Capa de aplicación

Existen varios modelos de servicios en la capa de transporte, los más importantes son el modelo **cliente-servidor**, el modelo **Peer-to-peer (P2P)** y un modelo **híbrido entre ambas P2P y cliente-servidor**. Para poder enviar paquetes se utilizan diferentes protocolos tales como **HTTP**, **FTP**, **SMTP**, **POP3**, **IMAP**, **DNS** entre otros.

Las aplicaciones se comunican a través de la red, estas deben ejecutarse en varios sistemas finales (sistemas con la aplicación) y el software solo debe ser escrito en los elementos que usen la aplicación, no en los elementos intermedios ya que estos no ejecutan aplicaciones.

En la arquitectura cliente-servidor tenemos dos elementos principales que son el **cliente** y el **servidor**.

* El cliente se comunica con el servidor, este se puede conectar con el servidor de manera intermitente, puede tener una IP dinámica y los clientes no se comunican entre sí.
* El servidor esta siempre en un host, su dirección IP es permanente y para escalar el sistema se utilizan granjas de servidores.

La arquitectura P2P es:

* Descentralizada, ya que no existe un servidor central para el control, los participantes se comunican entre sí y todos los nodos actúan como clientes y servidores.
* Distribuida, ya que la información no está alojada en un solo sitio.
* La carga esta balanceada entre todos los participantes.
* La información es redundante para que sea más accesible.
* La disponibilidad es muy alta porque la caída de un host no bloquea un servicio.
* Los recursos están optimizados.

Como ejemplos de arquitecturas híbridas tenemos a Skype y los servicios de mensajería instantánea.

Las aplicaciones que se comunican en la red comunican sus procesos. Estos procesos son programas en ejecución de un sistema informático. Para comunicar diferentes sistemas informáticos se utilizan mensajes. En la arquitectura cliente servidor, el **proceso cliente** inicia la comunicación y el **proceso servidor** espera peticiones de los clientes.

Los procesos envían y reciben mensajes por la red a través de sus sockets. Los sockets son análogos a “puertas” (puertos). Podemos definir a un socket como la interfaz software por la que se envían paquetes, en un socket suele haber dos procesos principales (puerta de una casa):

* Proceso transmisor, envía los mensajes por el puerto. Este confía en la infraestructura de transporte al otro lado del puerto, la cual lleva los mensajes al socket en el proceso receptor del destino.
* Proceso receptor, recibe el mensaje y actúa sobre este.

Un socket es la interfaz entre la capa de aplicación y la capa de transporte de un host. También se conoce como **Interfaz de programación de aplicaciones (API, Application Programming Interface)** que opera entre la aplicación y la red.

Para que un proceso reciba un mensaje, éste debe tener un identificador único. Un host tiene una dirección IP única, pero ¿es suficiente la dirección IP? No, es necesario un número de puerto para identificar a un proceso dentro de un host. El servidor HTTP usa el puerto 80, el servidor e-mail usa el puerto 25.

Hemos dicho que para enviar mensajes hacen falta protocolos, hay diferentes tipos de protocolos, **de dominio público** **(abiertos)**, permiten interoperabilidad y están definidos en RCFs, y **propietarios**, en general no permiten interoperabilidad.

Las aplicaciones pueden tener o no una transferencia de datos confiable, la tasa de transferencia en aplicaciones implica que pueda haber aplicaciones con un ancho de banda sensitivo o aplicaciones estáticas, el retardo de algunas aplicaciones debe de ser bajo para que estas puedan ser efectivas, por último, un aspecto importante es la seguridad.

Los dos principales protocolos son:

* TCP. Está orientado a la conexión ya que requiere de un acuerdo entre cliente y servidor. El transporte es confiable, sin pérdidas. El control de flujo impide que se sobrecargue el receptor. El control de congestión frena al transmisor cuando la red esta sobrecargada. No provee garantías de retardo ni ancho de banda mínimo.
* UDP. Transferencia de datos no confiable entre los procesos transmisor y receptor. **No** provee acuerdo entre los procesos, confiabilidad, control de flujo, control de congestión, ni garantías de retardo o ancho de banda.

Programación de sockets

Como mencionamos anteriormente, se usan APIs que permiten desarrollar aplicaciones para usar los servicios TCP.

Se definen los siguientes tipos de sockets en función de los servicios proporcionados:

* Stream sockets: Confiables, orientados a conexión. Utilizan los servicios de TCP. Los datos se envían sin errores y en el mismo orden que se envían. Hay control de flujo y no se imponen límites a los datos.
* Datagram Sockets: No confiables ni orientados a conexión. Utilizan los servicios de UDP. No hay ‘handshake’ antes de enviar los datos. Los datagramas se envían como paquetes independientes, el emisor añade explícitamente la IP y el puerto de destino a cada paquete, el receptor extrae la IP y puerto de destino del paquete. No hay garantías ni de que lleguen los paquetes ni de que lo hagan ordenados. No hay segmentación ni reconstrucción de los paquetes.
* Raw Sockets: Acceso niveles más bajos del protocolo TCP/IP. Accede a los niveles más bajos de la pila de protocolos (IP, ICMP). Se utiliza para probar nuevos protocolos.

