Description of structure created:

1). thread_pool: This structure represents the fork-join thread pool that contains one or more threads and complete the tasks that the driver submitted or the tasks that the worker threads submitted (if a task generate more tasks). In this structure, we have a boolean variable 'shutDown' indicating whether the pool is shutting down, a list of tasks (the global queue), a number indicates the number of threads in the thread pool, an array of pids of threads in the thread pool, a pool_lock that protects the fields in the thread_pool (particularly the global queue and the status of the thread pool) and a semaphore indicate whether the pool is empty.

2). Future: In this structure, we have a fork join task to be completed which is called 'task' (the function), a void* 'args' represents the argument to be passed into the function, a void* 'result' represents the result that the function returned, a semaphore represents if a particular tasked is completed, a thread_pool pointer that points back to the thread pool, a 'status' indicates the status of the future (0 - not completed; 1 – being processed; 2 – completed) and a lock to protect the future (particularly the status of the future).

Description of functions:

1). Thread_pool_new: This function just creates a new thread pool and initialize all fields in the structure. Then the function creates n threads that is required in the thread pool.

2). Thread_execute: This is the helper function that I created. Every thread in the thread pool execute this function. The function is called in the phtread_create in thread_pool_new. A while loop keeps the threads running until the thread pool is being shut down. A sem_wait at the beginning keeps the threads waiting if the thread pool is empty. Then we check if the pool is shut down, if so, the thread will return and exit. If the thread pool is not shut down, the threads will pop a future from the global queue and execute it. Then it changes the status of the future to 2 which is completed. Then sem_post towards the future is called and the thread in future_get will receive the signal and stop waiting for the future to be completed.

3). Thread_pool_shutdown_and_destroy: This function change the status of the threadpool to shut down, destroy all the locks and semaphores in the thread_pool structure, join all the threads and free all the memory allocated in the thread pool create function.

4). Thread_pool_submit: This function initializes all the fields in the future structure and put the future into the global queue in the thread pool.

5). Future_get: In the future_get function, we check the status of the future that is to be get. If the future is completed or in process, a sem_wait will block the thread until a signal is sent after the future is completed in the thread_execute. If not, the future is removed from the global queue, taken and completed by the thread calling future_get.

6). Future_free: destroy the lock and semaphore initialized and free all the dynamically allocated memory associated with the future structure.

Overall design: Generally, the threads in my thread pool will take tasks from the global queue in the thread pool and execute it if the global queue is not empty. To speed up the process, in the

future_get function, I check whether the future is completed or it’s in process, if so, I will wait until it is finished and return the future, if not, the thread will take the future and execute it.