Priority Scheduling

Task: Implement priority scheduling in Pintos.

When a thread is added to the ready list that has a higher priority than the currently running thread, the current thread should immediately yield the processor to the new thread. Similarly, when threads are waiting for a lock, semaphore, or condition variable, the highest priority waiting thread should be awakened first. A thread may raise or lower its own priority at any time, but lowering its priority such that it no longer has the highest priority must cause it to immediately yield the CPU.

Thread priorities range from PRI_MIN (0) to PRI_MAX (63). Lower numbers correspond to lower priorities, so that priority 0 is the lowest priority and priority 63 is the highest. The initial thread priority is passed as an argument to thread_create(). If there's no reason to choose another priority, use PRI_DEFAULT (31). The PRI_ macros are defined in threads/thread.h, and you should not change their values.

One issue with priority scheduling is "priority inversion". Consider high, medium, and low priority threads *H*, *M*, and *L*, respectively. If *H* needs to wait for *L* (for instance, for a lock held by *L*), and *M* is on the ready list, then *H* will never get the CPU because the low priority thread will not get any CPU time. A partial fix for this problem is for *H* to "donate" its priority to *L* while *L* is holding the lock, then recall the donation once *L* releases (and thus *H* acquires) the lock.

Implement priority donation.

You will need to account for all different situations in which priority donation is required.

Be sure to handle multiple donations, in which multiple priorities are donated to a single thread. You must also handle nested donation: if *H* is waiting on a lock that *M* holds and *M* is waiting on a lock that *L* holds, then both *M* and *L* should be boosted to *H*'s priority. If necessary, you may impose a reasonable limit on depth of nested priority donation, such as 8 levels.

You must implement priority donation for locks. You need not implement priority donation for the other Pintos synchronization constructs. You do need to implement priority scheduling in all cases.

You may have also modify or add the following functions:

Function: void thread_set_priority (int *new_priority*)
Sets the current thread's priority to *new_priority*. If the current thread no longer has the highest priority, yields.
Function: int thread_get_priority (void)
Returns the current thread's priority. In the presence of priority donation, returns the higher (donated) priority.
You need not provide any interface to allow a thread to directly modify other threads' priorities.

There are some test cases which may have been defined in PintOS. However, to make the assignment reasonably easier, you can write your own test programs to show how your scheduler works.

Hence you will need to provide a good design document which outlines the design choices.