Para que se produzca una conexión entre cliente y servidor:

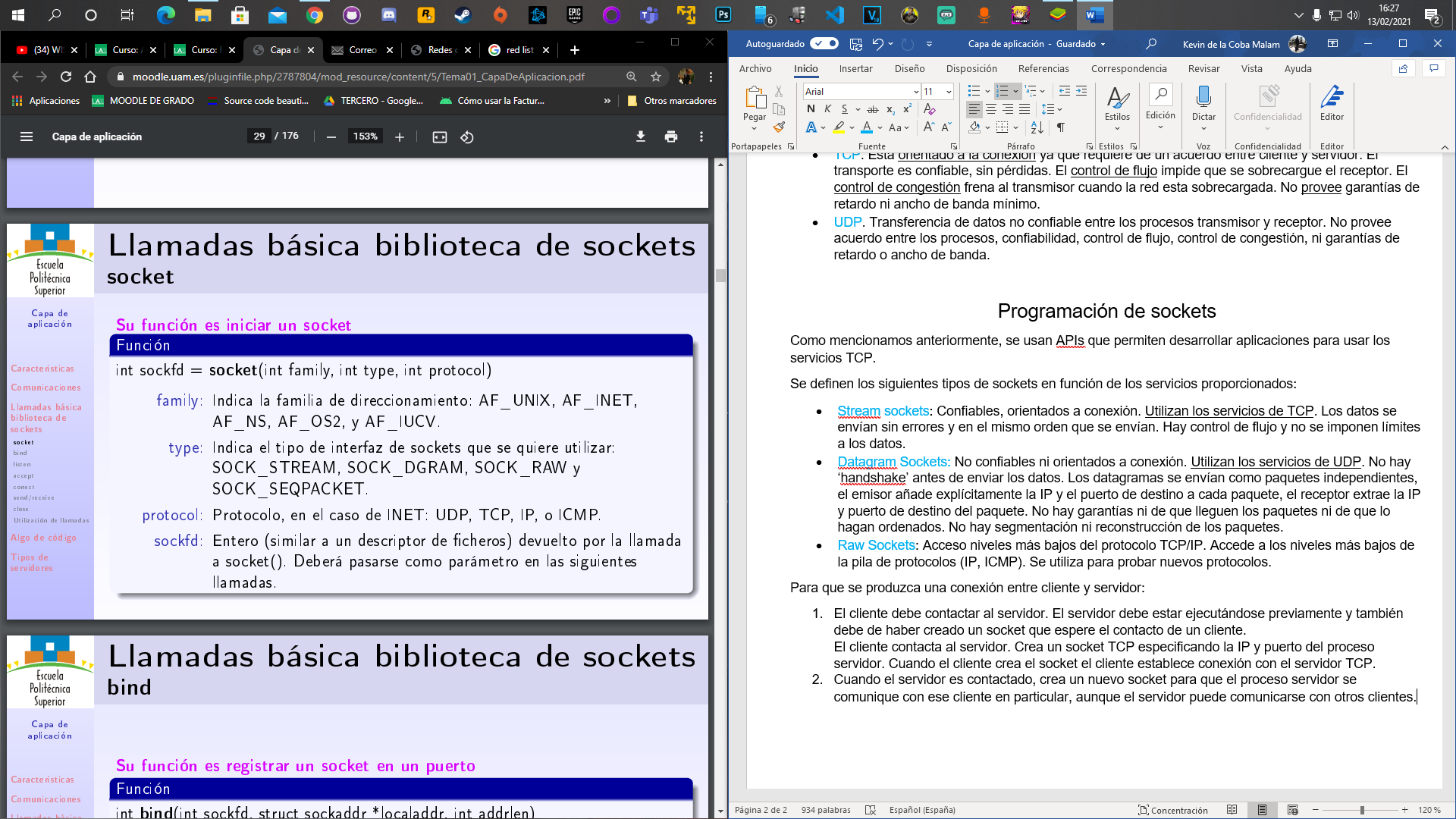
1. El cliente debe contactar al servidor. El servidor debe estar ejecutándose previamente y también debe de haber creado un socket que espere el contacto de un cliente.

