Erikson Sodergren

Concepts of Parallel and Distributed Systems – Project 3+4

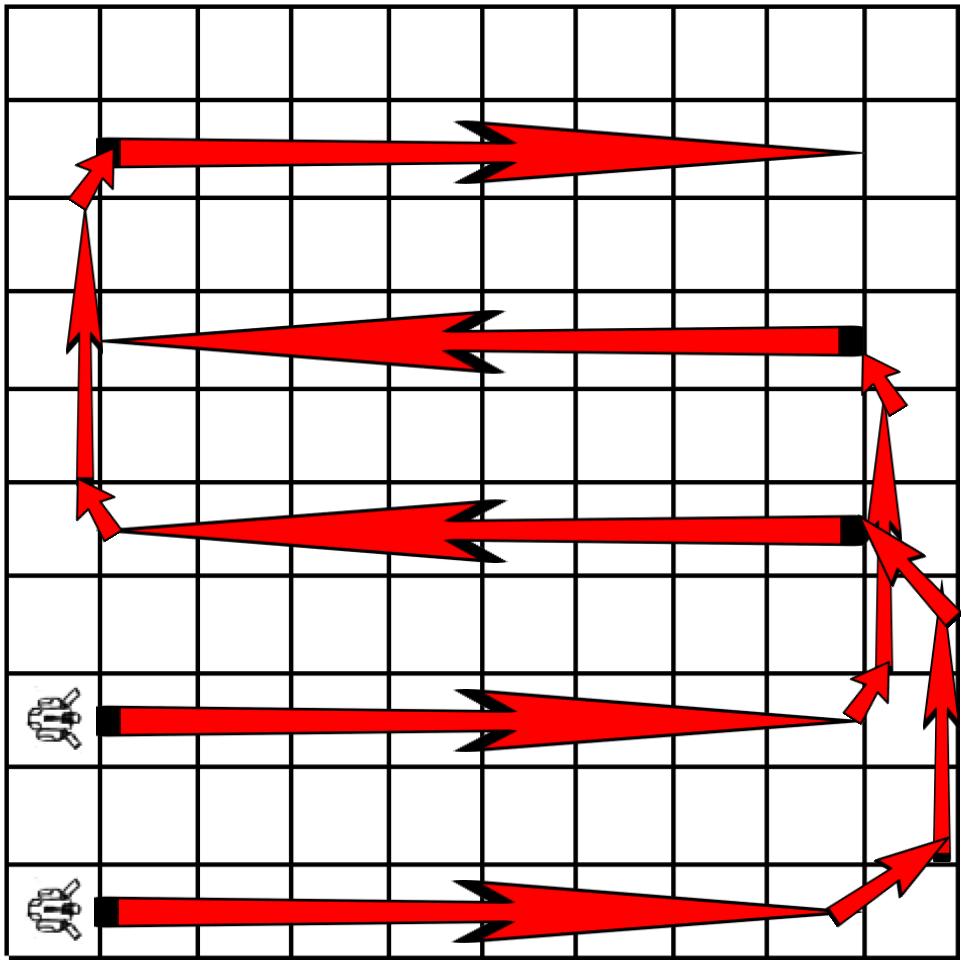
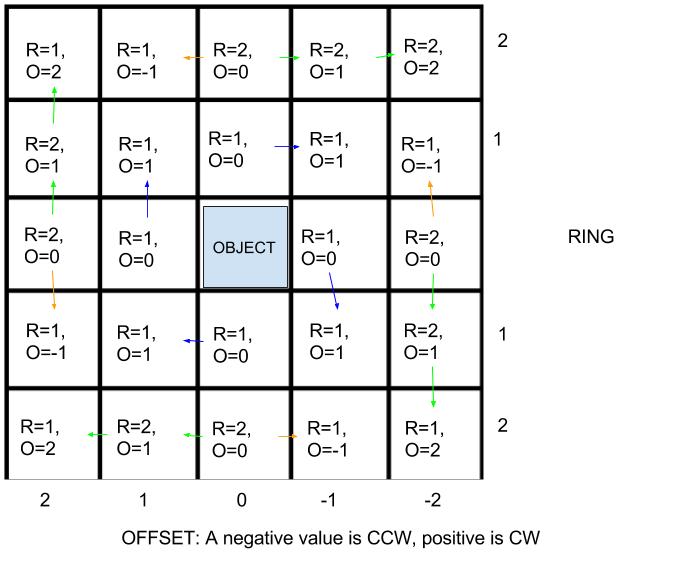
For this assignment, I used pthreads in C to implement the robot simulation task with shared memory parallelization. The program initializes a thread for each robot at the beginning of the program, and keeps them synchronized using pthread barriers to avoid certain steps happening at a time where required information is not available. Due to time constraints, not all parts of the program are completely perfect, but it works for most cases.

The program begins reading input and then by placing the static object and robots on random tiles in the board, then creates threads to represent the robots and starts them. The threads first begin a search algorithm(unless the mode is set to “A”), in which they form a dotted line at the bottom left and comb over the entire board. Image 1 is an example with 2 robots on a 10x10 board. Once one of the robots finds the object, all robots quickly exit the search function and robot zero, the first one made, is elected to determine which locations are considered to be surrounding the object. The algorithm it runs is visualized in image 2, but the basic idea is to pick locations in groups of 4 based on the “ring”, or the maximum difference in the object and any coordinate of a surround location, and the “offset”, how far clockwise the locations are compared to something exactly “ring” spaces away in a straight line.

After the locations are determined, each robot individually calculates how many moves it would take them to get to each location, and robot zero decides who goes to each location by finding the smallest sum of distances possible. This is determined by starting with each robot going to the location with the same index as their ID, and looping over the list comparing the current sum of each pair of robots distance from their current target, and each other’s targets. Once these decisions are complete, each robot moves towards its target location until the object is surrounded by robots. If a robot is unable to move at any time due to an obstacle, it will go around the obstacle by moving left, forward, forward, right in relation to its original direction. The movement function is controlled by a mutex lock to ensure this is done correctly. The program will output a large amount of information throughout the runtime, including the starting locations of each robot and their distances from each target location, and more, the final output being the number of times each robot moved including the search and surround functions.

The program completes the vast majority of times(no failures found in over 50 test runs each, over 2 input sets), and I believe it works decently quick, but sometimes the robots may take longer than needed due to collision avoidance. In addition, due to time constraints, the move function does not check to ensure robots do not walk off the board, so a robot might walk into (-1, 4) to go around another robot. The average number of moves for the last robot to finish was 15.4 with a 10x10 board and 4 robots, and 51.8 with a 20x20 board using 8 robots.

In conclusion, I believe this program runs decently well, but could use improvement if I had time to continue, including potentially adding the ability to visualize the state of the board and print that to the terminal, improving movement and collision detection, and the search algorithm.

 image 1image 2

Block/State diagram

