What is it?

1. This project implements a User-space thread library in C, like the POSIX threads library.
2. It implements “green threads", i.e. cooperative threads with no preemption.

Usage?

1. The functions provided by the thread library allow users to implement their own functions which accept a single argument, pass a pointer and a variable to thread on creation, and manage the thread execution with mutex variables, condition variables, and thread join functions.

What are its limitations?

1. Only developed and supported on Ubuntu for x86 and aarch64 systems. The context switching function (saving and switching between stack pointers between 2 threads) is written in assembly and supported in Ubuntu only.
2. Does not support pre-emption, only cooperative threading, with the expectation that the user of this threading library will use thread yielding function appropriately & frequently to avoid deadlock scenarios.
3. Does not support more than 32 sleeping threads at a single time.
4. Does not support more than 1 thread waiting at join for a thread to exit.

Thread Structure / Representation

```
struct qthread {
   /* your code here */;
   struct qthread *next;
   /* need this stack base pointer to let us free space allocated for stack once thread exits */
   void *stack;
   void *saved_sp;
   int exited_flag;
   void *exit_val;
   struct qthread *waiting_for_me; /* pointer to thread waiting for this thread to exit to do join */
   long wake_time;
};
```

Functions Provided:

Thread library initialization:

- void qthread_init(void) = function to initialize the threading library and its structures/variables for proper thread queueing/scheduling. Maintains a struct to represent the original OS created thread which calls this library.

Thread management:

- qthread_t qthread_create(f_1arg_t f, void *arg1) = function to create a thread structure with appropriate thread management variables and accepts a function pointer(f_1arg_t f), and parameter pointer (void *arg1), which is to be passed to function to be executed by the thread.

- void qthread_exit(void *val) = called by qthread_create after execution of the of the function passed in qthread_create. The exit function accepts a return value from the function executed.qthread_exit sets this value in the thread structure and puts any threads waiting for current thread exit to join on the active queue of threads.

- void qthread_yield(void) = function to put current running thread to the end of the active threads queue and execute thread at the front of the active threads queue.

- void *qthread_join(qthread_t thread) = function which accepts a pointer to a thread, and waits for the thread to exit and then get its return value. This function allows a thread to be joined by the current running thread.

Mutexes and condition variables:

- qthread_mutex_t *qthread_mutex_create(void) = creates a mutex queue & lock variable

- void qthread_mutex_lock(qthread_mutex_t *m) = acquires lock, or puts in mutex_queue

- void qthread_mutex_unlock(qthread_mutex_t *m) = releases lock

- void qthread_mutex_destroy(qthread_mutex_t *m) = destroys/frees mutex variable

- qthread_cond_t *qthread_cond_init(qthread_cond_t *c) = creates a condition variable queue

- void qthread_cond_wait(qthread_cond_t *c, qthread_mutex_t *m) = puts current thread onto condition variable queue

- void qthread_cond_signal(qthread_cond_t *c) = wakes up first thread in condition variable queue by putting it on the active queue

- void qthread_cond_broadcast(qthread_cond_t *c) = wakes up all threads in condition variable queue and puts them all on active queue

- void qthread_cond_destroy(qthread_cond_t *c) = destroys/frees condition variable

Thread State Diagram