Student Information:

Jeff Chen (chenjeff4840)

Jackson Burkey (jacksonb01)

Project description:

- Used work stealing approach. No additional optimizations mentioned in extra credit in project spec is implemented.

Description of Base Functionality:

- struct thread_pool* thread_pool_new(int nthreads)
    - Creates a dynamically allocated threadpool. Initializes it then returns it.
- void thread_pool_shutdown_and_destroy(struct thread_pool* pool)
    - Sets shutdown flag on. Broadcast to the threads using the same pathway used for signaling a new task (threads will read shutdown flag first before any other actions). Afterwards, destroy global queue. Then wait for all worker threads to shutdown. Then clean worker thread dynamically allocated data and remaining fields of the threadpool.
- void future_free(struct future* fut)
    - Destroy and clean up fields from fut.
- void* future_get(struct future* fut)
    - Check if is worker thread. If is worker thread, check if fut is queued. If queued, remove it from the queue it is in, update status, work on the queue, update its status again, and return result.
        - If worker thread and not queued or is not worker thread, if fut is not complete, wait until it is completed then return result.
- struct future* thread_pool_submit(struct thread_pool* pool, fork_join_task_t task, void* data)
    - Initializes an empty future. Set status to queued. If task is internal (called by worker thread), submit to the front of worker's queue. If external, submit to back of global queue. Update current queue field for the future. Notify threads that a task is available then return future.

Description of Private Functions:

- static void worker_init(struct worker* w)
    - Initialize a worker struct. No threads are created,
- static void thread_pool_init(struct thread_pool* pool, int nthreads)
    - Initialize threadpool with nthreads. All fields such as the pool's worker and worker threads are initialized here.
- static void future_init(struct future* fut, fork_join_task_t, void* args, struct thread_pool* pool)

    - Construct a future using the provided arguments. Arguments not provided such as current queue, state, and result are set to default values.
- static struct future* get_future_front(struct list* queue)
    - Returns list element in front of queue as a future. Remove the list element from the queue.
- static struct future* get_future_back(struct list* queue)
    - Same as previous function but back of queue

- static void queue_clean_all(struct list* queue)
    - Removes and frees all list elements within queue.
- static int search_all_queues(struct thread_pool* pool)
    - Searches all queues within pool for an available task. Checks worker's own queue first, then global queue, then from queues of other workers. If a task is found, the queue id it is found in is returned. If not found, an impossible queue id is returned.
- static void run_task(struct thread_pool* pool, int q)
    - Looks at queue id q within q and grabs a task. If q matches worker thread's id, it will grab task from the front of the worker's queue. If in any other queue, it will grab task from the back of the queues. The task is retrieved as a future. The future status is updated, ran, then updated again with run results,
- static void* worker_thread_job(void* args)
    - Worker receives data from args which is converted into poolID struct containing thread pool and worker_id assigned to the worker. After initializing worker's _Thread_local id, it will run in a loop until shut down flag is triggered. Immediately upon entering the loop, the worker thread waits for thread_pool_submit to be triggered. Once triggered, the worker will check if shutdown flag has been triggered. If triggered, it will exit the loop. If not, it will enter another loop where it will continuously search for a task within any of the queues then run them. When no more tasks exists, it will exit both loops. After exiting, the worker thread cleans up its worker_id struct and its data then exits.
- static bool remove_from_queue(struct list* queue, struct future* target)
    - Searches and removes target from queue starting from the front of the queue to the back of the queue. Only 1 list element matching target can be removed. Returns true if removed, false if not removed.

Structs:

```
/**
 * Used as worker threads.
 */
struct worker
{
        struct list local_queue;          // Queue held locally by worker
thread
        pthread_mutex_t local_queue_lock; // Lock for local queue
        pthread_t thread;                 // Worker's thread (may change
later on)
};

/*
```

```
 * Opaque forward declarations. The actual definitions of these
 * types will be local to your threadpool.c implementation.
 *
 * Has all kinds of locks, however, is necessary since there is a condition
 * and having a universal lock for threadpool and waiting on queued_cond is
bad news
 */
struct thread_pool
{
        uintptr_t id;              // ID of worker
        struct worker *workers;            // Worker threads working for
threadpool, unknown number of workers
                                           // Worker lock might not be needed
as only accessed for queue and queue has own lock
        struct list global_queue;          // Global queue accessed by all
workers
        pthread_mutex_t global_queue_lock; // Lock for global queue
        pthread_cond_t queued_cond;        // Conditional to wake workers up
when new task is queued
        pthread_mutex_t queued_lock;       // Lock that goes together with
queued conditional
        int worker_count;                  // Number of workers
        pthread_mutex_t worker_lock;       // Lock for worker arr, needed
since threads are started while some threads are active
        bool shutdown;                     // Shutdown flag, true to shut
down, false to not
        sem_t t_continue;                  // To allow broadcast to go
through
};

/**
 * Worker ID, used as _Thread_local
 */
struct workerID
{
        uintptr_t id; // ID of worker
};


struct future
{
        struct list_elem elem;        // List tag element
        int curr_queue;              // Current queue future is in, -1 for
global queue, <-1 for not set
        fork_join_task_t task;       // Task future will execute
        void *args;                  // arguments for task
        void *result;                // Results from running task
        int state;                   // Current state of task, 0 = queued, 1
= in progress, 2 = finished, -1 for unset
        struct thread_pool *pool;    // Threadpool instance that
contains/contained future
};


/*
 * Submit a fork join task to the thread pool and return a
 * future.  The returned future can be used in future_get()
```

```
 * to obtain the result.
 * 'pool' - the pool to which to submit
 * 'task' - the task to be submitted.
 * 'data' - data to be passed to the task's function
 *
 * Returns a future representing this computation.
 */
struct future *thread_pool_submit(
    struct thread_pool *pool,
    fork_join_task_t task,
    void *data);
```

Globals:

```
static _Thread_local struct workerID *worker_id; // External threads == NULL,
Worker threads
                                                 // ID corresponds to worker
placement in struct
                                                 // thread_pool's worker*
workers.
```