The GNU Transactional Memory Library

Short Contents

Introduction 1
1 Enabling libitm 3
2 C/C++ Language Constructs for TM 5
3 The libitm ABI 7
4 Internals 13
GNU Free Documentation License 19
Library Index 27

Table of Contents

Introduction 1

1 Enabling libitm 3

2 C/C++ Language Constructs for TM 5

3 The libitm ABI 7
- 3.1 [No changes] Objectives 7
- 3.2 [No changes] Non-objectives 7
- 3.3 Library design principles 7
  - 3.3.1 [No changes] Calling conventions 7
  - 3.3.2 [No changes] TM library algorithms 7
  - 3.3.3 [No changes] Optimized load and store routines 7
  - 3.3.4 [No changes] Aligned load and store routines 7
  - 3.3.5 Data logging functions 7
  - 3.3.6 [No changes] Scatter/gather calls 7
  - 3.3.7 [No changes] Serial and irrevocable mode 7
  - 3.3.8 [No changes] Transaction descriptor 7
  - 3.3.9 Store allocation 7
  - 3.3.10 [No changes] Naming conventions 8
  - 3.3.11 Function pointer encryption 8
- 3.4 Types and macros list 8
- 3.5 Function list 8
  - 3.5.1 Initialization and finalization functions 8
  - 3.5.2 [No changes] Version checking 8
  - 3.5.3 [No changes] Error reporting 8
  - 3.5.4 [No changes] inTransaction call 8
  - 3.5.5 State manipulation functions 8
  - 3.5.6 [No changes] Source locations 8
  - 3.5.7 Starting a transaction 8
    - 3.5.7.1 Transaction code properties 8
    - 3.5.7.2 [No changes] Windows exception state 9
    - 3.5.7.3 [No changes] Other machine state 9
    - 3.5.7.4 [No changes] Results from beginTransaction 9
  - 3.5.8 Aborting a transaction 9
  - 3.5.9 Committing a transaction 9
  - 3.5.10 Exception handling support 10
  - 3.5.11 [No changes] Transition to serial–irrevocable mode 10
  - 3.5.12 [No changes] Data transfer functions 10
  - 3.5.13 [No changes] Transactional memory copies 10
  - 3.5.14 Transactional versions of memmove 10
  - 3.5.15 [No changes] Transactional versions of memset 10

3.5.16 [No changes] Logging functions . . . 10
3.5.17 User-registered commit and undo actions . . . 10
3.5.18 [New] Transactional indirect calls . . . 11
3.5.19 [New] Transactional dynamic memory management . . . 11
3.6 [No changes] Future Enhancements to the ABI . . . 11
3.7 Sample code . . . 11
3.8 [New] Memory model . . . 12

4 Internals . . . 13

4.1 TM methods and method groups . . . 13
4.1.1 TM method life cycle . . . 13
4.1.2 Selecting the default method . . . 13
4.2 Nesting: flat vs. closed . . . 13
4.3 Locking conventions . . . 14
4.3.1 State-to-lock mapping . . . 14
4.3.2 Lock acquisition order . . . 15
4.3.3 Serial lock implementation . . . 16
4.3.4 Reentrancy . . . 16
4.3.5 Privatization safety . . . 17
4.3.6 Progress guarantees . . . 17

GNU Free Documentation License . . . 19

ADDENDUM: How to use this License for your documents . . . 26

Library Index . . . 27

Introduction

This manual documents the usage and internals of libitm, the GNU Transactional Memory Library. It provides transaction support for accesses to a process' memory, enabling easy-to-use synchronization of accesses to shared memory by several threads.

1 Enabling libitm

To activate support for TM in C/C++, the compile-time flag `-fgnu-tm` must be specified. This enables TM language-level constructs such as transaction statements (e.g., `__transaction_atomic`, see Chapter 2 [C/C++ Language Constructs for TM], page 5, for details).

