

Instituto Politécnico Nacional  
Escuela Superior de Cómputo  
Grupo: 3CV5  
Aplicaciones de Comunicaciones en Red

Práctica #3  
Servidor no bloqueante

Ana Paola Nava Vivas

# Introducción

Un socket (enchufe), es un método para la comunicación entre un programa del cliente y un programa del servidor en una red, se define, por tanto, como el punto final en una conexión.[1]

Este mecanismo surge a principios de los 80 con el sistema Unix de Berkeley, para proporcionar un medio de comunicación entre procesos y presentan la misma funcionalidad que tiene la comunicación por correo o por teléfono (de un buzón se extraen mensajes completos, mientras que el teléfono permite el envío de flujos de información que no tienen una estructura claramente definida), es decir permiten que un proceso hable (emita o reciba información) con otro incluso estando estos en distintas máquinas. Esta característica de interconectividad hace que el concepto de socket sea de gran utilidad.[1]

Un socket queda definido por un par de direcciones IP local y remota, un protocolo de transporte y un par de números de puerto local y remoto. Para que dos programas puedan comunicarse entre sí es necesario que se cumplan ciertos requisitos:

Que un programa sea capaz de localizar al otro.

Que ambos programas sean capaces de intercambiarse cualquier secuencia de octetos, es decir, datos relevantes a su finalidad.

Para ello son necesarios los tres recursos que originan el concepto de socket:

Un protocolo de comunicaciones, que permite el intercambio de octetos.

Un par de direcciones del Protocolo de Red (Dirección IP, si se utiliza el Protocolo TCP/IP), que identifica la computadora de origen y la remota.

Un par de números de puerto, que identifica a un programa dentro de cada computadora.

Los sockets permiten implementar una arquitectura cliente-servidor o peer to peer. La comunicación debe ser iniciada por uno de los programas que se denomina programa cliente. El segundo programa espera a que otro inicie la comunicación, por este motivo se denomina programa servidor.[1]

Un socket es un proceso o hilo existente en la máquina cliente y en la máquina servidora, que sirve en última instancia para que el programa servidor y el cliente lean y escriban la información. Esta información será la transmitida por las diferentes capas de red.[1]

Cuando un cliente conecta con el servidor se crea un nuevo socket, de esta forma, el servidor puede seguir esperando conexiones en el socket principal y comunicarse con el cliente conectado, de igual manera se establece un socket en el cliente en un puerto local.[1]

Una aplicación servidor normalmente escucha por un puerto específico esperando una petición de conexión de un cliente, una vez que se recibe, el cliente y el servidor se conectan de forma que les sea posible comunicarse entre ambos. Durante este proceso, el cliente es asignado a

un número de puerto, mediante el cual envía peticiones al servidor y recibe de este las respuestas correspondientes.[1]

Similarmente, el servidor obtiene un nuevo número de puerto local que le servirá para poder continuar escuchando cada petición de conexión del puerto original. De igual forma une un socket a este puerto local.[1]

## Objetivo

Generar un servidor no bloqueante, orientado a conexión, con el siguiente funcionamiento. Cada cliente se encargará de recolectar todas las imágenes, pdf, audios y videos que existan en la pc donde se está ejecutando.

Enviando la recopilación de información al servidor, el cual detectará de que cliente viene la información y la almacenará en una carpeta, que tendrá como nombre la dirección IP del cliente.

**Restricción:** No se pueden enviar archivos completos en un solo envío, cada archivo se enviará en 4 partes.

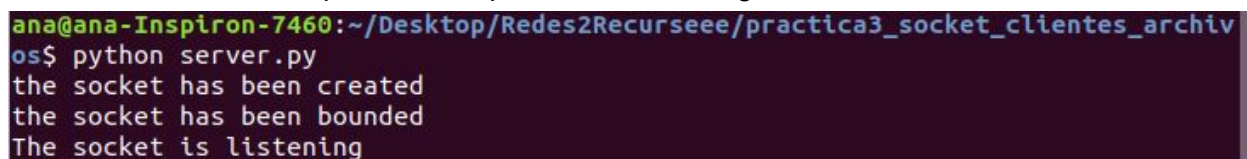
## Desarrollo

Para desarrollar la práctica se usó el lenguaje Python 3.6. Las bibliotecas: glob, socket, sys, threading, os.

