PROGRAMACIÓN DE SERVICIOS Y PROCESOS

TÉCNICO EN DESARROLLO DE APLICACIONESMULTIPLATAFORMA

Modelos de comunicaciones
Comunicación Cliente/Servidor

Arquitectura cliente/servidor

A nivel general, cuando se trata de aplicaciones de comunicaciones entre equipos, el modelo más utilizado es el **modelo Cliente/Servidor**, debido a que ofrece una enorme **flexibilidad, interoperabilidad y estabilidad** para acceder a recursos que están centralizados en un ordenador. Este modelo apareció por primera vez en los años 80, resolviendo uno de los grandes problemas de la época: un ordenador necesita un servicio que provee otro.

Aplicaciones que usan el modelo Cliente/Servidor:

- **Outlook** es un **cliente** de correo que sirve para leer los correos del servicio hotmail.
- **WhatsApp** o **Telegram** son **clientes** de servicios de chat, vídeo, audio.
- **Firefox** es un **cliente** web, **Chrome** también ("navegadores").
- **Internet Information Server** y **Apache Web Server** son **servidores** web.

Este modelo está **compuesto por los siguientes elementos**:

importante memorizar:

- **Cliente**:

  Es quien va a interactuar con el **usuario**, realizando las peticiones que sean necesarias al servidor y **mostrando** a posteriori los **resultados** a la persona con la que está interactuando. Las principales **funciones** que va a cumplir el cliente son:

  - Interactuar con el **usuario**.
  - **Procesar** todas las **peticiones** para comprobar que sean **válidas**, evitando enviar peticiones malintencionadas al servidor.
  - **Recibir** los **resultados** que le envía el servidor.
  - Dar un **formato correcto** a los **datos** recibidos para mostrárselos al usuario.

- **Servidor**:

  Este va a ser quien provea de **uno o varios servicios** a los clientes.

  Las **funciones** que deberá desempeñar son:

  - **Aceptar las peticiones** de los clientes.
  - **Procesar** en orden las **peticiones** de los clientes.
  - Dar un **formato correcto** a los **datos** enviados a los clientes.
  - Realizar **validaciones** de **datos**.
  - Debe asegurar que la **información** sea **consistente**.
  - Mantener **seguro el sistema**.

Variantes de modelo Cliente/Servidor

- ***1*** *cliente y* ***1*** *servidor.*
- ***1*** *servidor y* ***muchos*** *clientes*
- ***muchos*** *servidores y* ***muchos*** *clientes.*

Todo dependerá de los ***servicios*** *que se necesiten proporcionar, ya que esto condicionará la* ***complejidad*** *del propio sistema.*

Características modelo cliente/servidor

Características básicas:

• Existe una **combinación del cliente con los usuarios**, ya que estos son los que interactúan entre sí. También existe una **combinación entre el servidor y los recursos** que se van a ofrecer, es decir: es el **cliente** el que deberá proporcionar una **interfaz gráfica** al usuario y será el **servidor** el que permita el **acceso remoto a los recursos** que se compartirán.

• Las **operaciones** que podrán realizar los **clientes** y los **servidores** van a necesitar, claramente, unos **requisitos totalmente diferentes** en lo que respecta al tratamiento de información, por razones obvias, y será el **servidor** quien realice todo el **trabajo de procesamiento**.

• Existe una clara **relación entre el cliente y el servidor**, pudiéndose ejecutar en un **mismo ordenador o** en varios ordenadores **distribuidos en red**.

• Un servidor podrá dar **servicio a uno o muchos clientes**, realizando siempre las operaciones necesarias para que las **peticiones se ejecuten en orden** y que ningún cliente se quede sin atender.

• Los **servidores** van a tener un **rol pasivo** en este modelo, ya que estarán esperando a que los clientes les envíen sus peticiones, mientras que los **clientes** tendrán un **rol activo** en el modelo, ya que serán los que envíen las peticiones cuando las necesiten.

• La **comunicación** entre cliente y servidor **únicamente** se puede realizar a través del **envío de mensajes**, tantos como hagan falta.

• Los clientes podrán utilizar **cualquier plataforma** para conectarse a los servidores.

Cantidad necesaria de Servidores

Las aplicaciones que usan el modelo cliente/servidor deben ser capaces de dar servicio a todos los clientes que lo necesiten, da igual que sea uno o que sean millones.

Deben dar soporte a todos los clientes que lo necesiten y, para ello, **probablemente deberemos tener más de un servidor**. El uso de varios servidores en la arquitectura cliente/servidor **es la norma que predomina**, ya que es inconcebible no poder dar soporte a los clientes.

