NAME

smthread_t − SSM Thread Class

SYNOPSIS

```
#include <sm_vas.h>  // which includes smthread.h

typedef void st_proc_t(void *);

class smthread_t : public sthread_t {
public:
    NORET                   smthread_t(
       st_proc_t*              f,
       void*                    arg,
       priority_t              priority = t_regular,
       bool                    block_immediate = false,
       bool                    auto_delete = false,
       const char*             name = 0,
       long                    lockto = WAIT_FOREVER);

    NORET                   smthread_t(
       priority_t              priority = t_regular,
       bool                    block_immediate = false,
       bool                    auto_delete = false,
       const char*             name = 0,
       long                    lockto = WAIT_FOREVER);

    NORET                   ~smthread_t();

    virtual void            run() = 0;

    void                    attach_xct(xct_t* x);
    void                    detach_xct(xct_t* x);
    xct_t*                  xct();
    const xct_t*            const_xct() const;
    static smthread_t*            me();

    // set and get lock_timeout value
    long                    lock_timeout() const;
    void                    lock_timeout(long i);

    /*
     *  These methods are used to verify than nothing is
     *  left pinned accidentally.  Call mark_pin_count before an
     *  operation and check_pin_count after it with the expected
     *  number of pins that should not have been released.
     */
    void                    mark_pin_count();
    void                    check_pin_count(int change);
    void                    check_actual_pin_count(int actual) ;
    void                    incr_pin_count(int amount) ;

    /*
     *  These methods are used to verify that a thread
     *  is only in one ss_m::, scan::, or pin:: function at a time.
```

```
     */
    void                    in_sm(bool in);
    bool                    is_in_sm() const;

private:
    void                    user(); /* disabled sthread_t::user */
};
```

DESCRIPTION

Class **smthread_t** inherits from **sthread_t,** and extends it for use by the higher layers of the Shore Storage Manager. Any thread calling methods documented in *ssm* section manual pages (see **intro(ssm)** for a list) must be an **smthread_t** or derived from it.

sthread_t(priority, block_immediate, auto_delete, name, lock_timout)

See **sthread_t(sthread)** for details on the *priority, block_immediate, auto_delete* and *name* parameters. The *lock_timout* parameter specifies the default for how long a lock request by the smthread should block before it times out.

run()

This method is the body of the thread. See **sthread_t(sthread)** for more details. Users must provide their own **run** method.

Methods pertinent to Transactions

Threads often run on behalf of a transaction, so there are methods for associating a thread with a transaction. For more information on transactions, see **transaction(ssm)**

attach_xct(xct)

The **attach_xct** method attaches the thread to transaction *xct.* Any SSM operation, performed by this thread, that requires transaction information will use the *xct* transaction. For example, all locks acquired by operations will be for the *xct* transaction. The **ss_m::begin_xct** method automatically calls **attach_xct.** It is a fatal error to call **attach_xct** if the thread is already attached to a transaction.

More than one thread can operate on behalf of a given transaction at any time, but certain transaction-related activities are serialized with a synchronization variable. For example, only one of the threads can be writing log records for a top-level (compensated) operation at any time. Another example of such serialization involves the lock manager: if any single thread of a multi-threaded transaction waits on a lock, all of the transaction's threads that would block in the lock manager wait on the same lock (regardless what locks they are trying to acquire).

A VAS that attaches a transaction to more than one thread runs a high risk of getting latch-latch deadlocks among threads. It is the responsibility of the VAS to implement its own protocol for avoiding these deadlocks. An example of such a protocol is to allow threads of multi-threaded transactions to work on non-overlapping partitions of the database.

It is also the responsibility of the VAS to see that certain operations, including commit and abort, are not attempted while a transaction is attached to several threads.

detach_xct(xct)

The **detach_xct** method detaches the thread from transaction *xct.* It is a fatal error if the thread is not already attached to *xct.*

xct()

The **xct** method returns the transaction to which the thread is currently attached.

ERRORS

TODO

EXAMPLES

See **Writing Value-Added Servers with the Shore Storage Manager** for an example of how to use threads in a server.

VERSION

This manual page applies to Version 2.0 of the Shore Storage Manager.

SPONSORSHIP

The Shore project is sponsored by the Advanced Research Project Agency, ARPA order number 018 (formerly 8230), monitored by the U.S. Army Research Laboratory under contract DAAB07-91-C-Q518. Further funding for this work was provided by DARPA through Rome Research Laboratory Contract No. F30602-97-2-0247.

COPYRIGHT

Copyright (c) 1994-1999, Computer Sciences Department, University of Wisconsin -- Madison. All Rights Reserved.

SEE ALSO

intro(sthread) sthread_t(sthread) transaction(ssm) lock(ssm)