UNIVERSITY OF SOUTHERN CALIFORNIA

CS402 Assignment 01 – Commands

Michael Gerow

February 22, 2013

1 e Command

The e command works by calling a function called create_client() which calls the client_create function in order to allocate the client. After the client is created we then call add_client_to_thread_handler which adds the client to the thread handler's list of running clients. In order to do this it gets a lock on the list, adds itself to it, and then unlocks. After that we simpl call the launch_client_thread function which calls pthread_create on the thread, running the client_run code.
When the thread has finished it simply sets its (newly added) done flag to true and increments the thread handler's semaphore. This causes the thread handler to go through its list of clients looking for one to clean up. When it finds one it calls pthread_join on the thread and cleans up any of its other resources.

2 E Command

The E command works very similarly to the e command with the exception that it pulls additional arguments from the command command interface in order to acquire the input and output files it needs. Once it has these it simply calls create_non_interactive_client which calls client_create_no_window which we, like with the e command, simply add to the thread handler's list of running clients and then launch the client in much the same way.

3 s Command

The s command by using a pause flag in the thread handler's data. When the command is called we simply acquire the mutex for that flag, set the flag to true, and then unlock the mutex.

In the client threads we call a function called wait_on_pause which first checks to see if we are paused, and if we are, acquires the pause mutex and checks again. If we are still paused it waits on the pause_cond condition variable which is broadcasted when the g command is run.
The intention of this double checking is to prevent unnecessary locking and unlocking of the pause mutex, which I assumed would cause performance issues. In my tests, though, I have found only a moderate if any increase in performance due to using this technique.

4 g Command

The g command simply grabs the pause mutex, sets pause to false, and then broadcasts the pause_mutex, freeing all the threads blocking on this condition and allowing them to continue execution.

5 w Command

In order to handle the w command we first get a lock on the clients list and, if it is not empty, wait on the threads_done_cond condition variable, which is fired on the thread handler when it cleans up a thread and notices that there are no more threads in the list. Once this happens the thread continues through and allows us to execute additional commands.

6 EOF Handling

Although not a command exactly, EOF is handled as requested in the assignment. When we see an EOF we simply do the same thing as the w command (get a lock on the clients list and wait on a condition if it is not empty) except we exit after we block instead of waiting for new commands.

7 t and T Commands

In order to properly test the speed of the different implementations I have also added a t and T command. Running the t command will start a kind of timer (basically it will just record the current time as accurately as we can). If we later call the T command it will determine the amount of time that has passed since t was last called and the spit that value out to stdout. This is useful for accurately measuring the amount of time an implementation takes on a given data set.