

Instituto Politécnico Nacional

Escuela Superior de Cómputo

*Aplicaciones para comunicaciones de red*

*“Practica 6 – Servidor HTTP sockets No-bloqueantes”*

**Grupo:** 3CM15

**Integrantes:**

* Ramírez Olvera Guillermo.
* Sánchez Méndez Edmundo Josue.

**Fecha de entrega:** 03 de juniode 2021

**Profesor:** Moreno Cervantes Axel Ernesto



Existen diferentes tipos de sockets y sus tipos se clasifican según sus propiedades de comunicación. Las propiedades de comunicación están definidas por los protocolos que admiten. Por lo general, requiere un mismo tipo de sockets para establecer cualquier tipo de comunicación porque el conjunto estándar de reglas para transferir datos está definido por los protocolos que utilizan. Los protocolos son TCP o UDP. El programa especifica el protocolo que admite. Normalmente, un programa admite más de un protocolo porque el programador que lo ha escrito proporcionó el código necesario para admitir el tipo de socket en particular en este dominio. Si se ha estado preguntando el tipo de sockets admitidos por TCP / IP, son los siguientes:

|  |  |  |
| --- | --- | --- |
| Tipo de socket | Protocolo | Descripción |
| SOCK\_STREAM | TCP | Proporciona un servicio orientado a la conexión confiable |
| SOCK\_DGRAM | UDP | Proporciona un servicio sin conexión para datagramas. |
| SOCK\_RAW | IP, RAW, ICMP | Brinda soporte para protocolos de capa inferior, como IP |

Tanto los sockets bloqueantes como los no bloqueantes tienen sus usos, pero los sockets bloqueantes son más comunes porque son fáciles de manejar y son la elección predeterminada de la programación de sockets en Java. Para decirlo de una manera simple: en un programa de bloqueo de socket, las líneas de código se ejecutan una tras otra, pero se bloquean o se detienen en la instrucción donde se realizan las llamadas al sistema. Espera allí hasta que se recibe una respuesta adecuada, se agota el tiempo de espera o se encuentra algún error. El programa de socket no bloqueantes, por otro lado, no espera la respuesta y continúa la ejecución, aunque la llamada no se haya completado. Cualquier respuesta incompleta de la llamada al sistema invocada se trata por separado.

**Sockets no bloqueantes vs sockets bloqueantes**

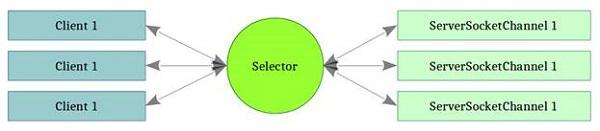
Los sockets bloqueantes son apropiados para una situación en la que una respuesta de la llamada al sistema es crucial. Por ejemplo, un programa cliente desea conectarse a un servidor. El programa debe responder solo cuando se establece la conexión o se ha producido algún error. Esto significa que el conector debe estar bloqueando el control del usuario hasta que se reciba una respuesta adecuada.

Por otro lado, considere que un cliente está intentando establecer múltiples conexiones y cada conexión puede hacerse exclusiva de otra. En tal caso, el socket no debe ser rígido para bloquear otra llamada al sistema para establecer la conexión. Este es un escenario para sockets no bloqueantes.

**Sockets no bloqueantes en Java**

Java tiene sockets TCP y UDP. Los métodos como connect(), accept(), read() y write() definidos en la clase ServerSocket y Socket se utilizan para bloquear la programación de sockets. Por ejemplo, cuando un cliente invoca el método read() para leer datos del servidor, el hilo se bloquea hasta que los datos estén disponibles. Esta situación es indeseable en algunas circunstancias. En cambio, lo que podemos hacer es utilizar el período de espera para realizar alguna otra tarea. El socket del cliente puede notificar cuando los datos están disponibles. Otro problema es que, en una conexión de varios sockets, cada cliente es un hilo independiente. Por lo tanto, hay una sobrecarga de mantener un grupo de subprocesos de cliente.