2 C/C++ Language Constructs for TM

Transactions are supported in C++ and C in the form of transaction statements, transaction expressions, and function transactions. In the following example, both `a` and `b` will be read and the difference will be written to `c`, all atomically and isolated from other transactions:

```
__transaction_atomic { c = a - b; }
```

Therefore, another thread can use the following code to concurrently update `b` without ever causing `c` to hold a negative value (and without having to use other synchronization constructs such as locks or C++11 atomics):

```
__transaction_atomic { if (a > b) b++; }
```

GCC follows the Draft Specification of Transactional Language Constructs for C++ (v1.1) (`https://sites.google.com/site/tmforcplusplus/`) in its implementation of transactions.

The precise semantics of transactions are defined in terms of the C++11/C11 memory model (see the specification). Roughly, transactions provide synchronization guarantees that are similar to what would be guaranteed when using a single global lock as a guard for all transactions. Note that like other synchronization constructs in C/C++, transactions rely on a data-race-free program (e.g., a nontransactional write that is concurrent with a transactional read to the same memory location is a data race).

3 The libitm ABI

The ABI provided by libitm is basically equal to the Linux variant of Intel's current TM ABI specification document (Revision 1.1, May 6 2009) but with the differences listed in this chapter. It would be good if these changes would eventually be merged into a future version of this specification. To ease look-up, the following subsections mirror the structure of this specification.

3.1 [No changes] Objectives

3.2 [No changes] Non-objectives

3.3 Library design principles

3.3.1 [No changes] Calling conventions

3.3.2 [No changes] TM library algorithms

3.3.3 [No changes] Optimized load and store routines

3.3.4 [No changes] Aligned load and store routines

3.3.5 Data logging functions

The memory locations accessed with transactional loads and stores and the memory locations whose values are logged must not overlap. This required separation only extends to the scope of the execution of one transaction including all the executions of all nested transactions.

The compiler must be consistent (within the scope of a single transaction) about which memory locations are shared and which are not shared with other threads (i.e., data must be accessed either transactionally or nontransactionally). Otherwise, non-write-through TM algorithms would not work.

For memory locations on the stack, this requirement extends to only the lifetime of the stack frame that the memory location belongs to (or the lifetime of the transaction, whichever is shorter). Thus, memory that is reused for several stack frames could be target of both data logging and transactional accesses; however, this is harmless because these stack frames' lifetimes will end before the transaction finishes.

3.3.6 [No changes] Scatter/gather calls

3.3.7 [No changes] Serial and irrevocable mode

3.3.8 [No changes] Transaction descriptor

3.3.9 Store allocation

There is no `getTransaction` function.

3.3.10 [No changes] Naming conventions

3.3.11 Function pointer encryption

Currently, this is not implemented.

3.4 Types and macros list

`_ITM_codeProperties` has changed, see [Starting a transaction], page 8. `_ITM_srcLocation` is not used.

3.5 Function list

3.5.1 Initialization and finalization functions

These functions are not part of the ABI.

3.5.2 [No changes] Version checking

3.5.3 [No changes] Error reporting

3.5.4 [No changes] inTransaction call

3.5.5 State manipulation functions

There is no `getTransaction` function. Transaction identifiers for nested transactions will be ordered but not necessarily sequential (i.e., for a nested transaction's identifier *IN* and its enclosing transaction's identifier *IE*, it is guaranteed that $IN >= IE$).

3.5.6 [No changes] Source locations

3.5.7 Starting a transaction

3.5.7.1 Transaction code properties

The bit `hasNoXMMUpdate` is instead called `hasNoVectorUpdate`. Iff it is set, vector register save/restore is not necessary for any target machine.

The `hasNoFloatUpdate` bit (`0x0010`) is new. Iff it is set, floating point register save/restore is not necessary for any target machine.

`undoLogCode` is not supported and a fatal runtime error will be raised if this bit is set. It is not properly defined in the ABI why barriers other than undo logging are not present; Are they not necessary (e.g., a transaction operating purely on thread-local data) or have they been omitted by the compiler because it thinks that some kind of global synchronization (e.g., serial mode) might perform better? The specification suggests that the latter might be the case, but the former seems to be more useful.

The `readOnly` bit (`0x4000`) is new. **TODO** Lexical or dynamic scope?

`hasNoRetry` is not supported. If this bit is not set, but `hasNoAbort` is set, the library can assume that transaction rollback will not be requested.

