen main page contents, namespace docs and any other documentation that has no other good place to sit.

Author:

Stephen Tether <tether@slac.stanford.edu>

Date created:

2010/03/29

Last commit:

$Date: 2010-04-02 15:20:51 -0700 (Fri, 02 Apr 2010) $ by $Author: tether $.

Revision number:

$Revision: 1133 $

Location in repository:

$HeadURL: file:///reg/g/npa/svnrepo/quarks/trunk/docmain.hh $

Credits:

SLAC

Definition in file docmain.hh.

Namespaces

- namespace quarks
- namespace quarks::concurrency
- namespace quarks::datastructures
- namespace quarks::io
- namespace quarks::service

8.6 InitLogging.hh File Reference

8.6.1 Detailed Description

Facility:

RCE Petacache

Abstract:

Defines and implements the InitLogging class.

Author:

Stephen Tether <tether@slac.stanford.edu>

Date created:

2009/03/26

Revision date:

$Date: 2010-03-29 14:35:22 -0700 (Mon, 29 Mar 2010) $

Revision number:

$Revision: 1118 $

Checkout/export tag:

$Name$

Credits:

SLAC

Definition in file InitLogging.hh.

```
#include "quarks/service/Logger.hh"
```

Namespaces

- namespace quarks
- namespace quarks::service

Classes

- struct quarks::service::InitLogging

  Set up logging in the constructor and tear it down in the destructor.

Defines

- #define QUARKS_SERVICE_INITLOGGING_HH

8.6.2 Define Documentation

8.6.2.1 #define QUARKS_SERVICE_INITLOGGING_HH

Definition at line 37 of file InitLogging.hh.

8.7 legal.cc File Reference

8.8 legal.hh File Reference

8.9 LlScFifo.hh File Reference

8.9.1 Detailed Description

Facility:

RCE Petacache

Abstract:

Define the LlScFifo class template.

Author:

Stephen Tether <tether@slac.stanford.edu>

Date created:

2010/05/07

Last commit:

$Date: 2010-05-18 17:57:37 -0700 (Tue, 18 May 2010) $ by $Author: tether $.

Revision number:

$Revision: 1153 $

Location in repository:

$HeadURL: file:///reg/g/npa/svnrepo/quarks/trunk/concurrency/rce405/LlScFifo.hh $

Credits:

SLAC

Definition in file LlScFifo.hh.

```
#include <memory>
#include "quarks/service/Logger.hh"
#include "quarks/concurrency/rce405/syncops.hh"
```

Namespaces

- namespace quarks
- namespace quarks::concurrency

Classes

- class quarks::concurrency::LlScFifo< T >

  A lock-free concurrent FIFO based on a circular array synchronized by the atomic operations Load and Lock (LL) and Store Conditional (SC).

Defines

- #define QUARKS_CONCURRENCY_RCE405_LLSCFIFO_HH

8.9.2 Define Documentation

8.9.2.1 #define QUARKS_CONCURRENCY_RCE405_LLSCFIFO_HH

Definition at line 37 of file LlScFifo.hh.

8.10 Logger.cc File Reference

8.10.1 Detailed Description

Facility:

RCE quarks

Abstract:

Implements Logger::log().

Author:

Stephen Tether <tether@slac.stanford.edu>

Date created:

2010/02/11

Revision date:

$Date: 2010-04-02 15:28:29 -0700 (Fri, 02 Apr 2010) $

Revision number:

$Revision: 1134 $

Checkout/export tag:

$Name$

Credits:

SLAC

Definition in file Logger.cc.

```
#include <cstdarg>
#include <tr1/memory>
#include "rce/service/Semaphore.hh"
#include "quarks/concurrency/CriticalSection.hh"
#include "quarks/service/Logger.hh"
#include "quarks/service/LoggerImpl.hh"
#include "quarks/service/LogMessage.hh"
```

Namespaces

- namespace quarks
- namespace quarks::service

8.11 Logger.hh File Reference

8.11.1 Detailed Description

Facility:

RCE quarks

Abstract:

Declares the Logger class.

