Assignment #2

Chapter: 3

Q1. Would it make sense to limit the no. of threads in server process?

Ans: Yes, for two reasons, the no. of threads should be limited in a server process. First reason: all threads require different memory slots for setting up their own private stack. So, having many threads may consume too much memory of server and in this case, server may not work to its full potentials. Secondly, every threads work independently which tends to work in a chaotic manner. It may be difficult to build a relatively stable working set which results in many page fault and therefore I/O. Because of page faults, it leads to performance degradation. Even if everything fits into memory we may easily see that memory is accessed following a chaotic pattern results useless caches. Therefore, to maintain the performance, we should limit the no. of threads in server process.

Q2. In this problem you are to compare reading a file using a single-threaded file server and a multithreaded server. It takes 5msec. to get a request for work, dispatch it and

do the rest of the necessary processing, assuming that the data needed are in cache in main memory. If a disk operation is needed, ~~as processing, assuming that the data needed are in a cache in main memory~~ as is the case one-third of the time, an additional 30msec 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?

Soln: In case of single-threaded, the cache hits take 5msec and cache misses take (5+30) = 35 msec.

∴ The weighted avg. $= \frac{2}{3} \times 5 + \frac{1}{3} \times 35$

$= \frac{10}{3} + \frac{35}{3} = \frac{45}{3}$

$= 15$ msec.

Request /sec $= \frac{1000}{15}$

$= 66.66$

∴ mean request for single threaded

case takes 15 msec and the server can do 66 requests per sec.

For multithreaded server, all the waiting for the disk is overlapped, so every request takes 5 msec. and server can do (1000/5) = 200 requests per second.

Q3 Consider a process Q that requires access to file f which is locally available on the machine where Q is currently running. When Q 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. A simple solution is to create a separate process P_1 that handles remote requests for file f. Process P_1 is offered the same interface to file f as before, in the form of proxy. Effectively the process P_1 will operate as a file server which responds requests of Q as a Client.

Chapter 5

Q4. The root node in ~~the~~ hierarchical location services may become a potential bottleneck. How can this problem be effectively circumvented?

Ans: Root node in hierarchical location services may become a ~~pote~~ potential & bottleneck. We can solve this ~~pro~~ problem by using random bit strings as identifiers. In that way, 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.

Q5. In a hierarchical location service with a depth of x, how many location records need to be updated at most when a mobile entity changes its location?

Ans: When a mobile entity changes its location, number of location records ~~can~~ in hierarchical location service with a depth of x can be described as the combination of an insert and a

delete operation. Insert operation requires at most x+1 location records to be changed. Delete operation also requires x+1 records where record in the root is shared b/w two operations insert and delete.

∴ Total records = (x+1) + (x+1) − 1

= 2x+1

Q6. High-level name servers in DNS, ie, 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. We cannot expect much improvement in high-level name servers in DNS because the high-level name servers form 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. So, caching will be highly effective and much long-haul communication will be avoided in this way. So, the recursive name resolution

for low-level name servers are important since so that name resolution can be kept local at the lower level domain in which resolution is happening.

Chapter 6

Q7. Consider the behavior of two machines in a distributed system. Both have clocks that are supposed to tick 1500 times per ms. One of them actually does, but the other ticks only 1400 times per ms. If UTC updates come in once a minute, what is the max^m clock skew that will occur?

Soln: The first clocks ticks 1500 times/msec and second clock ticks 1400 times/msec

We can see that the second clock is

$$\frac{1500-1400}{1500} \times 100 = \frac{10000}{1500} = 6.6\% \text{ slower}$$

than 1st clock.

∴ After a minute, it is off by

$$60 \times \frac{6.6}{100} = 60 \times 0.066 \text{ sec}$$

$$= 3.96 \text{ sec}$$

$$\approx 4000 \text{ msec} \quad \text{Ans}$$

Qs. 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 larger timestamp 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 larger timestamp. This guarantees that there are no more messages underway with a lower timestamp.

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

Ans. Coordination in a synchronous system with no failures is comparatively easy. However, if a system is asynchronous, messages may be delayed an indefinite amount of time or it may failed,

then coordination and agreement become much more challenging.
In centralized algorithms, 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 among the processes that form part of the algorithm.

Q 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: A distributed system may have multiple independent resources. ~~There is a sequential process of accessing the resources in distributed system~~ A process holding a resource may not attempt to access another resource, it means access resources is strictly sequentially. In this case, there is no way that

it can block while holding a resource that some other process could. Then we can say that the system is deadlock free.

On the other hand, if a process P_1 may hold resource R_1 and then try to access resource R_2, a deadlock can occur if some other process tries to acquire them in the reverse order. The Ricart and Agrawalas algorithm itself does not contribute to deadlock since each resource is handled independently of all the others.