Implementing Vacation Booking Agency with 2-Phase Commit

Note: Please read the description completely before developing the solution

In this project, you will implement a simple vacation booking portal with 2PC. The portal will be the coordinator and will interact with 2 participants, namely, *hotel reservation* and *concert reservation*. You will implement the coordinator as well as the two participants. Henceforth, the coordinator and the participants are referred to as entities (i.e., when you see the word entity you should assume that it refers to coordinator and/or participants). Your implementation will also have a provision to make the coordinator and the participants artificially fail and recover (see below).

The hotel will have limited number of rooms and the concert has limited number of tickets for each day (specified in respective configuration files – please see below). A vacation can be successfully booked only if a room is available for all of the days and concert tickets are available for all the days. Even if one of them is not available the booking should not be performed. The system should work correctly even in the face of failures of entities. Your program should implement the various primitives of the 2PC protocol and use them for implementing the system. Details of the coordinator and the participants are provided below.

Communication Substrate (CS)

In order to simulate asynchronous communication mode as well as failures and recoveries, each of the participants and the coordinator will have a communication substrate (CS). Each CS will send and receive messages with other substrates through sockets. You can assume that the substrates of the coordinator and the participants know one another's IP address and port numbers.

The substrates provide two message buffers to the respective coordinator/participant, namely, an incoming buffer and an outgoing buffer. The coordinator and participants put messages into the outgoing buffer whenever they want to send messages to others. Unless the respective entity is in a "FAILED" state (see below), CS will send messages from the outgoing buffer to the recipients. Analogously, when the entity has not FAILED, CS will receive messages from other entities and puts them in the incoming buffer. If the entity is in the FAILED state, CS will not send any messages. It will still receive messages. But, it will not put them in the incoming buffer. Instead, it discards all messages.

Notice that CS will have at least two threads – one to receive messages and the other to send message (you can have more threads depending upon your design).

Coordinator

The coordinator program will have one command-line parameter, namely, the configuration file name. The configuration file contains the IP addresses and port numbers of the participants and the name of the booking requests file. The coordinator will have at least two threads in addition to the CS threads (again, you can have more threads). One thread (called thread-A) will be used for emulating artificial failures and recoveries. The actual actions of the coordinator will be implemented in the second thread

(thread-B). A system-wide variable (SYSTEM_STATUS) indicates the current state of the system. SYSTEM_STATUS can be NORMAL, RECOVERY or FAILED

When the SYSTEM_STATUS is NORMAL, thread-B will read the next booking request from the booking requests file. The line of the file will be in format below.

```
Booking ID (Integer)   # of tickets (Integer)  [dates] (integers
separated by spaces and enclosed with in []).
```

Below is an example:

```
101  3 [1 3 5]
```

For simplicity, you can assume that each person stays in a separate room (i.e., number of rooms and number of tickets desired are the same). Also, for simplicity, we assume that the system allows for booking only 10 days in advance and the dates are integers from 1 through 10.

Once a line is read, the coordinator starts the transaction (for e.g., with begin_transaction or init_transaction messages to participants). It then sends the request to the participants, and then tries to commit the transaction (by asking for votes). The result of each transaction is written to a results file. The coordinator must implement time-out mechanism for identifying participant failures. It must use a log-file for recovery.

When the SYSTEM_STATUS is NORMAL, thread-B may be interrupted by thread-A to indicate failure. Upon interruption, thread-B will check the system status. If the system status is FAILED, it will stop its operation. It will then clear all its memory (i.e., the variables get set to default values), delete all messages from the buffers and starts to sleep.

When thread-B is sleeping, it may be interrupted by thread-A (to indicate the system should start recovering). When this happens, thread-B will first check the SYSTEM_STATUS to check if the system is indeed in RECOVERY. If so, it initiates the recovery process (reading the log file and possibly interacting with participants), and it rebuilds the system state. Once the system is successfully rebuilt, the system status is set to NORMAL and then it continues its operation.

Thread-A will receive user commands (FAIL/RECOVER) and react in the following manner. If the system is currently operational, and the user command is FAIL, this thread will set the system status to FAIL and interrupt thread-A. Conversely, if the system is currently in the FAILED mode, and the user command is RECOVER, it will set the system status to recover and interrupt the sleeping thread-B.

PARTICIPANTS

The participant programs will have one command-line parameter, namely, the configuration file name. The configuration file contains the IP addresses and port numbers of the coordinator and the total number of tickets/rooms for each day. An example is given below.

```
128.196.212.56  4256
1 5
2 5
```

```
3 5
…..
….
10 3
```

Upon booting up, the participant will initialize the system with the provided values.

The participants will also have two threads – thread-A and thread-B. As in the case of coordinator, thread-B is for implementing the participant logic and thread-A is for emulating failures. The SYSTEM_STATUS can again have one of three values – NORMAL, FAILED or RECOVERY.

When the system status is NORMAL Thread-B receives messages from the coordinator (through the incoming buffer) and performs the respective operations and sends messages by putting them in the outgoing buffer.

If thread-B is interrupted when the system is NORMAL, it will check the SYSTEM_STATUS. If the SYSTEM_STATUS is FAILED, it stops its operation, cleans up all the memory, deletes all messages from the buffers and enters the sleep mode.

If a thread is interrupted when it is sleeping, it checks the SYSTEM_STATUS. If the SYSTEM_STATUS is RECOVERY, it begins the recovery operation and brings back the system to a consistent state. It then sets SYSTEM_STATUS to NORMAL and then proceeds with normal operations.

Thread-A will receive user commands (FAIL/RECOVER) and react in the following manner. If the system is currently operational, and the user command is FAIL, this thread will set the system status to FAIL and interrupt thread-A. Conversely, if the system is currently in the FAILED mode, and the user command is RECOVER, it will set the system status to recover and interrupt the sleeping thread-B.

POINTS TO NOTE:

1. The above description outlines a high-level description of the system. You will have to work out a detailed design of the system (e.g., message formats, log formats, timeout details, etc.).
2. The participants may maintain the current available tickets/rooms in a DB or in some other form of persistent storage.
3. In order to simplify the design and implementation, you can assume that the coordinator processes requests sequentially.
4. You can also assume that the coordinator will not fail when the participants are in the ready state (waiting for a response from the coordinator). In other words, you do not need to handle the “indefinite wait” problem.
5. You can also assume that there will not be simultaneous coordinator and participant failures.
6. You can also assume that an entity will not fail when it or some other entity is in recovery mode.
7. You can make other assumptions as necessary provided that they do not contradict the basic design and nature of the 2PC protocol.
8. Submission will be through ELC. You will do a demo to me for evaluation.