Author:

Stephen Tether <tether@slac.stanford.edu>

Date created:

2009/12/19

Revision date:

$Date: 2010-04-02 15:28:29 -0700 (Fri, 02 Apr 2010) $

Revision number:

$Revision: 1134 $

Checkout/export tag:

$Name$

Credits:

SLAC

Definition in file Logger.hh.

```
#include <tr1/memory>
#include <cstdarg>
#include "rce/service/Semaphore.hh"
#include "quarks/concurrency/CriticalSection.hh"
```

Namespaces

- namespace quarks
- namespace quarks::service

Classes

- class quarks::service::Logger

  Handle log messages from applications.

Defines

- #define QUARKS_SERVICE_LOGGER_HH

8.11.2 Define Documentation

8.11.2.1 #define QUARKS_SERVICE_LOGGER_HH

Definition at line 37 of file Logger.hh.

8.12 LoggerImpl.cc File Reference

8.12.1 Detailed Description

Facility:

RCE Petacache

Abstract:

Provides some tools useful when implementing loggers.

Author:

Stephen Tether <tether@slac.stanford.edu>

Date created:

2010/02/24

Last commit:

$Date: 2010-03-26 19:54:05 -0700 (Fri, 26 Mar 2010) $ by $Author: tether $.

Revision number:

$Revision: 1111 $

Location in repository:

$HeadURL: file:///reg/g/npa/svnrepo/quarks/trunk/service/src/LoggerImpl.cc $

Credits:

SLAC

Definition in file LoggerImpl.cc.

```
#include "quarks/service/Logger.hh"
#include "quarks/service/LoggerImpl.hh"
```

Namespaces

- namespace quarks
- namespace quarks::service

8.13 LoggerImpl.hh File Reference

8.13.1 Detailed Description

Facility:

RCE Petacache

Abstract:

Declares class LoggerImpl.

Author:

Stephen Tether <tether@slac.stanford.edu>

Date created:

2009/12/19

Revision date:

$Date: 2010-04-02 15:28:29 -0700 (Fri, 02 Apr 2010) $

Revision number:

$Revision: 1134 $

Checkout/export tag:

$Name$

Credits:

SLAC

Definition in file LoggerImpl.hh.

```
#include <cstdarg>
#include <tr1/memory>
```

Namespaces

- namespace quarks
- namespace quarks::service

Classes

- class quarks::service::LoggerImpl

  Abstract base class for logging implementations.

Defines

- #define QUARKS_SERVICE_LOGGERIMPL_HH

8.13.2 Define Documentation

8.13.2.1 #define QUARKS_SERVICE_LOGGERIMPL_HH

Definition at line 37 of file LoggerImpl.hh.

8.14 LogMessage.cc File Reference

8.14.1 Detailed Description

Facility:

RCE Petacache

Abstract:

Implements the LogMessage class.

Author:

Stephen Tether <tether@slac.stanford.edu>

Date created:

2010/03/25

Last commit:

$Date: 2010-04-02 15:28:29 -0700 (Fri, 02 Apr 2010) $ by $Author: tether $.

Revision number:

$Revision: 1134 $

Location in repository:

$HeadURL: file:///reg/g/npa/svnrepo/quarks/trunk/service/src/LogMessage.cc $

Credits:

SLAC

Definition in file LogMessage.cc.

```
#include <cstdarg>
#include <stdio.h>
#include <time.h>
#include <vector>
#include "quarks/service/Logger.hh"
#include "quarks/service/LogMessage.hh"
```

Namespaces

- namespace anonymous_namespace{LogMessage.cc}
- namespace quarks
- namespace quarks::service

Functions

- void anonymous_namespace{LogMessage.cc}::addProlog (LogMessage &msg, Logger::Severity sev)

8.15 LogMessage.hh File Reference

8.15.1 Detailed Description

Facility:

RCE Petacache

Abstract:

Declares class LogMessage.

Author:

Stephen Tether <tether@slac.stanford.edu>

Date created:

2010/03/25

Last commit:

$Date: 2010-03-29 14:35:22 -0700 (Mon, 29 Mar 2010) $ by $Author: tether $.

Revision number:

$Revision: 1118 $

Location in repository:

$HeadURL: file:///reg/g/npa/svnrepo/quarks/trunk/service/LogMessage.hh $

Credits:

SLAC

Definition in file LogMessage.hh.

```
#include <vector>
#include "quarks/service/Logger.hh"
```

Namespaces

- namespace quarks
- namespace quarks::service

Classes

- class quarks::service::LogMessage

  Assembles a log message in a buffer.