“Glob”, para obtener por parte de los clientes un listado de los archivos, “socket” para los sockets, “sys” para hacer una salida “system exit” del programa en caso de error en el “bind” del servidor, “threading” para manejar cada cliente en un hilo diferente y “os” para crear los directorios desde el servidor.

Se usó una máquina virtual de una versión ligera de ubuntu, llamada lubuntu, para simular clientes en la red y no solo en el host.

Al correr el servidor, se queda a la espera/escucha de algún cliente:

A terminal window with a dark background and light-colored text. The prompt is 'ana@ana-Inspiron-7460:~/Desktop/Redes2Recurseee/practica3\_socket\_clientes\_archiv'. The user has entered 'python server.py'. The output shows four lines: 'the socket has been created', 'the socket has been bounded', and 'The socket is listening'.

```
ana@ana-Inspiron-7460:~/Desktop/Redes2Recurseee/practica3_socket_clientes_archiv
os$ python server.py
the socket has been created
the socket has been bounded
The socket is listening
```

Al correr el cliente, éste da la opción de iniciar con el envío:

```

ana@ana-Inspiron-7460:~/Desktop/Redes2Recursee/practica3_socket_clientes_archiv
os$ python client.py
Elementos en la lista: 108
-->[Press enter to send stuff]

```

Aquí ya envió y confirmó el recibimiento de “algo”

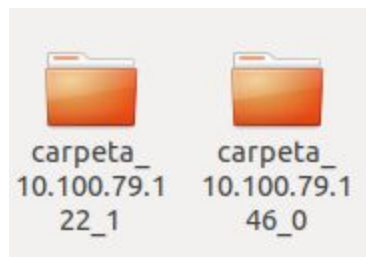
```

-->[Press enter to send stuff]
/home/ana/Pictures/doge.jpg

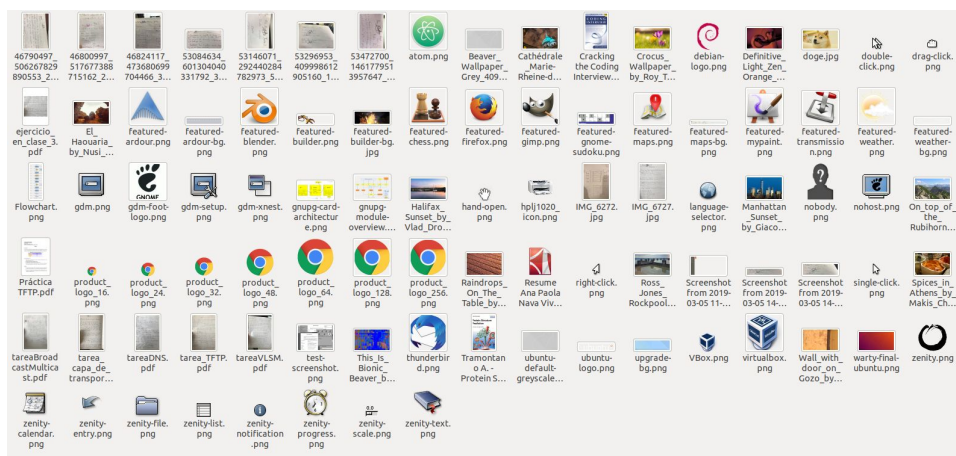
Largo del archivo: 49202
tamano aprox del paquete: 12300
tamano del buffer: 16384
b'recibido'
b'recibidotl'
test1
test2
b'recibido'
b'recibido'
b'recibido'
b'recibido'
/home/ana/Downloads/46800997_517677388715162_2519311104392298496_n.jpg

```

Estas son las carpetas que creó. Llevan el nombre de la ip del cliente y el número del hilo.



