CSC9006 Real-Time Systems (Spring 2021) Homework 8

** **Submission deadline: 9PM, June 27th, 2021.** **

Instruction

This is the final homework assignment for this semester. We will start from the result of Homework 7. You should reuse my solution for Homework 7 on the course repository (link), and make sure you read the README.txt file in that solution repository.

The total points in this assignment is 125 + 40 bonus points. The 40 bonus points account for the implementation of fault-tolerant strategies. Since, according to the school calendar, the summer break will begin at June 28th, I think we should set the submission deadline to be earlier than that. In this case, I think it would be very challenging for everyone to implement the fault-tolerance part in time (partly because I will not be able to be in time to supply a more detail instruction for that). We should nevertheless reward those who are willing to take the challenge. So, that part accounts for 40 points as a bonus.

In the following, like what we did in Homework 7, we will modify files in `$TAO_ROOT/orbsvcs/examples/RtEC/Simple` for our application and files in `$TAO_ROOT/orbsvcs/orbsvcs/Event` for our real-time fault-tolerant event service.

Assignment 1: System Behavior Observation and Analysis (45 points): In my solution for Assignment 5 in Homework 7, we've shown that the end-to-end latency might align with the suppliers' sending periods, since the proxy thread may need to wait until the next event arrival before it can take another look at the FIFO queue. Our implementation for this part is like what shows in Figure 1:

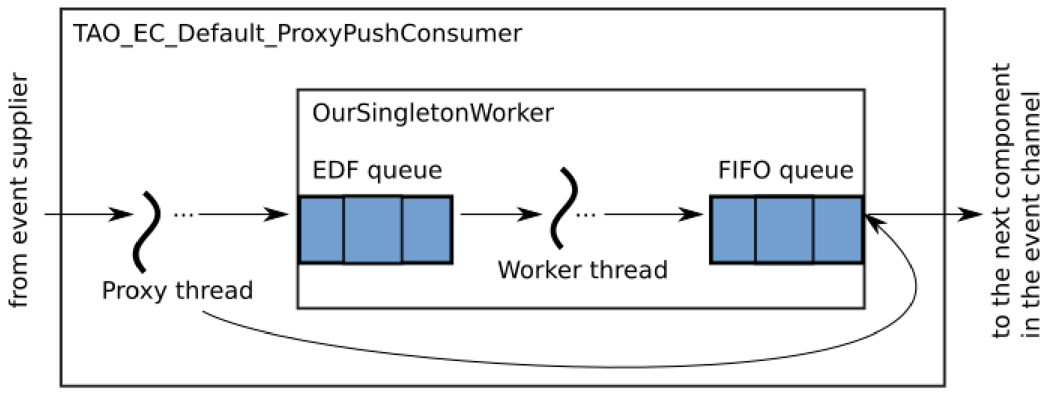

Figure 1

where the Proxy thread, upon taking an arriving event, will first push the event into the EDF queue and then will check out the FIFO queue to see if there is any event pending therein: if yes, the Proxy thread will take out the event and delivery it to the consumers; if no, the Proxy thread will return to wait for the next event arrival from suppliers. This implementation works like the system architecture in Figure 2 does:

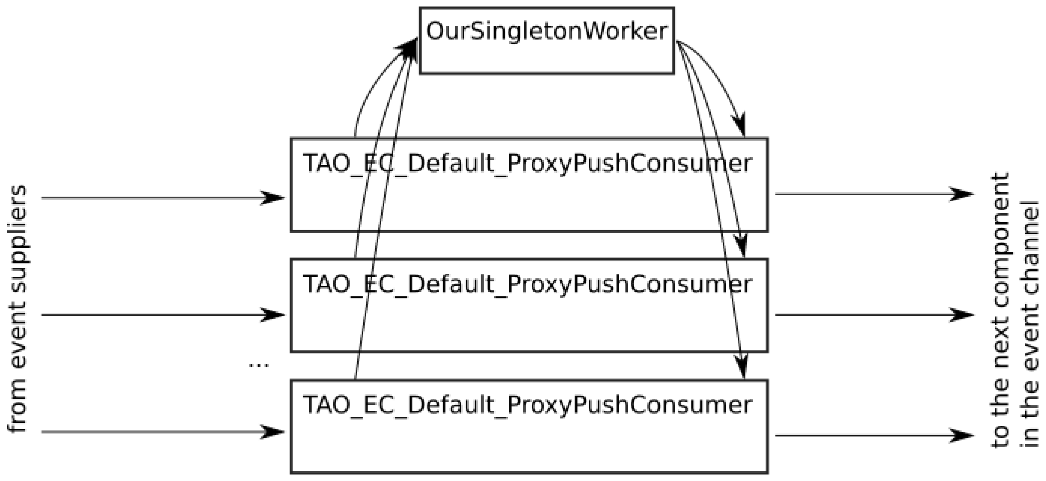
OurSingletonWorker
TAO_EC_Default_ProxyPushConsumer
TAO_EC_Default_ProxyPushConsumer
TAO_EC_Default_ProxyPushConsumer
from event suppliers
...
to the next component in the event channel

