CSC9006 Real-Time Systems (Spring 2021) Homework 4

CSC9006 Real-Time Systems (Spring 2021) Homework 4
1. Introduction
2. Building ACE and TAO
3. Running a built-in example
4. A first look at the event channel configuration
5. Things to submit to Moodle

** **Submission deadline: 9PM, May 1st, 2021.** **

1. Introduction

In this homework assignment, we will

1. setup the necessary programming environment for the rest of this semester;
2. have some initial study for the implementation of the TAO real-time event service.

The design of the real-time event service (a.k.a. real-time event channel) is described in the paper we've studied for Critique 1 and Critique 2:

> T. Harrison, D. Levine, and D.C. Schmidt, *The Design and Performance of a Real-time CORBA Event Service*, ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications, October 1997.

Let's begin :)

2. Building ACE and TAO

Here is the official website for TAO: http://www.dre.vanderbilt.edu/~schmidt/TAO.html

We will use **ACE+TAO-7.0.0**. Building ACE and TAO may take about 7 GB out of your disk space and the whole building process may take more than two hours to complete.

Download the zipped file *ACE+TAO-7.0.0.tar.gz* from this page. Type the following at the directory where you put the zipped file:

```
$ tar zxvf ACE+TAO-7.0.0.tar.gz
```

Then you should find a folder named *ACE_wrappers*. Follow the instruction on this page, under the subsection **On UNIX platforms**. Whenever it asks you to set an environment variable, use `export` rather than `setenv`, because on our Ubuntu Linux the default shell is BASH. Now, follow steps **2**, **3** (follow its example), **4** (follow its example), and **6** (skip step 5) on this page.

Then, go back to the previous page and follow the first bullet in step 2 under the subsection **On UNIX platforms**. The compilation process for ACE may take about five minutes to complete. After that, follow the rest of the instruction on the same page. For this course, we choose to compile the **complete** TAO distribution.

After you typed 'make' at $TAO_ROOT, take some rest, as it may take more than two hours to complete the whole compilation process. After the compilation, now you are ready to use ACE+TAO :)

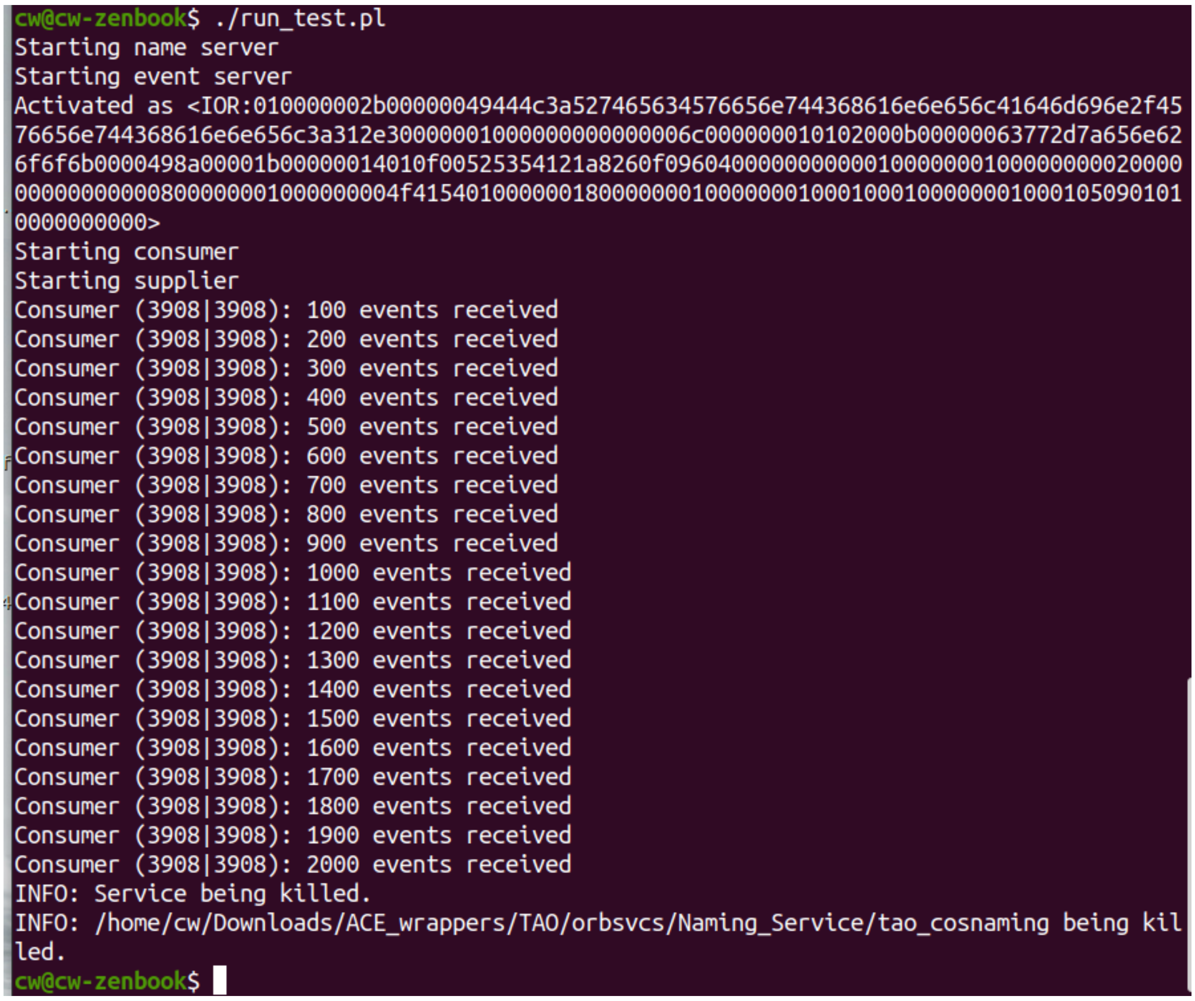
Optionally, you may do the following steps to keep all our setting of the environment variables; otherwise, after each reboot, you will need to export them again. Here we will modify the ~/.bashrc file to put our setting there. The system will execute the script in this file every time we log in. Just append the following three new lines at the end of ~/.bashrc (remember to change the path assigned to ACE_ROOT):