El bloqueo de sockets es sencillo debido a su ejecución secuencial. Los sockets no bloqueantes, por otro lado, no son secuenciales. Requieren una perspectiva diferente para implementarlos en la programación. En cierto modo, los programas de socket sin bloqueo son un poco complejos y una técnica un poco más avanzada de comunicación de socket.



**Desarrollo**

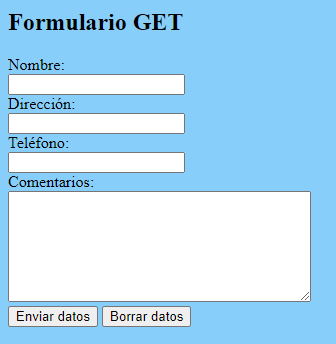
Esta práctica es igual a la práctica número 4 pero usando bloques no bloqueantes, permitiendo la conexión de varios clientes al mismo tiempo, es decir podemos hacer uso de los métodos POST y GET.

Como primer lugar veamos que son el método GET:

Cuando un usuario rellena un formulario en una página web los datos hay que enviarlos de alguna manera. Vamos a considerar las dos formas de envío de datos posibles: usando el método POST

En el ejemplo anterior la acción que se ejecutará cuando el usuario pulse el botón “Enviar datos” (submit) será el envío de los datos a la url especificada usando el método get.

Veamos el aspecto de un formulario cualquiera para hacernos una idea general.

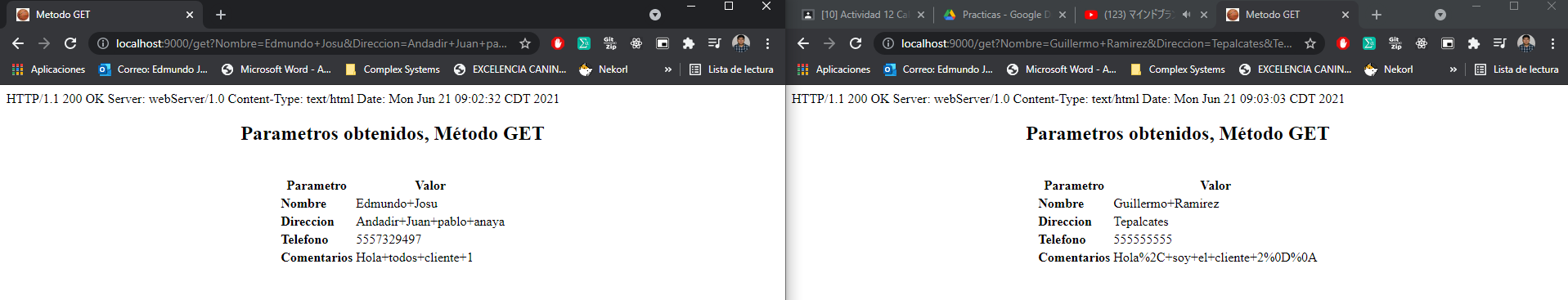


Este formulario consta de varios campos que al usuario se le solicitan como Nombre, Dirección, Teléfono y Comentarios.

