Homework Solutions for Chapters 1, 2, 3

I. SOLUTIONS TO CHAPTER 1 PROBLEMS

1) Q2: Explain what is meant by (distribution) transparency, and give examples of different types of transparency.
   A: Distribution transparency is the phenomenon by which distribution aspects in a system are hidden from users and applications. Examples include access transparency, location transparency, migration transparency, relocation transparency, replication transparency, concurrency transparency, failure transparency, and persistence transparency.
2) Q4: Why is it not always a good idea to aim at implementing the highest degree of transparency possible?
   A: Aiming at the highest degree of transparency may lead to a considerable loss of performance that users are not willing to accept.
3) Q15: Explain what false sharing is in distributed shared memory systems. What possible solutions do you see?
   A: False sharing happens when data belonging to two different and independent processes (possibly on different machines) are mapped onto the same logical page. The effect is that the page is swapped between the two processes, leading to an implicit and unnecessary dependency. Solutions include making pages smaller or prohibiting independent processes to share a page.
4) Q16: An experimental file server is up 3/4 of the time and down 1/4 of the time, due to bugs. How many times does this file server have to be replicated to give an availability of at least 99
   A: With k being the number of servers, we have that $(\frac{1}{4})^k < 0.01$, expressing that the worst situation, when all servers are down, should happen at most $\frac{1}{100}$ of the time. This gives us $k = 4$.

II. SOLUTIONS TO CHAPTER 2 PROBLEMS

1) 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.
2) Q7: Assume a client calls an asynchronous RPC to a server, and subsequently waits until the server returns a result using another asynchronous RPC. Is this approach the same as letting the client execute a normal RPC? What if we replace the asynchronous RPCs with one-way RPCs?
   A: No, this is not the same. An asynchronous RPC returns an acknowledgement to the caller, meaning that after the first call by the client, an additional message is sent across the network. Likewise, the server is acknowledged that its response has been delivered to the client. Two one-way RPCs may be the same, provided reliable communication is guaranteed. This is generally not the case.
3) Q16: Now suppose you could make use of only transient synchronous communication primitives. How would you implement primitives for transient asynchronous communication?

A: This situation is actually simpler. An asynchronous send is implemented by having a caller append its message to a buffer that is shared with a process that handles the actual message transfer. Each time a client appends a message to the buffer, it wakes up the send process, which subsequently removes the message from the buffer and sends it its destination using a blocking call to the original send primitive. The receiver is implemented similarly by offering a buffer that can be checked for incoming messages by an application.

4) Q20: How would you incorporate persistent asynchronous communication into a model of communication based on RMIs to remote objects?
   A: An RMI should be asynchronous, that is, no immediate results are expected at invocation time. Moreover, an RMI should be stored at a special server that will forward it to the object as soon as the latter is up and running in an object server.
5) Q: How could you guarantee a maximum end-to-end delay when a collection of computers is organized in a (logical or physical) ring?
   A: We let a token circulate the ring. Each computer is permitted to send data across the ring (in the same direction as the token) only when holding the token. Moreover, no computer is allowed to hold the token for more than T seconds. Effectively, if we assume that communication between two adjacent computers is bounded, then the token will have a maximum circulation time, which corresponds to a maximum end-to-end delay for each packet sent.
6) Q25: How could you guarantee a minimum end-to-end delay when a collection of computers is organized in a (logical or physical) ring?
   A: Strangely enough, this is much harder than guaranteeing a maximum delay. The problem is that the receiving computer should, in principle, not receive data before some elapsed time. The only solution is to buffer packets as long as necessary. Buffering can take place either at the sender, the receiver, or somewhere in between, for example, at intermediate stations. The best place to temporarily buffer data is at the receiver, because at that point there are no more unforeseen obstacles that may delay data delivery. The receiver need merely remove data from its buffer and pass it to the application using a simple timing mechanism. The drawback is that enough buffering capacity needs to be provided.
7) Q26: Imagine we have a token bucket specification where the maximum data unit size is 1000 bytes, the token bucket rate is 10 million bytes/sec, the token bucket size is 1 million bytes, and the maximum transmission rate is 50 million bytes/sec. How long can a burst of maximum speed last?
   A: Call the length of the maximum burst interval Δt. In an extreme case, the bucket is full at the start of the interval (1 million bytes) and another $10\Delta t$ comes in during that interval. The output during the transmission burst consists of $50\Delta t$ million bytes, which should be equal to $(1 + 10\Delta t)$. Consequently, Δt is equal to 25 msec.

III. SOLUTIONS TO CHAPTER 3 PROBLEMS

1) Q1: In this problem you are to compare reading a file using a single-threaded file server and a multithreaded server. It takes 15 msec to get a request for work, dispatch it, and do the rest of the necessary processing, assuming that the data needed are in a cache in main memory. If a disk operation is needed, as is the case one-third of the time, an additional 75 msec is required, during which time the thread sleeps. How many requests/sec can the server handle if it is single threaded? If it is multithreaded?

A: In the single-threaded case, the cache hits take 15 msec and cache misses take 90 msec. The weighted average is $\frac{2}{3} \times 15 + \frac{1}{3} \times 90$. Thus the mean request takes 40 msec and the server can do 25 per second. For a multi-threaded server, all the waiting for the disk is overlapped, so every request takes 15 msec, and the server can handle $66\frac{2}{3}$ requests per second.

2) Q12: Mention some design issues for an object adapter that is to support persistent objects.
A: The most important issue is perhaps generating an object reference that can be used independently of the current server and adapter. Such a reference should be able to be passed on to a new server and to perhaps indicate a specific activation policy for the referred object. Other issues include exactly when and how a persistent object is written to disk, and to what extent the objects state in main memory may differ from the state kept on disk.

3) Q19: Consider a process P that requires access to file F which is locally available on the machine where P is currently running. When P moves to another machine, it still requires access to F. If the file-to-machine binding is fixed, how could the systemwide reference to F be implemented?
A: A simple solution is to create a separate process Q that handles remote requests for F. Process P is offered the same interface to F as before, for example in the form of a proxy. Effectively, process Q operates as a file server.