Tenemos, incluso, casos en los que los propios **servidores van a actuar como clientes** que piden soporte a otros servidores para **obtener un servicio del que no disponen** y poder dar soporte a sus clientes.

Ventajas e inconvenientes del modelo cliente/servidor

Ventajas:

• Utiliza **clientes** que son **ligeros**, lo cual significa que no van a necesitar un hardware muy potente para funcionar, sino todo lo contrario, ya que quien realizará todos los **cálculos** necesarios será el **servidor**.

• Muestra una gran **facilidad** en la **integración entre sistemas** que comparten la información, lo cual va a permitir que dispongamos de unas **interfaces gráficas muy sencillas** de usar para el usuario.

• Predominan las interfaces gráficas que son interactivas, por lo que los usuarios pueden **interactuar con el servidor**. Haciendo uso de interfaces gráficas conseguiremos **transmitir únicamente los datos**, consiguiendo una **menor congestión en la red**.

• El **desarrollo y mantenimiento** de este tipo de aplicaciones es relativamente **sencillo**.

• Este modelo es **modular**, lo que implica que va a permitir la **integración de nuevos componentes** y tecnologías de una forma muy **sencilla**, permitiendo un crecimiento **escalable**.

• Se tiene un **acceso centralizado** de los recursos en el servidor.

• Los clientes pueden **acceder** de forma **simultánea al servidor**, compartiendo los datos entre ellos de una forma **rápida y sencilla**.

Inconvenientes:

• El **mantenimiento** de los sistemas es un poco **más complejo**, ya que interactúan **diferentes hardware y software**, lo cual puede ocasionar múltiples **errores**.

• Se tienen que **controlar** absolutamente todos los **errores** posibles del sistema.

• Se tiene que garantizar una **buena seguridad** en el sistema. Si cae el servidor, cae el servicio para todos sus clientes.

• Se debe garantizar la **consistencia** de la información transmitida.

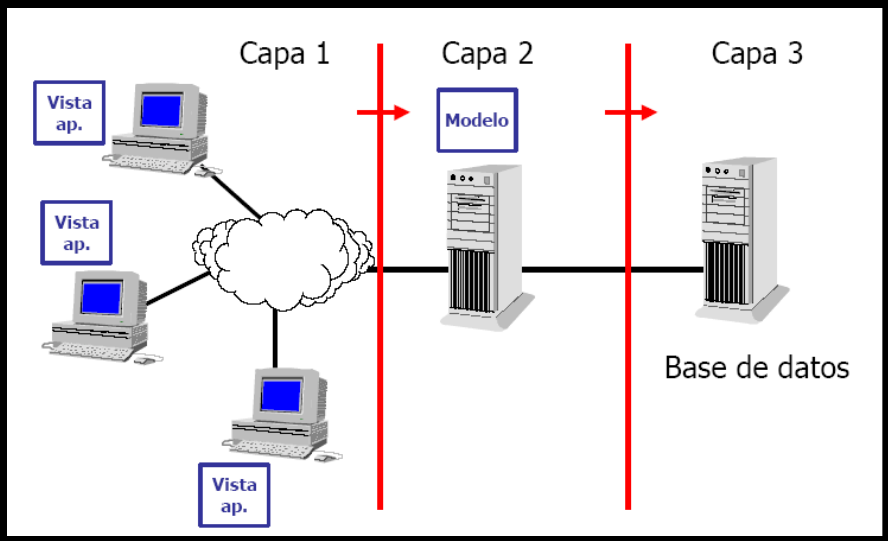
Modelos cliente/servidor

Según el número de capas (triers) que tenga, lo vamos a poder clasificar de las siguientes formas:

• Sistema de **1 capa**: Vamos a tener este tipo de modelo cuando tanto el cliente como el servidor se encuentren en el **mismo ordenador**. Este sistema **puede no llegar a considerarse** como un modelo **Cliente/Servidor**, ya que, en realidad, **no hay una conexión de red** para la comunicación del cliente y del servidor.

• Sistema de **2 capas**: Este es el **modelo tradicional** en lo que se refiere a Cliente/Servidor y es el que vamos a utilizar en nuestras aplicaciones. Aquí existen un cliente y un servidor totalmente identificables. Este modelo tiene un gran inconveniente: tiene una **bajísima escalabilidad**, pudiendo llegar a **sobrecargarse** si hay **muchos clientes** o si los clientes son pocos, pero mandan una **gran cantidad de peticiones** al servidor.

