CSC 412 Final Project (Box pushing) Report

Professor Hervé

December 20th 2018

Overall, we both found this assignment to be very educational and also enjoyable to complete. It was a good way to bring together many topics covered in the class like threads, locks and synchronization. For all of our versions we use a struct called Robot to hold all of the data a robot needs to push its box into the correct door (thread ID, its box index, the door for the box index and locations). Inside the edited initializeApplication function we allocate all of the memory we need using the information provided on the command line. The global data we used is the door locations (since they do not move), an array of robot struct (contains its own memory, but still must be accessed by the draw functions), and all of the locks we use.

There are separate functions for moving the robot, positioning the robot around the box, pushing the box, finding path to box and finding path to door. The two functions to find paths were just comparing to locations and was based on the pathfinding algorithm in the assignment sheet. It was ideal to keep all of these functions separate as it was easier to understand exactly where our locks were going to be needed. Both the move robot and push box functions are implemented in similar ways. They both are given a direction from the path finding functions and simplify modify location variables contained inside the robot struct which then is displayed as movement with some glut magic.

When it comes to locks version 2 and 3 needed a lock every time the robot was going to write to the log file to make sure no 2 robots did it at the same time. For version 3, since we had to take collision into account, we decided to use a grid of locks so each square could be locked and unlocked as robots and boxes moved across the grid. Based on the tests that we ran for version 1 and 2 of the program, there were no limitations that we could identify. Any limitations that had been present and identified in these versions, we worked to remove them from the program. The same is true for version 3 of the program, excluding potential deadlocks. However, the previous limitations that we worked to remove certainly can be put into the category of “difficulties we ran into.”

Initially, a problem that we had was a segmentation fault when trying to run to program with more than 7 robots, which became very hard to find the source of. As it turns out, when searching through the log file which would be partially populated before encountering the fault, we noticed that some of the values for door indices were garbage values which would then lead to a segmentation fault. We modified our randomization algorithm to solve this problem, but without the invaluable log file we may not have been able to resolve that limitation.

We attempted to do some deadlock detection and resolution (it is still in version 3 but commented out, you will see in the comments where I explain). These attempted implementations were very buggy but would sometimes getting crashing robots and boxes to resolve and find their way to the door. As stated previously, this project was quite enjoyable, and we believe our final product is quite satisfactory.