RTOS - Elevator

Project By Sean Connolly and Kehnin Dyer

This project was intended to simulate an express elevator, as describe by the lab specifications. We were unable to accomplish this goal in the intended way in the time allotted.

Even though I have stated we were not able to finish we still had great success in completing many of the various peripheral tasks of the device.   
Our design outline featured:

* UART control
* Message Serializer
* CLI interpreter
* Button scanner
* Frequency control
* Motor control
* Door control
* Main control

Message passing between the various parts was to be done primarily via the holding and releasing of semaphores with specific meaning, for instance when the button scanner detected and emergency stop being pressed it would release the semaphore for emergency and the frequency control would then be able to acquire it, once handled the frequency control would release it letting it return to its natural state.   
In the case of updating the Max speed and Acceleration of the elevator these messages are passed by a queue from the CLI interpreter to Frequency control, when there is a message the data is the new value.   
in most instances the semaphores were not directly sent from peripheral device to peripheral device, but instead sent and received by the Main control. The reason is because one message sent by a button would normally need to trigger multiple messages in sequence, for instance, if a call button was pressed, the main control would need to set the destination, then wait for arrival, then tell the door to open.

In short we completed every part of the project in isolation except for the Main Control. Integration of all components was complete except CLI interpreter and Main Control.

essentially what this means is the functionality for the door to open and close was handled correctly (according to our design) but there was nothing to tell the door to open. The elevator could handle speeding up, slowing down, manage changing destinations on the fly, emergency stop, changing of max

speed, or the acceleration (even mid transit) would all work and were tested to work, but without the main control to tell it that it has a new destination, it would be fated to sit and wait.

The buttons all were registered to send their various messages. The UART handled sending data. The Message serializer converted the elevator data and status into human readable text and gave it to the UART device to send. The motor controller read the speed and blinked on and off at 1Hz/10ft/s.

Everything was in place for the project to be finished.

In short the elevator API was complete, just not the connecting wires to make it 'go'