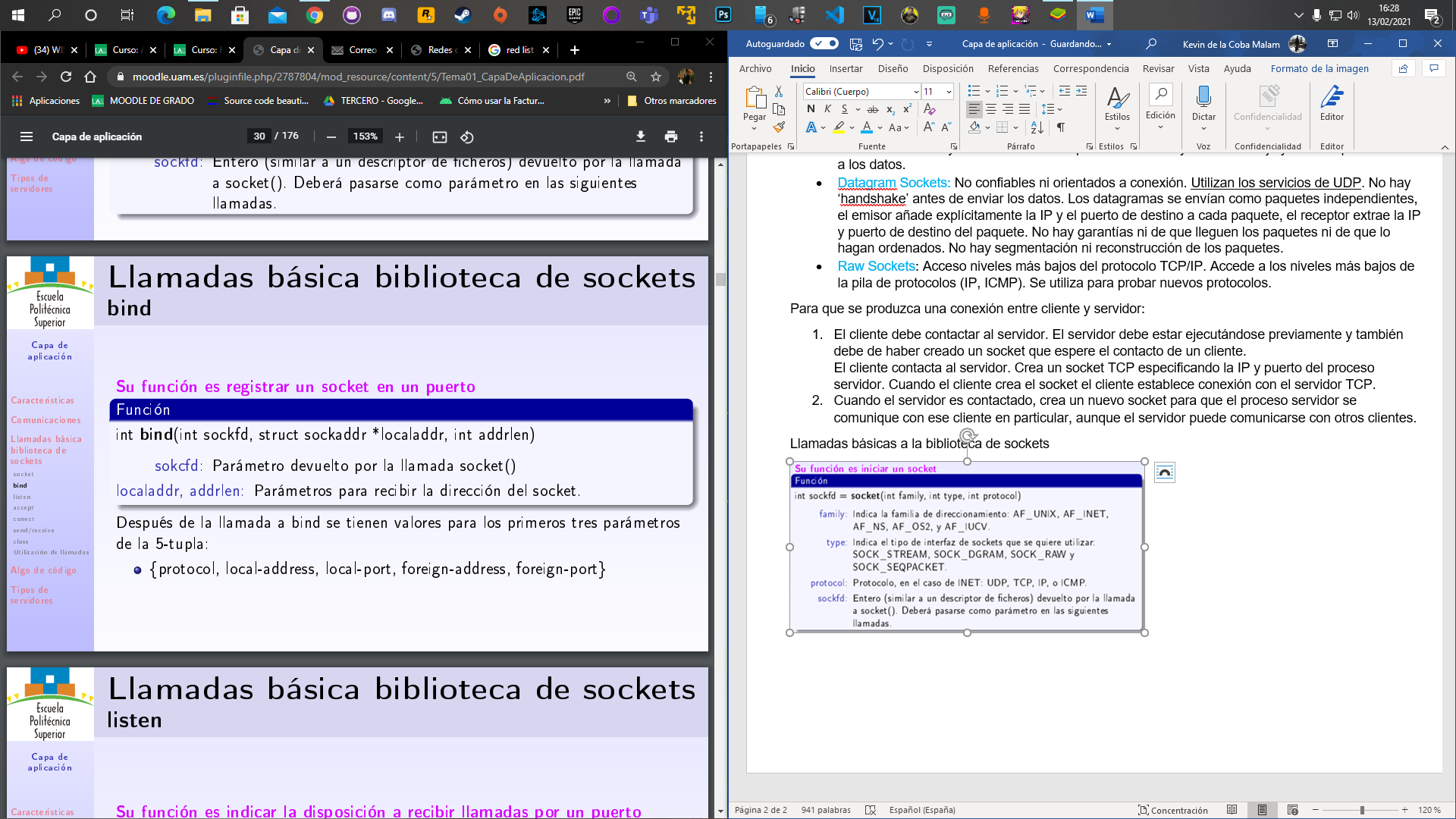
El cliente contacta al servidor. Crea un socket TCP especificando la IP y puerto del proceso servidor. Cuando el cliente crea el socket el cliente establece conexión con el servidor TCP.

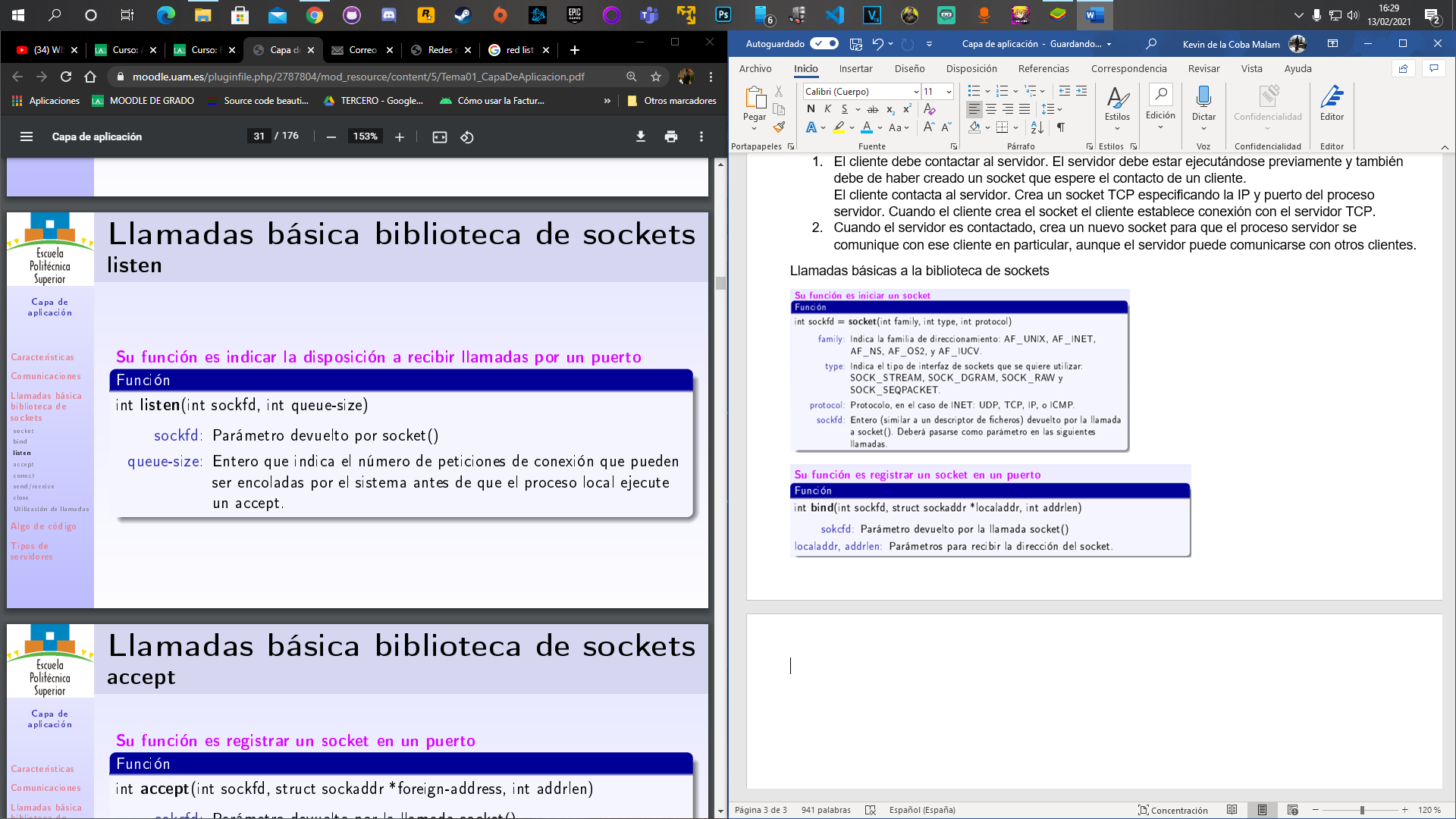
1. Cuando el servidor es contactado, crea un nuevo socket para que el proceso servidor se comunique con ese cliente en particular, aunque el servidor puede comunicarse con otros clientes.

