To change the supplied FCFS ready queue into a priority based ready queue:

thread.c (and thread.h):

- In thread_init, set the original priority to be the given priority.
- In thread_create, we force the current thread to yield if its priority is lower than the priority of the new thread. This is so that no lower priority thread runs while a higher priority thread is also ready to run.
- In thread_set_priority, we create a temporary thread (set to current thread). This is so we can save it and use its fields more easily. Then we force the current thread to yield if its priority is lower than the priority of the temporary thread. Again, this is so that no lower priority thread runs while a higher priority thread is also ready to run.
- I created function is_greater_priority that takes two threads and returns true if the first thread's priority is greater, or false if the second thread's priority is greater. This was done to help modularize because this function will be used often throughout the entire code. I made sure to include this function in thread.h to avoid a potential error for not being initialized.
- In thread_unblock, when inserting back into the list, instead of putting the thread at the end, we must insert threads in order of highest priority to lowest priority. The same is done when inserting in thread_yield. This is so that the ready queue can just pull the first thread from the front and ensure that it will be a high priority thread every time.
- All tests still failed so I decided to come back to this later.

To implement priority waiting for locks, semaphores, and condition variables:

semaphore.c:

- In semaphore_up, I created a boolean variable that would determine whether the thread should yield, determined by whether the thread from the front of the ready queue had a higher priority or not (it must yield after the semaphore value is incremented, otherwise wouldn't work). This is so that we can check if this is true after the semaphore is incremented to yield the thread.
- All tests still failed (technical difficulties) so I moved on to lock.c

lock.c:

- I changed the lock_try_acquire function to check if the semaphore_try_down was successful, then change the current thread's lock1 to null.

condvar.c (and condvar.h):

- In condvar_wait, I changed the part where the thread adds to the end of the list to rather insert into the ready queue in descending order of priority. This is so that higher priority threads are always at the front of the ready queue.
- I created a function compare_semaphores that would be used in condvar_signal to sort the ready queue according to the respective semaphores. This was added in condvar.h accordingly.

- At this point I was experiencing technical difficulties with testing, and found it was because my Filezilla application was not letting me transfer the files from my local machine correctly.

To implement single-level priority donation, including multiple donations:

thread.c (and thread.h):

- For init_thread, I initialized variables for a donation list, originalP (specified below) and lock. These were added to the thread.h file accordingly to prevent any problems.
- I decided to work more on the thread_set_priority. I needed to also check if the current thread's priority had changed, and backup the old priority (originalP) before taking the thread from the from of the ready list. This way, I saved the priority to switch back to what it was previously set to once donation was done. This also works in case of multiple donations.
- I made a function called remove_lock that takes the lock off the donation list.

lock.c:

- For lock_release, I added 2 lines such that after lock's holder was set to null, the lock would be removed and priority would be increased. I also made it so the lock was removed from the donation list. Thread priority was updated accordingly.

semaphore.c:

- In semaphore_down, I added code so that when the semaphore value was zero, it would donate priority and insert into the list in order of priority before blocking the thread.

More testing:

- To investigate why the tests were timing out, I went back and remembered that they worked until I added the function that compared two threads' priorities. When I went back to the function and changed the way it returned, the tests ran smoothly.
- I ran make in my terminal and had a few syntax errors (undeclared variables etc) that were a quick fix.