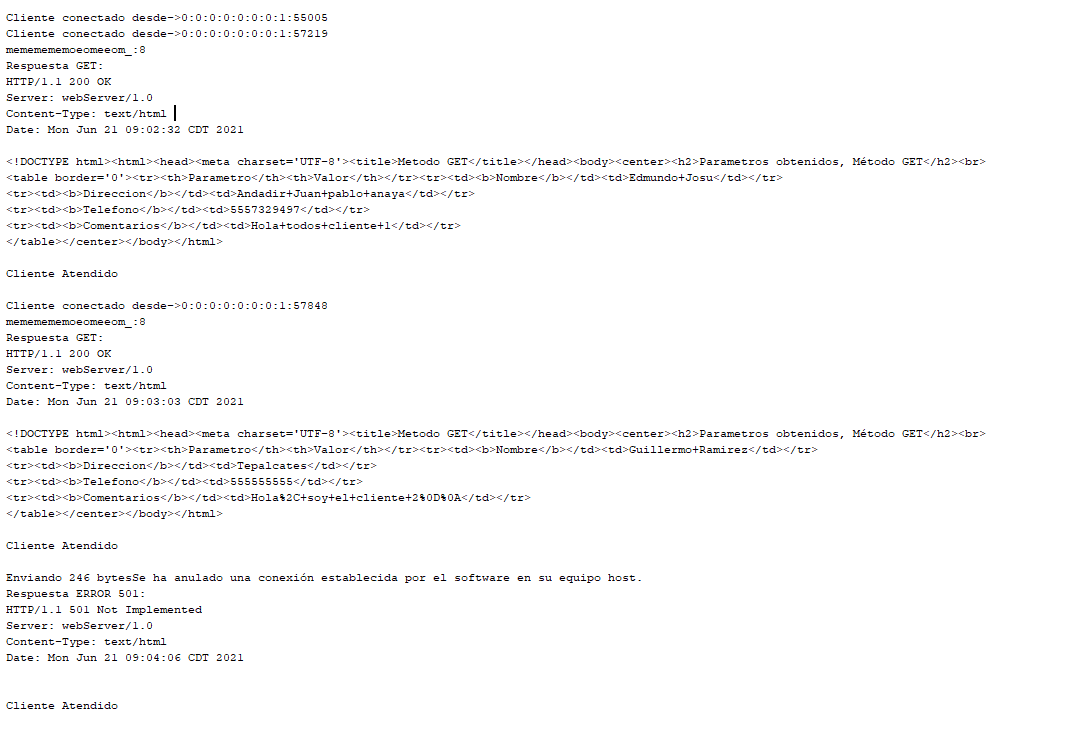
La diferencia entre los métodos get y post radica en la forma de enviar los datos a la página cuando se pulsa el botón “Enviar”. Mientras que el método GET envía los datos usando la URL, el método POST los envía de forma que no podemos verlos (en un segundo plano u "ocultos" al usuario).

Un resultado usando el método GET, a modo de ejemplo, podría es el siguiente: http://localhost:7777/get?Nombre=Edmundo+Josu&Direccion=Andadir+Juan+pablo+anaya&Telefono=5557329497&Comentarios=Hola+a+todos+%3D%29

El símbolo ? indica dónde empiezan los parámetros que se reciben desde el formulario que ha enviado los datos a la página y la pagina nos devuelve el siguiente resultado, en este caso es para dos clientes:



Resultado que no arroja el servidor:

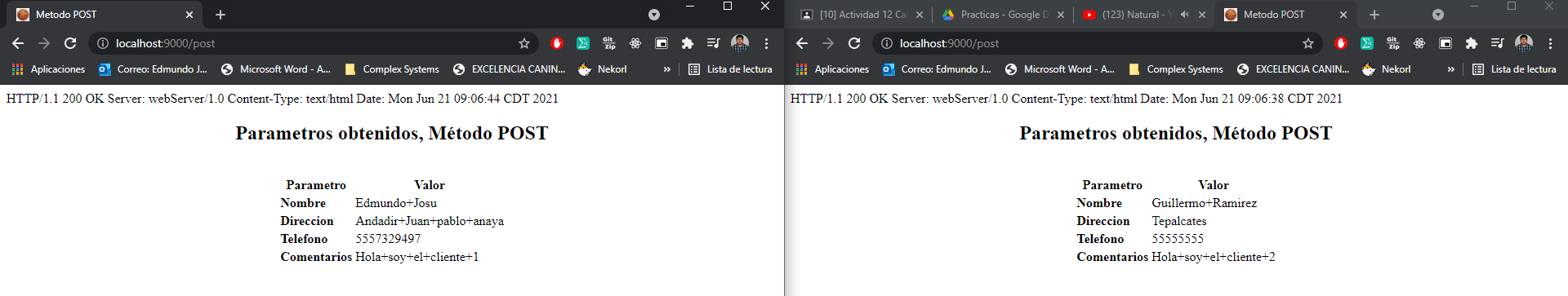


Ahora veamos el método POST:

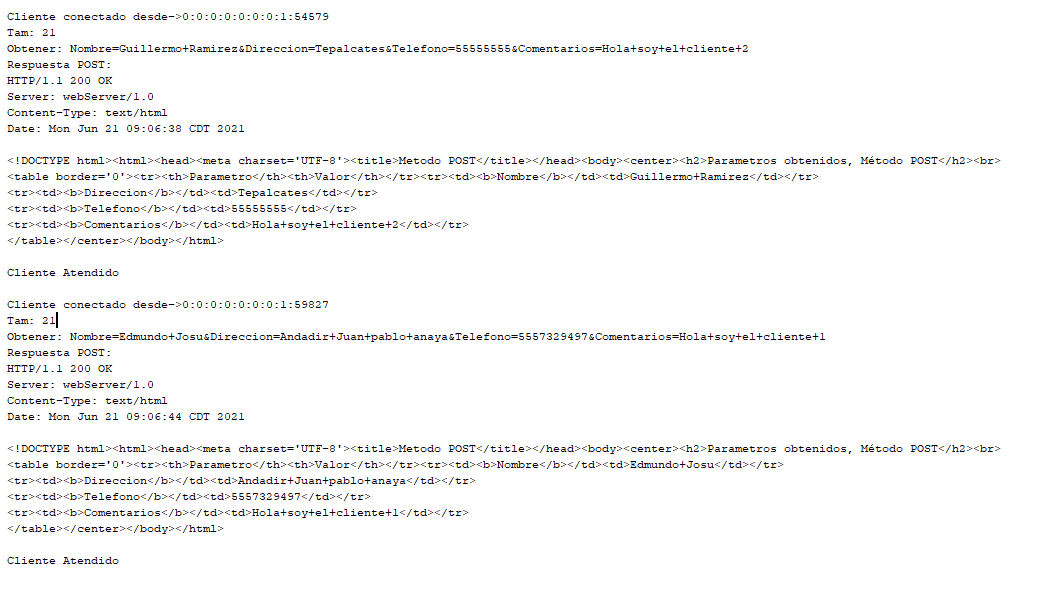
Hemos visto el resultado de un envío por el método GET. En el caso de un envío de datos usando el método POST, aunque estos datos también serán enviados (de una forma que podemos denominar “oculta”), no los podremos ver en la URL. Para poder recuperar los valores de los campos en el caso de un envío con el método POST necesitaríamos otras herramientas (por ejemplo, valernos del lenguaje PHP para recuperar el valor de esos campos).

El resultado final con ambos métodos podemos decir que es el mismo: la información se transmite de un lado a otro. La diferencia radica en que con el método GET podemos ver directamente los parámetros pasados ya que están dentro de la URL mientras que con el método POST los parámetros quedan ocultos y para rescatarlos hay que usar otras herramientas.

Un ejemplo de uso del método post sería este: <form action=" http://localhost:7777/post" method ="post">

A continuación, se muestran los parámetros obtenidos por el método POST, para ambos clientyes: 

El resultado en consola es el siguiente:



Así mismo el servidor proporciona recursos disponibles como lo son imágenes, música, etc, que el usuario puede visualizar o acceder a ellos.



Y si el servidor borra uno de estos archivos y un cliente quiere acceder a él se muestra:



En esta práctica se puede observar de manera perfecta el uso de sockets no bloqueante, ya que este permite varias conexiones sin usar el pool de hilos, en esta ocasión para levantar un servidor, ademas, pudimos crear un servidor HTTP basado en el propio protocolo y hacer la réplica del servidor con los estándares mime para poder hacer la visualización de cualquier elemento que tengamos en nuestra computadora, como, por ejemplo: Un archivo con extensión mp3, o con extensión PDF.