> export ACE_ROOT=/home/cw/Downloads/ACE_wrappers
> export TAO_ROOT=$ACE_ROOT/TAO
> export LD_LIBRARY_PATH=$ACE_ROOT/lib:$LD_LIBRARY_PATH

3. Running a built-in example

Now we will run a simple example to get some sense of how this real-time event service works. We will cover other details in the future when needed.

The example we will run here is located at `$TAO_ROOT/orbsvcs/examples/RtEC/Simple`. Take a look at the README file in the same directory. If you've accomplished all the setup, you should be able to run the script **`run_test.pl`** and should see output similar to the following:



```
cw@cw-zenbook$ ./run_test.pl
Starting name server
Starting event server
Activated as <IOR:010000002b00000049444c3a527465634576656e744368616e6e656c41646d696e2f45
76656e744368616e6e656c3a312e30000000100000000000006c00000001010200b00000063772d7a656e62
6f6f6b0000498a00001b00000014010f00525354121a8260f0960400000000000010000000100000000020000
0000000000080000001000000004f41540100000018000000010000000100010001000000001000105090101
0000000000>
Starting consumer
Starting supplier
Consumer (3908|3908): 100 events received
Consumer (3908|3908): 200 events received
Consumer (3908|3908): 300 events received
Consumer (3908|3908): 400 events received
Consumer (3908|3908): 500 events received
Consumer (3908|3908): 600 events received
Consumer (3908|3908): 700 events received
Consumer (3908|3908): 800 events received
Consumer (3908|3908): 900 events received
Consumer (3908|3908): 1000 events received
Consumer (3908|3908): 1100 events received
Consumer (3908|3908): 1200 events received
Consumer (3908|3908): 1300 events received
Consumer (3908|3908): 1400 events received
Consumer (3908|3908): 1500 events received
Consumer (3908|3908): 1600 events received
Consumer (3908|3908): 1700 events received
Consumer (3908|3908): 1800 events received
Consumer (3908|3908): 1900 events received
Consumer (3908|3908): 2000 events received
INFO: Service being killed.
INFO: /home/cw/Downloads/ACE_wrappers/TAO/orbsvcs/Naming_Service/tao_cosnaming being kil
led.
cw@cw-zenbook$
```

In the above example, we created an event supplier and had it push 2,000 events through an event channel to an event consumer. The event supplier, the event channel, and the event consumer are three independent processes. The event channel would register itself to a naming service, and then both event supplier and event consumer may get to know the reference of the event channel by querying the naming service. Then, after both the consumer and the supplier connected to the event channel, the supplier can start to send events to the consumer.

