Homework 2

Md Ali
CS 550: Advanced Operating Systems

October 7, 2020

Exercise 1. Consider a chain of processes $P_1, P_2, ..., P_n$ implementing a multitiered client-server architecture. Process P_i is client of process P_{i+1}, and P_i will return a replay to P_{i-1} only after receiving a reply from P_{i+1}. What are the main problems with this organization when taking a look at the request-reply performance at process P_1?

Proof. Performance will be particularly terrible for large values of n, due to the fact that each process must take into account previous processes. For instances, if we take into account P_1, we can see that the performance between P_1 and P_2 is directly determined by $n-2$ request-reply performance between the other layers. There is another fault here, since we know that the system seems linear in nature and this will lead to the problem if one machine is performing badly or is completely down then the entire system will have either a low or no performance entirely. □

Exercise 2. Describe precisely what is meant by a scalable system. Scalability can be achieved by applying different techniques. What are these techniques?

Proof. A scalable system is a system that can be scaled out to one or more domains without losing performance of the following: number of components, geographical size, or size of administrative domains. In regards with scalability, the three techniques that can be applied are distribution, replication, and caching. □

Exercise 3. If a client and a server are placed far apart, we may see network latency dominating overall performance. How can we tackle this problem?

Proof. This question solely depends on the client. We can rearrange the client in a way that it can do other work after it sent a request to a server. Another possible solution would be to replace the synchronous client-server communication with a asynchronous one-way communication. Lastly, we can divide the client side code into smaller parts that can run separately, which one part is waiting for the server to response, we can go ahead schedule another part. □

Exercise 4. Imaging a Web server that maintains a table in which client IP addresses are mapped to the most recently accessed Web pages. When a client connects to the server, the server looks up the client in its table, and if found, returns the registered page. Is this server stateful or stateless?

Proof. This server seems to be a stateless server. This is due to the fact that stateless designs is not that any information is maintained by the server on its clients, but whether that information is needed for correctness. For instance in this example, for whatever reason if the table is lost, the client and server can still properly interact as if nothing really happened. On the contradiction, if this server was a stateful then the interaction would only be possible after the server had recovered from a fault of some kind. □

Exercise 5. Consider a BitTorrent system in which each node has an outgoing link with a bandwidth capacity B_{out} and an incoming link with bandwidth capacity B_{in}. Some of these nodes (called seeds) voluntarily offer files to be downloaded by others. What is the maximum download capacity of a BitTorrent client if we assume that it can contact at most one seed at a time?

Proof. With the assumption that each client can contact at most one seed at a time, let's make the variable S as the seeds and C as the clients. The outgoing capacity would then equal to $S \cdot B_{out}$. With the introduction of the clients, the immediate download capacity will be $\frac{S \cdot B_{out}}{C}$. Now, as we know the download capacity of BitTorrent clients is dictated by the outgoing capacity, so as a result the total download capcity will $\frac{S \cdot B_{out}}{C + B_{out}}$ □

Exercise 6. 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?

Proof. You would be effectively remove transparency from the system. This is because, having different headers would mean that data passed from layer $k + 1$ down to layer k would contain both header and data, but later k cannot tell which is which. Having one universal header would effectively make changes in one layer visible to all other layers which is undesirable. □

Exercise 7. 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?

Proof. If call-by-reference is used, i will be called by a pointer to *incr*. This means that i will be incremented two times, which comes to the result of two. In regards to copy/restore, i will be passed by value twice. Both will be incremented meaning that both values will be one. Then both will be copied back, with the second cope overwriting the first one hence the final result will be one instead of what the call-by-reference value of two. □

Exercise 8. Describe how connectionless communication between a client and a server proceeds when using sockets.

Proof. In connectionless communication between a client and a server, they will both create a socket, but only the server binds the socket to a local endpoint. The server then does a blocking read call to wait for incoming data from any client to that specified socket. Once a client connects, the client simply does a blocking call to write data to the server. □

Exercise 9. Suppose that you could make use of only transient asynchronous communication primitives, including only an asynchronous receive primitive. How would you implement primitives for transient synchronous communications?

Proof. We can utilize the send primitive to send a message to the server using asynchronous communications, which will let the caller continuously poll for an response from the server. Now, if the local operating system stores any incoming message into a local buffer, then a work around would be to block the caller until it receives a specified signal from the operation system that a message has arrived, which in turn the caller will do an asynchronous receive. □

Exercise 10. 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 synchronous RPCs?

Proof. The first question is asking if executing an asynchronous RPC is the same as executing a normal RPC and that will be an affirmative no. This is due to that an asynchronous RPC return an acknowledgment to the caller. In an normal RPC which is an synchronous operation which requires the requesting program to be suspended until the results of the remote procedure returned. Hence, there are not the same approach. The next question is regarding in replacing the asynchronous RPCs with a synchronous RPCs which still wouldn't be the same as instead of having process run in the background, we would have to halt a process and wait for the completed task requested. □