• Sistema de **3 capas**: Este modelo fue diseñado para mejorar las limitaciones del modelo de 2 capas, añadiendo, para ello, una nueva capa con un **nuevo servidor**, que puede ser o bien un **servidor de aplicaciones**, el cual se encargará de **interactuar** con los clientes y de enviar las peticiones al servidor de datos; o bien un **servidor de datos**, que será el encargado de recibir las peticiones del servidor de aplicación con lo requerido por los clientes para **resolverlas**. Sería posible agregar **tantos servidores de datos como necesitemos** sin problema alguno para mejorar el **rendimiento** del sistema.



• Sistema de **n capas**: Este modelo fue diseñado para mejorar las limitaciones del modelo de 3 capas y, a partir de aquí, podremos añadir **tantas capas de servidores como sean necesarias**, permitiendo **separar funcionalidades** y mejorar el rendimiento del sistema de una forma considerable. Un ejemplo de este tipo de sistemas son los conocidos programas o **aplicaciones P2P**, como **Napster, Gnutella, KaZaA, eMule, BitTorrent, BitCoin**...

Comunicación en el modelo cliente/servidor

El modelo Cliente/Servidor tiene un sistema de comunicación íntegro a través de intercambio de mensajes. Estos mensajes contendrán toda la información necesaria para poder ser identificados y enviados a donde se les solicite.

En este sistema, la pérdida de mensajes es un factor importantísimo a tener en cuenta, ya que podríamos estar enviando un fichero que ocupa 3 MB en varios mensajes. Si se pierden unos cuantos por el camino, porque no sean bien redireccionados, por ejemplo, el resultado de la reconstrucción del fichero de 3 MB estaría corrupto, por lo que quedaría inutilizable. Para evitar que se pierdan mensajes en este tipo de comunicaciones, **se enviará por cada mensaje recibido** un mensaje de **confirmación al emisor**. Estos son los **mensajes** conocidos como **ACK** o **acknowledgement** (reconocimiento) en inglés. En el caso de que **un mensaje no llegue** correctamente al receptor o se **demore mucho** tiempo su llegada (se considerará que se ha producido algún **error**), **no se enviará** el mensaje de ACK al emisor; en consecuencia, este entenderá que el mensaje no ha llegado, por lo que lo **volverá a enviar** y **esperará** de nuevo la llegada de su **ACK**.

El método de los envíos de ACK es muy efectivo y útil, pero la verdad es que es bastante lento, ya que deberemos esperar absolutamente todos los ACK de todos los mensajes que enviemos, además de que se **producirá más tráfico** en la red, con la posible congestión que puede implicar. Esto se puede **mejorar** de una forma muy simple, ya que, si permitimos que el receptor **envíe varios ACK a la vez**, en lugar de enviarlos de uno en uno, conseguiremos una **mejora** considerable en la **transmisión** de la información. Con este método, el emisor podrá enviar **paquetes de X mensajes** y esperará la **llegada** de sus correspondientes **ACK en grupo**. Debemos tener en cuenta que con este método de enviar varios paquetes y recibir varios ACK, puede suceder que tanto los paquetes como los ACK lleguen **desordenados**, o que, incluso, alguno de ellos **se pierda** por el camino, siendo necesario llevar un **control de errores** en esta técnica. Para gestionar estos errores, podremos utilizar un **vector de ACK**, compuesto por **pares** que indican el **número del mensaje enviado** y si se ha **recibido o no** su correspondiente **ACK**.

Procedimiento para programar una aplicación con el modelo Cliente/Servidor, por parte del **servidor**:

1. Decidir un puerto: El **puerto** elegido será utilizado tanto por el **servidor** como por los **clientes**.
2. Esperar peticiones: Será el servidor quien quede escuchando permanentemente que se conecte un cliente. Una vez **conectado** el cliente **crearemos un Socket** para la comunicación entre cliente y servidor.
3. Envío y recepción de información: Una vez conectados el cliente y el servidor por medio de un Socket deberemos **obtener los flujos** de escritura y lectura necesarios.
4. Cerrar el Socket: Una vez que se ha terminado la trnasmisión de información entre el cliente y el servidor se deberá cerrar el Socket **para evitar errores** indeseados. Este es un de los

pasos más importantes, ya que si se nos olvida cerrar el Socket nuestro programa va a estar sujeto a errores indeseados.