Defines

- #define QUARKS_SERVICE_LOGMESSAGE_HH

8.15.2 Define Documentation

8.15.2.1 #define QUARKS_SERVICE_LOGMESSAGE_HH

Definition at line 39 of file LogMessage.hh.

8.16 NoInterruptFifo.hh File Reference

8.16.1 Detailed Description

Facility:

RCE Petacache

Abstract:

Define the NoInterruptFifo class template.

Author:

Stephen Tether <tether@slac.stanford.edu>

Date created:

2010/05/17

Last commit:

$Date: 2010-05-19 14:04:59 -0700 (Wed, 19 May 2010) $ by $Author: tether $.

Revision number:

$Revision: 1157 $

Location in repository:

$HeadURL: file:///reg/g/npa/svnrepo/quarks/trunk/concurrency/NoInterruptFifo.hh $

Credits:

SLAC

Definition in file NoInterruptFifo.hh.

```
#include <memory>
#include "quarks/service/Logger.hh"
#include "quarks/concurrency/syncops.hh"
```

Namespaces

- namespace quarks
- namespace quarks::concurrency

Classes

- class quarks::concurrency::NoInterruptFifo< T >

  A concurrent FIFO based on a circular array synchronized by locked critical sections. Locking is achieved by disabling interrupts within the critical sections.

Defines

- #define QUARKS_CONCURRENCY_NOINTERRUPTFIFO_HH

8.16.2 Define Documentation

8.16.2.1 #define QUARKS_CONCURRENCY_NOINTERRUPTFIFO_HH

Definition at line 37 of file NoInterruptFifo.hh.

8.17 PtyLogger.cc File Reference

8.17.1 Detailed Description

Facility:

RCE Petacache

Abstract:

Implements class PtyLogger.

Author:

Stephen Tether <tether@slac.stanford.edu>

Date created:

2009/12/19

Revision date:

$Date: 2010-04-02 15:28:29 -0700 (Fri, 02 Apr 2010) $

Revision number:

$Revision: 1134 $

Checkout/export tag:

$Name$

Credits:

SLAC

Definition in file PtyLogger.cc.

```
#include <errno.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include "quarks/io/rwall.hh"
#include "quarks/service/Logger.hh"
#include "quarks/service/LoggerImpl.hh"
#include "quarks/service/PtyLogger.hh"
```

Namespaces

- namespace anonymous_namespace{PtyLogger.cc}
- namespace quarks
- namespace quarks::service

Enumerations

- enum { anonymous_namespace{PtyLogger.cc}::Stderr = 2 }

8.18 PtyLogger.hh File Reference

8.18.1 Detailed Description

Facility:

RCE Petacache

Abstract:

Declares the PtyLogger class.

Author:

Stephen Tether <tether@slac.stanford.edu>

Date created:

2009/12/19

Revision date:

$Date: 2010-04-02 15:28:29 -0700 (Fri, 02 Apr 2010) $

Revision number:

$Revision: 1134 $

Checkout/export tag:

$Name$

Credits:

SLAC

Definition in file PtyLogger.hh.

```
#include <tr1/memory>
#include "quarks/service/Logger.hh"
#include "quarks/service/LoggerImpl.hh"
```

Namespaces

- namespace quarks
- namespace quarks::service

Classes

- class quarks::service::PtyLogger

  Write log messages to /dev/pty0. If the PTY can't be opened then log messages will go to fd 1 (stdout).

Defines

- #define QUARKS_SERVICE_PTYLOGGER_HH

8.18.2 Define Documentation

8.18.2.1 #define QUARKS_SERVICE_PTYLOGGER_HH

Definition at line 37 of file PtyLogger.hh.

8.19 rwall.cc File Reference

8.19.1 Detailed Description

Facility:

RCE Petacache

Abstract:

Defines the functions readAll() and writeAll().

Author:

Stephen Tether <tether@slac.stanford.edu>

Date created:

2010/04/01

Last commit:

$Date: 2010-04-02 15:20:51 -0700 (Fri, 02 Apr 2010) $ by $Author: tether $.

Revision number:

$Revision: 1133 $

Location in repository:

$HeadURL: file:///reg/g/npa/svnrepo/quarks/trunk/io/src/rwall.cc $

Credits:

SLAC

Definition in file rwall.cc.

```
#include <cstddef>
#include <unistd.h>
#include "quarks/io/rwall.hh"
```

Namespaces

- namespace quarks
- namespace quarks::io

Functions

- ptrdiff_t quarks::io::readAll (int fd, void *buffer, size_t nbytes)
- ptrdiff_t quarks::io::writeAll (int fd, const char *buffer, size_t nbytes)

8.20 rwall.hh File Reference

8.20.1 Detailed Description

Facility:

RCE Petacache

Abstract:

Declares the functions readAll() and writeAll().

