Threadpool

Program Description and What You Need to Do:

You will write one source file, threadpool.c (the executable file should be called pool). In order to enable multithreaded program, the program should create threads that handle the tasks it needs to do. Since, the program should maintain a limited number of threads, it construct a thread pool. In other words, the program creates the pool of threads in advanced and each time it needs a thread to handle a task, it take one from the pool or enqueue the request if there is no available thread in the pool.

Command line usage: pool <pool-size> <number-of-tasks> <max-number-of-request>

- pool-size is the number of threads in the pool
- number-of-tasks is the number of tasks to do
- number-of-request is the maximum number of tasks your program will handle before it terminates (we count also unsuccessful tasks). This parameter implies that the program does not run forever.

The threadpool

The pool is implemented by a queue. When the program creates a task, it should put the task in the queue. When there will be available thread (can be immediately), it will handle this task.
You should implement the functions in threadpool.h.
You should have a main function that should first init the thread pool by calling the function create_threadpool(int). This function gets the size of the pool.

create_threadpool should:

1. Check the legacy of the parameter.
2. Create threadpool structure and initialize it:
    a. num_thread = given parameter
    b. qsize=0
    c. threads = pointer to <num_thread> threads
    d. qhead = qtail = NULL
    e. Init lock and condition variables.
    f. shutdown = dont_accept = 0
    g. Create the threads with *do_work* as execution function and the *pool* as an argument.

do_work should run in an endless loop and:

1. If destruction process has begun, exit thread
2. If the queue is empty, wait (no job to make)
3. Check again destruction flag.
4. Take the first element from the queue (*work_t)
5. If the queue becomes empty and destruction process wait to begin, signal destruction process.
6. Call the thread routine.

dispatch gets the pool, pointer to the thread execution routine and argument to thread execution routine. dispatch should:

1. Create work_t structure and init it with the routine and argument.
2. If destroy function has begun, don't accept new item to the queue
3. Add item to the queue

destroy_threadpool

1. Set don't_accept flag to 1
2. Wait for queue to become empty
3. Set shutdown flag to 1
4. Signal threads that wait on empty queue, so they can wake up, see shutdown flag and exit.
5. Join all threads
6. Free whatever you have to free.

Program flow:

1. The main function creates a pool of threads, threads wait for jobs.
2. Create a task function that prints the thread id 1000 times, add sleep for 100 milliseconds after each print.
3. dispatch a job - call dispatch number-of-tasks times with the task function. dispatch will add work_t item to the queue.
4. When there is an available thread, it will take a job from the queue and run the task function.

Error handling:

1. In any case of wrong command usage, print

"Usage: pool <pool-size> <number-of-tasks> <max-number-of-request>\n"

2. In any case of a failure before running the task function, use perror(“error: <sys_call>\n”) and exit the program.

3. In any case of a failure within the thread, close the current thread.

Compile the program:

Remember that you have to compile with the –lpthread flag.

What to submit:

You should submit a tar file called ex2_a.tar with threadpool.c and README. Find README instructions in the course web-site. **DON'T SUBMIT threadpool.h**

tar –cvf ex2_a.tar threadpool.c README

Test your code:

Run the program with different parameters, start with one thread and one task and invrease the parameters slowly.

Good Luck!