**Threading – Part 3**

Readers-Writers Problem

A common problem occurs when some threads wish to read a particular file while other threads are attempting to update, or write, to that file.

Although synchronization is needed to prevent the interleaving of read and write access, multiple threads may read the file simultaneously without causing any errors.

Moreover, depending on the need of the users or the system, reading may be given precedence over writing, or vice versa.

When readers are given precedence, a waiting reader will be permitted access to the critical section if another reader is already in the critical section, even if a writer is already waiting.

When writers are given precedence, however, a waiting writer blocks access to all subsequent readers.

Hence, a policy must be established that either favors readers or writers, and this policy must be implemented as part of the synchronization solution.

Example: readers and writers synchronization that prefers readers (taken from Bryant & O'Hallaron, p. 970):

- readers-and-writers.c

Functions that do not reference any shared memory during execution are known as reetrant functions.

Because multiple threads may execute a reetrant function simultaneously without causing errors, reetrant functions are, by definition, thread safe.

Functions that reference shared memory may be made thread safe by guarding the access to shared memory with a mutex.

Hence, all reetrant functions are thread safe, but not all thread safe functions are reetrant.

Nonetheless, some people refer to any thread safe function as a reetrant function because the guarantee against computation errors is identical.

Reentrant functions in Unix are designated with a trailing "_r", such as `rand_r`.