EXERCISES

4.1 Is it conceivably useful for a port to have several receivers? *page 148*

4.2 A server creates a port that it uses to receive requests from clients. Discuss the design issues concerning the relationship between the name of this port and the names used by clients. *page 148*

4.3 The programs in Figure 4.3 and Figure 4.4 are available at www.cdk5.net/ipc. Use them to make a test kit to determine the conditions in which datagrams are sometimes dropped. Hint: the client program should be able to vary the number of messages sent and their size; the server should detect when a message from a particular client is missed. *page 150*

4.4 Use the program in Figure 4.3 to make a client program that repeatedly reads a line of input from the user, sends it to the server in a UDP datagram message, then receives a message from the server. The client sets a timeout on its socket so that it can inform the user when the server does not reply. Test this client program with the server program in Figure 4.4. *page 150*

4.5 The programs in Figure 4.5 and Figure 4.6 are available at www.cdk5.net/ipc. Modify them so that the client repeatedly takes a line of user's input and writes it to the stream and the server reads repeatedly from the stream, printing out the result of each read. Make a comparison between sending data in UDP datagram messages and over a stream. *page 153*

4.6 Use the programs developed in Exercise 4.5 to test the effect on the sender when the receiver crashes, and vice-versa. *page 153*

4.7 Sun XDR marshals data by converting it into a standard big-endian form before transmission. Discuss the advantages and disadvantages of this method when compared with CORBA CDR. *page 160*

4.8 Sun XDR aligns each primitive value on a 4-byte boundary, whereas CORBA CDR aligns a primitive value of size n on an n-byte boundary. Discuss the trade-offs in choosing the sizes occupied by primitive values. *page 160*

4.9 Why is there no explicit data typing in CORBA CDR? *page 160*

4.10 Write an algorithm in pseudo-code to describe the serialization procedure described in Section 4.3.2. The algorithm should show when handles are defined or substituted for classes and instances. Describe the serialized form that your algorithm would produce when serializing an instance of the following class, *Couple*:

```
class Couple implements Serializable{
        private Person one;
        private Person two;
        public Couple(Person a, Person b) {
            one = a;
            two = b;
        }
}
```
page 162

4.11 Write an algorithm in pseudo-code to describe deserialization of the serialized form produced by the algorithm defined in Exercise 4.10. Hint: use reflection to create a class from its name, to create a constructor from its parameter types and to create a new instance of an object from the constructor and the argument values. *page 162*

4.12 Why can't binary data be represented directly in XML, for example, by representing it as Unicode byte values? XML elements can carry strings represented as *base64*. Discuss the advantages or disadvantages of using this method to represent binary data. *page 164*

4.13 Define a class whose instances represent remote object references. It should contain information similar to that shown in Figure 4.13 and should provide access methods needed by higher-level protocols (see request-reply in Chapter 5, for example). Explain how each of the access methods will be used by that protocol. Give a justification for the type chosen for the instance variable containing information about the interface of the remote object. *page 168*

4.14 IP multicast provides a service that suffers from omission failures. Make a test kit, possibly based on the program in Figure 4.14, to discover the conditions under which a multicast message is sometimes dropped by one of the members of the multicast group. The test kit should be designed to allow for multiple sending processes. *page 170*

4.15 Outline the design of a scheme that uses message retransmissions with IP multicast to overcome the problem of dropped messages. Your scheme should take the following points into account:

i) There may be multiple senders.

ii) Generally only a small proportion of messages are dropped.

iii) Recipients may not necessarily send a message within any particular time limit.

Assume that messages that are not dropped arrive in sender order. *page 173*

4.16 Your solution to Exercise 4.15 should have overcome the problem of dropped messages in IP multicast. In what sense does your solution differ from the definition of reliable multicast? *page 173*

4.17 Devise a scenario in which multicasts sent by different clients are delivered to two group members in different orders. Assume that some form of message retransmission is in use, but that messages that are not dropped arrive in sender order. Suggest how recipients might remedy this situation. *page 173*

4.18 Revisit the Internet architecture as introduced in Chapter 3 (see Figures 3.12 and 3.14). What impact does the introduction of overlay networks have on this architecture, and in particular on the programmer's conceptual view of the Internet? *page 175*

4.19 What are the main arguments for adopting a super node approach in Skype? *page 177*

4.20 As discussed in Section 4.6, MPI offers a number of variants of *send* including the *MPI_Rsend* operation, which assumes the receiver is ready to receive at the time of sending. What optimizations in implementation are possible if this assumption is correct and what are the repercussions of this assumption being false? *page 180*