Assignment-2

NAME: MOHD SADIQUE
CWID: A20380442

Chapter 3

1. Would it make sense to limit the number of threads in a server process?

- **Ans.** Yes, the number of threads in a server process should be limited for two reasons.
  First, all threads require different memory slots for setting up their own private stack. Therefore, having many threads may consume too much memory of server so that server may not work to its full potentials.

  Second, in operating systems every threads works independently which tends to operate in a chaotic manner. Like in a virtual memory system, it may be difficult to build a relatively stable working set, resulting in many page faults and thus I/O. Having many threads cause page faults may thus lead to a performance degradation resulting from page thrashing. Even in those cases where everything fits into memory, we may easily see that memory is accessed following a chaotic pattern rendering caches useless. Therefore, performance may be degraded in comparison to the single-threaded case.

2. In this problem you are to compare reading a file using a single-threaded file server and a multithreaded server. It takes 8 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 40 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?

- **Ans.** For single-threaded case, the cache hits take 8 msec and cache misses take 48 msec (8msec + 40msec). The weighted average is 2/3 ´ 8 + 1/3 ´ 48. Thus the mean request for single threaded case takes 21.3msec and the server can do 46.8 requests per second.
  Request/sec = 1000/21.3 = 46.8requests per second = 46req/sec.

  For a multithreaded server, all the waiting for the disk is overlapped, so every request takes 8 msec, and the server can handle 125 requests per second.

3. 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 system-wide reference to F be implemented?

- **Ans**. When P moves from one machine to another machine and its need file from previous machine to solve this problem, it create a separate process *S* 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 *S* operates as a file server which responds requests of P as a client .

Chapter 5

4. The root node in hierarchical location services may become a potential bottleneck. How can this problem be effectively circumvented?

➢ **Ans:** Root node is become a potential bottleneck in hierarchical location services to solve this problem we need to use **random bit strings as identifiers.** As a result, we can easily partition the identifier space and install a separate root node for each part. In addition, the partitioned root node should be spread across the network so that accesses to it will also be spread.

5. **5.** In a hierarchical location service with a depth of k, how many location records need to be updated at most when a mobile entity changes its location?

➢ **Ans:** Changing location in hierarchical location with a depth of k can be described as the combination of an insert and a delete operation. An insert operation requires that at worst *k* +1 location records are to be changed. Likewise, a delete operation also requires changing *k* +1 records, where the record in the root is shared between the two operations insert and delete
Total = (k+1) +(k+1)   - 1 (entity is shared in both operation)
        = 2k + 1
 Therefore total of 2*k* +1 records needed to be updated.

6. **6.** High-level name servers in DNS, that is, name servers implementing nodes in the DNS name space that are close to the root, generally do not support recursive name resolution. Can we expect much performance improvement if they did?

➢ **Ans:** Probably not: we cannot expect much performance in improvement in high-level name servers in DNS because the high-level name servers constitute the global layer of the DNS name space, it can be expected that changes to that part of the name space which do not occur often. Therefore, caching will be highly effective, and much long-haul communication will be avoided in this way. So that recursive name resolution for low-level name servers is important, because in that case, name resolution can be kept local at the lower-level domain in which the resolution is taking place.

Chapter 6

7. Consider the behavior of two machines in a distributed system. Both have clocks that are supposed to tick 2000 times per millisecond. One of them actually does, but the other ticks only 1900 times per millisecond. If UTC updates come in once a minute, what is the maximum clock skew that will occur?

➢ **Ans:** The First clock ticks 2000,000 times per second and second clock ticks 1900,000 times per second, giving an error of 50 msec per second. In a minute this error has grown to 3000 msec or 3sec.
Another way, the second clock is 5% slow, so after a minute it is off by 0. 05 * 60 sec = 3sec or 3000 msec.

8. To achieve totally-ordered multicasting with Lamport timestamps, is it strictly necessary that each message is acknowledged?

➢ **Ans:** No, it is not necessary that each message is acknowledged by totally-ordered multicasting with Lamport timestamps, it is only sufficient to multicast any other type of message, as long as that message has a timestamp larger than the received message. The condition for delivering a message *m* to the application, is that another message has been received from each other process with a large timestamp. This guarantees that there are no more messages underway with a lower timestamp.

9. **9**. Many distributed algorithms require the use of a coordinating process. To what extent can such algorithms actually be considered distributed? Discuss.

- **Ans:** Coordination in a synchronous system with no failures is comparatively easy. However, if a system is asynchronous, meaning that messages may be delayed an indefinite amount of time, or failures may occur, then coordination and agreement become much more challenging.
  In a centralized algorithm, there is often one fixed process that acts as coordinator. Distribution comes from the fact that the other processes run on different machines. In distributed algorithms with a nonfixed coordinator, the coordinator is chosen (in a distributed fashion) among the processes that form part of the algorithm. The fact that there is a coordinator does not make the algorithm less distributed.

10. **10**. A distributed system may have multiple, independent resources. Imagine that process 0 wants to access resource A and process 1 wants to access resource B. Can Ricart and Agrawalas algorithm lead to deadlocks? Explain your answer.

- **Ans:** In distributed system may have multiple independent resources so processes access resources strictly sequentially, that is, a process holding a resource may not attempt to access another one, then there is no way that it can block while holding a resource that some other process wants. The system is then deadlock free.
  On the other hand, if process 0 may hold resource *R1* and then try to access resource *R2*, a deadlock can occur if some other process tries to acquire them in the reverse order. The Ricart and Agrawala algorithm itself does not contribute to deadlock since each resource is handled independently of all the others.