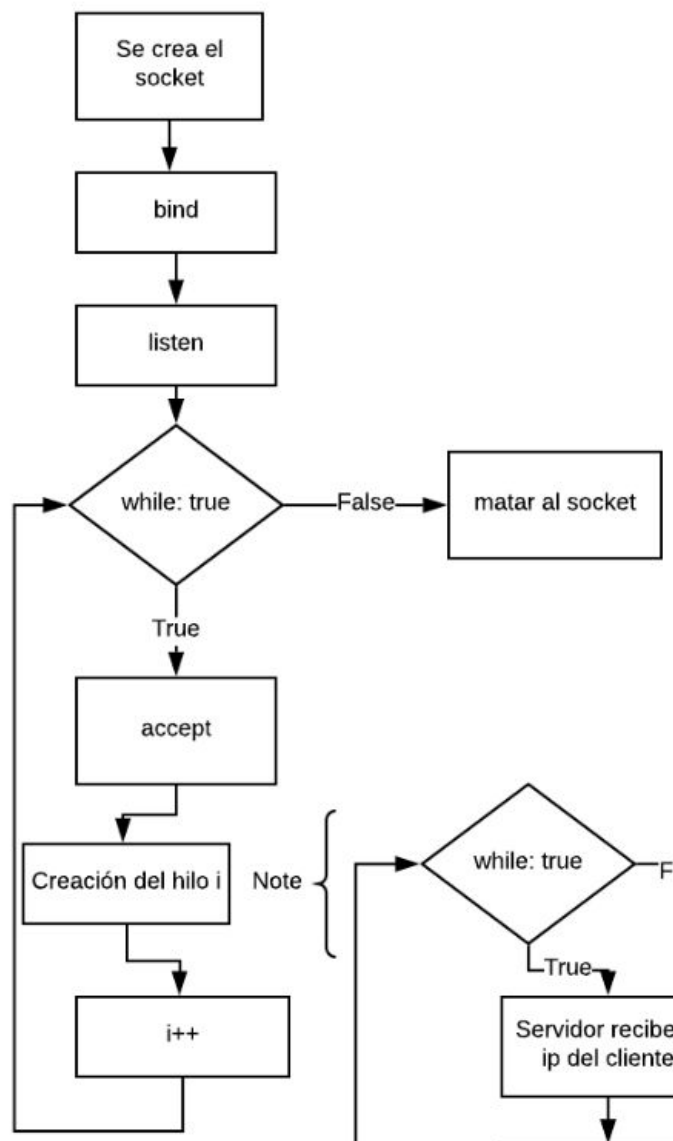
Esta es una carpeta con todas las cosas que recibió.



## Diagramas

### Servidor:

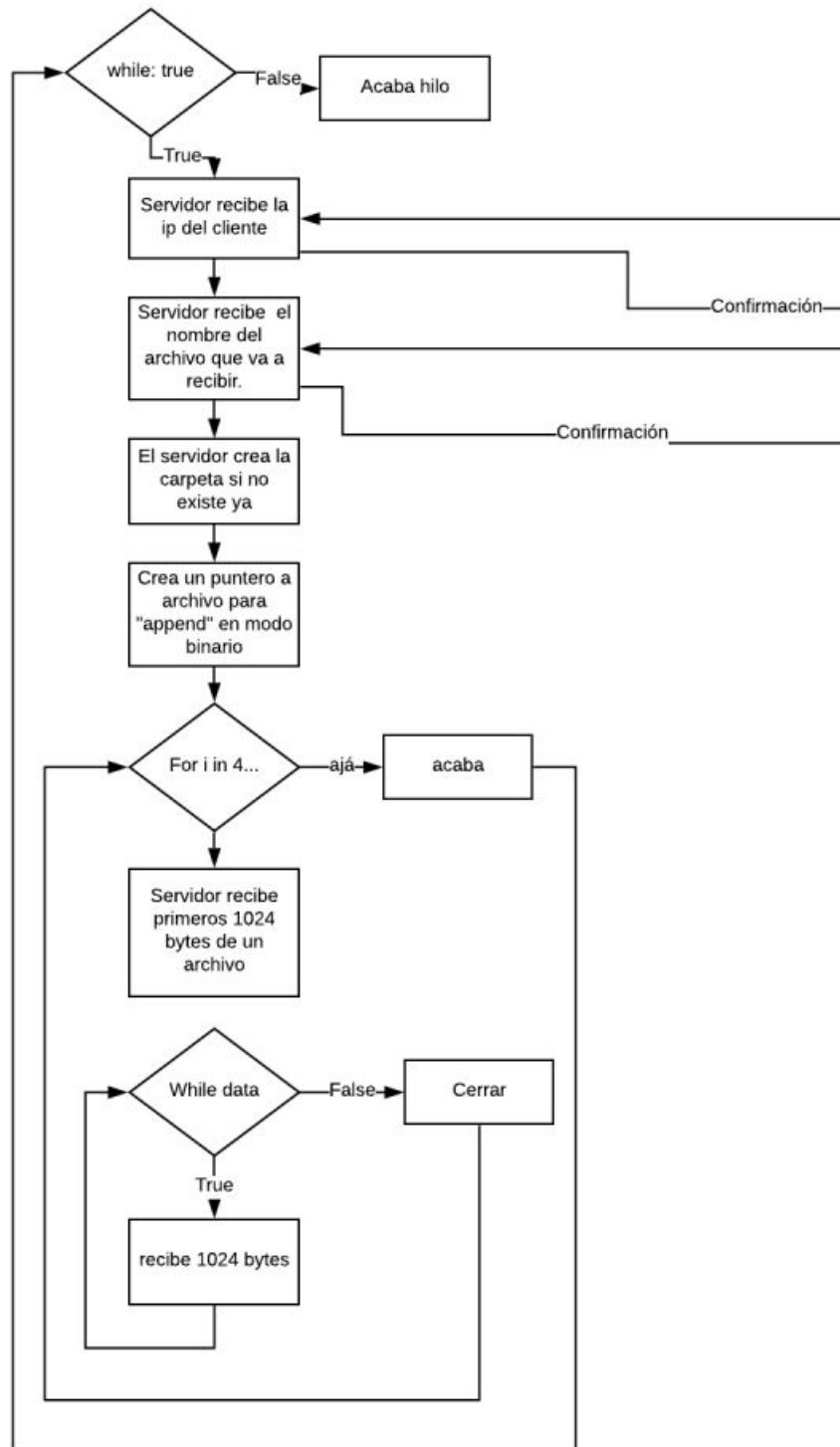
Se crea el socket no bloqueante, se intenta hacer el bind y en caso de que no se pueda se imprime la excepción. Se define el número máximo de clientes y se inicializan 3 arreglos con el espacio reservado para guardar “algo”. Un arreglo para los clientes, un arreglo para los objetos socket, y otro arreglo para las tuplas de “addr” que traen el host y port para los clientes.



