2.2.2 POSIX® Mutexes and Semaphores

Given the goals of MRAPI, the MRAPI working group considered POSIX® mutexes and semaphores (IEEE Standard 1003.1b) as having relevant functionality. However, the working group determined that condition variables and signaling should be considered within the scope of the future Multicore Association Multicore Task Management API (MTAPI) working group rather than the MRAPI working group. The rationale for this decision is that in order to properly implement condition variables and signaling one would require the ability to manage threads or processes, and this is what MTAPI will provide. Therefore the functionality should be considered on the Multicore Association roadmap, but deferred until MTAPI becomes available.

The POSIX® standard provides two forms of semaphores: mutexes (binary semaphores), and semaphores (counting semaphores).

2.2.2.1 POSIX® Mutexes

POSIX® mutexes are declared as part of the POSIX® Threads (pthreads) package. POSIX® mutexes are only guaranteed to work within a single process. It is possible on some systems to declare mutexes as global by setting the 'process-shared' attribute on the mutex, but implementations are not required to support this.

The following mutex types are defined within the POSIX® standard:

- PTHREAD_MUTEX_NORMAL - This type of mutex does not detect deadlock. A thread attempting to relock this mutex without first unlocking it shall deadlock. Attempting to unlock a mutex locked by a different thread results in undefined behavior. Attempting to unlock an unlocked mutex results in undefined behavior.

- PTHREAD_MUTEX_ERRORCHECK - This type of mutex provides error checking. A thread attempting to relock this mutex without first unlocking it shall return with an error. A thread attempting to unlock a mutex which another thread has locked shall return with an error. A thread attempting to unlock an unlocked mutex shall return with an error.

- PTHREAD_MUTEX_RECURSIVE - A thread attempting to relock this mutex without first unlocking it shall succeed in locking the mutex. The relocking deadlock which can occur with mutexes of type PTHREAD_MUTEX_NORMAL cannot occur with this type of mutex. Multiple locks of this mutex shall require the same number of unlocks to release the mutex before another thread can acquire the mutex. A thread attempting to unlock a mutex which another thread has locked shall return with an error. A thread attempting to unlock an unlocked mutex shall return with an error.

- PTHREAD_MUTEX_DEFAULT - Attempting to recursively lock a mutex of this type results in undefined behavior. Attempting to unlock a mutex of this type which was not locked by the calling thread results in undefined behavior. Attempting to unlock a mutex of this type which is not locked results in undefined behavior. An implementation may map this mutex to one of the other mutex types.

2.2.2.2 Mutex Analysis

After reviewing the pthreads API and semantics, the working group came to the following conclusions:

- POSIX® mutexes cannot always be shared between processes, it depends on the implementation
- forking a process that has POSIX® mutexes has pitfalls when mutexes are process-shared; for example the new child could inherit held locks from threads in the parent that do not exist in the child because fork always creates a child with one thread under the standard