Assignment 1

Due: 21 October 2016 (through Blackboard)

This assignment consists of designing and building a set of programs to simulate a building entry controller system, and provide a testing harness. The system is similar to that discussed during the lectures – a two door system with card scanners on each door, a scale that weighs the person after they have entered the space between the two doors, and a human guard that uses a switch to open each of the doors.

Details:

1. People enter the entry control space from the left to enter the building, and enter the control space from the right to exit the building.
2. Each person requesting to enter or exit the building scans their unique personal id card containing an integer **person_id** number. The scanner sends a message to the controller with the input information (which door opening is being requested, and which person_id is being used.; (e.g., "left 12345")
3. Only 1 person at a time should be able to be inside the lock.
4. Assume that the door is self-closing, and that an event (see below under Step 4.a.) will be sent to the controller when the status of the door changes.

Steps:

1. (20 points) Design a state machine for the system. Provide the following:
   a. A list of inputs, including persistent saved data (e.g., the weight of the person on entry)
   b. A list of outputs, including persistent saved data.
   c. A list of conditions / events that cause transitions between states.
   d. A list of states
   e. A diagram showing the state machine.
2. (20 points) Write a program *ass1_display* that displays the status of the system – which door is open / closed, if there is a user waiting to enter from the left or right, etc. The program should run in the background and print out status information to the console each time a status update is sent to it using a message from the *ass1_controller* program. The *ass1_display* program can print out its process id when it first starts up (as in lab4).
3. (40 points) Write a program *ass1_controller* that runs in the background, operates the state machine for the controller, and directly maintains the persistent data and status for the controller. It should have separate functions for the state handler for each state. Each state handler should perform the actions required for that state, send a message to the *ass1_display* program to update the display (as required), and then check the exit conditions for that state. When an exit condition is met, the state handler should return the function pointer for the next state handler. This does not have to be done using a lookup table, but could be implemented that way. The *ass1_controller* program should print out its process id when it first starts up.

4. (20 points) Write a program *ass1_inputs* that prompts the user for inputs to the controller. This program is simulating all of the input events from the devices; e.g., card readers, door latches, scale.

   Prompt for the input:

   a. The first prompt should be:
      Enter the event type (ls= left scan, rs= right scan, ws= weight scale, lo =left open, ro=right open, lc = left closed, rc = right closed , gru = guard right unlock, grl = guard right lock, gll=guard left lock, glu = guard left unlock)
   b. If the event is the lo, ro, lc, rc, glu, gll, gru, or grl, no further prompt is required.
   c. If the event is ls or rs, prompt for the person_id.
      Enter the person_id
   d. If the event is ws, prompt for the weight.
      Enter the weight

   Once the input is finished, send a message to the *ass1_controller* program to provide the input "event", and loop back to prompt again.

   Note: You can store input scenarios in text files and then run them using the command:
   # ./ass1_inputs < myinputsfile &

   With an input file "myinputsfile" containing:

   ls
   12345
   glu
   lo
   w
   70
   lc
   gll
   gru
   ro
   rc
   grl

Example Startup:

#./ass1_display &

The display is running as process_id 77234.

[status update : initial startup]

#./ass1_controller 77234 &

The controller is running as process_id 77235.

#./ass1_inputs 77235

Enter the device (ls= left scan, rs= right scan, ws= weight scale, lo =left open, ro=right open, lc = left closed, rc = right closed , gru = guard right unlock, grl = guard right lock, gll=guard left lock, glu = guard left unlock)

…..

Steps:

1. Get Lab 4 working! (Use it as a model).
2. Design the state machine.
3. Create new projects named "Ass1_xxxx" inside the GIT repository for your laboratory work.
4. Create an include file "mystruct.h" to define the typedefs for structs to be used for sending requests and receiving responses, and for storing the status information.
5. Create the *ass1_controller, ass1_display,* and *ass1_inputs* programs.
6. Test your programs.
7. Save and commit the changes to your repository.
8. Go to the working directory for your GIT repository (the one named with your Algonquin username) and zip it.
    a. If you open the "Window>Open Perspective>Other…>Git Repository Exploring" You can see where it is physically stored.
9. Demo your programs in the lab.
10. Submit your laboratory for grading by submitting it online using Blackboard under the "Handing in Work" link, with the zip file as an attachment.

Grading Scheme

Marked out of 100. (as shown above).