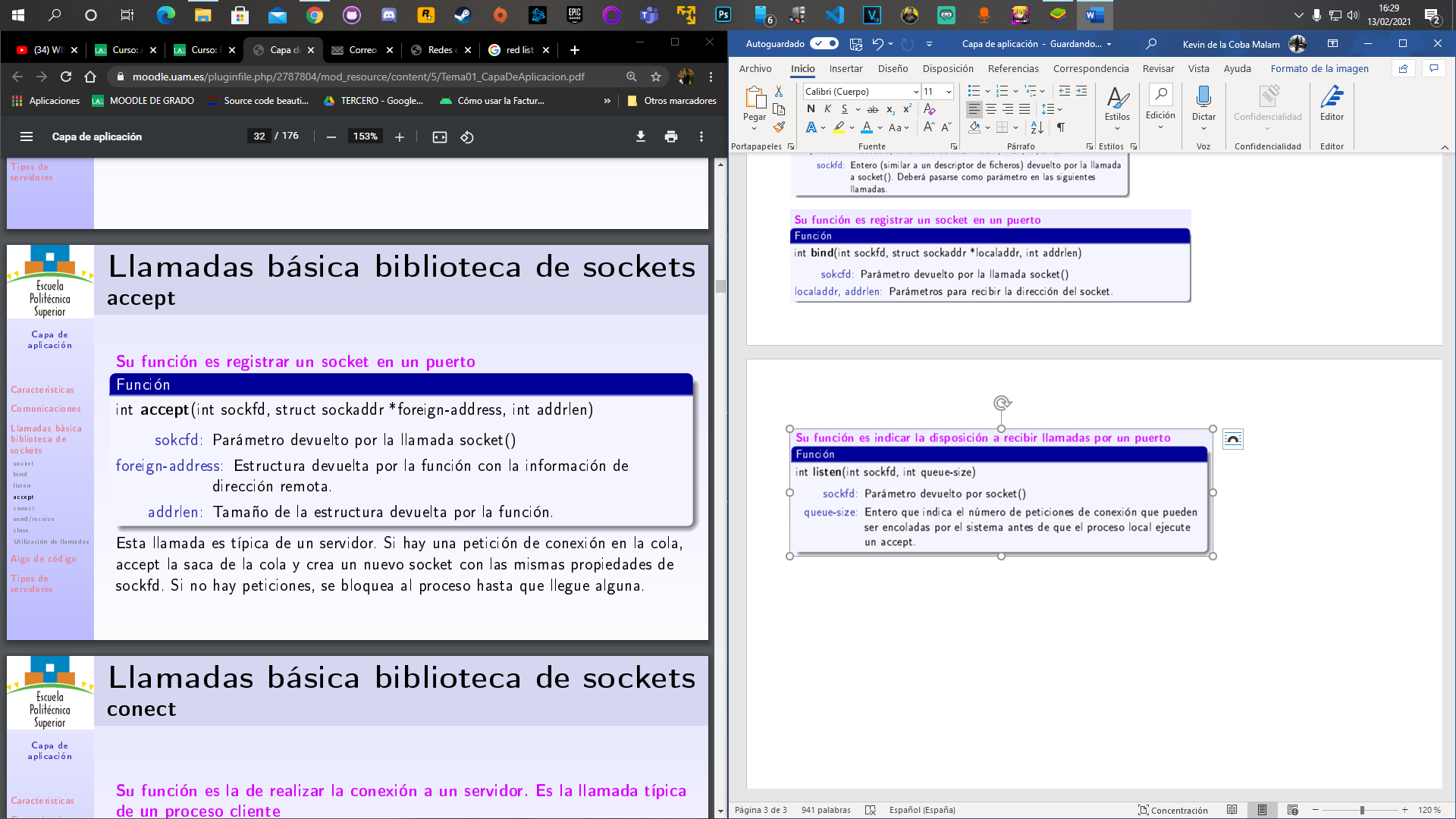
Los servidores pueden ser **iterativos** y procesar peticiones de una en una, o pueden ser **concurrentes**, por lo que pueden atender a varios clientes a la vez.

