Assignment 7

Problem 1.

Consider a system that uses persistent queues for requests and replies, but where the queues do not support transactions. Each queue simply supports the atomic operations Enqueue(r), Dequeue(r), and ReadTop(r), where ReadTop(r) is a non-destructive read of the first element in the queue. Suppose the system adopts the three-transaction request-processing model as described in the lecture slides where Txn1 enqueues a request, Txn2 executes a request and enqueues a reply, and Txn3 dequeues a reply.

a) Suggest an implementation of Txn2 such that each request executes at least once.

b) Suggest another implementation of Txn2 such that each request executes at most once.

c) Suggest another implementation of Txn2 such that each request executes exactly once, under the assumption that each transaction executed by Txn2 is testable.

Problem 2.

Consider a system that uses persistent queues that do not support transactions. Each queue supports the atomic operations

- Enqueue(r, qn) – enqueue element r to the end of queue named qn,
- Dequeue(r, qn) – dequeue the top element from the queue named qn and return it in r,
- ReadTop(r, qn) - a non-destructive read of the first element in the queue named qn.

Suppose there is one single-threaded server that processes requests from the request queue. It operates as follows:

```
While (true) do {
        Write(Last = 0); // 0 is a value that is different from any request
        StartTransaction;
                ReadTop(r, RequestQueue);
                s = ProcessRequest(r);
                Write(Last = r);
                Write(Reply = s);
        Commit;
        Enqueue(Reply, ReplyQueue);
        Dequeue(x, RequestQueue);
}
```

In the above server program, Last and Reply are transactional data items in stable storage, so Write operations on them are undone if the transaction that invokes them aborts. If the server fails, the values of local variables r, s, and x are lost. After the server recovers, before restarting execution of the above program it runs the following recovery program:

```
        ReadTop(r, RequestQueue)
        If Last = r then {Dequeue(x, RequestQueue); Enqueue(Reply, ReplyQueue)}
```

Although the server fails occasionally, it is highly available. Assume that ProcessRequest() supports transactions. If its transaction aborts, it is undone as if it was never called.

For the following questions, answer true or false and provide a brief explanation.

a) The above programs ensure that for each request r, it is processed exactly once, even in the presence of server failures (assuming no other kinds of failures).

b) The above programs ensure that each reply is enqueued at least once, even in the presence of server failures (assuming no other kinds of failures).

c) Consider the atomic operation EnqueueDequeue(r, qn1, s, qn2) which enqueues r on queue qn1 and dequeues the top element of qn2 and returns it in s. In the server program, replace the last two lines by EnqueueDequeue(s, ReplyQueue, x, RequestQueue). In the recovery program, replace the then-clause by EnqueueDequeue(Reply, ReplyQueue, x, RequestQueue). Now the programs ensure that each reply is enqueued exactly once, even in the presence of server failures (assuming no other kinds of failures).

Problem 3.

Redo problem 2 assuming that ProcessRequest() has side effects, and does not execute atomically.