Procedimiento para programar una aplicación con el modelo Cliente/Servidor, por parte del **cliente**:

1. Conectarse con el servidor: El cliente deberá conectarse al servidor mediante su **IP o nombre y un puerto** previamente **conocido (el mismo que use el servidor)**. Una vez que se haya realizado correctamente dicha **conexión** se creará el **Socket** para la comunicación.
2. Envío y recepción de información: Una vez conectados el cliente y el servidor por medio de un Socket deberemos **obtener los flujos** de escritura y lectura necesarios.
3. Cerrar el Socket: Una vez que se ha terminado la transmisión de información entre cliente y servidor se deberá cerrar el Socket para **evitar posibles errores** indeseados, al igual que lo debía de cerrar el servidor.

Diseño de una aplicación cliente/servidor

El modelo cliente/servidor tiene dos partes: la del cliente y la del servidor.

Cuando se diseña una aplicación de este tipo, lo más normal es que sea bastante compleja, por lo que habrá **dos equipos de desarrollo**, uno para desarrollar la parte del **servidor** y otro que se encargue de desarrollar la parte del **cliente**. Con este tipo de desarrollo, se gana mucho tiempo, ya que lo único en lo que tendrán que **estar de acuerdo** será en la **forma en la que se comunican** los clientes con el servidor, si hay algún **mensaje de inicio, de confirmación, de parada**... y de situaciones sobre comunicación por el estilo.

Sería recomendable seguir este tipo de diseño, aunque la aplicaióc sea simple, **definiendo cada uno una parte y poniéndolas en común**.

Diagramas de estados

Cuando utilizamos el modelo Cliente/Servidor para programar aplicaciones, uno de los **fallos** principales con respecto a la **seguridad** que nos vamos a encontrar son:

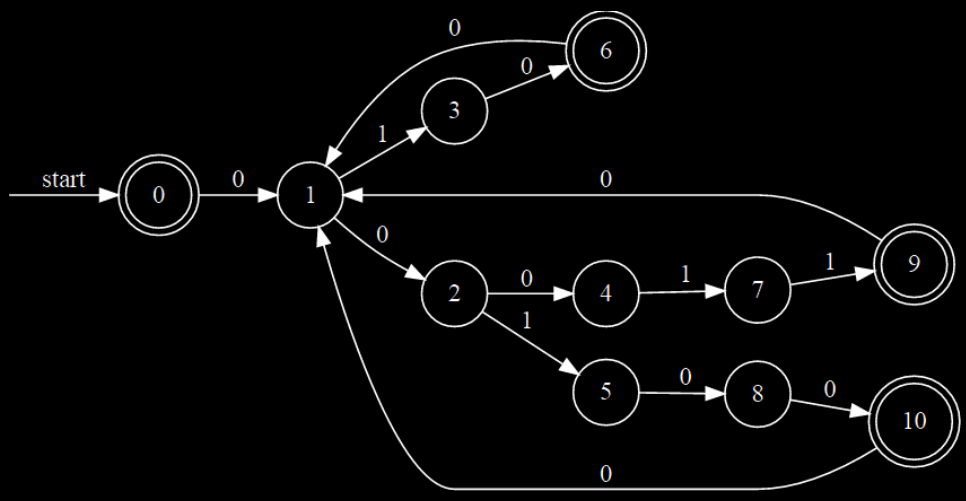
> • Que un **cliente** pueda realizar **operaciones no autorizadas**. En este caso, el servidor no va a poder procesar una petición enviada por un **cliente que no tiene privilegios para ello**, por ejemplo: un cliente que se encarga de realizar peticiones de precios de productos, de repente empieza a realizar peticiones de login.
>
> • Que los **mensajes no** estén **bien formados**. Puede ocurrir, por ciertos eventos, que un cliente envíe al servidor un mensaje y que éste no esté formado, bien porque **le falte** información necesaria o porque la información que contenga **sea errónea**.

Para evitar estos errores es de vital importancia que se utilice un **diagrama de estados o autómata**, para crear el **comportamiento que desarrollará el servidor**.

Con ellos podremos definir todos y cada uno de los **estados** o comportamientos por los que **va a pasar nuestro servidor**, definiendo qué **información debe llegar** a cada uno de ellos, hacia qué **estado se dirigirá** cuando termine su ejecución y qué **comportamientos** deberá tener cuando **llegue información errónea** a cada uno de los estados posibles en su ejecución.