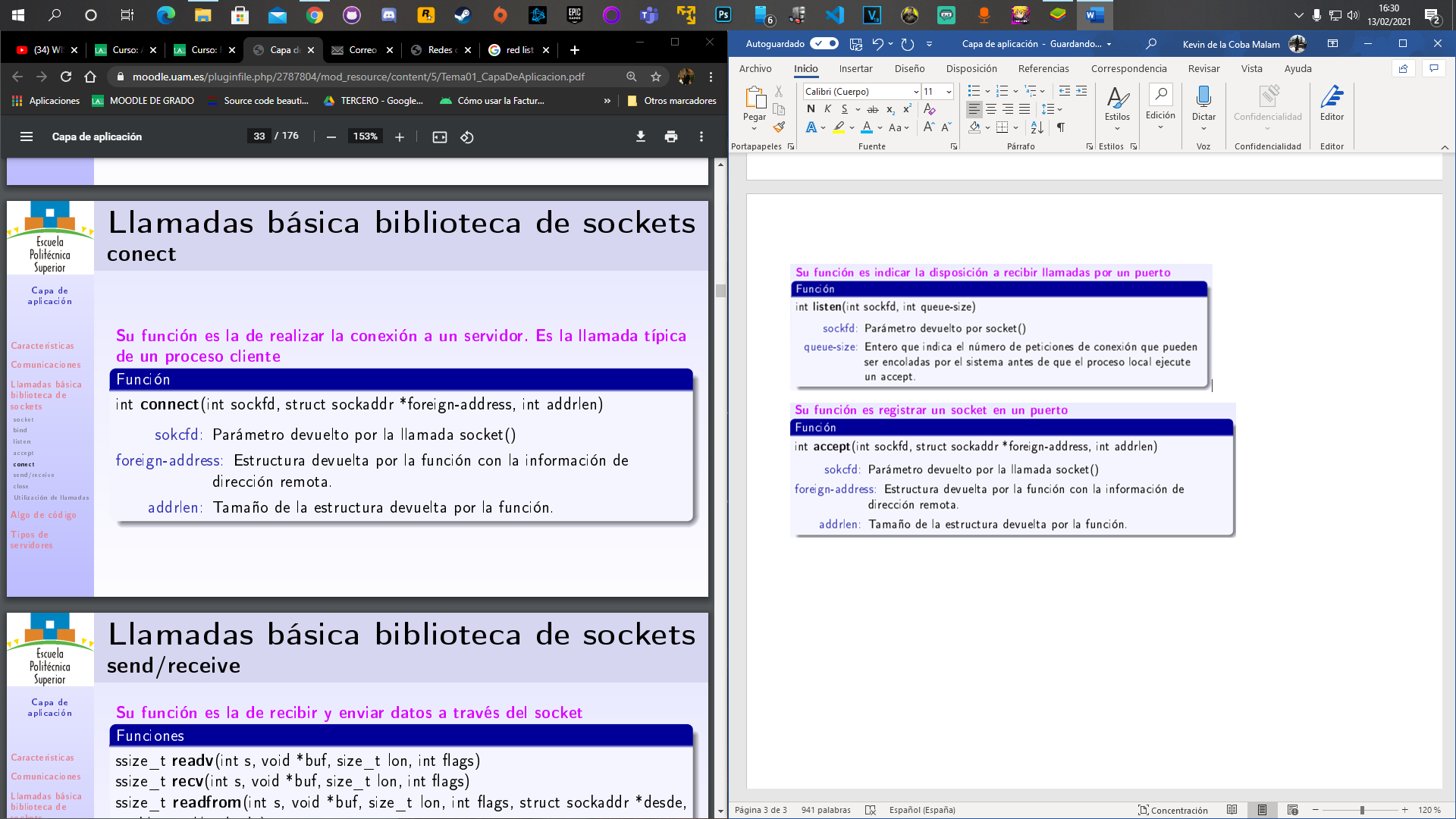
Llamadas básicas a la biblioteca de sockets

