CSCI 201L Final - Programming
Spring 2015
13% of course grade

Using the existing factory code posted on the course web site, you are going to modify the functionality as follows.

Factory Server

Currently, the recipes on which the workers operate are read from the file system. We are still going to read the recipes from the file system, but this will now be performed in a new server program you create. The server program should listen on port **`6789`**, and it should allow more than one factory to connect.

To simplify the file reading, create a directory in your project directory called **`factory`**, which will be the directory that contains the .factory, .rcp, and .dep (explained in a later section) files. This can be hard-coded into your server as the directory that contains the files to be read. The number and names of the .rcp files may be different, but the directory can be hard-coded.

Once a client connects to the server, send back the contents of just the .factory file.

Factory Client (i.e. existing factory)

The existing factory code will now need to be modified so that it is reading the .factory and .rcp files from the server. Remove the "Open File…" menu item and create an "Open Factory…" menu item. This menu item should display a dialog box with two fields – server name and server port. Make the default values of these fields be **`localhost`** and **`6789`**, respectively.

Once a connection is established, the server will send back the contents of the .factory file. Once this file comes back, individual workers can start requesting recipes similarly to how they operate now. The recipes will be requested from the server though. The worker will request a recipe, and if the server has a recipe that needs to be created, it will send it back to the worker when requested (read the next section before implementing this though). The worker should then add that recipe to the task board and start creating it. If there are no recipes left to create, the worker should be notified of that.

Once all of the workers have finished the recipes on which they are working and there are no other recipes to be fulfilled, the program will display the same message it does now, saying that it has completed.

Dependencies

The above code is testing your knowledge of networking and multi-threading, but this section will test your knowledge of synchronization. There is a new file added into the directory called **`recipes.dep`**. This file will contain any dependent recipes that must be created before a recipe can be created. The format of the file is similar to the format of the .rcp files. Here is a sample recipes.dep file:

```
[Gadget] [Cog x2] [Widget x1]
[Widget] [Cog x1]
```

This file specifies that a Gadget needs two Cogs and one Widget before it can be created. A Widget needs one Cog before it can be created. We are going to try to make this easy on you and say that recipes can be reused. For example, once one Cog is created, you will be able to create a Widget. Once one more Cog is created, then you will be able to create a Gadget. Note that you don't need two additional Cogs created before creating a Gadget.

Each line in the file could contain multiple recipes that must be created before the other recipe is created. You can assume the file is formatted correctly.

This situation causes a synchronization issue because one recipe may not be able to be created until another one (or more than one) is created. Instead of ordering the recipes on the server, this seems like something that can be solved with conditions. From the above file, you notice that a Gadget needs to wait (or await, hint hint) on two Cogs and one Widget. Once those recipes have been completed, it would be rather convenient if the Gadget knew about it (or was signaled, hint hint).

NOTE: This is not changing the number of recipes that are created. You are still creating the number of recipes as stated in the .rcp files. You just now will have an ordering to creating them. The client should not realize this dependency either. The client is just requesting a recipe, and the server sends back the one that is able to be created.

Since the server can only send recipes out based on other recipes that have been completed, you will need the workers to send a message back to the server when a recipe has been completed. The server should print out to the command line which recipes have been sent out to workers and which ones have been completed. ***Based on the above example, we should never see that a Gadget was sent out to be created before two Cogs and one Widget have been completed.***

Grading Criteria

1.0% Server program listening to port 6789
1.0% Server program reading files from factory directory
1.0% Server program allowing multiple clients to connect simultaneously
0.5% Server program sending .factory file back after connection is established
0.5% Server program sending .rcp files back when workers request them
4.0% Server program having synchronization so that recipes are not sent back if the dependent recipes have not been created
0.5% Server program outputting when recipes have been sent and when they have been completed
0.5% Client program has Open Factory menu item
0.5% Client program has GUI for specifying hostname and port, defaulting to localhost and 6789
0.5% Client program creating factory after connecting to serer
1.0% Client program has workers request recipes from server and add them to the task board
1.0% Client program has workers send a message back to the server when the recipe has been completed
1.0% Client eventually terminates when all workers have completed all recipes on server