Study Log

Tommy Leedberg

Table of Contents

[Summary 3](#_Toc529805078)

[Twenty Concepts 4](#_Toc529805079)

[1. Synchronous Programming 4](#_Toc529805080)

[2. Asynchronous Programming 4](#_Toc529805081)

[3. Thick Client 4](#_Toc529805082)

[4. Thin Client 4](#_Toc529805083)

[5. Code Migration 5](#_Toc529805084)

[6. Event Hub Architecture 5](#_Toc529805085)

[7. Distributed Hash Table 5](#_Toc529805086)

[8. Distributed Systems 5](#_Toc529805087)

[9. Open Systems 5](#_Toc529805088)

[10. Peer to Peer Architecture Style 6](#_Toc529805089)

[11. Bluetooth 6](#_Toc529805090)

[12. Asymmetric Encryption 6](#_Toc529805091)

[13. Network Address Translation 6](#_Toc529805092)

[14. Symmetric Encryption 7](#_Toc529805093)

[15. Hash Function 7](#_Toc529805094)

[16. Serialization 7](#_Toc529805095)

[17. One Time Pad 7](#_Toc529805096)

[18. Code 7](#_Toc529805097)

[19. RPC 7](#_Toc529805098)

[20. Proxy Server 8](#_Toc529805099)

[Forum Postings 9](#_Toc529805100)

[Say Hello Forum My Entry: 9](#_Toc529805101)

[Reply to Chris Williams Introduction: 9](#_Toc529805102)

[Reply to Chuan Chang in Hello: Hi John, 9](#_Toc529805103)

[General Homework Submissions: 9](#_Toc529805104)

[Provide automated grading scripts for submissions double check 9](#_Toc529805105)

[Network Labs Secreenshots? 9](#_Toc529805106)

[JokeServer: “Address Already In Use” 10](#_Toc529805107)

[Reply in Joke Server to “Maintaining State/Programming Form Question:” 10](#_Toc529805108)

[Reply in Networks Lab to “Some other network tools to check out” 10](#_Toc529805109)

[JokeServer Joke Server Architecture 10](#_Toc529805110)

[General CodeMigration: 11](#_Toc529805111)

[General CodeMigration - Reply 12](#_Toc529805112)

# Summary

Word count for final (Twenty Concepts Segment): 2476

Total word count (whole document, including beginning): 4668

# Twenty Concepts

## Synchronous Programming

Synchronous programming is when you write a method that blocks while waiting for a sub-method to return. In the world of single machine programming the sub-method could return in a matter of nanoseconds depending on what it is doing. However, in a distributed system there are external factors that can slow the call down from returning. Some of these factors include network latency and dropped messages. The later could end up causing your process to deadlock indefinitely, or until it is reset. One way to prevent such deadlocks if you must use synchronous programming is to build timeouts into the system.

## Asynchronous Programming

Asynchronous programming is the counter to synchronous programming. In this method type after a method will call a sub-method and then continue processing without waiting for the sub-method to return. For this to work affectively the program requires a way for the sub-method to call back into the calling method with whatever response it is supposed to supply and then for the calling method to be able to process that response. This means that programming asynchronously is better able to prevent deadlocks while waiting on a response from a called method. It also means that writing an asynchronous method is far more complex than writing synchronous ones because you have to program for what to do with the response whenever it comes, whether it be a couple nanoseconds or a couple minutes. The easiest way to write an asynchronous method is “fire and forget” where you call the method and then continue without actually caring what the response is.

## Thick Client

A thick client, also known as a heavy client, are clients that are fully (or almost fully) functional with or without access to their server. There are many varieties of thick clients the heaviest being all but the database being present in the client. This form of architecture has many advantages. Some of these advantages are that they are more responsive to users, they cause a lot less network traffic, and most if not all the processing can be done on the client without interaction with the server. Some of the major drawbacks are that it is very difficult to update the clients as they can function while disconnected from a network, they are much harder to administer, and versions between multiple clients can get out of sync.