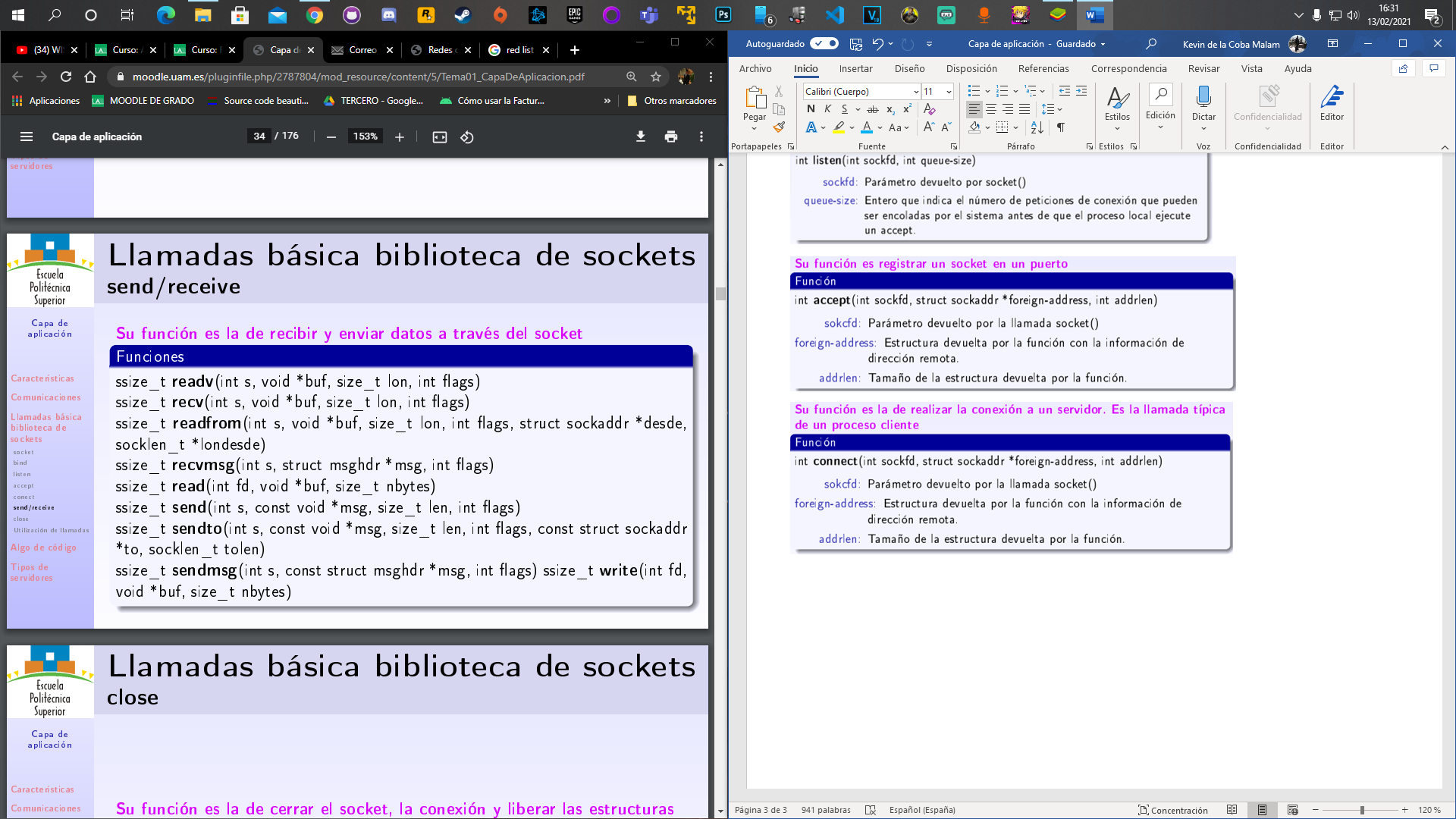






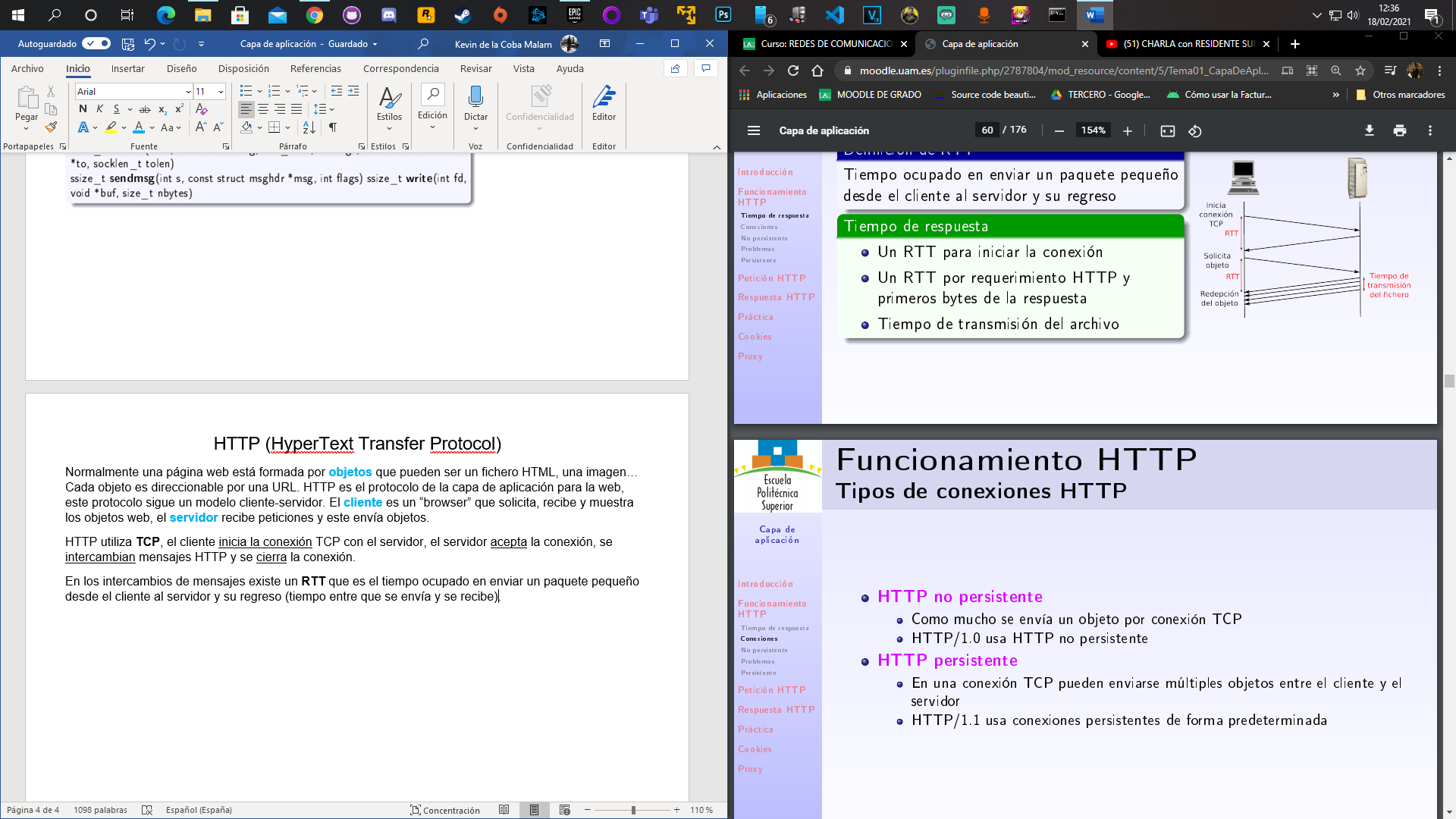






HTTP (HyperText Transfer Protocol)

Normalmente una página web está formada por **objetos** que pueden ser un fichero HTML, una imagen… Cada objeto es direccionable por una URL. HTTP es el protocolo de la capa de aplicación para la web, este protocolo sigue un modelo cliente-servidor. El **cliente** es un “browser” que solicita, recibe y muestra los objetos web, el **servidor** recibe peticiones y este envía objetos.