It would be useful if the absence of externally-triggered rollbacks would be reported for the dynamic scope as well, not just for the lexical scope (`hasNoAbort`). Without this, a library cannot exploit this together with flat nesting.

`exceptionBlock` is not supported because exception blocks are not used.

3.5.7.2 [No changes] Windows exception state

3.5.7.3 [No changes] Other machine state

3.5.7.4 [No changes] Results from beginTransaction

3.5.8 Aborting a transaction

`_ITM_rollbackTransaction` is not supported. `_ITM_abortTransaction` is supported but the abort reasons `exceptionBlockAbort`, `TMConflict`, and `userRetry` are not supported. There are no exception blocks in general, so the related cases also do not have to be considered. To encode `__transaction_cancel [[outer]]`, compilers must set the new `outerAbort` bit (`0x10`) additionally to the `userAbort` bit in the abort reason.

3.5.9 Committing a transaction

The exception handling (EH) scheme is different. The Intel ABI requires the `_ITM_tryCommitTransaction` function that will return even when the commit failed and will have to be matched with calls to either `_ITM_abortTransaction` or `_ITM_commitTransaction`. In contrast, gcc relies on transactional wrappers for the functions of the Exception Handling ABI and on one additional commit function (shown below). This allows the TM to keep track of EH internally and thus it does not have to embed the cleanup of EH state into the existing EH code in the program. `_ITM_tryCommitTransaction` is not supported. `_ITM_commitTransactionToId` is also not supported because the propagation of thrown exceptions will not bypass commits of nested transactions.

```
void _ITM_commitTransactionEH(void *exc_ptr) ITM_REGPARM;
void *_ITM_cxa_allocate_exception (size_t);
void _ITM_cxa_throw (void *obj, void *tinfo, void *dest);
void *_ITM_cxa_begin_catch (void *exc_ptr);
void _ITM_cxa_end_catch (void);
```

`_ITM_commitTransactionEH` must be called to commit a transaction if an exception could be in flight at this position in the code. `exc_ptr` is the current exception or zero if there is no current exception. The `_ITM_cxa...` functions are transactional wrappers for the respective `__cxa...` functions and must be called instead of these in transactional code.

To support this EH scheme, libstdc++ needs to provide one additional function (`_cxa_tm_cleanup`), which is used by the TM to clean up the exception handling state while rolling back a transaction:

```
void __cxa_tm_cleanup (void *unthrown_obj, void *cleanup_exc,
                       unsigned int caught_count);
```

`unthrown_obj` is non-null if the program called `__cxa_allocate_exception` for this exception but did not yet called `__cxa_throw` for it. `cleanup_exc` is non-null if the program is currently processing a cleanup along an exception path but has not caught this exception yet.

caught_count is the nesting depth of __cxa_begin_catch within the transaction (which can be counted by the TM using _ITM_cxa_begin_catch and _ITM_cxa_end_catch); __cxa_tm_cleanup then performs rollback by essentially performing __cxa_end_catch that many times.

3.5.10 Exception handling support

Currently, there is no support for functionality like __transaction_cancel throw as described in the C++ TM specification. Supporting this should be possible with the EH scheme explained previously because via the transactional wrappers for the EH ABI, the TM is able to observe and intercept EH.

3.5.11 [No changes] Transition to serial–irrevocable mode

3.5.12 [No changes] Data transfer functions

3.5.13 [No changes] Transactional memory copies

3.5.14 Transactional versions of memmove

If either the source or destination memory region is to be accessed nontransactionally, then source and destination regions must not be overlapping. The respective _ITM_memmove functions are still available but a fatal runtime error will be raised if such regions do overlap. To support this functionality, the ABI would have to specify how the intersection of the regions has to be accessed (i.e., transactionally or nontransactionally).

3.5.15 [No changes] Transactional versions of memset

3.5.16 [No changes] Logging functions

3.5.17 User-registered commit and undo actions

Commit actions will get executed in the same order in which the respective calls to _ITM_addUserCommitAction happened. Only _ITM_noTransactionId is allowed as value for the resumingTransactionId argument. Commit actions get executed after privatization safety has been ensured.