En un ciclo while se itera tratando de aceptar las conexiones. La función `accept()` devuelve un objeto socket que sirve para enviar y recibir data en la conexión, y una tupla “addr” que es la dirección host y port desde el otro lado de la conexión.

Se crea el hilo y se le pasan como datos el número de iteración para su nombre, y una tupla con el objeto socket y la tupla "addr".

El contador aumenta. Dentro de la función del hilo se entra en un loop While.



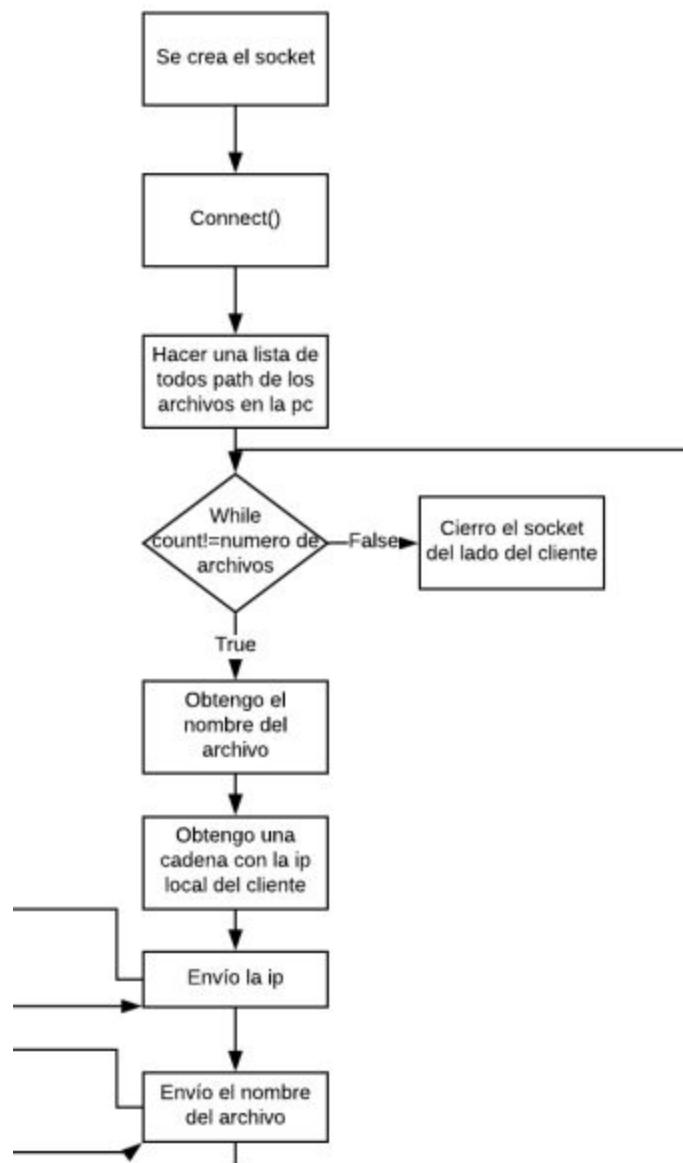
En el loop while se itera mientras se reciban datos.

