CSI 3344Distributed Systems

Workshop Solution 03

Q1. In many layered protocols, each layer has its own header. Surely it would be more efficient to have a single header at the front of each message with all the control in it than all these separate headers. Why is this not done?

A: Each layer must be independent of the other ones. The data passed from layer *k*+1 down to layer *k* contains both header and data, but layer *k* cannot tell which is which. Having a single big header that all the layers could read and write would destroy this transparency and make changes in the protocol of one layer visible to other layers. This is undesirable.

Q2. Why are transport-level communication services often inappropriate for building distributed applications?

A: They hardly offer distribution transparency meaning that application developers are required to pay significant attention to implementing communication, often leading to proprietary solutions. The effect is that distributed applications, for example, built directly on top of sockets are difficult to port and to interoperate with other applications.

Q3. A reliable multicast service allows a sender to reliably pass messages to a collection of receivers. Does such a service belong to a middleware layer, or should it be part of a lower-level layer?

A: In principle, a reliable multicast service could easily be part of the transport layer, or even the network layer. As an example, the unreliable IP multicasting service is implemented in the network layer. However, because such services are currently not readily available, they are generally implemented using transport- level services, which automatically places them in the middleware. However, when taking scalability into account, it turns out that reliability can be guaranteed only if application requirements are considered. This is a strong argument for implementing such services at higher, less general layers.

The following questions are *Optional* (for those who are good at C):

Q4. Consider a procedure *incr* with two integer parameters. The procedure adds one to each parameter. Now suppose that it is called with the same variable twice, for example, as *incr*(*i*, *i*). If *i* is initially 0, what value will it have afterward if call-by-reference is used? How about if copy/restore is used?

A: If call by reference is used, a pointer to *i* is passed to *incr*. It will be incremented two times, so the final result will be two. However, with copy/restore, *i* will be passed by value twice, each value initially 0. Both will be incremented, so both will now be 1. Now both will be copied back, with the second copy overwriting the first one. The final value will be 1, not 2.

Q5. C has a construction called a union, in which a field of a record (called a struct in C) can hold any one of several alternatives. At run time, there is no sure-fire way to tell which one is in there. Does this feature of C have any implications for remote procedure call? Explain your answer.

A: If the runtime system cannot tell what type value is in the field, it cannot marshal it correctly. Thus unions cannot be tolerated in an RPC system unless there is a tag field that unambiguously tells what the variant field holds. The tag field must not be under user control.

END OF THE WORKSHOP SOLUTION