Undo actions will get executed in reverse order compared to the order in which the respective calls to _ITM_addUserUndoAction happened. The ordering of undo actions w.r.t. the roll-back of other actions (e.g., data transfers or memory allocations) is undefined.

_ITM_getThreadnum is not supported currently because its only purpose is to provide a thread ID that matches some assumed performance tuning output, but this output is not part of the ABI nor further defined by it.

_ITM_dropReferences is not supported currently because its semantics and the intention behind it is not entirely clear. The specification suggests that this function is necessary because of certain orderings of data transfer undos and the releasing of memory regions (i.e., privatization). However, this ordering is never defined, nor is the ordering of dropping references w.r.t. other events.

3.5.18 [New] Transactional indirect calls

Indirect calls (i.e., calls through a function pointer) within transactions should execute the transactional clone of the original function (i.e., a clone of the original that has been fully instrumented to use the TM runtime), if such a clone is available. The runtime provides two functions to register/deregister clone tables:

```
struct clone_entry
{
  void *orig, *clone;
};

void _ITM_registerTMCloneTable (clone_entry *table, size_t entries);
void _ITM_deregisterTMCloneTable (clone_entry *table);
```

Registered tables must be writable by the TM runtime, and must be live throughout the life-time of the TM runtime.

TODO The intention was always to drop the registration functions entirely, and create a new ELF Phdr describing the linker-sorted table. Much like what currently happens for `PT_GNU_EH_FRAME`. This work kept getting bogged down in how to represent the *N* different code generation variants. We clearly needed at least two—SW and HW transactional clones—but there was always a suggestion of more variants for different TM assumptions/invariants.

The compiler can then use two TM runtime functions to perform indirect calls in transactions:

```
void *_ITM_getTMCloneOrIrrevocable (void *function) ITM_REGPARM;
void *_ITM_getTMCloneSafe (void *function) ITM_REGPARM;
```

If there is a registered clone for supplied function, both will return a pointer to the clone. If not, the first runtime function will attempt to switch to serial–irrevocable mode and return the original pointer, whereas the second will raise a fatal runtime error.

3.5.19 [New] Transactional dynamic memory management

```
void *_ITM_malloc (size_t)
       __attribute__((__malloc__)) ITM_PURE;
void *_ITM_calloc (size_t, size_t)
       __attribute__((__malloc__)) ITM_PURE;
void _ITM_free (void *) ITM_PURE;
```

These functions are essentially transactional wrappers for `malloc`, `calloc`, and `free`. Within transactions, the compiler should replace calls to the original functions with calls to the wrapper functions.

3.6 [No changes] Future Enhancements to the ABI

3.7 Sample code

The code examples might not be correct w.r.t. the current version of the ABI, especially everything related to exception handling.

3.8 [New] Memory model

The ABI should define a memory model and the ordering that is guaranteed for data transfers and commit/undo actions, or at least refer to another memory model that needs to be preserved. Without that, the compiler cannot ensure the memory model specified on the level of the programming language (e.g., by the C++ TM specification).

For example, if a transactional load is ordered before another load/store, then the TM runtime must also ensure this ordering when accessing shared state. If not, this might break the kind of publication safety used in the C++ TM specification. Likewise, the TM runtime must ensure privatization safety.

4 Internals

4.1 TM methods and method groups

libitm supports several ways of synchronizing transactions with each other. These TM methods (or TM algorithms) are implemented in the form of subclasses of `abi_dispatch`, which provide methods for transactional loads and stores as well as callbacks for rollback and commit. All methods that are compatible with each other (i.e., that let concurrently running transactions still synchronize correctly even if different methods are used) belong to the same TM method group. Pointers to TM methods can be obtained using the factory methods prefixed with `dispatch_` in `libitm_i.h`. There are two special methods, `dispatch_serial` and `dispatch_serialirr`, that are compatible with all methods because they run transactions completely in serial mode.

4.1.1 TM method life cycle

The state of TM methods does not change after construction, but they do alter the state of transactions that use this method. However, because per-transaction data gets used by several methods, `gtm_thread` is responsible for setting an initial state that is useful for all methods. After that, methods are responsible for resetting/clearing this state on each rollback or commit (of outermost transactions), so that the transaction executed next is not affected by the previous transaction.

