Prioritized Mutual Exclusion Support for RT-CORBA

Relevant Constraint by OMG RT-CORBA Specification:

- Fixed Priority Scheduling only.
- RT-ORB relies on RTOS to schedule threads that represent activities being processed and to provide mutexed to handle resource contention.
- End-to-end predictability should be supported.
- End-to-end predictability:

  Respecting thread priorities between client and server for resolving resource contention during the processing of CORBA invocations.

  Bounding the duration of thread priority inversions during end-to-end processing.

  Bounding the latencies of operation invocations.

- RT-CORBA is not restricted/relied to the OS that implements POSIX real-time extensions (Which ensures end-to-end predictability).
- RT-CORBA view of a thread is compatible with POSIX definition of a thread.
- Mutex:

  An implementation of RTCORBA::Mutex that implements some form of Priority Inheritance Protocol (PIP / PCP).

  Mutex must have same Priority Inheritance Properties as those used by the ORB to protect resources.

- Invocation timeouts to improve predictability.
- May rely on OS for notion of thread priorities.
- ORB must use a Priority Inheritance Protocol (PIP / PCP) during the executions of components.
- Mutex interface:

  Consists of lock(), unlock() and trylock()

  Created by create_mutex(), destroy_mutex() of RTCORBA interface

- If RT-ORB runs on shared memory multiprocessors, atomicity of the lock operations should be guaranteed.
- The Implementation should conform to OMG CORBA/IIOP specification.
- Priority models:

  Server declared: Predictable performance

Client propagated: No priority inversion

- Object level granularity for priority.
- OMG is standardizing dynamic scheduling techniques, such as deadline based or value based scheduling

Relevant RT Notification RFP by Objective Inteface Systems

- No changes being proposed to existing CORBA, RT-CORBA specifications.
- Event is a schedulable entity to ensure end-to-end QoS for RT-CORBA.
- Dynamic scheduling is an emerging facility.
- Existing algorithms in Notification service are O(N2) and unbounded due to flaws in the specification.
- Predictability for the existing Notification service in delivery of events is the main issue.
- Integration with RT-CORBA for configuration of priorities and scheduling.

Relevant Concurrency service specification

- Locks can be acquired in one of two ways:

  On behalf of transactions: transactions service

  On behalf of the current thread: Must be executing outside the scope of transaction (User's responsibility).

- An object is a shared resource. The implementation can use concurrency service to provide control.
- Lock modes: Read, Write, Intent locks
- Lock granularity
- Two phase commit
- Lock release at transaction completion: Lock coordinator
- Upgrading locks
- FIFO for waiting clients
- Multiple possessions

Directions to take

- TAO is an implementation of RT-CORBA. So we have to identify the components of TAO and may be ACE which take care of prioritized mutual exclusion.

- Once identified and studied, we have 2 choices:
  Add our implementation as a configurable option into TAO

  Replace the existing implementation with ours

- Depending on what we chose we have to understand the internals of ACE and TAO to be able to modify them. We have to make the tests run correctly before we can start on the study of the code.

- ORB is not a process or thread. So the Dining philosopher implementation will not quite fit the context of ORB. But that is intended to reduce risk and to have thorough understanding of the implementation requirements. The init() method of ORB interface can be modified to take care of the lock initializations. We don't have mutex_init() !!!
- We have to take care of multiplexing and demultiplexing messages of the protocol as the same port is used for the All the components on one end systems.
- Issue of upgrading locks already held by a thread.

Nirmit Desai
nvdesai@unit.ncsu.edu
10-19-2001