Distributed Systems: Concepts and Design

Chapter 4 Exercise Solutions

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

4.1 Ans.

If several processes share a port, then it must be possible for all of the messages that arrive on that port to be received and processed independently by those processes.

Processes do not usually share data, but sharing a port would requires access to common data representing the messages in the queue at the port. In addition, the queue structure would be complicated by the fact that each process has its own idea of the front of the queue and when the queue is empty.

Note that a port group may be used to allow several processes to receive the same message.

4.2 A server creates a port which 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.

4.2 Ans.

The main design issues for locating server ports are:

(i) How does a client know what port and IP address to use to reach a service?

The options are:

- use a name server/binder to map the textual name of each service to its port;
- each service uses well-known location-independent port id, which avoids a lookup at a name server.

The operating system still has to look up the whereabouts of the server, but the answer may be cached locally.

(ii) How can different servers offer the service at different times?

Location-independent port identifiers allow the service to have the same port at different locations. If a binder is used, the client needs to reconsult the client to find the new location.

(iii) Efficiency of access to ports and local identifiers.

Sometimes operating systems allow processes to use efficient local names to refer to ports. This becomes an issue when a server creates a non-public port for a particular client to send messages to, because the local name is meaningless to the client and must be translated to a global identifier for use by the client.

4.3 The programs in Figure 4.3 and Figure 4.4 are available on 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.

4.3 Ans.

For a test of this type, one process sends and another receives. Modify the program in Figure 4.3 so that the program arguments specify i) the server's hostname ii) the server port, iii) the number, *n* of messages to be sent and iv) the length, *l* of the messages. If the arguments are not suitable, the program should exit immediately. The program should open a datagram socket and then send *n* UDP datagram messages to the

server. Message *i* should contain the integer *i* in the first four bytes and the character '*' in the remaining l-4 bytes. It does not attempt to receive any messages.

Take a copy of the program in Figure 4.4 and modify it so that the program argument specifies the server port. The program should open a socket on the given port and then repeatedly receive a datagram message. It should check the number in each message and report whenever there is a gap in the sequence of numbers in the messages received from a particular client.

Run these two programs on a pair of computers and try to find out the conditions in which datagrams are dropped, e.g. size of message, number of clients.

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 in Figure 4.4.

4.4 Ans.

The program is as Figure 4.4 with the following amendments:

```
DatagramSocket aSocket = new DatagramSocket();
aSocket.setSoTimeout(3000);// in milliseconds
while (// not eof) {
try{
   // get user's input and put in request
  .....
   aSocket.send(request);
   ........
  aSocket.receive(reply);
}catch (InterruptedIOException e){System.out.println("server not responding");}
```

4.5 The programs in Figure 4.5 and Figure 4.6 are available at 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.

4.5 Ans.

The changes to the two programs are straightforward. But students should notice that not all sends go immediately and that receives must match the data sent.

For the comparison. In both cases, a sequence of bytes is transmitted from a sender to a receiver. In the case of a message the sender first constructs the sequence of bytes and then transmits it to the receiver which receives it as a whole. In the case of a stream, the sender transmits the bytes whenever they are ready and the receiver collects the bytes from the stream as they arrive.

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

4.6 Ans.

Run them both for a while and then kill first one and then the other. When the reader process crashes, the writer gets IOException - broken pipe. When writer process crashes, the reader gets EOF exception.

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's CDR.

4.7 Ans.

The XDR method which uses a standard form is inefficient when communication takes place between pairs of similar computers whose byte orderings differ from the standard. It is efficient in networks in which the byte-

ordering used by the majority of the computers is the same as the standard form. The conversion by senders and recipients that use the standard form is in effect a null operation.

In CORBA CDR senders include an identifier in each message and recipients to convert the bytes to their own ordering if necessary. This method eliminates all unnecessary data conversions, but adds complexity in that all computers need to deal with both variants.

4.8 Sun XDR aligns each primitive value on a four 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.

4.8 Ans.

Marshalling is simpler when the data matches the alignment boundaries of the computers involved. Four bytes is large enough to support most architectures efficiently, but some space is wasted by smaller primitive values. The hybrid method of CDR is more complex to implement, but saves some space in the marshalled form. Although the example in Figure 4.8 shows that space is wasted at the end of each string because the following long is aligned on a 4- byte boundary.

4.9 Why is there no explicit data-typing in CORBA CDR?

4.9 Ans.

The use of data-typing produces costs in space and time. The space costs are due to the extra type information in the marshalled form (see for example the Java serialized form). The performance cost is due to the need to interpret the type information and take appropriate action.

