Distributed Systems

Chapter 4

Designing of a Client/Server-Software

4.1 The Client/Server Model

- The design of distributed applications today is often based on the so-called client/server model. In this interaction model, the units communicating with each other are divided into clients and servers. A **server** is a <u>service provider that provides a specific functionality.</u> The **client** is a <u>service consumer who requests the execution of a specific service from a server</u>. Because the client and server run independently of one another - often on different computer systems - communication between the two must be **synchronized**. Otherwise, data exchange <u>may never occur between them.</u>

Rendezvous problem

- The problem addressed is also known as the rendezvous problem. If you start the client and server on different machines in rapid succession, the following processes can occur.
    1. The client starts first. It tries to contact the appropriate server. If this has not yet been started, the client sees it as finished. The server is also started immediately after the client. If the server cannot find its communication partner (this has already been terminated), it also sees that it is terminated. No interaction can take place.
    2. The server starts first. He tries (just like in the first case) to find his communication partner. Unfortunately, the client is not yet active, which is why the server is shut down. The client started immediately after the server cannot find its communication partner either and is therefore terminated.

- The rendezvous problem arises because both sides are actively trying to find their communication partner.
- To solve the problem, only one of the two sides is allowed to actively establish communication. The other side must wait passively for a request to arrive. The definition of client and server introduced must be expanded to include this aspect.

Definitions:

- Server: A server is a service provider that provides a specific functionality for a client in the form of a service and passively waits for a client to make a request to it.
- Client: A client is a service consumer who actively requests a service from a server and then waits for the server to provide the requested service.

Servers are often more complex than clients. In addition to communicating with the client and providing the specific service, they also have to carry out other tasks:

- Authentication: The server must be able to uniquely identify the client requesting a service.
- Authorization: Once the identity of a client has been determined unequivocally, it must be checked whether it is allowed to request the desired service.
- Data protection: Clients should not be allowed to freely access personal data.
- Data security: All information offered by a server should be protected against manipulation and destruction.
- Protection of the operating system: The server must prevent a client from accessing any resources of the operating system and being able to misuse them.

There are a number of advantages in favor of the wide distribution and use of the client/server model. The main advantages of the client/server model are:

- The client/server model can be used in heterogeneous environments. It therefore enables communication despite different computer and operating systems.
- The terms client and server have prevailed in theory and practice. These are fixed terms that enable clear communication among IT professionals.
- Because of the clear separation between service provider and service user, the interaction between the two sides is clearly defined.
- The client/server model can be used not only for software development but also for hardware design. For example, when developing systems for controlling production plants, one hardware component can act as a client (e.g. sensor that measures temperature) and another as a server (e.g. components for processing the measurement data)

The disadvantages of the client/server model are:

- Although the client/server model supports heterogeneous environments, it does not hide them from the programmer (no transparency). The programmer must consider heterogeneity when implementing distributed applications.
- In practice, there are often applications that do not allow for a clear separation of client and server. For example, a server can simultaneously act as a client for another server.
- The client/server model is relatively old and inadequate for many modern forms of communication. New communication models have to be developed here, such as remote method calls or message-oriented communication.

- The messages exchanged between client and server are also generally referred to as **requests and responses.**
  - After the server has started, it passively waits for a request to arrive.
  - As soon as a request arrives, it provides the requested service and sends a response back to the client.
  - The process described is shown in Figure 4.1 (according to [Tanenbaum 2003] p. 62).

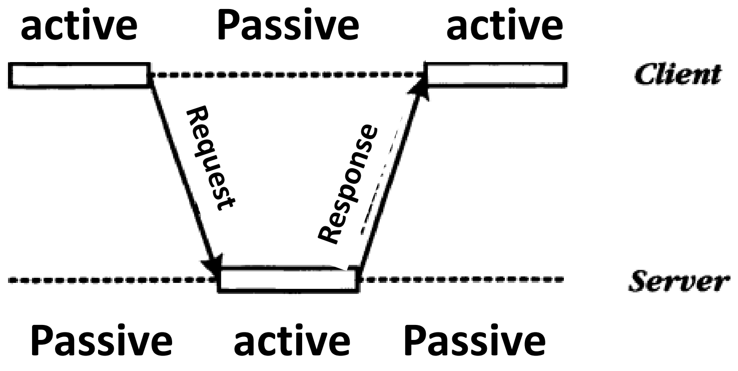


Figure 4.1: Structure of the client/server communication

4.1.1 Client Design According to [Comer 2001],

three different aspects must be distinguished when designing clients:

- Parameterizable vs. non-parameterizable clients
- iterative vs. parallel clients
- connectionless vs. connection-oriented clients

Parameterizable vs. non-parameterizable clients:

- First of all, the question arises as to whether the client should be freely configurable or not.
    - A freely configurable client makes it possible to select the server to be addressed via parameters - for example command line parameters - and to set other communication parameters.
        - The main advantage of such a client is the possibility of establishing a connection between different servers. If, for example, a server cannot be reached due to network problems, the request can simply be sent to another server.
    - A client that cannot be freely parameterized always communicates with the same server. The user can only start the client but cannot provide any command cell parameters. Such clients offer a high degree of security because they only allow communication with a server classified as trustworthy.
    - In practice, one often finds freely parameterizable clients, since a connection to several servers - selectable by the user - is necessary.

If you implement a freely parameterizable client, the programmer must specify where the client gets its parameters from. There are several possibilities here:

- The client evaluates the transferred command line parameters.
- The user can pass the configuration parameters to the client via a GUI.
- The parameters are stored in a local configuration file that is read by the client.
- Relevant parameters are made available to the client via a directory service (e.g. Lightweight Directory Access Protocol (LDAP) / X.500).

iterative vs. parallel clients

- The second important design aspect is the question of an iterative or parallel implementation of clients.
- An iterative client has only one thread of control, i.e: it can only do one specific task at a time.
- A parallel client, on the other hand, has several execution fields and can therefore process several tasks (quasi) in parallel.

A parallel client is advantageous if the following tasks are to be fulfilled:

- The client should be able to interact with several servers at the same time. For example, a client can send a request to multiple servers at the same time. The response from the server that responds first is used, or the responses provided can be averaged for greater accuracy.
- During the client's communication with the server, the user wants to make further entries. The client must therefore be able to process the communication and the inputs (quasi) in parallel.

connectionless vs. connection-oriented clients:

- The third and final design aspect when implementing a client is the question of the transport protocol used.
- In today's distributed systems, the connection-oriented TCP (Transmission Control Protocol) or the connectionless UDP (User Datagram Protocol) are predominantly used as the transport protocol.
- Accordingly, a distinction is made between connectionless and connection-oriented clients. Whether a client works connectionless or connection-oriented depends very much on the server. The client must always use the same transport protocol as the server.

4.1.2 Design of Servers

- When designing Servers, there are similar issues to be discussed as when designing clients. The most important design aspect is whether the server should work iteratively or in parallel.

iterative server

- An iterative server processes incoming requests in the order in which they arrive in a strictly sequential manner,
- i.e. according to the first-come-first-served (FCFS) principle, which is also known from operating systems.
- With an iterative server, requests that require a relatively short period of time to be processed may have to wait for requests that take a long time to process.
- From a user's point of view, such a way of working is unsatisfactory. Intuitively (and rightly so), every user expects a "small" request to be processed quickly.

parallel server

- A parallel server, on the other hand, processes several requests (quasi-)parallel,
- i.e. the requests are processed in parallel (several processors available) or according to a round-robin process (only one processor available). In order to implement a parallel server, the operating system must offer some form of parallelism. Two common approaches to creating parallelism are using multiple processes or using multiple threads (see Chapter 2).
- It should be mentioned here that there are also other methods, such as parallelism via asynchronous I/O.