HTTP utiliza **TCP**, el cliente inicia la conexión TCP con el servidor, el servidor acepta la conexión, se intercambian mensajes HTTP y se cierra la conexión.

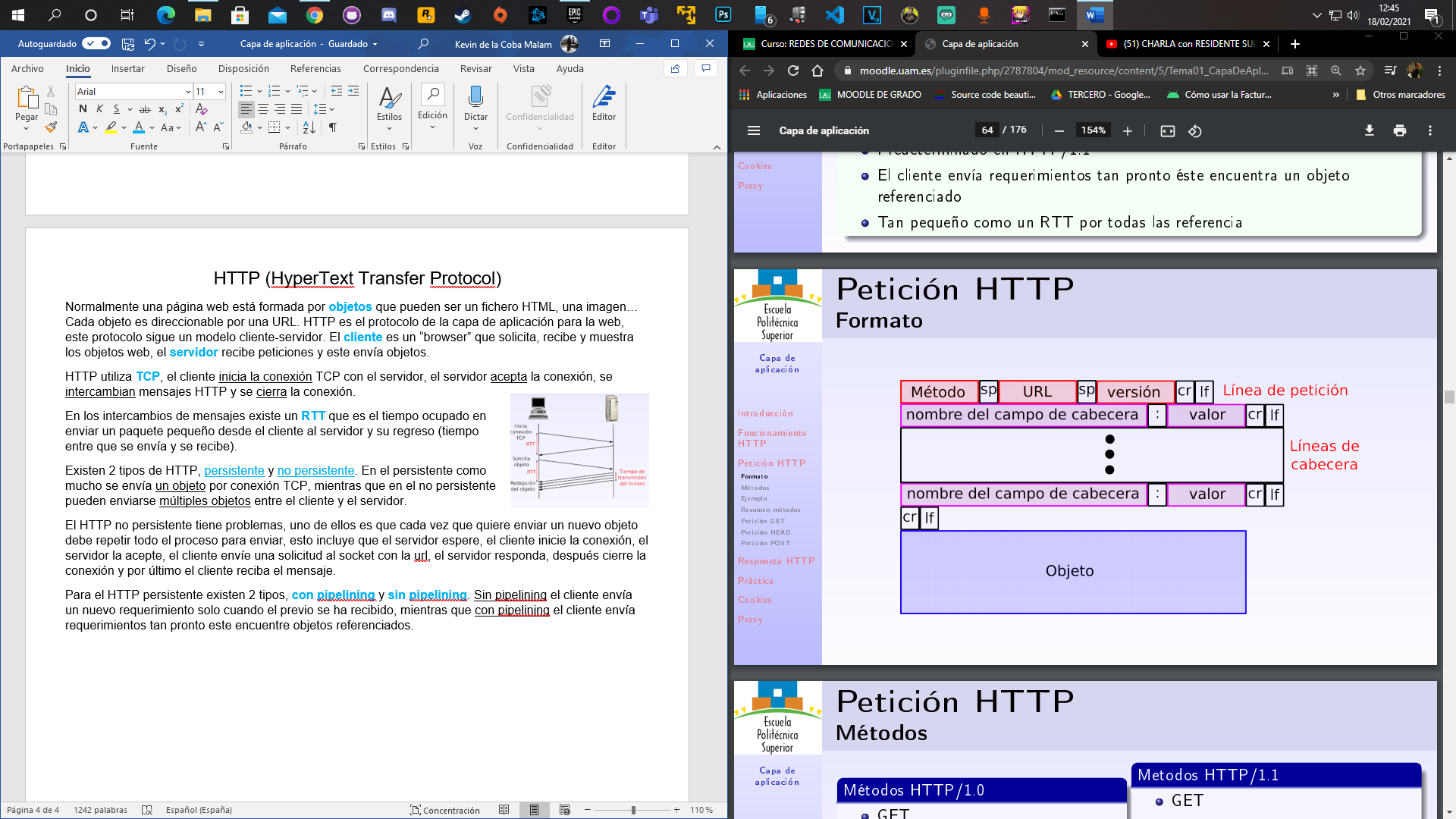
En los intercambios de mensajes existe un **RTT** que es el tiempo ocupado en enviar un paquete pequeño desde el cliente al servidor y su regreso (tiempo entre que se envía y se recibe).

Existen 2 tipos de HTTP, persistente y no persistente. En el persistente como mucho se envía un objeto por conexión TCP, mientras que en el no persistente pueden enviarse múltiples objetos entre el cliente y el servidor.

El HTTPno persistente tiene problemas, uno de ellos es que cada vez que quiere enviar un nuevo objeto debe repetir todo el proceso para enviar, esto incluye que el servidor espere, el cliente inicie la conexión, el servidor la acepte, el cliente envíe una solicitud al socket con la url, el servidor responda, después cierre la conexión y por último el cliente reciba el mensaje.

