CSC9006 Real-Time Systems (Spring 2021) Homework 5

** **Submission deadline: 9PM, May 18th, 2021.** **

1. Introduction

In this homework, we will have some hands-on exercise for the topics covered in class:

1. The use of IDL;
2. The singleton design pattern.

Be sure you get the *TAO Developer's Guide*, available from Object Computing, Inc. You may request a free copy here. The latest version is 2.2a, patch 7. Optionally, for a more detailed exposition of IDL and CORBA, check out the book *Advanced CORBA Programming with C++*, by Michi Henning and Steve Vinoski.

2. Using IDL

We will take the `time` example located at $TAO_ROOT/examples/Simple/, which implements a simple server and a client. The client may get the current time of the server and may request to shutdown the server.

Now, make a copy of the `time` folder and name it as `csc9006hw5`:

```
$ cp -r time csc9006hw5
```

In this homework assignment, we will modify files in `csc9006hw5`. Now, switch to that folder and let's clean up the folder by removing some unrelated files:

```
$ rm *vs20* Time_Client.java
```

Read through the `README` file in the folder. The file describes both the purpose and the instruction regarding this example. We will not use the `run_test.pl` script; instead, we do the following to launch the server and the client.

To launch the server, type:

```
$ ./server -o server_ior
```

To run a client program to get the current time of the server, type:

```
$ ./client -f server_ior
```

To have a client requesting to shutdown the server, type:

```
$ ./client -f server_ior -x
```

Read the `README` file for details.

Now, take a look at file `Time.idl`. This is the file where use IDL to define the Time interface. The TAO's IDL compiler will take this file and generate the corresponding stub and skeleton. The makefile in this folder will handle all the building process. To see this, type the following in sequence:

`$ touch Time.idl`

`$ make`

Then among the output log messages you should see the following one at the beginning:

> ../../../../bin/tao_idl -Wb,pre_include=ace/pre.h -Wb,post_include=ace/post.h -I ../../.. -Sa -St -DGEN_IDL_MAKE_DEPS Time.idl

This shows the invocation of the IDL compiler. This TAO IDL Compiler User's Guide describes the options and features of the IDL compiler.

The IDL compiler will generate stub `TimeC` and skeleton `TimeS`. `Time_i` implements the Time IDL interface, and `Time_Client_i` implements the client calls. Spend some time to review the basic concept of ORB, and do some code tracing to see how the source code in each file relates to one another. Notice that this example uses a template for client and server, defined in `../Simple_util.h` and implemented in `../Simple_util.cpp`.

It may take some nontrivial amount of time to really figure out how this example is implemented. This is a necessary step. Take heart, and post your questions on the Moodle forum if any.

Assignment: Take a look at file `Time_Client_i.cpp`, line 55. Explain under what condition will `client_.do_shutdown()` evaluate to 1, and what an application user would do to make that happen. Write your answer in a plain-text file and name it `code_study.txt`.

Now, we will add a new operation to our server, so that a client may query the server for the total number of `current_time()` requests it received so far. We will name the operation `count_requests()`. The updated Time.idl should look like the following:

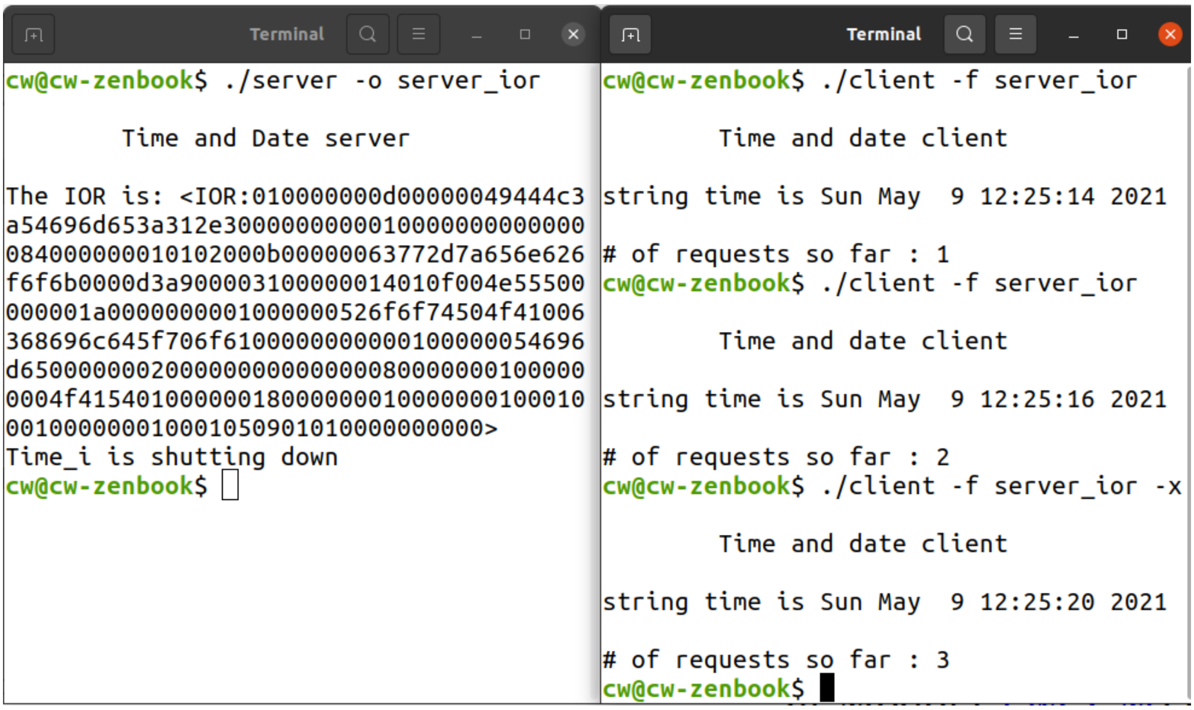
```
/* -*- C++ -*- */
interface Time
{
  // = TITLE
  //    Defines an interface that encapsulates operations that
  //    return the current time of day and shuts down the server.

  long current_time ();
  // Returns the current time of day on another machine.

  long count_requests ();
  // Returns the total number of current_time() requests.

  oneway void shutdown ();
  // Shutdown the application.
};
```

Assignment: Update related files in folder `csc9006hw5`, to complete the implementation of this new operation. The following execution is an example of the use of this operation:



```
cw@cw-zenbook$ ./server -o server_ior

        Time and Date server

The IOR is: <IOR:010000000d00000049444c3
a54696d653a312e30000000000100000000000000
084000000010102000b00000063772d7a656e626
f6f6b0000d3a900003100000014010f004e55500
000001a0000000001000000526f6f74504f41006
368696c645f706f610000000000010000005469 6
d650000000200000000000000080000000100000
0004f41540100000018000000010000000100010
001000000010001050901010000000000>
Time_i is shutting down
cw@cw-zenbook$ 
```

```
cw@cw-zenbook$ ./client -f server_ior

        Time and date client

string time is Sun May  9 12:25:14 2021

# of requests so far : 1
cw@cw-zenbook$ ./client -f server_ior

        Time and date client

string time is Sun May  9 12:25:16 2021

# of requests so far : 2
cw@cw-zenbook$ ./client -f server_ior -x

        Time and date client

string time is Sun May  9 12:25:20 2021

# of requests so far : 3
cw@cw-zenbook$ 
```

3. Using the singleton pattern

Here is an example C++ singleton class:

```
class ourSingleton {
public:
    ~ourSingleton(){ /*...*/ }
    static ourSingleton *getInstance(){
        if(uniqueInstance == NULL){
            uniqueInstance = new ourSingleton();
        }
        return uniqueInstance;
    }
    /*...*/
private:
    static ourSingleton *uniqueInstance;
    ourSingleton(){ /*...*/ }
    /*...*/
};
```

Assignment: Update the `Time_Client_i` class to make it a singleton class.

Assignment: Now update the client implementation to use two periodic threads to access the server via the `Time_Client_i` singleton. You may reuse some implementation in the previous homework assignments. Describe your design in a plain-text file and name it `design_explained.txt`.

Assignment: (review the 4/22 lecture) The above singleton class example has a *heisenbug*: at run-time, this singleton class could have two objects (?!). Create a plain-text file named `heisenbug_singleton.txt`. In the file, give a two-thread execution scenario to illustrate why a program at run-time could have two objects of the singleton class.

4. Moodle submission

Package everything in your `csc9006hw5` folder into one single zip file, and name the zip file using your student ID. Your Moodle submission will be graded according to the following criteria:

- (5 points) Your `code_study.txt`
- (35 points) The query functionality using `count_requests()`;
- (20 points) The related updates to use `Time_Client_i` as a singleton;
- (25 points) The implementation of using two periodic threads to access the server via the unique `Time_Client_i` object, and the `design_explained.txt` file;
- (15 points) Your `heisenbug_singleton.txt` file.

Post your questions on the Moodle forum.