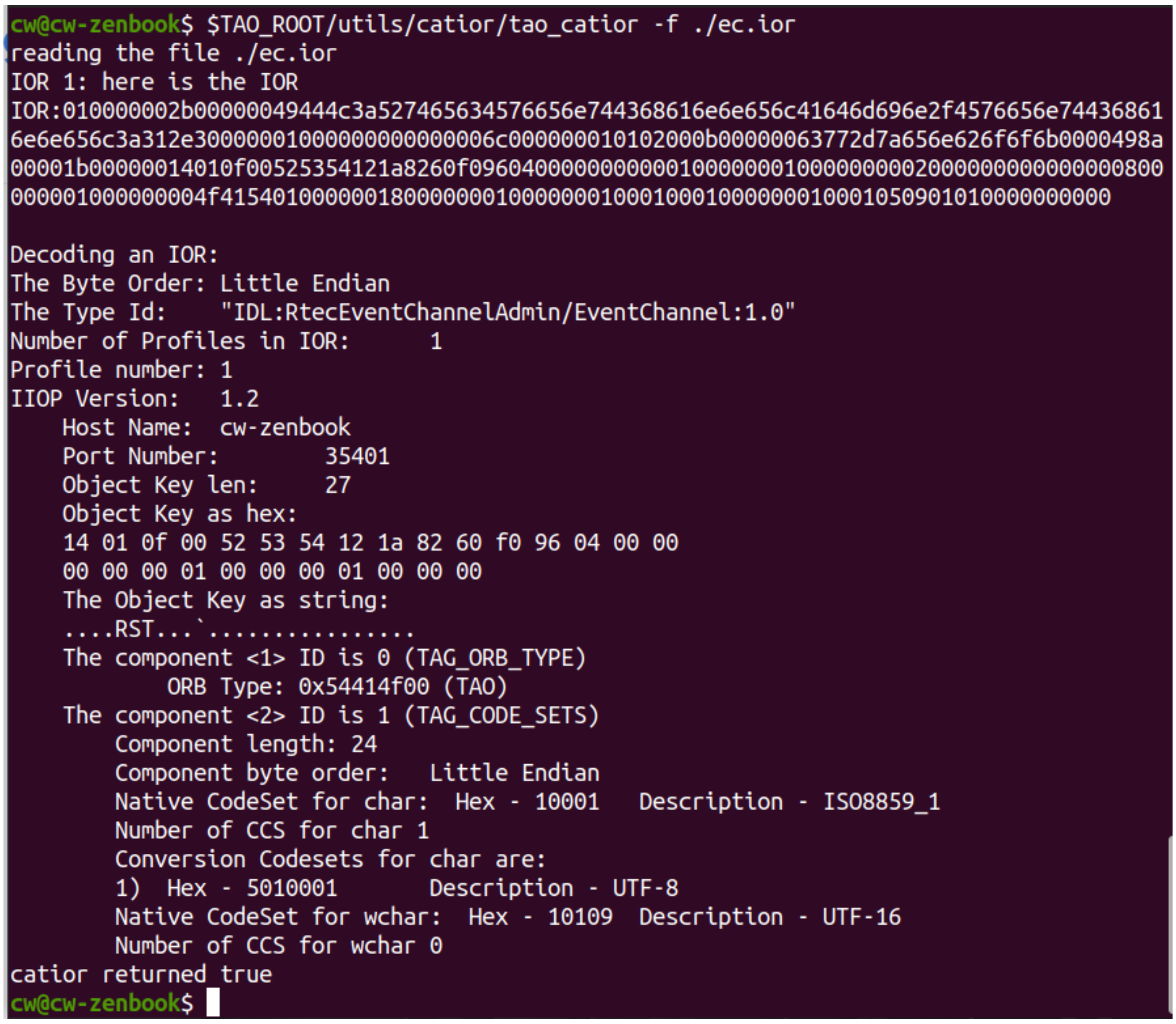
Assignment: take a screen shot for your execution of the script **`run_test.pl`** and name it `rtec_simple.png`.