There is also global state associated with each method group, which is initialized and shut down (`method_group::init()` and `fini()`) when switching between method groups (see `retry.cc`).

4.1.2 Selecting the default method

The default method that libitm uses for freshly started transactions (but not necessarily for restarted transactions) can be set via an environment variable (`ITM_DEFAULT_METHOD`), whose value should be equal to the name of one of the factory methods returning abi_dispatch subclasses but without the "dispatch_" prefix (e.g., "serialirr" instead of `GTM::dispatch_serialirr()`).

Note that this environment variable is only a hint for libitm and might not be supported in the future.

4.2 Nesting: flat vs. closed

We support two different kinds of nesting of transactions. In the case of *flat nesting*, the nesting structure is flattened and all nested transactions are subsumed by the enclosing transaction. In contrast, with *closed nesting*, nested transactions that have not yet committed can be rolled back separately from the enclosing transactions; when they commit, they are subsumed by the enclosing transaction, and their effects will be finally committed when the outermost transaction commits. *Open nesting* (where nested transactions can commit independently of the enclosing transactions) are not supported.

Flat nesting is the default nesting mode, but closed nesting is supported and used when transactions contain user-controlled aborts (`__transaction_cancel` statements). We assume that user-controlled aborts are rare in typical code and used mostly in exceptional situations. Thus, it makes more sense to use flat nesting by default to avoid the performance overhead of the additional checkpoints required for closed nesting. User-controlled aborts will correctly abort the innermost enclosing transaction, whereas the whole (i.e., outermost) transaction will be restarted otherwise (e.g., when a transaction encounters data conflicts during optimistic execution).

4.3 Locking conventions

This section documents the locking scheme and rules for all uses of locking in libitm. We have to support serial(-irrevocable) mode, which is implemented using a global lock as explained next (called the *serial lock*). To simplify the overall design, we use the same lock as catch-all locking mechanism for other infrequent tasks such as (de)registering clone tables or threads. Besides the serial lock, there are *per-method-group locks* that are managed by specific method groups (i.e., groups of similar TM concurrency control algorithms), and lock-like constructs for quiescence-based operations such as ensuring privatization safety.

Thus, the actions that participate in the libitm-internal locking are either *active transactions* that do not run in serial mode, *serial transactions* (which (are about to) run in serial mode), and management tasks that do not execute within a transaction but have acquired the serial mode like a serial transaction would do (e.g., to be able to register threads with libitm). Transactions become active as soon as they have successfully used the serial lock to announce this globally (see [Serial lock implementation], page 16). Likewise, transactions become serial transactions as soon as they have acquired the exclusive rights provided by the serial lock (i.e., serial mode, which also means that there are no other concurrent active or serial transactions). Note that active transactions can become serial transactions when they enter serial mode during the runtime of the transaction.

4.3.1 State-to-lock mapping

Application data is protected by the serial lock if there is a serial transaction and no concurrently running active transaction (i.e., non-serial). Otherwise, application data is protected by the currently selected method group, which might use per-method-group locks or other mechanisms. Also note that application data that is about to be privatized might not be allowed to be accessed by nontransactional code until privatization safety has been ensured; the details of this are handled by the current method group.

libitm-internal state is either protected by the serial lock or accessed through custom concurrent code. The latter applies to the public/shared part of a transaction object and most typical method-group-specific state.

The former category (protected by the serial lock) includes:

- The list of active threads that have used transactions.
- The tables that map functions to their transactional clones.
- The current selection of which method group to use.
- Some method-group-specific data, or invariants of this data. For example, resetting a method group to its initial state is handled by switching to the same method group, so the serial lock protects such resetting as well.

In general, such state is immutable whenever there exists an active (non-serial) transaction. If there is no active transaction, a serial transaction (or a thread that is not currently executing a transaction but has acquired the serial lock) is allowed to modify this state (but must of course be careful to not surprise the current method group's implementation with such modifications).