## Thin Client

The opposite of a thick client is a thin client. This is where essentially all the processing is done of the server. The lightest client you can have is where the client is just a UI. Like the thick client this has many advantages and disadvantages. Some of the advantages are that they are far easier to update because they are typically much smaller than a thick client. Not only are they easier to update but they also typically require fewer updates since they are so small. Also, any exception handling is likely done on the server which means you have a centralized exception handler. Some of the drawbacks of this are that you are sending a lot more packets back and forth between the server and the client which can cause security vulnerabilities. Also, if the client cannot handle larger packet sizes the server will be forced to break up the packets so they can be re-assembled on the client.

## Code Migration

There are two major forms of code migration. Weak mobility migration and strong mobility migration. Weak mobility migration is the easiest form of code migration. The basic principle behind it is that when your code migrates from one location to another it simply restarts and reinitializes the process without maintain any of its state. Strong mobility migration is much more complex but allows for a couple different paths for the state to be migrated from location to another. Three different ways code migration can be handled is by stopping the current virtual machine, migrating the memory to the new machine, and then starting the new machine. Start up the new virtual machine and while the original is still running start migrate memory pages from the old to the new. Once all the pages have been migrated recheck and pages that were modified during the migration process and migrate them. Once all pages have been migrated shut down the original virtual machine. The final method is to start the new virtual machine and then to let it get the memory pages from the original on an as needed basis.

## Event Hub Architecture

The event bus(hub) architecture allows you to send messages without ever directly connecting your systems, services, or components together. The event bus facilitates multiple components connecting to it to either receive or send messages. Once a message is sent to the event bus any component that is listening on it will get that message. There are a couple different forms of event buses the simplest being similar in form to fire and forget when one system sends the message and if anyone is listening they get it if not the message is dropped. Another form of event bus would have a data store backend to persist messages that have come to it. A great example of this is in Microsoft’s cloud space. The component in azure called the event hub.

## Distributed Hash Table

Distributed hash tables are a cluster of machines working together sharing data. One example of these that we worked with in class is a blockchain system. In a Distributed Hash Table (DHT) any machine can join or leave the cluster at their leisure. Due to this face they are extremely scalable as any machine can join them.

## Distributed Systems

A distributed system is a collection of independent computers that to a user look and feel like one system. Distributed systems are far more scalable, flexible, and robust than a single computer system because. When you need more resources you simply add another computer to your system(also known as scaling). Distributed systems also allow you to have the computers that make up the system be anywhere in the world. One of the pitfalls of distributed systems in the complexity in which managing them becomes. Managing a single system is quiet easy but managing a collection of them that are not localized or even physical machines can become much harder.

## Open Systems

Open systems have many advantages one of the best of them being that anyone can use them and they don’t cost anything. In an open system you will typically have a publicly known interface that any user can build off of. Open systems have a handful of requirements and they are that the system has to be fully defined so that anyone using it is working with the same framework, the system has to be stable for a reasonable length of time, the interface must be publicly accessible, and the system must also not be under the control of any one entity.

## Peer to Peer Architecture Style

The peer to peer architecture style is most famously known as being used by bit torrent systems. In a peer to peer architecture any number of systems can share data with any machine in the cluster. Typically, you need to enforce good behavior in these scenarios as you don’t have control over the machines and some of them may consume more than they produce. The way the architecture works is as a machine wants a particular piece of data (say a cd album) it can request it from all machines connected to the cluster and will get pieces of the album from any machine that has it until the full album has been constructed. At the same time that it is getting these chunks of data it may also be sharing the chunks it has with any other machine that may want the same album.

## Bluetooth

Bluetooth is a means of wireless connection from one device to another. It is used most commonly with cell phones and/or wireless headsets. It has very low battery consumption which is mostly dependent on the range you want your device to be able to connect at. It uses a Spread Spectrum Technology which switches between the 79 channels available to blue tooth around 1600 times a second.

## Asymmetric Encryption

Asymmetric encryption uses two keys, a public key and a private key, to encrypt your data. These keys are interchangeable however the private key should remain private while the public key may be known by many. The way that the encryption works is you sign a document (for the use of this explanation a document is easiest to understand but it could be a string, an object, a file etc.) using either the private or the public key depending on the reason you are signing it. Then when you want to decrypt the document you would use the alternate key. An example of signing the document with a public key would be if you want to validate that someone is agreeing to the document. Once it’s signed with their public key the only one way to decrypt the document at that point is using the persons private key.

