**Data Structures:**

\*All data structures required for the kernel-side of the mutex api are stored in proc.h.

Mutex information is stored in a global array of type 'struct mutex' with size MUX\_MAXNUM. This array is indexed by the mutex id returned by mutex\_create. Additionally, the array uses xv6's spinlock to ensure atomic read/writes. The mutex struct contains the name of the mutex that was passed into mutex\_create, and a boolean indicator of the state of the mutex:

0 => mutex not taken

1 => mutex taken

A given mutex is considered an "empty mutex", name is null and state is -1, if it has not been created by calling mutex\_create

Each process has a mutex reference table; this is an array of pointers w/ size MUX\_MAXNUM. This array is indexed by mutex id. If the process has access to a particular mutex, the pointer at the corresponding index points to the mutex in the global array. If the process does not have access, the corresponding pointer in the array is 0 (null). When a parent forks a child, the child inherits this mutex reference table from the parent - therefore descendant processes share mutex access with the root process.

Lastly, blocking process are placed into a wait queue. The wait queue is an array protected by a spinlock.

**Locking & Unlocking:**

When a process calls mutex\_take, it first verifies that is has access to the requested mutex; if the process's mutex-reference table does not contain a reference to the requested mutex, then this process does not have access to it.

In order to acquire a lock, the process uses its reference table to look up the mutex in the global array and see the mutex's state. If the state of the mutex is 0, it takes the mutex by setting the state to 1 - note that all reading from/writing to the global mutex array is done atomically. If the mutex state is 1, it blocks waiting for the mutex to become available. We use a wait queue in which enqueuing and dequeuing are atomic.

**\*Note about manipulating header inclusions:**

In order to expose the proper parts of the kernel (proc.c and sysproc.c) to my global mutex structures, I declared these structures in "proc.h". I chose "proc.h" because the proc struct in "proc.h" needed to be modified with mutex information.

Since the mutex struct contains a spinlock, the "proc.h" header file must be exposed to the "spinlock.h" header file in some way. I decided to include "spinlock.h" in "types.h" because "types.h" is always included before "proc.h" throughout the kernel. Consequently, it was necessary to remove the inclusion of "spinlock.h" from all programs and header files that already include "types.h". Here is the list of altered files for future reference:

types.h

console.c

bio.c

file.c

fs.c

ide.c

kalloc.c

log.c

pipe.c

sleeplock.c

spinlock.c

sysfile.c

trap.c

uart.c