Para el HTTP persistente existen 2 tipos, **con pipelining** y **sin pipelining**. Sin pipelining el cliente envía un nuevo requerimiento solo cuando el previo se ha recibido, mientras que con pipelining el cliente envía requerimientos tan pronto este encuentre objetos referenciados.

PETICIÓN HTTP

Los métodos HTTP/1.0 son GET, POST y HEAD.

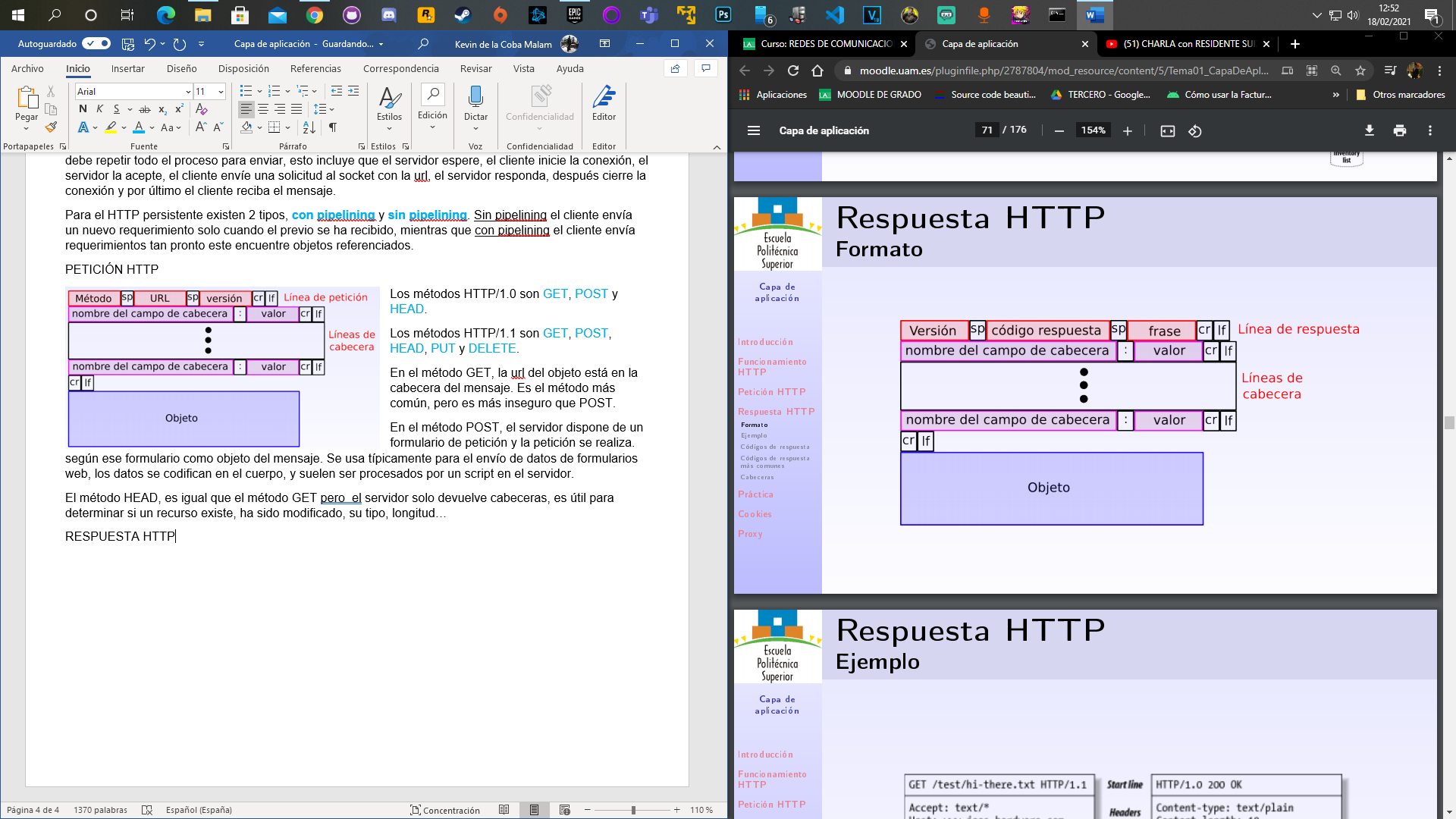
Los métodos HTTP/1.1 son GET, POST, HEAD, PUT y DELETE.

En el método GET, la url del objeto está en la cabecera del mensaje. Es el método más común, pero es más inseguro que POST.

En el método POST, el servidor dispone de un formulario de petición y la petición se realiza. según ese formulario como objeto del mensaje. Se usa típicamente para el envío de datos de formularios web, los datos se codifican en el cuerpo, y suelen ser procesados por un script en el servidor.

El método HEAD, es igual que el método GET pero el servidor solo devuelve cabeceras, es útil para determinar si un recurso existe, ha sido modificado, su tipo, longitud…

RESPUESTA HTTP

200 OK

301 Moved permanently

400 Bad request

404 Not Found

505 HTTP version not supported

Las **cookies** siguen el protocolo RFC 2965, son datos que se guardan en el cliente y los utiliza el servidor para dar una mejor experiencia al usuario. Las cookies contienen, carritos, sugerencias, estado de la sesión. Las cookies permiten aprender sobre el usuario, se puede proveer nombre y correo.

Para hacer mas amena la conexión existen **servidores proxy**/**web cache**. Estos actúan como clientes y servidores. Estos reducen el tiempo de respuesta de las peticiones del cliente, reducen el tráfico de los enlaces de internet y permiten a proveedores de contenido “pobre” entregar contenido de forma eficiente.