Rather than using a naming service, we can make different objects directly operate with each other using *Interoperable Object References (IORs)*. Conceptually, an IOR is a string that encode necessary information regarding an object, and other objects may learn those information from the IOR string.

In the above example, the IOR string for our event channel is stored in file `ec.ior`:

> IOR:010000002b00000049444c3a527465634576656e744368616e6e656c41646d696e2f4576
> 656e744368616e6e656c3a312e3000000001000000000000006c000000010102000b000000637
> 72d7a656e626f6f6b0000498a00001b00000014010f00525354121a8260f0960400000000000
> 1000000010000000002000000000000000800000001000000004f41540100000018000000001
> 0000000100010001000000001000105090101000000000

There is a tool called **`tao_catior`** to decode an IOR string. It is located at `$TAO_ROOT/utils/catior/`:



```
cw@cw-zenbook$ $TAO_ROOT/utils/catior/tao_catior -f ./ec.ior
reading the file ./ec.ior
IOR 1: here is the IOR
IOR:010000002b00000049444c3a527465634576656e744368616e6e656c41646d696e2f4576656e74436861
6e6e656c3a312e3000000001000000000000006c000000010102000b00000063772d7a656e626f6f6b0000498a
00001b00000014010f00525354121a8260f096040000000000001000000010000000002000000000000000800
000001000000004f415401000000180000000010000000100010001000000001000105090101000000000

Decoding an IOR:
The Byte Order: Little Endian
The Type Id:    "IDL:RtecEventChannelAdmin/EventChannel:1.0"
Number of Profiles in IOR:      1
Profile number: 1
IIOP Version:   1.2
    Host Name:  cw-zenbook
    Port Number:        35401
    Object Key len:     27
    Object Key as hex:
    14 01 0f 00 52 53 54 12 1a 82 60 f0 96 04 00 00
    00 00 00 01 00 00 00 01 00 00 00
    The Object Key as string:
    ....RST...`................
    The component <1> ID is 0 (TAG_ORB_TYPE)
            ORB Type: 0x54414f00 (TAO)
    The component <2> ID is 1 (TAG_CODE_SETS)
        Component length: 24
        Component byte order:   Little Endian
        Native CodeSet for char:  Hex - 10001   Description - ISO8859_1
        Number of CCS for char 1
        Conversion Codesets for char are:
        1)  Hex - 5010001       Description - UTF-8
        Native CodeSet for wchar:  Hex - 10109  Description - UTF-16
        Number of CCS for wchar 0
catior returned true
cw@cw-zenbook$ 
```

Assignment: take a screen shot for your execution of **`tao_catior`** and name it `catior.png`.

We will learn more about the event channel and the IOR in the future.

4. A first look at the event channel configuration

Before you continue, spend some time and review some fundamentals in the C++ object-oriented programming; in particular, refresh your memory for these terms:

- class vs. object
- inheritance: the base class and the derived class
- class constructor and class destructor
- the 'this' pointer

If you've never learned object-oriented programming before, learn the above terms first. Learn their concepts and the corresponding C++ syntax. This should be sufficient for you to complete this homework assignment.

As described in the paper for Critique 1 and Critique 2, the event channel can be configured for different purposes. This is realized by the use of some *design patterns*. We shall discuss some design patterns later this semester. For now, it is sufficient to just know that TAO can configure the event channel at run-time using a configuration file. In our example `Simple` (which you've just tested in Section 3), the configuration file is `ec.conf` and it is located in the same directory. It contains the following information:

> static EC_Factory "-ECDispatching reactive -ECFiltering basic -ECSupplierFiltering per-supplier -ECProxyConsumerLock thread -ECProxySupplierLock thread -ECConsumerControl reactive -ECSupplierControl reactive -ECConsumerControlPeriod 50000 -ECSupplierControlPeriod 50000"

This essentially specifies which ones of possible options will be used to build components in the event channel (review Figure 6 in the paper for Critique 1 and Critique 2). Take a look at this page for a complete description of those options.

