Data Structures

I didn't need to add or change any data structures, I did and the abstract methods `void ensure_max_priority(void);` and `bool compare_priority(const struct list_elem *a, const struct list_elem *b);` in thread.h, and `bool compare_sema_priority(const struct list_elem *a, const struct list_elem *b);` in synch.h, these methods will be detailed in the algorithms section.

Algorithms

My solution for part A is based around the ensure_max_priority and compare_priority methods added to thread.c.

ensure_max_priority begins by checking if there are threads on the ready list, if there are no threads the method can exit saving execution time. The method then retrieves the top of the ready list and compares it's priority to the current thread, if the current thread has lower priority then it yields the processor. The ready list is assumed to be ordered by priority so only the top entry needs to be checked.
ensure_max_priority is used after a new thread is created and after a threads priority is changed. thread_set_priority will exit before making changes and calling ensure_max_priority if the new priority equals the current priority.

The ready list is kept ordered because threads are added to it with the list_insert_ordered method from list.c which runs in O(n) as an average case. The ordering is determined by the compare_priority method.

compare_priority is a simple method that takes two list elements, extracts the threads then compares their priority. compare_priorit returns true if the first element has a higher priority than the second otherwise it returns false.

My solution for part B reuses the ensure_max_priority and compare_priority methods and also adds the compare_sema_priority method in synch.c.

compare_sema_priority is used to sort the list of waiting semaphores for a conditional variable, sorting is done in the cond_signal method with list_sort from list.c which runs in O(n lg n) time.
compare_sema_priority takes two semaphores and compares their priority returning true if the first has higher priority and false otherwise.
compare_sema_priority is much like compare_priority except first it checks that there are threads waiting on both semaphores. If b has no waiting threads then the method can return immediately, either false if a also has nothing waiting or true if it does. Likewise if a has no threads but b does then false can be returned immediately. These checks save processing time on sorting the waiting threads. The waiting threads are sorted using list_sort with compare_priority from part A. Once sorted the top threads can have priorities compared as in compare_priority.

In sema_up waiting threads are sorted using list_sort with compare_priority before the top waiting thread is unblocked. ensure_max_priority is used at the end of the method to ensure that the unblocked thread will gain the cpu if it has higher priority than the current thread.

Synchronization

To avoid race conditions ensure_max_priority disables interrupts, as do thread_set_priority and thread_get_priority.