XXXIII Неделя науки СПбГПУ. Материалы межвузовской научно-технической конференции. Ч.IX: С.79, 2005.

УДК 681.3.07

Feng Guian, Vaibhav Shah (graduate students, Intelligent Systems Program),
M.V.Khloudova, PhD, associate prof.

SYNCHRONIZATION FACILITIES IN POSIX

POSIX is an API standard. It stands for Portable Operating System Interface. The POSIX API is based on the UNIX process model (i.e., a single thread of control per process). But among the interface functions defined by POSIX 1003.1c are those for creating, initializing and destroying threads; managing thread resources; and scheduling their executions. When we have multiple threads running, they will invariably need to communicate with each other in order to synchronize their execution. There are few possible methods for synchronizing threads: Mutual Exclusion (Mutex) Locks, Condition Variables and Semaphores. In this report, we have focused on Mutex which is used in POSIX. It is the simplest construct used in the POSIX thread library.

A Mutex, or Mutex lock, is a special variable that can be either in the locked state or the unlocked state. Mutex is one of the most important primitives in synchronization for multithreading the kernel. These primitives are very similar to those used in other operating systems. The Mutex primitive provides mutual exclusion for one or more data objects.

As usual there are two versions of the Mutex primitives: spin Mutexes and sleep Mutexes. Spin Mutexes are a simple spin lock. If the lock is held by another thread when a thread tries to acquire it, the second thread will spin waiting for the lock to be released. Due to this spinning nature, a context switch cannot be performed while holding the spin Mutex to avoid deadlocking in case of the thread owning spin lock not being executed on a CPU and all other CPUs spinning on that lock. A second Mutex primitive, sleep Mutex, provides a context switch when a thread blocks on a Mutex. Since a thread that contests on a sleep Mutex blocks instead of spinning, it is not susceptible to the first type of deadlock with spin locks. If the Mutex is locked, it has a distinguished thread that holds or owns the Mutex.

If a Mutex is unlocked, we say that the Mutex is free or available. The Mutex also has a queue of threads that are waiting to hold the Mutex. POSIX does not require that this queue be accessed FIFO. A Mutex is meant to be held for only a short period of time. It is usually used to protect access to a shared variable. A typical lifecycle of a Mutex can be viewed as, lock Mutex, the critical section, unlock the Mutex

After the attributes for a Mutex are configured, you initialize the Mutex itself. Functions are available to initialize or destroy, lock or unlock, or try to lock a Mutex:

- **Initializing Mutex Attribute Object**
- **Destroying a Mutex Attribute Object**
- **The Scope of a Mutex**
- **Creating and initializing a Mutex**
- **Locking a Mutex**
- **Destroying a Mutex**

Mutex attributes may be associated with every thread. To change the default Mutex attributes, you can declare and initialize a Mutex attribute object and then alter specific values much like we have seen in the last chapter on more general POSIX attributes. Often, the Mutex attributes are set in one place at the beginning of the application so they can be located quickly and modified easily.