El servidor recibe un string en bytes con la ip local del cliente, recibe el nombre del archivo, y si no existe todavía crea una carpeta con esos datos. Para cada cosa que recibe envía una confirmación.

El servidor crea un puntero para file de tipo append que va a leer bytes.

Itera 4 veces para recibir los 4 bloques, y como hay un límite de tamaño para el protocolo TCP, divide los bloques y los lee de 1024 en 1024 bytes en un While.

### Ciente:



Se crea el socket del lado del cliente, se conecta al servidor, se hace una lista con todos los path de los archivos.

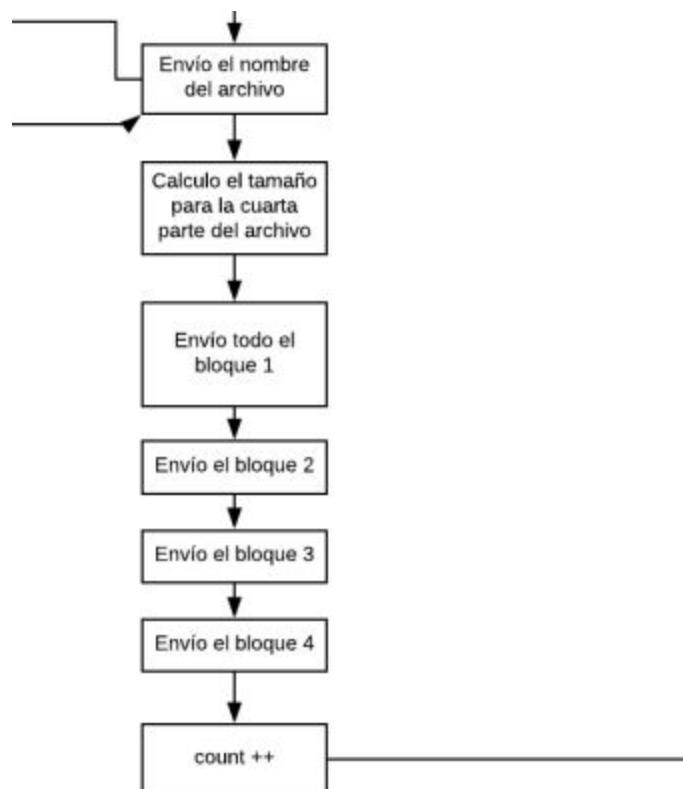
Se itera por cada elemento en la lista.

Dentro del while, se obtiene el nombre del archivo en el path, se codifica a bytes para su envío a través del socket. Se calcula el tamaño del archivo a través de una función que itera en cada bytes del archivo.

Se obtiene un string de la ip a través de una función.

Se envía el string en forma de bytes a través del socket y se espera por una confirmación.

Se envía el nombre del archivo en forma de bytes a través del socket y se espera por la confirmación.



Se calcula el tamaño de los bloques y abro un puntero a archivo que leerá en bytes los archivos.



Envío el bloque1 y espero confirmación, y así para todos los demás, aumento el contador y regreso al while para enviar el siguiente archivo.

## Conclusiones

En el caso de esta práctica, el socket era orientado a conexión, lo cual hace el envío de datos más seguro pero más lento.

Con el protocolo TCP se garantiza la transmisión de todos los octetos sin errores ni omisiones y se garantiza que todo octeto llegará a su destino en el mismo orden en que se ha transmitido. Estas propiedades son muy importantes para garantizar la corrección de los programas que tratan la información.

## Referencias

[1] <https://www.ecured.cu/Socket>