## Network Address Translation

Network address translation (or NAT) is a means to increase the number of useable IP addresses by way of giving a single IP as an entry point into your network. It acts as an intermediary between the internet and your private network. This means rather than have 10, 20, 30, 1000, etc IP addresses assigned for a company they get 1 IP address that is exposed to the internet. This is very important in the world today because the number of available IPv4. The way that NAT works is that when a request is made from a machine on the LAN to an external the request goes through the NAT router. The router takes the message and pulls off sender’s port and IP address. It then replaces the IP address with the external(real) IP address of the network and the port with a value in its table where it also stores the port the sending machine is using. This way when the response comes back it simple replaces the port with the senders port and the IP address with the senders IP address so the message can make it back to the correct location

## Symmetric Encryption

Symmetric encryption is the workhorse of encryption. It is the simplest and fastest form of encryption. In symmetric encryption there is only one key used to both encrypt and decrypt a document. It has a major drawback of distributing the key which is one of the reasons asymmetric encryption is more widely used.

## Hash Function

A hash function is a simple tool that allows you to map a large data set into a fixed data set size. For example, if you had a data set that was 256 bits long a hash function could be used to map that data down to 32 bits. Hashing data has many advantages. Some of those are the ability to validate your data as it makes it so that it is extremely hard to generate the same hashed data from multiple data sets. We used this in our block chain assignment. It also allows for duplicate detection and faster lookup in hash tables.

## Serialization

XML or the Extensible Markup Language is a common external data format alongside JSON or JavaScript Object Notation. With serialization you can use formats like these to transpose your object from a data type/class into a string object. Once it has been converted into a string it is then often passed from one process or machine to another. Both of these data formats allow you to define the object they represent with some for of named properties.

## One Time Pad

A One-Time Pad is a form of perfect encryption. To do it you create a pad of bits and take your message and then XOR it with the pad. This gives you a completely new string of bits and the only way to get back the original message is to XOR the new message with the same pad. While this is perfect encryption is has a similar drawback that symmetric encryption is also stuck with in that you have to distribute the pad somehow and once it’s distributed you can only use it once or it wont be affective anymore.

## Code

Code is similar to a One-Time Pad. While a One-Time Pad is perfect encryption, code is perfect security for up to a single use. In order to use a Code you must know about the messages ahead of time and also about the keys to each message, which can be a completely arbitrary word. Once you have those you create a table of keys and the messages and then when someone says a keyword you can look up the message for it.

## RPC

RPC or Remote Procedure Call is a protocol that a process can use to make requests to a process on another machine either on the LAN or on the internet. It has 3 different models. Request, Request and Reply, or Request, Reply, and Acknowledge. Request is the simplest of the 3 different models it’s essentially a fire and forget from one process to another where the request is sent and the processing making the request continues on without caring about the response. The Request Reply model is similar however in it the process making the request also requires a response from the process it has sent the request too. The final model, Request, Reply, and Acknowledge, takes this one step further and requires that the process that is making the request also sends an acknowledgement to the sub process that it got the sub processes reply. While this is the most process/time intensive operation it also guarantees delivery of a message from one process to the next.

## Proxy Server

A proxy server handles network requests between a client and a server allowing the client and the server to know nothing about each other but still interact. It will intercept requests and replies coming in either direction and forward them off as needed. It is similar in concept to the Proxy Pattern where not only will it forward off a request from the client/server it may also modify that request. An very simple example of why it would want to modify the request is say that a client can only send out requests in a JSON format but the server can only receive requests in an XML format. The Proxy may have the ability to take in the JSON and translate it to XML before sending the request off to the server and vice versa on the way back.

# Forum Postings

### Say Hello Forum My Entry:

Hello everyone,

       My name is Thomas Leedberg but I've gone by Tommy for just about my whole life. I'm 31 years old and am a transplant from Massachusetts and have I've lived in Chicago for a little over a year and a half now. I received my undergrad in Computer Science at the University of Massachusetts Lowell, but prior to that I was attending the University of West Virginia. I currently work for Schneider Electric and I've been with them for a little over 8 years now. This is only the second course that I've taken at DePaul and so far I'm enjoying the school. In my spare time I like to travel, I've been a lot of places but there are so many more I want to see and experience. I love being a part of cultures that are not my own. I also enjoy being outdoors, growing up in northern Massachusetts there is plenty of opportunity to go hiking and camping in the mountains of New Hampshire. I also love music and going to music festivals/concerts.

