CSC 412 - Programming Assignment Final, Fall 2019

Jordan Coats

***Abstract*— The objective of this assignment is to get combine several things from earlier in the semster including pthreads, mutex locks, mild deadlock management.**

I. INTRODUCTION

This assignment was to create a robots on a grid that push boxes. They are implemented as threads and have a very basic pathfinding AI to move to the box and to push the box to a door. Early on in the assignment I went to TA help hours with Chris to get as much insight into the best approach as possible. As such Chris did decide the best approach was to modify some areas that probably should have been left alone such as functions inside the gl files being moved to main.cpp and the elimination of many extern globals.

II. VERSIONS

To start off with I would like to explain that I only have two versions of the program. When creating Version One I just decided to implement the multithreading since we already accomplished that with our Traveler program. I started and built everything with that in mind, since I basically had finished Version 2 with the completion of my first version I decided to not waste time dumbing it down to a Version 1. I did ask Chris if this was alright and he seemed to confirm, but I won't hold it to him, that all you actually need is a working Version 3 for all credit. So with that in mind I did include my Version 2 and Version 3 apps as I feel like my Version 3 is not complete.

III. ROBOTS.H

To start with I would like discuss the header file. As it is the backbone for the entire program I figured it is best to start with. There are several structs that I use throughout the program the most being the RobotAndBox struct. Initially I had these as two structs but since Robots and Boxes are always paired it made more sense to create them as one. Inside this struct I keep everything that the program might need to know about this pair such as locations, assigned door, and distance from each other. To run each RobotAndBox I have a RobotThread which has pointers it's RobotAndBox data, the thread ids, and it points to another struct titled RobotScene. RobotScene is a struct that encapsulates the entire grid data inside along with lists of the doors and robots and the mutex locks I use throughout the program. Since scene makes up a lot of the behind the scenes and can be used in the main.cpp and robot.cpp I figured it was best to keep the mutex locks in a place that both main.cpp and robots.cpp had access too. Apart from that my header file has the function protoypes which will be discussed further on and an enum for directions.

IV. MAIN.CPP

Apart from the obvious changes that were mandated by the assignment some of my person changes were to eliminate the use of Rand(). I had issues with it before and looked into possible alternatives. Apparently among people on StackOverflow Rand() should never be used. As such I decided to look at GeeksforGeeks and found instructions on using ¡random¿ along with random_device and default_random_engine functions. While they are new to me they seem to work exactly as described to I was happy to eliminate Rand() for my generated locations.

V. ROBOTS.CPP - ROBOT_THREAD

Next is my robots.cpp code, it starts with the thread function robot_thread. This opens the text file to write to then while the robot is going it runs the path function, which will then run move, push, and end.

VI. PATH MOVE

This probably the most complicated part of the entire program. It is boiled down to a bunch of if/else statements to make sure the box knows where it is going, how it will get there, and how long it will take. In a simplified scenario it if it still has either a negative or postive value in rowDistanceFromBox it knows to either go North or South until it hits 0. The algorithm always goes North and South first before turning and going West or East. This was done to minimize the amount of rotations they might have to do around the boxes. The move function is mostly simple as in version 3 it will just check if a space is locked, if it isn't locked, it will lock it then move that direction. It does have two rotations built in and a stop function. The stop function is only used to end a movement to begin pushing. The special rotations are NS and SN which stand for North to South and South to North.

```
rt->owner->robotDir = WEST;
move(rt);

rt->owner->robotDir = NORTH;
move(rt);

rt->owner->robotDir = NORTH;
move(rt);

rt->owner->robotDir = EAST;
move(rt);
break;
```

There was probably a much easier method to this but for my version it will move West, South, South, East to make a rotation from North to South and vice-versa for the other function. They're only called with the push function. Initially I had them in a rotation function but it felt cluttered to have a new struct data and function just for two movements. One of the biggest issues aside from that rotation was actually the movement itself, sometimes multiple movements happened at once and the robot just jumps. To relieve that I added usleep() calls in between movements. I now think this would have helped the jitteriness of the previous Travelers program.

VII. PUSH

Push is very similar to move and might have been able to call move for some of it's movements but I chose to have both the pushing motion and robot follow up here together. Otherwise like path it is a lot of if statements for various positional issues and it even has the rotation lite section where instead of a full rotation it's only a half one. Again I left it in this function because I felt like having half-rotations in move would have been very cluttered but NS/SN were big enough to justify it. It does lock the box's forward space, then unlocks the robots space to move into that freed space. This can actually cause issues where the block is pushed and for just a moment another robot moves their block into that space and they're now deadlocked.

VIII. END

The simplest function in the entire program it merely releases the locks held by the robot and block, lowers the displayed numLiveThreads after locking the state_mutex, then it terminates the robot.

IX. UNSOLVED PROBLEMS AND QUIRKS

For this program I have run into a few issues that cause problems, hopefully there aren't many more than this:

- The initialized boxes aren't locked, so the robots walk over them. (I believe this could be fixed with a few more conditional statements but I just ran out of time and focused on everything else.)
- It sometimes crashes if the input Col is bigger than Row. (I believe this has something to do with the graph being drawn but this one has completely escaped me.)
- As said earlier the deadlock issues, which luckily we didn't have to troubleshoot completely, but again sometimes boxes and robots don't move fast enough together and their spot get stolen.
- Occasionally the full rotation displays twice but seems to correct itself. (I would guess that it's just meeting that if statement again before the rotation flag is turned off.)

X. CONCLUSION

This was another fun exercise in programming. I wish we had more time to work on it before the end of the semester so I could have gone to more office hours and got help on future issues after there were no more.