Author:

Stephen Tether <tether@slac.stanford.edu>

Date created:

2010/04/01

Last commit:

$Date: 2010-04-02 15:20:51 -0700 (Fri, 02 Apr 2010) $ by $Author: tether $.

Revision number:

$Revision: 1133 $

Location in repository:

$HeadURL: file:///reg/g/npa/svnrepo/quarks/trunk/io/rwall.hh $

Credits:

SLAC

Definition in file rwall.hh.

```
#include <cstddef>
```

Namespaces

- namespace quarks
- namespace quarks::io

Defines

- #define QUARKS_IO_RWALL_HH

Functions

- std::ptrdiff_t quarks::io::readAll (int fd, void *buffer, std::size_t nbytes)

  Attempt to read all the bytes requested before returning.

- std::ptrdiff_t quarks::io::writeAll (int fd, const char *buffer, std::size_t nbytes)

  Attempt to write all the bytes requested before returning.

8.20.2 Define Documentation

8.20.2.1 #define QUARKS_IO_RWALL_HH

Definition at line 37 of file rwall.hh.

8.21 syncops.hh File Reference

8.21.1 Detailed Description

Facility:

RCE Petacache

Abstract:

Define thread (task) synchronization operations for the PowerPC 405 uniprocessor found in RCEs.

Author:

Stephen Tether <ether@slac.stanford.edu>

Date created:

2010/05/05

Last commit:

$Date: 2010-05-19 12:49:28 -0700 (Wed, 19 May 2010) $ by $Author: tether $.

Revision number:

$Revision: 1156 $

Location in repository:

$HeadURL: file:///reg/g/npa/svnrepo/quarks/trunk/concurrency/rce405/syncops.hh $

Credits:

SLAC

Definition in file rce405/syncops.hh.

```
#include "rtems.h"
```

Namespaces

- namespace quarks
- namespace quarks::concurrency

Classes

- struct quarks::concurrency::IsA32BitType< B >
- struct quarks::concurrency::IsA32BitType< true >
- class quarks::concurrency::InterruptGuard

  The constructor saves the state of the interrupt enables and disables interrupts. The destructor restores the prior state of the interrupt enables.

Defines

- #define QUARKS_CONCURRENCY_RCE405_SYNCOPS_HH

Functions

- template<typename T>
  T quarks::concurrency::LL (const T &x) __attribute__((always_inline))

  Atomic Load and Lock. Read a value from a given address and reserve (lock) that location.

- template<typename T>
  bool quarks::concurrency::SC (T &x, T newval) __attribute__((always_inline))

  Atomic Store Conditional. Place a new value in the location previously reserved via LL() UNLESS the reservation has been cancelled. Cancel the reservation.

- template<typename T>
  bool quarks::concurrency::CAS (T &var, T oldVal, T newVal) __attribute__((always_inline))

  Atomic Compare And Swap. If the value at the given address has the given value then change it to the given new value and return true; else leave the value unchanged and return false.

- template<typename T>
  T quarks::concurrency::FetchAndAdd (T *x, T v) __attribute__((always_inline))

  Atomic Fetch And Add. Add a given amount to a variable, returning the old value.

- template<typename T>
  T quarks::concurrency::FetchAndSub (T *x, T v) __attribute__((always_inline))

  Atomic Fetch And Sub. Subtract a given amount to a variable, returning the old value.

- void quarks::concurrency::yield () __attribute__((always_inline))

  Stop running the current task and place it in the ready chain after all other tasks with the same priority.

- unsigned quarks::concurrency::getMsr () __attribute__((always_inline))

  Get the contents of the Machine Status Register.

- unsigned quarks::concurrency::setMsr (unsigned newMsr) __attribute__((always_inline))

  Set the contents of the Machine Status Register.

- unsigned quarks::concurrency::setMsrMasked (unsigned newMsr, unsigned mask) __attribute__((always_inline))

  Write new values to the MSR bits specified by a 32-bit mask. newMSR = (oldMSR & ~mask) | (newMSR & mask).