### Reply to Chris Williams Introduction:

Hi Chris,  
        How's it going? I live pretty close to uptown, I was in Ravenswood and now I'm over on Irving Park. I'm a pretty big gym advocate myself. I spend a lot of my free time at Uptown Fitness. I've always had some interest in mobile development, I've only looked into it lightly though most of my focus ends up being cloud based. I have done a couple small projects using cordova for work hackathons that I thought was interesting. I do like that you can touch and feel mobile applications in a way you can't really do with other platforms.

JokeServer and Inet  
Just a suggestion for if you're having trouble getting things running correctly or understanding what's going on in the inet server/client code. In IntelliJ you can host both the server and the client in a debug session simultaneously. I always find it's a lot easier to understand something when I can step through it.

### Reply to Chuan Chang in Hello: Hi John,

Congratulations on graduating this quarter! What do you plan to do after you're done?

General Homework Submissions:  
Hello,

   Is there a file name format we should give our zip files when we submit our programs? I was trying to find it in the docs/content but I couldn’t find anything specific.

### Provide automated grading scripts for submissions double check

Eh what production system" It works on my machine" lol.

### Network Labs Secreenshots?

Some of the network lab questions require you to provide screen shots, but we're only supposed to be submitting our answers in the HTML check list which doesnt have images unless we host them somewhere and provide the link for them. So my question is are we still supposed to provide these screenshots and how?

### JokeServer: “Address Already In Use”

You can also download the free windows tool [TCPview](https://docs.microsoft.com/en-us/sysinternals/downloads/tcpview" \t "_blank)to see what ports are in use and kill it that way, that's how I do it when my program doesn't shut down properly so the port remains open.

### Reply in Joke Server to “Maintaining State/Programming Form Question:”

The short answer is what you are proposing will work just fine. If it's a static final object you wont need to pass it around though, just access it. My suggestion would be to use a Dictionary or Map object over an ArrayList if you want to implement it this way though, then you can have the keys be the user id's and the values be your jokes/proverbs. This would make for much faster lookup.  
  
There is one thing to keep in mind. If you do the above and I read your post right you're structure would look something like the following:  
  
[{ "id1": [ JA, JB, JC, JD]},  { "id2": [ JA, JB, JC, JD]} ,  { "id3": [ JA, JB, JC, JD]} ]  
  
This is good for a small set of users, but imagine having 50-100k users. You're now keeping a list of every joke for every user and could end up being a resource issue.  
  
There are a lot of ways around this but if you want to keep a similar structure one possible option would be to create 2 objects rather than 1. One to maintain the jokes and one to maintain the users. The jokes could be a mapping of every joke you have and an ID for that joke and then the users would be a mapping of user ids and a list of joke objects containing the jokes id and if it's been heard yet. It adds a little more overhead in processing for lookup up the jokes and also determining if they've been heard or not but you save a ton of space by not replicating the jokes. Also, if you want to change a joke you only have to do it in 1 location.

Reply in Networks Lab to “Some other network tools to check out”  
I've actually used both PostMan and Fiddler a lot. One thing I like about fiddler is you can script a lot of stuff into it. For my job part of our platform is a REST API that requires a auth0 bearer token in order to allow you to have access. What I ended up doing was changing fiddler so that when a request was made to any url that contained the platforms url it automatically made a secondary request to get Auth0 to get bearer token and then changed the request that was going to our REST api to add the authentication header. It tooks a little figuring out at first because the fiddler scripting language is kinda funky about ordering but it ended up being a major time saver. I've also done something similar in POSTMan by chaining requests together, although I admit I've used it a lot less though.

JokeServer Joke Server ArchitectureI think we're all on the same page about wishing to be able to use multiple files rather than keep everything in one file. Even with folding it's still a really big jumble. It does make me appreciate all those classes way back when I started that tell you to break things out though lol.  
  