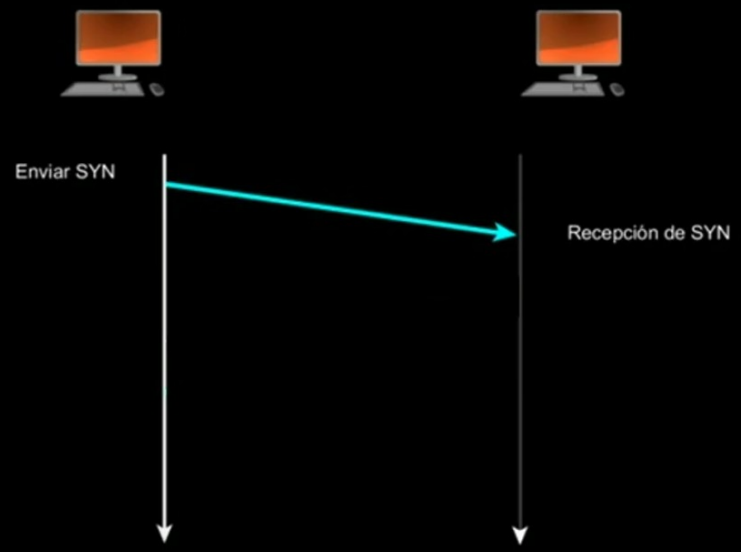
Para ello, deberemos definir en nuestro servidor **dos variables**, para almacenar:

- el **estado** en el que nos encontramos,
- el **comando** que queremos ejecutar de una lista de posibles casos.

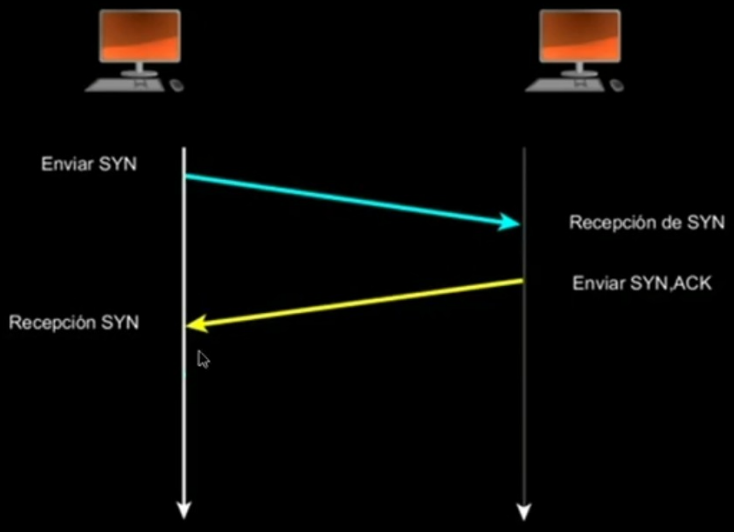


Conexión TCP simple paso a paso

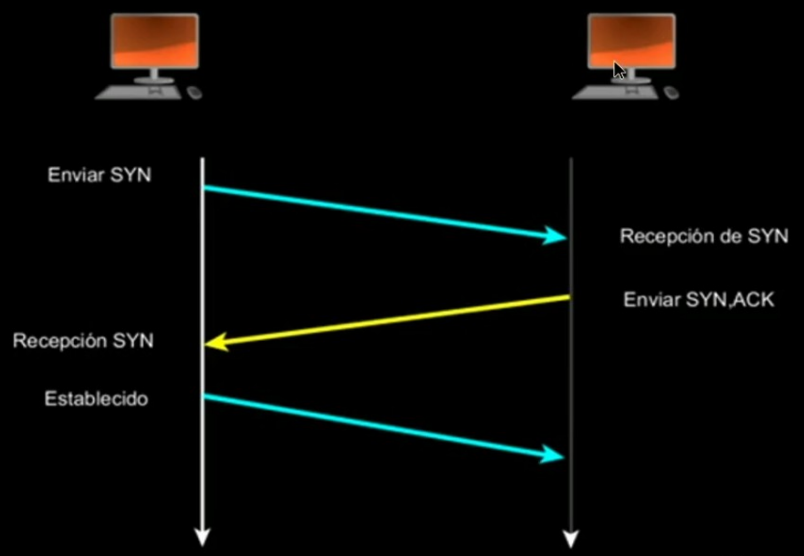
1. Sincronización del cliente y del servidor. Se envía un mensaje de sincronización, senominado SYN. Una vez realizada la sincronización podrán enviar y recibir información sin problemas. El cliente enviará un mensaje al servidor (SYN)



2. Una vez que el servidor recibe el SYN correctamente, este envía el ACK del mensaje al cliente (mensaje de confirmación al emisor-cliente de que ha llegado correctamente el paquete).



3. Una vez que el cliente recibe el ACK de la sincronización, pueden empezar a enviar paquetes. El envío de paquetes seguirá este procedimiento, se envía un mensaje y se espera su ACK.


Enviar SYN
Recepción de SYN
Enviar SYN,ACK
Recepción SYN
Establecido