8.21.2 Define Documentation

8.21.2.1 #define QUARKS_CONCURRENCY_RCE405_SYNCOPS_HH

Definition at line 37 of file rce405/syncops.hh.

8.22 syncops.hh File Reference

8.22.1 Detailed Description

Facility:

RCE Petacache

Abstract:

Define hardware-specific in-line synchronization operations for multi-threaded code.

Author:

Stephen Tether <ether@slac.stanford.edu>

Date created:

2010/05/05

Last commit:

$Date: 2010-05-18 17:54:11 -0700 (Tue, 18 May 2010) $ by $Author: tether $.

Revision number:

$Revision: 1152 $

Location in repository:

$HeadURL: file:///reg/g/npa/svnrepo/quarks/trunk/concurrency/syncops.hh $

Credits:

SLAC

Definition in file syncops.hh.

Defines

- #define QUARKS_CONCURRENCY_SYNCOPS_HH

8.22.2 Define Documentation

8.22.2.1 #define QUARKS_CONCURRENCY_SYNCOPS_HH

Definition at line 37 of file syncops.hh.

8.23 testBitSet.cc File Reference

8.23.1 Detailed Description

Facility:

RCE Petacache

Abstract:

Run unit ests for classes BitSet and BitPq.

Author:

Stephen Tether <tether@slac.stanford.edu>

Date created:

2010/03/24

Last commit:

$Date: 2010-03-26 20:01:40 -0700 (Fri, 26 Mar 2010) $ by $Author: tether $.

Revision number:

$Revision: 1113 $

Location in repository:

$HeadURL: file:///reg/g/npa/svnrepo/quarks/trunk/datastructures/test/testBitSet.cc $

Credits:

SLAC

Definition in file testBitSet.cc.

```
#include "quarks/datastructures/BitSet.hh"
#include "quarks/service/Logger.hh"
```

Functions

- void testBitSet ()

8.23.2 Function Documentation

8.23.2.1 void testBitSet ()

Definition at line 43 of file testBitSet.cc.

Referenced by testDatastructures().

8.24 testcalls.cc File Reference

Functions

- testService ()
- testDatastructures ()
- testConcurrency ()

8.24.1 Function Documentation

8.24.1.1 testConcurrency ()

Definition at line 38 of file testConcurrency.cc.

8.24.1.2 testDatastructures ()

Definition at line 38 of file testDatastructures.cc.

8.24.1.3 testService ()

Definition at line 40 of file testService.cc.

8.25 testCircularFifo.cc File Reference

8.25.1 Detailed Description

Facility:

RCE Petacache

Abstract:

Unit tests for class CircularFifo. Checks to make sure that no entries are lost, duplicated, altered or dequeued in the wrong order.

Author:

Stephen Tether <tether@slac.stanford.edu>

Date created:

2010/05/17

Last commit:

$Date: 2010-02-16 13:34:42 -0800 (Tue, 16 Feb 2010) $ by $Author: tether $.

Revision number:

$Revision: 1075 $

Location in repository:

$HeadURL: file:///reg/g/npa/svnrepo/quarks/trunk/boilerplate/legal.cc $

Credits:

SLAC

Definition in file testCircularFifo.cc.

```
#include <algorithm>
#include <map>
#include <tr1/memory>
#include <set>
#include <vector>
#include "quarks/concurrency/CircularFifo.hh"
#include "quarks/service/Logger.hh"
#include "rtems.h"
#include <unistd.h>
```

Namespaces

- namespace anonymous_namespace{testCircularFifo.cc}

Classes

- struct anonymous_namespace{testCircularFifo.cc}::Node

  The values placed in the queue.

- struct anonymous_namespace{testCircularFifo.cc}::Thread

  The abstract base class for producers and consumers. Non-copyable.

- struct anonymous_namespace{testCircularFifo.cc}::Completion

  Used to perform cleanup after a thread's run() member exits.

- struct anonymous_namespace{testCircularFifo.cc}::Producer

  Enqueues a fixed number of Nodes, yielding after each enqueue, then exits. Each node contains the producer ID and a monotonically increasing sequence number.