For my implementation I also did a pretty thin client and heavy server. My original plan was to do a mix mode implementation where a small amount of data was stored on the server and the rest was stored on the client but some of the requirements for the assignment made me change my mind in that.  
  
Anyway, onto the actual implementation:  
  
Rather than do a separate class implementation for both joke and proverb mode I kept it pretty simple and did an enum that was then used to determine what the server was supposed to serve up to the client. I would have likely followed your decision to do this in separate classes had there been more than just strings being returned though.   
  
I created class for storing Joke and Proverb information, named JokeProverb for lack of better naming ideas. It was a pretty basic data structure that contained a body and an id originally it contained more data than that but i pulled it out. I could have just used another HashMap here for the jokeId/Key as well. 

This data structure was stored in two arrays, one for Jokes and one for Proverbs, it allowed me to keep 1 single copy of every joke and proverb rather than a copy in each user set. Then I created 2 LinkedHashMaps with the following signature LinkedHashMap<String, LinkedHashMap<String, Boolean>>. The outer map had a key for the UUID of the user( in my case i just used the users name ) and a value of a second LinkedHashMap that had a key for the joke or proverbs id and a boolean value indicating if it had been heard or not yet. I had originally thought to include a flag in the JokeProverb class for if it had been heard or not but this is actually a faster lookup and a bit easier to implement.  
NOTE: Something I actually learned in this project was that you need to use a LinkedHashMap if you want order to maintain. I was originally using a HashMap and when i went to randomize the jokes/proverbs they were constantly being reorganized by key which was not helpful.  
  
Communication between the Server and the Client was done through a data structure I created called a ServerRequest. It had members for a UUID and a String request, I had originally thought we would need to send a true request to the server not just a hit return to get the next joke, this way works for that as well but it's just a bit more extensible if I need it for another project later.  
  
I had to build in some basic serialization/deserialization into the class because I was only passing strings back and forth from the client to the server. Typically, i'd have just a 3rd party library for this though.  
  
When a request came in I would call a helper method that then gets the next joke or proverb by looping through the list of proverbs or jokes associated with that user and finding the first key( the id of the joke/proverb) who's value is false, this bit of work was kind of ugly and could probably use some refactoring, once it has the key it will do a lookup in the Jokes/Proverbs array and then return that joke. Finally there was a string builder method that put everything together and returned it to the client which simply displayed it.  
  
Some other ideas i had originally played around with for this project was opening multiple sockets for the adminclient, client, and server and having a separate socket for send and receive on each. This would have allowed for true asynchonicity. I was a little confused about the requirements though and wasnt sure if we could open sockets out side of the spec so i dropped this idea. The other thought was to ObjectStreams to allow for serialization, i dont remember why i dropped this though.

### General CodeMigration:

I thought this topic was kind of interesting mainly in the fact that at my current company we are working on an Azure Cloud Platform that is built on top of the [Service Fabric Cluster](https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-deploy-anywhere). This also happens to be the platform that HALO the video game is built on.  
  
At a 1000 mile high view the Service Fabric Cluster is a cluster of nodes that run microservices. It has the capability of migrating which service is run on which node based on a set of rules that by default are determined by the resource availability on any given node at any given time. It has 2 service types, one is Stateless one is Stateful. For this outline I'll just talk about stateless services because they are far simpler. A statless service is exactly what it sounds like, a service that doesn't maintain state( Think of it like an API Gateway or Event Processor). In the SF Cluster world you can specify how many instances of this service you need then it would balance those instances across the nodes potentially shutting down the service in one node and then starting it up in completely different node based. This is pretty simple for stateless services because you dont have to worry about anything in memory.   
  
A stateful service is a little bit of a different animal but they way they get around it is by maintaining the state of the service in multiple locations, called replicas, then when the service has to move it already has all the state that the original node had when it starts up( Again this is the most basic terms ).

### General CodeMigration - Reply

If you have access to Pluralsite this is a really good lecture on SF. It also explains where it falls in between container services and functions.  
<https://app.pluralsight.com/library/courses/azure-service-fabric-programming-models/table-of-contents>