4.3.2 Lock acquisition order

To prevent deadlocks, locks acquisition must happen in a globally agreed-upon order. Note that this applies to other forms of blocking too, but does not necessarily apply to lock acquisitions that do not block (e.g., trylock() calls that do not get retried forever). Note that serial transactions are never return back to active transactions until the transaction has committed. Likewise, active transactions stay active until they have committed. Per-method-group locks are typically also not released before commit.

Lock acquisition / blocking rules:

- Transactions must become active or serial before they are allowed to use method-group-specific locks or blocking (i.e., the serial lock must be acquired before those other locks, either in serial or nonserial mode).
- Any number of threads that do not currently run active transactions can block while trying to get the serial lock in exclusive mode. Note that active transactions must not block when trying to upgrade to serial mode unless there is no other transaction that is trying that (the latter is ensured by the serial lock implementation.
- Method groups must prevent deadlocks on their locks. In particular, they must also be prepared for another active transaction that has acquired method-group-specific locks but is blocked during an attempt to upgrade to being a serial transaction. See below for details.
- Serial transactions can acquire method-group-specific locks because there will be no other active nor serial transaction.

There is no single rule for per-method-group blocking because this depends on when a TM method might acquire locks. If no active transaction can upgrade to being a serial transaction after it has acquired per-method-group locks (e.g., when those locks are only acquired during an attempt to commit), then the TM method does not need to consider a potential deadlock due to serial mode.

If there can be upgrades to serial mode after the acquisition of per-method-group locks, then TM methods need to avoid those deadlocks:

- When upgrading to a serial transaction, after acquiring exclusive rights to the serial lock but before waiting for concurrent active transactions to finish (see [Serial lock implementation], page 16, for details), we have to wake up all active transactions waiting on the upgrader's per-method-group locks.
- Active transactions blocking on per-method-group locks need to check the serial lock and abort if there is a pending serial transaction.
- Lost wake-ups have to be prevented (e.g., by changing a bit in each per-method-group lock before doing the wake-up, and only blocking on this lock using a futex if this bit is not group).

TODO: Can reuse serial lock for gl-*? And if we can, does it make sense to introduce further complexity in the serial lock? For gl-*, we can really only avoid an abort if we do -wb and -vbv.

4.3.3 Serial lock implementation

The serial lock implementation is optimized towards assuming that serial transactions are infrequent and not the common case. However, the performance of entering serial mode can matter because when only few transactions are run concurrently or if there are few threads, then it can be efficient to run transactions serially.

The serial lock is similar to a multi-reader-single-writer lock in that there can be several active transactions but only one serial transaction. However, we do want to avoid contention (in the lock implementation) between active transactions, so we split up the reader side of the lock into per-transaction flags that are true iff the transaction is active. The exclusive writer side remains a shared single flag, which is acquired using a CAS, for example. On the fast-path, the serial lock then works similar to Dekker's algorithm but with several reader flags that a serial transaction would have to check. A serial transaction thus requires a list of all threads with potentially active transactions; we can use the serial lock itself to protect this list (i.e., only threads that have acquired the serial lock can modify this list).

We want starvation-freedom for the serial lock to allow for using it to ensure progress for potentially starved transactions (see [Progress Guarantees], page 17, for details). However, this is currently not enforced by the implementation of the serial lock.

Here is pseudo-code for the read/write fast paths of acquiring the serial lock (read-to-write upgrade is similar to write_lock:

```
// read_lock:
tx->shared_state |= active;
__sync_synchronize(); // or STLD membar, or C++0x seq-cst fence
while (!serial_lock.exclusive)
  if (spinning_for_too_long) goto slowpath;

// write_lock:
if (CAS(&serial_lock.exclusive, 0, this) != 0)
  goto slowpath; // writer-writer contention
// need a membar here, but CAS already has full membar semantics
bool need_blocking = false;
for (t: all txns)
  {
    for (;t->shared_state & active;)
      if (spinning_for_too_long) { need_blocking = true; break; }
  }
if (need_blocking) goto slowpath;
```

Releasing a lock in this spin-lock version then just consists of resetting `tx->shared_state` to inactive or clearing `serial_lock.exclusive`.