- struct anonymous_namespace{testCircularFifo.cc}::ProdRecord
- struct anonymous_namespace{testCircularFifo.cc}::Consumer

Typedefs

- typedef map< int, ProdRecord > anonymous_namespace{testCircularFifo.cc}::RecordMap

Functions

- bool anonymous_namespace{testCircularFifo.cc}::operator== (const Node &lhs, const Node &rhs)
- void testCircularFifo ()

8.25.2 Function Documentation

8.25.2.1 void testCircularFifo ()

Definition at line 270 of file testCircularFifo.cc.

Referenced by testConcurrency().

8.26 testConcurrency.cc File Reference

8.26.1 Detailed Description

Facility:

RCE Petacache

Abstract:

Run the unit tests for quarks::concurrency.

Author:

Stephen Tether <tether@slac.stanford.edu>

Date created:

2010/05/17

Last commit:

$Date: 2010-02-16 13:34:42 -0800 (Tue, 16 Feb 2010) $ by $Author: tether $.

Revision number:

$Revision: 1075 $

Location in repository:

$HeadURL: file:///reg/g/npa/svnrepo/quarks/trunk/boilerplate/legal.cc $

Credits:

SLAC

Definition in file testConcurrency.cc.

Functions

- void testCircularFifo ()
- void testConcurrency ()

8.26.2 Function Documentation

8.26.2.1 void testCircularFifo ()

Definition at line 270 of file testCircularFifo.cc.

References quarks::concurrency::CircularFifo< T >::dequeue(), quarks::concurrency::CircularFifo< T >::empty(), and quarks::concurrency::CircularFifo< T >::enqueue().

8.26.2.2 void testConcurrency ()

Definition at line 38 of file testConcurrency.cc.

8.27 testDatastructures.cc File Reference

8.27.1 Detailed Description

Facility:

RCE Petacache

Abstract:

Runs the unit tests for quarks::datastructures.

Author:

Stephen Tether <tetheer@slac.stanford.edu>

Date created:

2010/03/24

Last commit:

$Date: 2010-03-26 20:01:40 -0700 (Fri, 26 Mar 2010) $ by $Author: tether $.

Revision number:

$Revision: 1113 $

Location in repository:

$HeadURL: file:///reg/g/npa/svnrepo/quarks/trunk/datastructures/test/testDatastruct
$

Credits:

SLAC

Definition in file testDatastructures.cc.

Functions

- void testBitSet ()
- void testDatastructures ()

8.27.2 Function Documentation

8.27.2.1 void testBitSet ()

Definition at line 43 of file testBitSet.cc.

8.27.2.2 void testDatastructures ()

Definition at line 38 of file testDatastructures.cc.

8.28 testdefs.hh File Reference

Functions

- void testService ()
- void testDatastructures ()
- void testConcurrency ()

8.28.1 Function Documentation

8.28.1.1 void testConcurrency ()

Definition at line 38 of file testConcurrency.cc.

References testCircularFifo().

8.28.1.2 void testDatastructures ()

Definition at line 38 of file testDatastructures.cc.

References testBitSet().

8.28.1.3 void testService ()

Definition at line 40 of file testService.cc.

8.29 testLogging.cc File Reference

8.29.1 Detailed Description

Facility:

RCE quarks

Abstract:

Unit tests for the logging service.

Author:

Stephen Tether <tether@slac.stanford.edu>

Date created:

2010/02/11

Revision date:

$Date: 2010-03-26 19:54:05 -0700 (Fri, 26 Mar 2010) $

Revision number:

$Revision: 1111 $

Checkout/export tag:

$Name$

Credits:

SLAC

Definition in file testLogging.cc.

```
#include "quarks/service/Logger.hh"
#include "quarks/service/LogMessage.hh"
```

Functions

- void testLogging ()

8.29.2 Function Documentation

8.29.2.1 void testLogging ()

Definition at line 43 of file testLogging.cc.

Referenced by testService().

8.30 testQuarks.cc File Reference

8.30.1 Detailed Description

Facility:

RCE quarks

Abstract:

Unit tests for the quarks project.

Author:

Stephen Tether <tether@slac.stanford.edu>

Date created:

2010/02/11

Revision date:

$Date: 2010-03-26 20:04:53 -0700 (Fri, 26 Mar 2010) $

Revision number:

$Revision: 1115 $

Checkout/export tag:

$Name$

Credits:

SLAC

Definition in file testQuarks.cc.

```
#include <stdio.h>
#include "quarks/mergedlib/testdefs.hh"
#include "quarks/service/InitLogging.hh"
#include "quarks/service/Logger.hh"
#include "quarks/service/PtyLogger.hh"
#include "quarks/mergedlib/testcalls.cc"
```

Functions

- int main ()

8.30.2 Function Documentation

8.30.2.1 int main ()

Definition at line 52 of file testQuarks.cc.

