a. Additional assumptions not specifically mentioned in project1.pdf include:
- Only suspended threads can be resumed
- Only READY or RUNNING thread(s) can be suspended
- Only one thread can be blocked trying to join on a specific thread ID.

b. I did not add any customized APIs

c. Input / output parameters are passed to a thread's entry function (called start_routine in the implementation files) using a void pointer. This void pointer is initially passed by the programmer to uthread_create, which then passes it to the TCB constructor, where it then gets passed to a stub function using the makecontext() function, at which point it is finally passed to the thread's entry function. To invoke the new thread execution, makecontext must create the initial thread context. Specifically, makecontext must initialize the thread's stack which is stored in that thread's corresponding TCB. This includes initializing the return address on the stack to point to the location of stub's instructions, along with placing the arguments—which include the thread's actual entry function, and the void pointer which references the input/output paramenters—on the same TCB stack with correct alignment.

d. Given that the uthread library does not support parallelism, the main performance benefit of shorter time slices can be seen in instances where asynchronous blocking system calls are used by the programmer. In such situations, shorter time slices allow for more frequent polling of blocked threads, which might provide lower latency. This does, however, mean that context switches will happened more frequently which is lost time that could be used by non-blocked threads. It is for this reason that longer time slices might result in better performance for programs in which thread's are expected to run to completion with infrequent blocking—since there will be fewer unnecessary context switches.

e. With the exception of uthread_init and uthread_self, interrupts are disabled almost immediately after entering any uthread library functions, and re-enabled just before returning from that call. The reason for this is that the uthread library functions all involve multiple reads and writes to shared data structures. As such, unanticipated context switches could potentially lead to undesirable race conditions. Consider, for example, what would happen if uthread_suspend did not disable interrupts before checking to see if the ready queue is empty. Imagine that uthread_suspend is trying to suspend the currently running thread. This is not a problem if there is at least one thread in the ready queue which can be switched to. However, if an interrupt comes between the check that ensures the ready queue is not empty, and the call to popFromReadyQueue, there is no guarantee that ready queue has not be modified by another thread, potentially resulting in an exception being thrown for trying to pop from an empty queue.

f. I did not implement a more advanced scheduling algorithm.