The RMI protocol for which CDR is designed is used in a situation in which the target and the invoker know what type to expect in the messages carrying its arguments and results. Therefore type information is redundant. It is of course possible to build type descriptors on top of CDR, for example by using simple strings.

4.10 Write an algorithm in pseudocode 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;
}
}
```

4.10 Ans.

The algorithm must describe serialization of an object as writing its class information followed by the names and types of the instance variables.Then serialize each instance variable recursively.

```
serialize(Object o) {
    c = class(o);
    class_handle = get_handle(c);
    if (class_handle==null) // write class information and define class_handle;
    write class_handle
    write number (n), name and class of each instance variable

    object_handle = get_handle(o);
    if (object_handle==null)  {
        define object_handle;
        for (iv = 0 to n-1)
            if (primitive(iv) ) write iv
            else serialize( iv)
    }
    write object_handle
}
```

To describe the serialized form that your algorithm would produce when serializing an instance of the class *Couple*.

For example declare an instance of Couple as

Couple t1 = new Couple(new Person("Smith", "London", 1934),
new Person("Jones", "Paris", 1945));

The output will be:

Serialized values				*Explanation*
Couple	8 byte version number		h0	*class name, version number, handle*
2	Person one	Person two		*number, type and name of instance variables*
Person	8 byte version number		h1	*serialize instance variable one of Couple*
3	int year	java.lang.String name	java.lang.String place	
1934	5 Smith	6 London	h2	
h1				*serialize instance variable two of Couple*
1945	5 Jones	5 Paris	h3	*values of instance variables*

4.11 Write an algorithm in pseudocode 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.

4.11 Ans.

Whenever a handle definition is read, i.e. a class_info, handle correspondence or an object, handle correspondence, store the pair by method map. When a handle is read look it up to find the corresponding class or object.

```
Object deserialize(byte [] stream) {
    Constructor aConstructor;
    read class_name and class_handle;
    if (class_information == null) aConstructor = lookup(class_handle);
    else {
        Class cl = Class.forName(class_name);
        read number (n) of instance variables
        Class parameterTypes[]= new Class[n];
        for (int i=0 to n-1) {
            read name and class_name of instance variable i
            parameterTypes[i] = Class.forName(class_name);
        }
        aConstructor = cl.getConstructor(parameterTypes);
        map(aConstructor, class_handle);
    }
    if (next item in stream is object_handle) o = lookup(object_handle);
    else {
        Object args[] = new Object[n];
        for (int i=0 to n-1) {
            if (next item in stream is primitive) args[i] = read value
            else args[i]  = deserialize(//rest of stream)
        }
        Object o = cnew.newInstance(args);
        read object_handle from stream
        map(object, object_handle)
        return o;
    }
}
```

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.

4.12 Ans.

Binary data can't be represented directly in XML, because somewhere the embedded binary data will include the representation of a special character such as '<' or '>'.

Even if binary is represented as CDATA, it might include the terminator for CDATA: ']]'.

Binary data can be encoded in base64 and represented in XML as a *string*.

Base64 encoding takes three bytes, each consisting of eight bits, and represents them as four printable characters in the ASCII standard (padding when necessary).

Thus a disadvantage in using base64 is that the quantity of data is increased by a factor of 4/3 and the translation process at both ends can take time. In addition, the encoded value isn't really a string and in some cases may be passed on to an application as a string.

The only advantage of base64 is that it can be used when necessary. It is in fact used in XML Security to represent digital signatures and encrypted data. In both cases, the data is enclosed in elements that specify, for example, *signature* or *encrypted data*.

4.13 Define a class whose instances represent remote object references. It should contain information similar to that shown in Figure 4.10 and should provide access methods needed by the request-reply protocol. 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.

4.13 Ans.

