README for Lift Programming Assignment in CptS 580 Concurrency

1. Deadlock-preventive using token for Floor agents

A token process is used to distribute a unique token to all requesting Floor agents in which only one can send request (up or down) signal to all Lift agents.

Whenever a floor button is pressed, its Floor agent will ask for the token from Token process. If issued to the token, it'll send all request messages to all Lift agents. After sending work, it sends back a release message to Token process and the latter can now issue the token to other Floor agents.

So this token approach prevents racing condition of the communication among Floor agents and Lift agents. The locked Lift agent (by previous request message from some Floor agent) absolutely responds with propose messages to the Floor agent with its own proposed wait time. So the sequential sending order among all Floor agents guarantees a short period to separate request message from various Floor agents. In addition, the Floor agent with token in hand quickly releases the token right after sending request to all Lift agents instead of waiting for propose messages from Lift agents. In a word, a quick, exclusive access to token avoids race condition (deadlock) for locked Lift agents who only respond to accept or reject messages from a specific Floor agent.

In the implementation, the Token process is spawned and waits for any ask message. If issued, the token will only be recycled by release message from the Lift agent with the token in hand.

2. Button light

The Floor agent list must also be sent to each Lift agent for the purpose of Button light as each Lift agent list must be hold by each Floor agent for the purpose of requesting estimated wait time.

So as soon as a floor button is pressed (the manual operation), the corresponding Floor agent lets it on by outputting a light-on printout sentence. Then when the Cabin signal of stop comes to each Lift, the button-off signal will be sent back to corresponding floor if the lift stops at the given floor.

3. File lists (new or modified)

(1) simulation.erl: the primary program;
(2) lift.erl: control Lift agent;
(3) floor.erl: control Floor agent;
(4) token.erl: control Token process;
(5) insert.erl: insert logic;
(6) waitTime.erl: wait time estimate logic;
(7) stopAt.erl: stop at logic for inner button signal in lifts;
(8) testcase.erl: test case for insert and waitTime functions. The test case is created and shared by Qing Chang, Min Shao and me together.

4. Drawbacks

(1) stopAt logic

Not fully tested. Also, I construct the stopAt logic according to my own understanding. Refer to my comment in that file please.

(2) no time to implement lock against propose message in Lift agent

The lock after proposing wait time within Lift agent is not implemented. The idea is simple: set a propose flag within the state (now three-element tuple state for Lift agent); whenever a request message comes from a Floor agent, the Lift agent will only accept either accept or reject messages from the same Floor agent by flipping the flag (from true to false). As soon as the accept or reject message comes from the Floor agent, flip it back (from false to true).

(3) waitTime and insert test cases

I don’t think I consider all the paths. Only part of test cases is thought seriously since it’s too complex to pinpoint every edge of the test set.

(4) concurrent running test

Beyond my capability to understand some complex running cases. I don’t think some simple test cases can be used to detect any concurrent weakness in my program.