Response Time and Processing Time

- The decision whether to implement an iterative or parallel server essentially depends on the response time from the client's point of view and the processing time from the server's point of view.
- The processing time is the time that a server needs to process a request. It is largely determined by the scope of the request.
- The response time is the time that elapses from the client's point of view until the corresponding response to a request arrives.
- The response time consists of the processing time, the transmission time of request and response via the connection network and the waiting time in buffers at the server (Figure 4.2).
- The answer is then always longer than the processing time. An iterative server processes requests strictly sequentially. If several requests arrive at the same time, only one will be processed directly. All others are saved in a queue. This naturally increases the response time. If the response time becomes so long that it is no longer acceptable for the user, you should switch from an iterative to a parallel server.

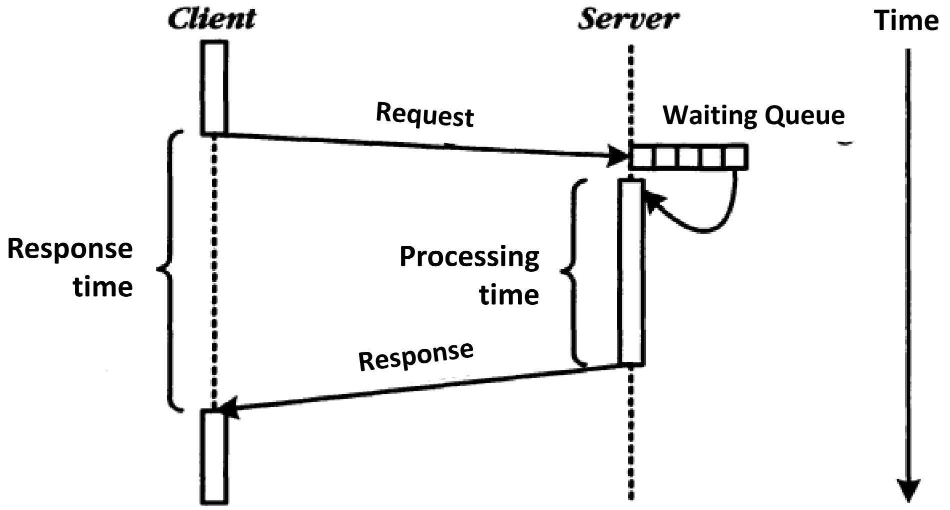
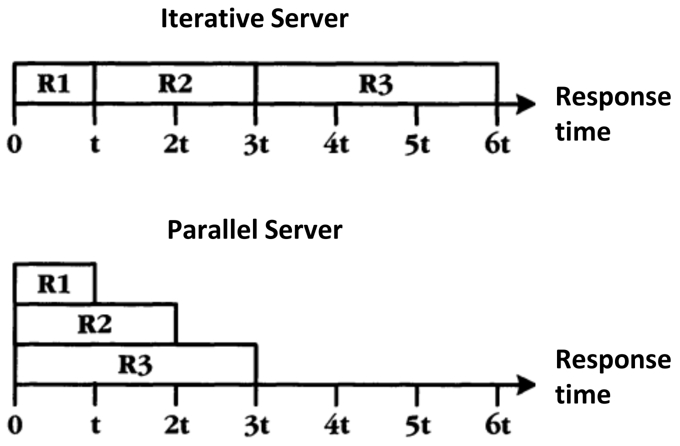


Figure 4.2: Response and processing timeline

Figure 4.3: Response time with iterative and parallel server



- Figure 4.3 shows how the response time differs between an iterative and a parallel server (the transmission of the request and response over the connection network and the creation of parallelism are not taken into account).
- Assuming at time 0, three different requests - Request 1 (R1), Request 2 (R2) and Request 3 (R3) - arrive at the server at the same time. R1 has a processing time of t time units, R2 of 2t time units and R3 of 3t time units. If one chooses an iterative server design, the response time for R1 is t time units, for R2 3t time units and for R3 6t time units.
- The average response time is therefore (t + 3t + 6t / 3) = 3.33t time units.
- A parallel implementation of the server results in a response time of t time units for R1, 2t time units for R2 and 3t time units for R3.
- The average response time is only (t + 2t + 3t) / 3 = 2t time units. The parallel processing of the queries has improved the individual and average response times of the clients. On the server side, this time advantage is paid for by an additional overhead for generating and managing parallelism.