The major portion of the real-time event channel implementation is located in the directory `$TAO_ROOT/orbsvcs/orbsvcs/Event`. **`TAO_EC_Event_Channel_Base`** is the base class of class **`TAO_EC_Event_Channel`**. The former class is defined in `EC_Event_Channel_Base.h`, and the latter class is defined in `EC_Event_Channel.h`.

The event channel uses this specification in function `TAO_EC_Default_Factory::init` starting at line 68 in file `EC_Default_Factory.cpp`. Then function `TAO_EC_Event_Channel_Base::create_strategies` starting at line 81 in file `EC_Event_Channel_Base.cpp` calls functions to create instances of the classes that corresponds to each component as specified. The `create_strategies` function is invoked in the constructor of class **`TAO_EC_Event_Channel`** in file `EC_Event_Channel.cpp`. To see how `TAO_EC_Default_Factory::init` relates to this, notice that in the same constructor, before we called `create_strategies`, we have created an object of class **`TAO_EC_Default_Factory`**, which inherits class **`TAO_EC_Factory`** derived from class **`ACE_Service_Object`**. When we in the constructor of **`TAO_EC_Event_Channel`** invoke `ACE_NEW(...)` to create an object of **`TAO_EC_Default_Factory`**, the member function `init(...)` in class **`TAO_EC_Default_Factory`** will be called because that function is the entry point for an object of class **`ACE_Service_Object`**. This completes the picture.

The above description could be pretty confusing at the first sight. Before we move on, let's take a breath, and make sure we are on the same page. It is helpful to consult the Doxygen documentation for ACE and TAO; for example, read the documentation for **`TAO_EC_Default_Factory`**, in the category TAO_RTEvent.

Assignment: In the example located at `$TAO_ROOT/orbsvcs/examples/RtEC/Simple`, as specified in its `ec.conf`, for the `per-supplier` strategy for option `-ECSupplierFiltering`, find out **the name** of the exact supplier-filter-builder class that will be used, such that an object of the class will be created when we call `TAO_EC_Event_Channel_Base::create_strategies()` in file `EC_Event_Channel_Base.cpp`. To answer this question, take a look at files in directory `$TAO_ROOT/orbsvcs/orbsvcs/Event`. Write the name of that class in a plain text file and name it `first_trace.txt`.

You may use any tools you like. But I recommend using Vim+Cscope for code tracing. Here are some helpful pages for Vim:

- Official website
- A modern tutorial
- A classic tutorial

And here are some helpful pages for Cscope:

- Official website
- A step-by-step tutorial of using Vim+Cscope
- Using Cscope for a large project

Following the last bullet, if you will use Cscope, for the purpose of this homework assignment you should build a Cscope database by typing the following command at `/`:

```
$ find /home/cw/Downloads/ACE_wrappers/TAO/orbsvcs/orbsvcs/Event/ \( -name '*.cpp' -o -name '*.inl' -o -name '*.h' \) > /home/cw/cscope/cscope.files
```

Remember to modify the path according to your own setting. Here are some particular helpful hot-keys when using Cscope+Vim:

- `Ctrl + \ + g` : show a list of global definitions that bear the name identical to our target function;
- `Ctrl + \ + c` : show a list of functions that invoke our target function;
- After you've found an entry you want to jump to, there's no need to move all the way down to the end of the list. Just press `q` and it will directly ask you to enter the entry number;
- `Ctrl + \ + t` : jump back to where we were.

Remember to set environment variable `CSCOPE_DB` to the path of your `cscope.out`.

Finally, remember to consult this on-line source code documentation for ACE+TAO: https://www.dre.vanderbilt.edu/Doxygen/7.0.0/html/

5. Things to submit to Moodle

Package the following items into one single zip file and name the file using your student ID:

- (40 points) `rtec_simple.png`;
- (40 points) `catior.png`;
- (20 points) `first_trace.txt`.

Should you have any questions, don't hesitate to post them on the Moodle forum.

At this point, you might feel that you really need to *read a lot* in order to be able to work with a large project such as TAO. This is true, but don't feel discouraged. In empirical study, it is often the case that we seem to make very slow progress at the beginning. But after a while, we will see that we did gain a great deal of useful experiences and learn to use many useful tools, and that could be very rewarding. So, start this journey early. Take your time. Both passion and patience are very important.