```
class RemoteObjectReference{
    private InetAddress ipAddress;
```

```
        private int port;
        private int time;
        private int objectNumber;
        private Class interface;
        public InetAddress getIPaddress() ( return ipAddress;}
        public int getPort() { return port;);
    }
```

The server looks up the client port and IP address before sending a reply.

The variable interface is used to recognize the class of a remote object when the reference is passed as an argument or result. Chapter 5 explains that proxies are created for communication with remote objects. A proxy needs to implement the remote interface. If the proxy name is constructed by adding a standard suffix to the interface name and all we need to do is to construct a proxy from a class already available, then its string name is sufficient. However, if we want to use reflection to construct a proxy, an instance of Class would be needed. CORBA uses a third alternative described in Chapter 17.

4.14 IP multicast provides a service that suffers from omission failures. Make a test kit, possibly based on the program in Figure 4.17, 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.

4.14 Ans.

The program in Figure 4.17 should be altered so that it can run as a sender or just a receiver. A program argument could specify its role. As in Exercise 4.3 the number of messages and their size should be variable and a sequence number should be sent with each one. Each recipient records the last sequence number from each sender (sender IP address can be retrieved from datagrams) and prints out any missing sequence numbers. Test with several senders and receivers and message sizes to discover the load required to cause dropped messages.The test kit should be designed to allow for multiple sending processes.

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) unlike the request-reply protocol, recipients may not necessarily send a message within any particular time limit.

Assume that messages that are not dropped arrive in sender ordering.

4.15 Ans.

To allow for point (i) senders must attach a sequence number to each message. Recipients record last sequence number from each sender and check sequence numbers on each message received.

For point (ii) a negative acknowledgement scheme is preferred (recipient requests missing messages, rather than acknowledging all messages). When they notice a missing message, they send a message to the sender to ask for it. To make this work, the sender must store all recently sent messages for retransmission. The sender re-transmits the messages as a unicast datagram.

Point (iii) - refers to the fact that we can't rely on a reply as an acknowledgement. Without acknowledgements, the sender will be left holding all sent messages in its store indefinitely. Possible solutions: a) senders discards stored messages after a time limit b) occasional acknowledgements from recipients which may be piggy backed on messages that are sent.

Note requests for missing messages and acknowledgments are simple - they just contain the sequence numbers of a range of lost messages.

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?

4.16 Ans.

Reliable multicast requires that any message transmitted is received by all members of a group or none of them. If the sender fails before it has sent a message to all of the members (e.g. if it has to retransmit a message) or if a gateway fails, then some members will receive the message when others do not.

4.17 Devise a scenario in which multicasts sent by different clients are delivered in different orders at two group members. Assume that some form of message retransmissions are in use, but that messages that are not dropped arrive in sender ordering. Suggest how recipients might remedy this situation.

4.17 Ans.

Sender1 sends request r1 to members m1 and m2 but the message to m2 is dropped

Sender2 sends request r2 to members m1 and m2 (both arrive safely)

Sender1 re-transmits request r1 to member m2 (it arrives safely).

Member m1 receives the messages in the order r1;r2. However m2 receives them in the order r2;r1.

To remedy the situation. Each recipient delivers messages to its application in sender order. When it receives a message that is ahead of the next one expected, it hold it back until it has received and delivered the earlier re-transmitted messages.

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?

4.18 Ans.

The initial Internet architecture was devised for a relatively small number of applications (such as electronic mail and file transfer) to run over relatively homogeneous styles of network. The definition and standardization of the (relatively simple) Internet architecture has been very successful in supporting these applications and other emerging applications. As the Internet grows though in terms of scale and diversity, there is a need to support richer application styles and also to operate more effectively over an increasing variety of network types. The introduction of overlay networks introduces an element of extra complexity into the Internet architecture but also an increasing degree of sophistication to meet these two requirements. For example, overlay networks can be introduced to support new styles of Internet applications, for example related to streaming of continuous media content. The Internet can also be extended to operate more optimally over new network types, for example ad hoc networks.

The key elements in terms of the conceptual view offered by the Internet is that programmers now see the architecture as extensible and also see the potential to have more application-specific network services. They also see a move from rigid standardization to a more experimental environment.

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

4.19 Ans.

The main reason for having super nodes is to implement efficient and reliable search, a key enabling function in Skype. Through this approach, search is supported only by nodes that have the required capabilities in terms of bandwidth, reachability and availability. The alternative is to adopt a pure peer-to-peer approach where all nodes cooperate to provide this function. The downside of this pure approach is that certain nodes involved in the implementation of search may be disconnected, or have weak connectivity in terms of bandwidth, leading to a more erratic service. The trade-off is that super nodes will encounter more traffic due to search, but they are selected on the basis they can absorb this.

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?

4.20 Ans.

The key optimization is that the sender does not have to check that the receiver is ready to receive a message and this therefore avoids a handshake between sender and receiver, increasing the performance of the implementation. If the receiver is not ready, the outcome of the send operation is unpredictable in terms of the buffer being ready and the MPI documentation deems the outcome undefined. It may be for example that values will be over-written before being consumed.