6.033 Lecture 4
Operating Systems Part IV

Americo De Filippo

May 16, 2023

We are going to get enfored modularity via client-server organization. The last time we have seen how the linker works with modularity, he perform some operation with more programs and outputs some result that is the exec file. It implemented name mapping with Table mapping, Search throught contexts.

1 Interface between modules

The different modules interact with one another via **procedure contract**.

When you call a procedure The caller and callee interact with each other via returning arguments in the registers. The important thing is that this contract has to obej to a principle called **stack discipline**, the callee has to leave the stack has it has founded when been called.

how can the discipline be violeted The Callee corrupts the stacks, Callee crashes. This is called fate sharing, so this modularity is **soft**, cause there is not shielding between called and callee.

2 Client and Server organization

Taking as example the previus situation, the Client is the caller and the Server is the callee. The basic abstraction is that the two are running on different machine so they are in some way indipendent, for this reason for make them communicate we have to implement some kind of way of communication between the two. We are going to have a Server-Client way of communicating (take TCP as example), in this way the two are indipendent, for this reason the fail of one do not interfere with the stack of the other, this we call **enforced modularity**. One problem with this implementation is the case which we want to decide what something exacly happends, for example we are not going to know whether a client is taking long time processing something or is crashed. This is one of the trade-offs that we have to make for have a more secure system.

3 Enforced modularity

For solving the problem of knowing what is happening, enforeced modularity comes with a timer called **Watchdog timer**, the server is going to have an idea of what is happening at the client, if the server does not return in a precise amount of time the server is going to ask himself whether the client has failed or is computing. Another nice proprety of this organization which is that clients geeet enforced modularity, the idea is that many client can access the same server but they are indipendent from each other. So you can you the server to has a **trusted intermidiary**, the clients do not trust each other but all of them trust the server.

4 How to implement this stuff

There are differnt ways of implmenting, all of them are different mainly by ways that they send messages for making the communication. A pretty standard way of implementing is called **remote procedure call (RPC)**. An RPC implementation that is used is: **XML RPC**

how to we solve the problem of dead vs slow? We have three ways of communicating that we are gonna call Exacly-once semantic (one request-one return), At-least once semantic (meaning that we are gonna try until we get a success). The latter is good when the server has idem-port semantic (when you run an operation more than once you get the same answer). The third is called At-Most once (if you dont get a responce you will handle the fail directly, if you get a responce good!)

async procedure call for performance feature we are gonna have an async communication (like in TCP), let's say that the client makes some requests it will not be there and wait instead it will go on with other task and it will handle the responce along with a request number (sequence number in TCP) that enables the two to be even more decouple.

Intermediary The client could send a request to the server when he its down, in this case there will be some kind of intermidiary that will handle the requests at behalf of the server that then is going to handle and compute the request.