Assignment 3 Internet Programming

Marc Went (2507013) and Ferry Avis (1945653)

October 23, 2015

1 A paper storage server

The `rpcfunc.x` file contains the definition of the the functions the `paperserver.c` and `paperclient.c` need to implement. Opaque had to be typedef-ed so pointer references could be used, without the typedef this was not possible; when we tried it.

The `paperserver.c` reuses the structures defined in `rpcfunc.x` so that when sending data over to the `paperclient.c` there is no need for converting structures. When sending the list of papers, a duplicate linkedlist is created where the papers are not stored, since the client does not need to receive all the papers at once.

The `paperclient.c`'s readFile first looks at the end of the file to determine the length. Then allocates that length in memory, after which the file is read to memory. The length and the buffer are saved in the *fileParams* struct so that the values can be passed properly into the opaque variable which is sent to the server. By doing so the pointers and values are set properly, without this some pointer problems arose with setting the opaque varables.

2 A hotel reservation server

Each hotel type is represented by an instance of the class `HotelRoomType`, that maintains the guest list for that type. In `HotelImpl.java` an array of `HotelRoomTypes` is defined.

Remote method invocations can run concurrently on the same object. Therefore, the booking process per `HotelRoomType` object is placed in a `synchronized` block. At reading the guest list, this block is not necessary. As no bookings can be canceled, the worst that can happen is that there is an extra booking between obtaining the current size of the guest list and placing the bookings in an array for returning the result. The extra booking would be missing in the result, but this has no severe consequences.

Both the gateway and client need code for invoking the remote methods and printing the output. The methods to do this are shared by using the abstract class `HotelDisplayLogic`. Output is printed to a `PrintWriter` object. For the client, output is redirected to the standard output, while the gateway prints it to the output stream of the socket. To make the prompt appear before any output of RMI's, `out.flush();` needed to be called.

The help message is implemented at client side, as server should not prescribe the user interface that must be used to provide the hotel reservation services to the users.

In this way, different types or versions of clients can be supported, such as the gateway and client. Both are responsible to provide an user interface to call the right methods at the server by RMI, but provide different ways to do this.

3 Answers to questions

3.1 Question A

By creating a (doubly) linked list you can send an arbitrary number of papers without knowing how many there will be.

3.2 Question B

The problem is that you have to send data of arbitrary length over the network. Knowing when the whole file has finished up- or downloading is the tricky part. By using the SunRPC *opaque* structure the problem is fixed, because not only the file-data is sent, but also the file-data-length. It also doesn't stop at the *null*-character which the *string* structure does.

3.3 Question C

The `HotelGateway` is a threaded server. Telnet sessions are long-living TCP connections and iterative server can only handle one connection at the time. If one user is connected to the server, another user would have to wait until the current user has closed the connection. This can take a long time.

In the threaded server, for each new TCP connection a new instance of a `HotelGatewayThread` object is created and placed in a new thread. Now, requests can be handled simultaneously. As each request gets a dedicated object, there are no concurrency issues, other than those in the server.

3.4 Question D

You would be able to send files over a text based protocol, but the length of the file should be send in advance. In C, for example, strings are terminated with a null character, but the null character can be a valid character in any binary file represented as text. By sending the length of the file we can read until the specified amount of bytes have been read, without relying on a terminating character.