CSC9006 Real-Time Systems (Spring 2021) Homework 7

CSC9006 Real-Time Systems (Spring 2021) Homework 7
- Instruction
- Moodle submission

** **Submission deadline: 9PM, June 12th, 2021.** **

Instruction

In this and the future homework assignments, we are going to build our real-time fault-tolerant event service. We will learn to put into practice what we've learned so far:

- real-time scheduling and latency measurement
- fault tolerance using the primary-backup model
- fault injection and software testing

In this homework assignment, we will first setup some necessary data structures; in the next homework assignment, we will implement the primary and the backup event service. Overall, we will make use some of the ideas described in the FRAME paper:

> Wang, C., Gill, C., & Lu, C. (2019, July). Frame: Fault tolerant and real-time messaging for edge computing. In 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS) (pp. 976-985). IEEE.

For ease of code maintenance, you should create a new git branch and do your work in the branch. In addition, we will modify the example at directory `$TAO_ROOT/orbsvcs/examples/RtEC/Simple`, and you may create another git repository for that. In the following, whenever we mention event suppliers or event consumers, we mean those in the above example.

Assignment 1 (15 points): In order to measure latency performance, we will encapsulate the creation time of an event into its payload. In particular, in the implementation of an event supplier, use C++'s `chrono` library to take a time stamp before *each* push of an event (line 92), and put the time stamp information into the payload of `event[0]`. To achieve so, first we need to make the size of the payload equal to the size of the `chrono` time point, and you should do that by adding this statement at line 89:

```
event[0].data.payload.length (sizeof(std::chrono::system_clock::time_point));
```

Recall that you've been using the `chrono` library since Homework 1. You should review its usage. You may use function `memcpy` to update the payload: http://www.cplusplus.com/reference/cstring/memcpy/ ; besides using `sizeof()` to specify the length of the copy, you may alternatively use `event[0].data.payload.length ()` to get the length you specified at line 89.

Assignment 2 (15 points): Now do the following modification on event suppliers and event consumers, respectively. We will implement five event suppliers. Use the `source` field in the event header to distinguish different event suppliers (line 87 in `Supplier.cpp`). Then change line 82 for different sending periods. Here is our specification of event suppliers:

Supplier source #	Sending period of the supplier (milliseconds)	Relative end-to-end deadline of events for the supplier (milliseconds)
0	100	100
1	100	50
2	100	200
3	200	150
4	200	70

We will use only one event consumer. Modify the push function in `Consumer.cpp` (lines 109-126) to take a time stamp and calculate and print the end-to-end latency for each event received, one event per line. For each event, also print out the source number of its supplier. Each output line should follow the format:

`(source #) + single space + (end-to-end latency in milliseconds)`

Assignment 3 (25 points): In the event service we will implement the EDF scheduling policy. Following Assignment 5 in Homework 6, in `TAO_EC_Default_ProxyPushConsumer::push (const RtecEventComm::EventSet& event){...}` now push the incoming event into a priority queue. The events in the queue are sorted according to their *absolute* end-to-end deadlines, and the one with the earliest deadline will be popped first. Use C++ standard library's `priority queue` to achieve the purpose: http://www.cplusplus.com/reference/queue/priority_queue/ . You should use the encapsulated event creation time from Assignment 1 *and* you should take into account the time spent before the event arrived at this function. Here is an example class for you:

```
// An example class that implements the priority-queue's comparing function
// lhs: left-hand-side; rhs: right-hand-side
// You may use typedef to define type task
class myComparison
{
public:
  myComparison() {};
  bool operator() (const task& lhs, const task& rhs) const
  {
    if (std::chrono::duration_cast<std::chrono::microseconds>(lhs - rhs).count()
> 0)
      return true;
    else
      return false;
  }
};
```

You should think about how you would do software testing to verify the correctness of such a priority queue, and you should do the planned testing yourself.

Assignment 4 (30 points): Let's call the thread that pushes events into the priority queue the *proxy thread*. Following Assignment 3, now let's use another thread, which we call the *worker thread*, to carry out EDF processing. Implement the worker thread such that it would take events from the priority queue, run a synthetic workload for 30 milliseconds, and then push the taken event into a FIFO queue and notify the proxy thread for that. Once notified, the proxy thread (i.e., the thread that did the push of the priority queue) should *empty* events from the FIFO queue by pushing them to the event consumer. Use C++ standard library's `FIFO queue` to implement the

FIFO queue: http://www.cplusplus.com/reference/queue/queue/ . Use `condition variables` to synchronize the accesses of the priority queue. Use `mutex` to protect the FIFO queue. Use a share variable for the worker thread to notify the proxy thread. The proxy thread should not wait for the notification; in the absence of notification, the proxy thread should keep processing incoming events from event suppliers.

Assignment 5 (15 points): Now run your event service with the consumer and all the five suppliers. Pin your event service to one dedicated CPU core, and make the consumer and the suppliers run on the other cores. Plot the CDF of the end-to-end latency for events from each supplier. Your CDF plot should include five curves, one for each event supplier. Label the latency in milliseconds.

Moodle submission

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

- `Supplier.cpp` (just attach your code for Supplier #0)
- `Consumer.cpp`
- `EC_Default_ProxyConsumer.cpp` and `EC_Default_ProxyConsumer.h` (i.e., your complete implementation for both Assignments 3 and 4)
- Your CDF plot figure.

Post your questions on the Moodle forum.