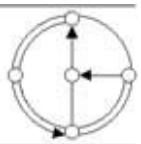
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich



SS 2006 Prof. Gustavo Alonso / Angela Nicoară / Patrick Stüedi

Computer Networks
Exercise 5

Start date: May 12, 2006
Due date: May 26, 2006

1. Task: Implementation of a Persistent Message Queue

The objective of this assignment is to implement a persistent message queue server and two clients, as shown in Figure 1. The server can receive / send messages from / to the clients. All messages and queues are persistently stored by the server. In case the server crashes, it can recover by using the persistent storage and restore the queues and messages in the queues. Additionally, the server can create new queues and delete the old queues according to the clients' requests.

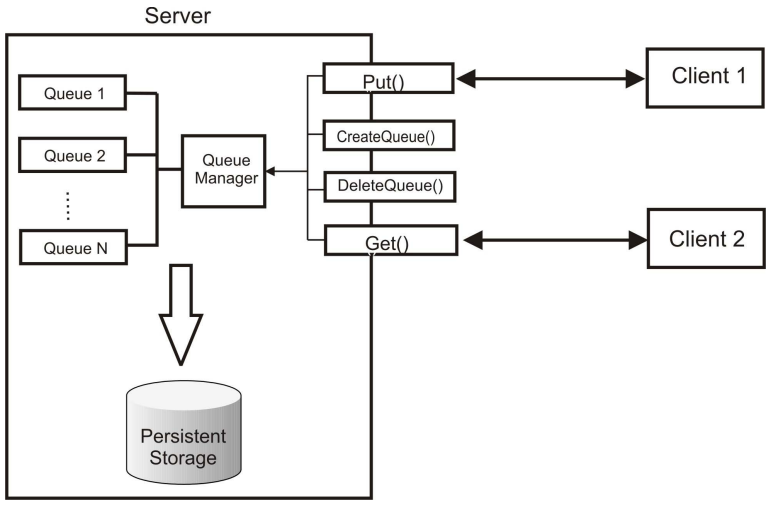


Figure 1: The Message Server

- The server **`PersistMessagePoolServer`** implements four remote methods defined in the **`PersistMessagePool`** remote interface:

```
package server;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface PersistMessagePool extends Remote {

   public void createQueue(String name) throws RemoteException;
   public void deleteQueue(String name) throws RemoteException;

   public void put(String msg, String queueName) throws RemoteException;
   public String get(String queueName) throws RemoteException;

}
```

Figure 2. The remote interface **`PersistMessagePool.java`**

- the **`put()`** method accepts a **`String`** message and a queue name from the client, finds the queue and stores the message into the queue on the server. In case the queue is not found, a **`QueueNotFoundException`** is thrown. If the queue is full, the **`put()`** operation will fail, and a **`QueueFullException`** will be thrown. In case the message from the client is null, the server will throw a **`MessageNullException`**.

- the **`get()`** method retrieves the message from the queue specified by the name to the client which invokes it. In case the queue is not found, a **`QueueNotFoundException`** is thrown. The retrieved message will be deleted from the queue. In case the queue is empty, the **`get()`** operation will fail and a **`QueueEmptyException`** will be thrown.

- the **`createQueue()`** method creates a queue with the corresponding **`queueName`**. In case the queue already exists, a **`QueueDuplicateException`** is thrown. Otherwise **`true`** is returned.

- the **`deleteQueue()`** method deletes a queue with the corresponding **`queueName`**. In case the queue cannot be found, a **`QueueNotFoundException`** is thrown.

- The server maintains a two-layer architecture for queue management: a queue manager **`PersistMessageQueueManager`** and a **`PersistMessageQueue`** queue.

  - **`PersistMessageQueueManager`** stores and manages a collection of queues and their states. Creating, deleting, and accessing the queues have to be controlled by **`PersistMessageQueueManager`**.

  - **`PersistMessageQueue`** is a FIFO queue having a size of 100 messages. It implements the **`enqueue()`** and **`dequeue()`** methods.

- Two clients **`MessageGetClient`** and **`MessagePutClient`** are to be implemented:
  - **`MessageGetClient`** calls the remote method **`get()`** to get a message from a specific queue. For example, call **`get("queue1")`** to get a message from the **`"queue1"`** queue. The client retrieves messages periodically (1 message per 2

seconds); In case the specified queue cannot be found and the queue is empty, the client will wait and then try again.

- **`MessagePutClient`** calls the remote method **`put()`** to send a message to a specific queue. The message could be the timestamp of the client or any random generated string. For example, call **`put("helloworld", "queue1")`** to send the *"helloworld"* message to the **`queue1`** queue. If the queue is not found, the client calls **`createQueue("queue1")`** to generate the queue first, and then send the message. The client generates messages periodically (1 message per second). In case the queue is full, the client will wait and then try again.

- The following exceptions will be thrown as **`RemoteException`** in various situations. Make sure they are handled properly.

  - **`QueueFullException`** is thrown when the queue is full and there is a message to be put into the queue.
  - **`QueueEmptyException`** is thrown when the queue is empty and there is a retrieve operation.
  - **`QueueDuplicateException`** is thrown when trying to create a queue with a name that already exists.
  - **`QueueNotFoundException`** is thrown when trying to find or delete a queue which cannot be found.
  - **`MessageNullException`** is thrown when trying to put an empty message into the queue.

- Persistency and fault tolerance: The server should be robust and fault tolerant. If the server crashes and restarts, it should be able to recover the previous state, including all the queues that have been generated, all the messages that have been stored in individual queues, and the states of the messages and queues. Once the server restarts, it serves the clients' requests as if the crash never happened.
  - The recovery mechanism of the server needs a persistent storage of all the data and states. Here are some hints:
  - For each queue, there is a queue file storing the data. The queue files are stored in a specific directory ***`queues`*** with the ***`.que`*** extension. A queue has the same name as its queue file name. For example, ***`queue1`*** has its file called ***`queue1.que`***.
  - The queue manager checks the available queue files under the ***`queues`*** directory, and restores those queues.

- For this exercise the program should be single threaded.