However, we can't rely on a pure spinlock because we need to get the OS involved at some time (e.g., when there are more threads than CPUs to run on). Therefore, the real implementation falls back to a blocking slow path, either based on pthread mutexes or Linux futexes.

4.3.4 Reentrancy

libitm has to consider the following cases of reentrancy:

- Transaction calls unsafe code that starts a new transaction: The outer transaction will become a serial transaction before executing unsafe code. Therefore, nesting within serial transactions must work, even if the nested transaction is called from within uninstrumented code.
- Transaction calls either a transactional wrapper or safe code, which in turn starts a new transaction: It is not yet defined in the specification whether this is allowed. Thus, it is undefined whether libitm supports this.
- Code that starts new transactions might be called from within any part of libitm: This kind of reentrancy would likely be rather complex and can probably be avoided. Therefore, it is not supported.

4.3.5 Privatization safety

Privatization safety is ensured by libitm using a quiescence-based approach. Basically, a privatizing transaction waits until all concurrent active transactions will either have finished (are not active anymore) or operate on a sufficiently recent snapshot to not access the privatized data anymore. This happens after the privatizing transaction has stopped being an active transaction, so waiting for quiescence does not contribute to deadlocks.

In method groups that need to ensure publication safety explicitly, active transactions maintain a flag or timestamp in the public/shared part of the transaction descriptor. Before blocking, privatizers need to let the other transactions know that they should wake up the privatizer.

TODO Ho to implement the waiters? Should those flags be per-transaction or at a central place? We want to avoid one wake/wait call per active transactions, so we might want to use either a tree or combining to reduce the syscall overhead, or rather spin for a long amount of time instead of doing blocking. Also, it would be good if only the last transaction that the privatizer waits for would do the wake-up.

4.3.6 Progress guarantees

Transactions that do not make progress when using the current TM method will eventually try to execute in serial mode. Thus, the serial lock's progress guarantees determine the progress guarantees of the whole TM. Obviously, we at least need deadlock-freedom for the serial lock, but it would also be good to provide starvation-freedom (informally, all threads will finish executing a transaction eventually iff they get enough cycles).

However, the scheduling of transactions (e.g., thread scheduling by the OS) also affects the handling of progress guarantees by the TM. First, the TM can only guarantee deadlock-freedom if threads do not get stopped. Likewise, low-priority threads can starve if they do not get scheduled when other high-priority threads get those cycles instead.

If all threads get scheduled eventually, correct lock implementations will provide deadlock-freedom, but might not provide starvation-freedom. We can either enforce the latter in the TM's lock implementation, or assume that the scheduling is sufficiently random to yield a probabilistic guarantee that no thread will starve (because eventually, a transaction will encounter a scheduling that will allow it to run). This can indeed work well in practice but is not necessarily guaranteed to work (e.g., simple spin locks can be pretty efficient).

Because enforcing stronger progress guarantees in the TM has a higher runtime overhead, we focus on deadlock-freedom right now and assume that the threads will get scheduled eventually by the OS (but don't consider threads with different priorities). We should support starvation-freedom for serial transactions in the future. Everything beyond that is highly related to proper contention management across all of the TM (including with TM method to choose), and is future work.

TODO Handling thread priorities: We want to avoid priority inversion but it's unclear how often that actually matters in practice. Workloads that have threads with different priorities will likely also require lower latency or higher throughput for high-priority threads. Therefore, it probably makes not that much sense (except for eventual progress guarantees) to use priority inheritance until the TM has priority-aware contention management.

GNU Free Documentation License

0. PREAMBLE

   The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

   This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

   We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

   This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

   A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

   A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

   The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The “Cover Texts” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

The “publisher” means any person or entity that distributes copies of the Document to the public.

A section “Entitled XYZ” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “Acknowledgements”, “Dedications”, “Endorsements”, or “History”.) To “Preserve the Title” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
D. Preserve all the copyright notices of the Document.
E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
H. Include an unaltered copy of this License.
I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

   However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

   Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

   Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See `http://www.gnu.org/copyleft/`.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

    "Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

    "CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

    "Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

    An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

    The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.3
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Library Index

F

FDL, GNU Free Documentation License 19

I

Introduction 1