Figure 2

As illustrated in Figure 2, there is one `TAO_EC_Default_ProxyPushConsumer` object per supplier, and `OurSingletonWorker` is a singleton. In this way, we may schedule those events arrived from suppliers, i.e., we may change the ordering of event delivery. Now, answer the following three questions:

1. In the implementation shown in Figure 1, if we do not make `OurSingletonWorker` a singleton, does this use of EDF queue and FIFO queue guarantee *an EDF ordering of event delivery*, from multiple suppliers to multiple consumers? Explain your answer.
2. In Assignment 5 in Homework 7, we specified that between the popping of EDF queue and the pushing of the FIFO queue, the worker thread will perform a workload with WCET equal to 30 milliseconds. Now update my implementation with just the following change: *set the workload to WCET=1 milliseconds instead*. Run the same experiment, and you should observe that, unlike Homework 7, the resulting end-to-end latencies in this case do not align with the sending period anymore. Generate and attach your CDF plot for this case. Name the CDF plot `CDF_1_1ms.png`.
3. Following Question 2, explain why those five curves in your CDF plot turn out to be the way as you see them.

Assignment 2: Analysis for Alternative Solutions (30 points): Now we see that our solution for Assignment 5 in Homework 7 is sensitive to the change in workload. Here is a plausible way to fix such an annoying issue: Let the Worker thread send a signal to interrupt the Proxy thread once the Worker thread pushed an event to the FIFO queue. This way, the Proxy thread may respond and deliver the queued event sooner. But this approach also has its shortcoming.

4. Name one shortcoming of the above alternative solution and explain your answer. You may implement and experiment with it to get some ideas of what might be the shortcoming.

Now, here is yet another solution to our problem. Consider the following architecture (Figure 3):

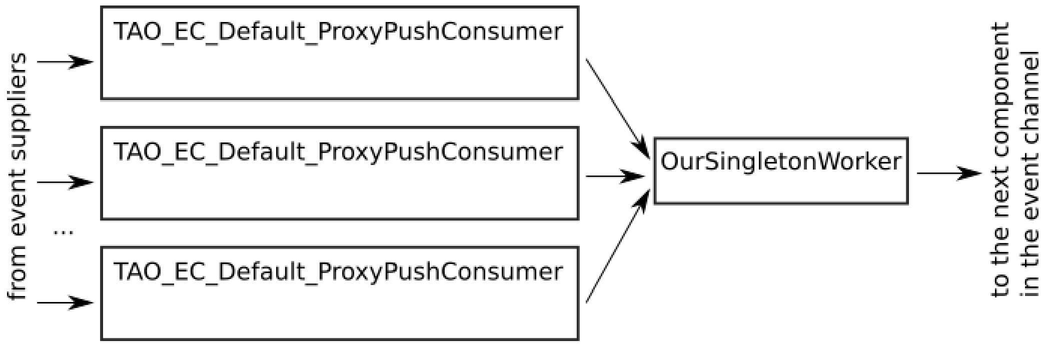
TAO_EC_Default_ProxyPushConsumer
TAO_EC_Default_ProxyPushConsumer
TAO_EC_Default_ProxyPushConsumer
OurSingletonWorker
from event suppliers
...
to the next component in the event channel

Figure 3

which can be implemented as shown in Figure 4:

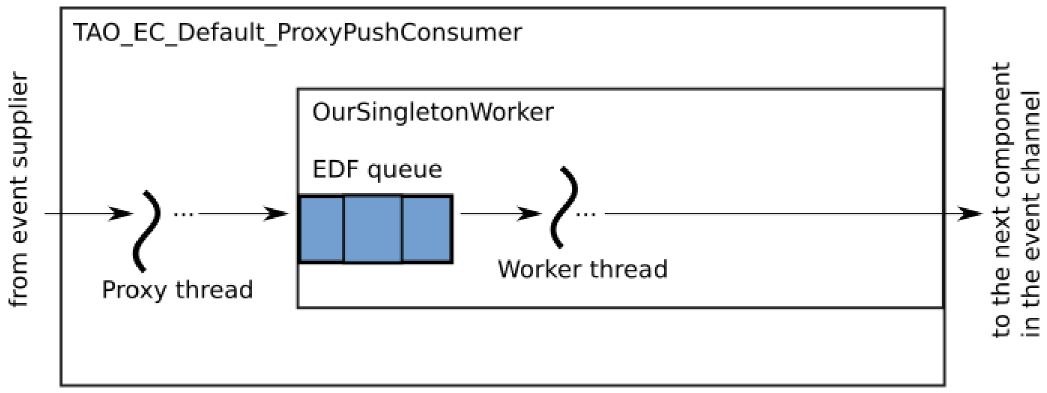
TAO_EC_Default_ProxyPushConsumer
OurSingletonWorker
EDF queue
from event supplier
Proxy thread
Worker thread
to the next component in the event channel

Figure 4

Notice how it differs from the implementation illustrated in Figure 1. We remove the FIFO queue and let the Worker thread deliver the events taken from the EDF queue. The Proxy thread in this case will bid farewell to the event once it pushed the event into the EDF queue.

5. Explain why the implementation like Figure 4 may be better than the previous two.

Write your answer for the above five questions. Write to a plain text file and name it `analysis.txt`.

Assignment 3: Implementation (20 points): Now, implement the solution illustrated in Figure 4. You should just change some part of the code in `EC_Default_ProxyConsumer.cpp` and `EC_Default_ProxyConsumer.h` from my solution for Homework 7.

Assignment 4: Empirical Performance Evaluation (30 points): Now let's run some experiments using our new implementation. Use the same experiment setting and run the following two cases:

1. workload with WCET = 30 millisecond; name your CDF plot `CDF_2_30ms.png`;
2. workload with WCET = 1 millisecond; name your CDF plot `CDF_2_1ms.png`;

Compare `CDF_1_1ms.png`, `CDF_2_30ms.png`, and `CDF_2_1ms.png` and explain what cause the difference. Write your explanation in a plain text file and name it `exp_explained.txt`.

Assignment 5: Implement The Primary-Back Model (40 bonus points): To make the fault tolerance work, here is a list of requirements:

- With respect to the primary event service, the backup event service should act as a consumer and should receive a copy for each event that the primary service will push to the regular consumers. The backup event service will keep those event copies and will only send them to the regular consumers once the primary crashed.
- The regular consumers should connect to both the primary event service and the backup event service, so that the consumers can receive events from the backup event service once the primary crashed.
- The regular suppliers should connect to both the primary event service and the backup event service, but should only push events to the primary at first; the suppliers will push events to the backup event service only after the primary has crashed (you can use either `kill` or `killall` to crash the primary).
- The backup event service should detect the crash of the primary; the regular suppliers should also detect the crash of the primary. Think about how you would do this. You may first try to crash the primary and see what the error type(s) the backup/supplier will output, and then use them as the condition(s) for the failover operation.
- You are very welcome to post your questions regarding how to implement to meet the above requirements. In your post, remember to both explain what you've tried and show what's

the error message, if any.

Moodle submission

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

- `CDF_1_1ms.png`, `CDF_2_30ms.png`, and `CDF_2_1ms.png`;
- `analysis.txt` and `exp_explained.txt`;
- Your updated `EC_Default_ProxyConsumer.cpp` and `EC_Default_ProxyConsumer.h` for Assignment 3;
- A sub-folder including all the source files you changed or created for Assignment 5; also, in the sub-folder include a `readme.txt` file explaining what you've done so far for the Assignment 5 above. In the `readme.txt` file, use plain language to clearly describe what you thought and what you did. We will give partial scores according to your description.

Post your questions on the Moodle forum.

Concluding remarks

I hope that from this course you have learned some flavors of real-time systems. It is a challenging subject in computing science and engineering indeed. It is also a beautiful and practical topic to ponder upon and to try with our hands. In doing so, sometimes I find it a joy to see how my analysis may predict a system's behavior, and how my experiment may confirm the predicted performance; in many other times, I just find it surprising that the system implementation, with so many details that could go wrong, actually works. I think before such sophisticate systems we ought to humble ourselves.