CMPSCI 677: Operating Systems

Homework 2 Solution

Spring 2002

1. 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 the file server have to be replicated to give an availability of at least 99 percent?

   Answer —

   With k being the number of servers, we have that $(1/4)^k < 0.01$, expressing that the worst situation, when all servers are down, should happen at most $1/100$ of the time. This gives us $k = 4$.

2. Consider the following last-one call semantics for RPCs: a caller repeatedly calls the server until a response is received.
   (a). Give an example of a service for which this semantics is appropriate.
   (b). Give an example of a service for which this semantics is NOT appropriate. Explain why and indicate the semantics that would be appropriate for your example.

   Answer —

   (a) An example: a client repeatly requesting some name-lookup service until a response is received.

   (b) An example: a client requesting $100 transfer from account A to account B.

   In this case, ”exactly-once” semantics would be appropriate.

3. A company’s sales brochure touts that its software libraries provide support for Atmost-once multicast RPC.
   (a). Give an example of an application where such an RPC might be useful.
   (b). Suggest how such an RPC might be realized.

   Answer —

   (a) An example: a business man (RPC client) wants to fax/print a memo to multiple offices (RPC servers). Atmost-once multicast RPC could be used here.

   (b) A simple best-effort multicast transmission protocol (such as UDP using IP multicast) could be used for underlying communication. The client multicasts it’s request. The servers do the requested procedure and no reply needed.

4. Not to be done.

5. Java and other languages support exceptions, which are raised when an error occurs. How would you implement exceptions in RPCs and RMIs?

   Answer —

   Because exceptions are initially raised at the server side, the server stub can do nothing else but catch the exception and marshal it as a special error response back to the client. The client stub, on the other hand, will have to unmarshal the message and raise the same exception if it wants to keep access to the server transparent. Consequently, exceptions now also need to be described in an interface definition language.

6. Suppose you could make use of only transient synchronous communication primitives. How would you implement primitives for transient asynchronous communication?

   Answer —

   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.

7. 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?

   Answer —

   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.

8. In this problem, you are to compare reading a file using a single-threaded file server and a multi-threaded 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 a 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 multi-threaded?

   Answer —

   In the single-threaded case, the cache hits take 15 msec and cache misses take 90 msec. The weighted average is $2/3 \times 15 + 1/3 \times 90$. Thus the mean request takes 40 msec and the server can do 25 per second. For a multithreaded 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.

9. Statically associating a single thread with a light weight process is not such a good idea. Why not?

   Answer —

   Such an association effectively reduces to having only kernel-level threads, implying that much of the performance gain of having threads in the first place, is lost.

10. Strong mobility in Unix systems could be supported by allowing a process to fork a child on a remote machine. Explain how this would work.

    Answer —

Forking in UNIX means that a complete image of the parent is copied to the child, meaning that the child continues just after the call to fork. A similar approach could be used for remote cloning, provided the target platform is exactly the same as where the parent is executing. The first step is to have the target operating system reserve resources and create the appropriate process and memory map for the new child process. After this is done, the parents image (in memory) can be copied, and the child can be activated. (It should be clear that we are ignoring several important details here.)

11. Consider a process P that requires access to file F that 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?

    Answer —

    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.