8.31 testService.cc File Reference

8.31.1 Detailed Description

Facility:

RCE quarks

Abstract:

Unit tests for quarks/service.

Author:

Stephen Tether <tether@slac.stanford.edu>

Date created:

2010/02/11

Revision date:

$Date: 2010-02-12 16:32:44 -0800 (Fri, 12 Feb 2010) $

Revision number:

$Revision: 1072 $

Checkout/export tag:

$Name$

Credits:

SLAC

Definition in file testService.cc.

Functions

- void testLogging ()
- void testTiming ()
- void testService ()

8.31.2 Function Documentation

8.31.2.1 void testLogging ()

Definition at line 43 of file testLogging.cc.

References quarks::service::LogMessage::add().

8.31.2.2 void testService ()

Definition at line 40 of file testService.cc.

References testLogging(), and testTiming().

8.31.2.3 void testTiming ()

Definition at line 50 of file rce405/testTiming.cc.

8.32 testTiming.cc File Reference

8.32.1 Detailed Description

Facility:

RCE Petacache

Abstract:

Unit test for TimerGuard. Times 100 divwu instructions and 100 add instructions.

Author:

Stephen Tether <tether@slac.stanford.edu>

Date created:

2010/06/10

Last commit:

$Date: 2010-02-16 13:34:42 -0800 (Tue, 16 Feb 2010) $ by $Author: tether $.

Revision number:

$Revision: 1075 $

Location in repository:

$HeadURL: file:///reg/g/npa/svnrepo/quarks/trunk/boilerplate/legal.cc $

Credits:

SLAC

Definition in file rce405/testTiming.cc.

```
#include "quarks/service/Logger.hh"
#include "quarks/service/TimerGuard.hh"
```

Classes

- struct SayTime

Functions

- void testTiming ()

8.32.2 Function Documentation

8.32.2.1 void testTiming ()

Definition at line 50 of file rce405/testTiming.cc.

Referenced by testService().

8.33 testTiming.cc File Reference

8.33.1 Detailed Description

Facility:

RCE Petacache

Abstract:

Pull in the correct testTiming.cc file for the platform.

Author:

Stephen Tether <tether@slac.stanford.edu>

Date created:

2010/06/10

Last commit:

$Date: 2010-02-16 13:34:42 -0800 (Tue, 16 Feb 2010) $ by $Author: tether $.

Revision number:

$Revision: 1075 $

Location in repository:

$HeadURL: file:///reg/g/npa/svnrepo/quarks/trunk/boilerplate/legal.cc $

Credits:

SLAC

Definition in file testTiming.cc.

8.34 TimerGuard.hh File Reference

Namespaces

- namespace quarks
- namespace quarks::service

Classes

- class quarks::service::TimerGuard< Accumulator >

  Time a block of code between the exiting of the TimeGuard constructor and the start of the TimeGuard destructor. Always in-line.

Defines

- #define QUARKS_SERVICE_RCE405_TIMERGUARD_HH

8.34.1 Define Documentation

8.34.1.1 #define QUARKS_SERVICE_RCE405_TIMERGUARD_HH

Definition at line 2 of file rce405/TimerGuard.hh.

8.35 TimerGuard.hh File Reference

8.35.1 Detailed Description

Facility:

RCE Petacache

Abstract:

Defers the definition of the class template TimerGuard to the appropriate system dependent header.

Author:

Stephen Tether <tether@slac.stanford.edu>

Date created:

2010/06/06

Last commit:

$Date: 2010-05-18 17:54:11 -0700 (Tue, 18 May 2010) $ by $Author: tether $.

Revision number:

$Revision: 1152 $

Location in repository:

$HeadURL: file:///reg/g/npa/svnrepo/quarks/trunk/concurrency/CircularFifo.hh $

Credits:

SLAC

Definition in file TimerGuard.hh.

Defines

- #define QUARKS_SERVICE_TIMERGUARD_HH

8.35.2 Define Documentation

8.35.2.1 #define QUARKS_SERVICE_TIMERGUARD_HH

Definition at line 38 of file TimerGuard.hh.

Index