Nathan Larson
CSC 412

Program 05

3.1: Movement of Travelers

Each of the travelers start at a random position on the grid with a random orientation. Each loop of their execution, a traveler chooses a random orientation perpendicular to their current orientation, then calculates a distance within the dimensions of the grid. If there is enough of the corresponding ink available in the ink tank, then the traveler will take enough ink to travel the total distance in that direction, otherwise it will not travel and choose a new orientation & distance on the next loop. Due to the implementation of the last part of extra credit, travelers can get stuck in a “deadlock”, where each traveler is trying to acquire the others lock. When a traveler reaches one of the four corners in the grid, the traveler “leaves” the grid and the thread is terminated.

3.2: Color Trails

Each of the travelers leave a trail of their corresponding color following their path along the grid. In order to travel a calculated “random” distance, a traveler must check to see if there is enough ink in the corresponding colors ink tank. If there is enough ink, then the traveler will remove the total required amount of ink to travel the distance then leave that color trail behind them as they travel. If there is not enough ink to travel the total current distance, the traveler will simply skip the rest of the loop without moving and start from the beginning again, calculating an orientation, distance, then attempting to acquire ink again hopefully with enough resources this time.

3.3: Traveler Threads

Each of the travelers has its own thread that controls its movement inside of the grid. This thread controls the orientation, distance, and whether or not the proper amount of resources are available. When a traveler reaches the

4.1: Color Levels

To show that multiple travelers have traveled over a given grid square, each time a traveler passes over a given grid square it adds 64 of its corresponding color to the RGB value of that grid square up to a max of 255. For example if green travelers keep passing over a single grid square, the square will continue to get a brighter and brighter shade of green until it reaches the max value of 255. If a couple red and a couple blue travelers pass over the same square, the square will display some approximate value relative to the color purple.

4.2: Maintain and Synchronize Traveler Info

In order to maintain and synchronize traveler info, I added an array of pthread mutex locks and allocated space for the size of the number of travelers. I originally implemented the locks to acquire before the moveTraveler() method that handles the grid color modification and next grid square movement, but since I also implemented the mutex locks for each grid square this caused problems. Since that implementation opened up the possibility of deadlocks between travelers, and the traveler info locks were acquired before the moveTraveler() method and released after the moveTraveler() method, if even one pair of travelers encountered a deadlock then the whole collection of travelers would freeze, since one of them are waiting to acquire a lock that will never be released resulting in the traveler info lock never being released and the whole display will freeze. To fix this, I had each thread acquire the traveler info lock after the corresponding grid locks are acquired and released, then modify the column or row to move the traveler, then release the traveler info lock. Since the row and column values are the only ones that are both

modified by the traveler threads and read by the gl_frontend.c script, I decided it was only necessary for the traveler threads to acquire the locks when they are modifying either the row or column value, since modifying the other values have no effect on the front end display.

4.3: Add Ink Producing Threads

Each of the colors have a tank that can be filled by “producers” and emptied by “consumers”. The consumers in this case are the travelers, which require ink to travel along the grid squares a leave a trail of their color. With this section I added ink producing threads (6 total, 2 for each color in my hard-coded example). The ink producing threads were fairly simply, containing an infinite while loop that first attempts to acquire its corresponding ink color mutex lock, then attempting to fill the tank relative to the MAX_FILL_INK value if there is enough room. The producer then releases the ink mutex lock and sleeps for a predetermined amount of time that can be modified by the user by pressing “.” and “,”.

4.4: Synchronize Access to Grid Squares

Implementing this section was similar to the implementation in program 04 that required each grid square to have an associated mutex lock. It was a straightforward implementation, which could easily be determined to be properly implemented since the deadlock problem quickly became apparent, where 2 travelers facing each other would be stuck trying to acquire each others lock before releasing their own. This would sometimes result in more travelers attempting to acquire one of the “deadlocked” grid squares, forming a group of travelers that could not move due to being locked out of a grid square since they cannot acquire the next lock.