UNIVERSIDAD LUTERANA SALVADOREÑA

Faculta de ciencia del hombre y la naturaleza

Catedrático: Manuel Flores Villatoro

Asignatura: Redes I

Proyecto: Cliente Servidor

Estudiante: Leonel Osmin Hernández López

N° de carnet: HL01132411

SAN SALVADOR 28 DE MAYO DE 2016

Índice

Introducción.

En este presente proyecto daremos a conocer el desarrollo de la aplicación cliente servidor y sobre su implementación en su funcionamiento, ya que se puede tener una comunicación entre en el cliente-servidor, ya que estos pueden interactuar enviándose mensaje y el cliente le podrá mandar archivos al servidor quien los va a recibir.

Esto es posible gracias a un lenguaje de programación y al desarrollo de las aplicaciones cliente-servidor, con el lenguaje python esta desarrollado dicho programa y a la misma ves utilizando la librería socket, los socket vienen siendo una clase de tecnología robusta por la forma hay que ponerse a investigar y practicar sobre los distintas opciones que ofrece dicha tecnología.

La aplicación consiste en que el servidores estará a la espera que un cliente se conecte para así corresponder el llamado del cliente a viendo comunicado el cliente con el servidor por medio de un puerto y de una dirección IP el cliente podrá interactuar el con el servidor y los mismo sucede con el servidor que debe de estar con el mismo puesto que esta el cliente a la espera del llamado del cliente.

La red cliente-servidor es una red de comunicaciones en la cual los clientes están conectados a un servidor, en el que se centralizan los diversos recursos y aplicaciones con que se cuenta; y que los pone a disposición de los clientes cada vez que estos son solicitados. Esto significa que todas las gestiones que se realizan se concentran en el servidor, de manera que en él se disponen los requerimientos provenientes de los clientes que tienen prioridad.

La separación entre cliente y servidor es una separación de tipo lógico, donde el servidor no se ejecuta necesariamente sobre una sola máquina ni es necesariamente un sólo programa. Los tipos específicos de servidores incluyen los servidores web, los servidores de archivo, los servidores del correo, etc. Mientras que sus propósitos varían de unos servicios a otros, la arquitectura básica seguirá siendo la misma.
Aclareciendo lo ya señalado lo que se pretende es desarrollar una aplicación cliente servidor. Que nos permita tener una comunicación del cliente hacia el servidor, para llevar a cabo este proceso utilizaremos la tecnología de los socket, esta tecnología nos permite que haiga una comunicación entre el cliente y el servidor.

Objetivos.

Objetivo General.

-Dar a conocer la aplicación cliente-servidor y demostrar que interactúan entre ellas por medio de envío de mensaje y archivos.

Objetivo Específico.

- Demostrar el funcionamiento de la aplicación cliente-servidor con su funcionalidad que realiza.

- Establecer la comunicación entre el cliente-servidor, y esta pueda interactuar entre el cliente-servidor.

- Utilizando los protocolos de comunicación de los socket es posible estable comunicación entre el cliente-servidor.

- Aplicando las herramientas de desarrollo de python y utilizando la librería socket es posible la interacción de las aplicaciones

Descripción del producto
Dar a conocer la aplicación cliente-servidor donde el agente que solicita la información se denomina cliente, mientras que le componente software que responde a esa solicitud es el que se conoce como servidor.
Es un proceso habitual el cliente será el que inicie el intercambio de información solicitando al servidor como el servidor estará a la espera que el cliente se conecte o comunique con el. Además de la transferencia de datos real.

Modelo Cliente Servidor
El modelo cliente-servidor consiste básicamente en un cliente que realiza peticiones a otro programa (el servidor) que le da respuesta.
La tecnología denominada Cliente -Servidor es utilizada por todas las aplicaciones de Internet/Intranet. Un cliente funciona en su ordenador local, se comunica con el servidor remoto, y pide a éste información. El servidor envía la información solicitada.

CARACTERISTICAS DE LA ARQUITECTURA CLIENTE-SERVIDOR :

El Cliente y el Servidor pueden actuar como una sola entidad y también pueden actuar como entidades separadas, realizando actividades o tareas independientes.

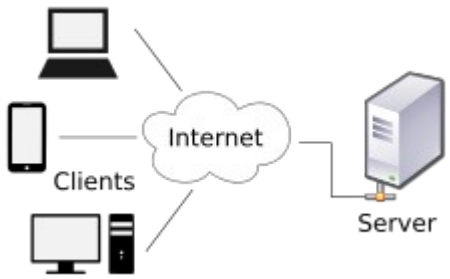
Las Funciones de Cliente y Servidor pueden estar en plataformas separadas, o en la misma plataforma.

Espera y recibe las respuestas del servidor.

Normalmente interactúa directamente con los usuario.

La arquitectura cliente-servidor es un modelo de aplicación distribuida en el que las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y los demandantes, llamados clientes. Un cliente realiza peticiones a otro programa, el servidor, quien le da respuesta. Esta idea también se puede aplicar a programas que se ejecutan sobre una sola computadora, aunque es más ventajosa en un sistema operativo multiusuario distribuido a través de una red de computadoras.

Ejemplos de aplicaciones computacionales que usen el modelo cliente-servidor son el Correo electrónico, un Servidor de impresión y la World Wide Web.



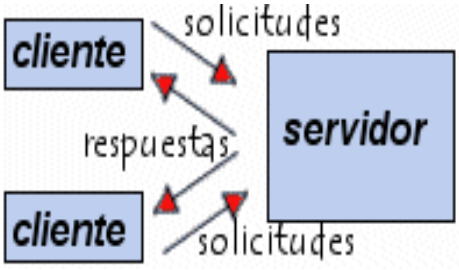
En esta arquitectura la capacidad de proceso está repartida entre los clientes y los servidores, aunque son más importantes las ventajas de tipo organizativo debidas a la centralización de la gestión de la información y la separación de responsabilidades, lo que facilita y clarifica el diseño del sistema.

La separación entre cliente y servidor es una separación de tipo lógico, donde el servidor no se ejecuta necesariamente sobre una sola máquina ni es necesariamente un sólo programa. Los tipos específicos de servidores incluyen los servidores web, los servidores de archivo, los servidores del correo, etc. Mientras que sus propósitos varían de unos servicios a otros, la arquitectura básica seguirá siendo la misma.

La red cliente-servidor es una red de comunicaciones en la cual los clientes están conectados a un servidor, en el que se centralizan los diversos recursos y aplicaciones con que se cuenta; y que los pone a disposición de los clientes cada vez que estos son solicitados. Esto significa que todas las gestiones que se realizan se concentran en el servidor, de manera que en él se disponen los requerimientos provenientes de los clientes que tienen prioridad.

Funcionamiento del sistema cliente/servidor

Un sistema cliente/servidor funciona tal como se detalla en el siguiente diagrama:



- El cliente envía una solicitud al servidor mediante su dirección IP y el puerto, que está reservado para un servicio en particular que se ejecuta en el servidor.
- El servidor recibe la solicitud y responde con la dirección IP del equipo cliente y su puerto.

Las funciones que lleva a cabo el proceso cliente se resumen en los siguientes puntos:

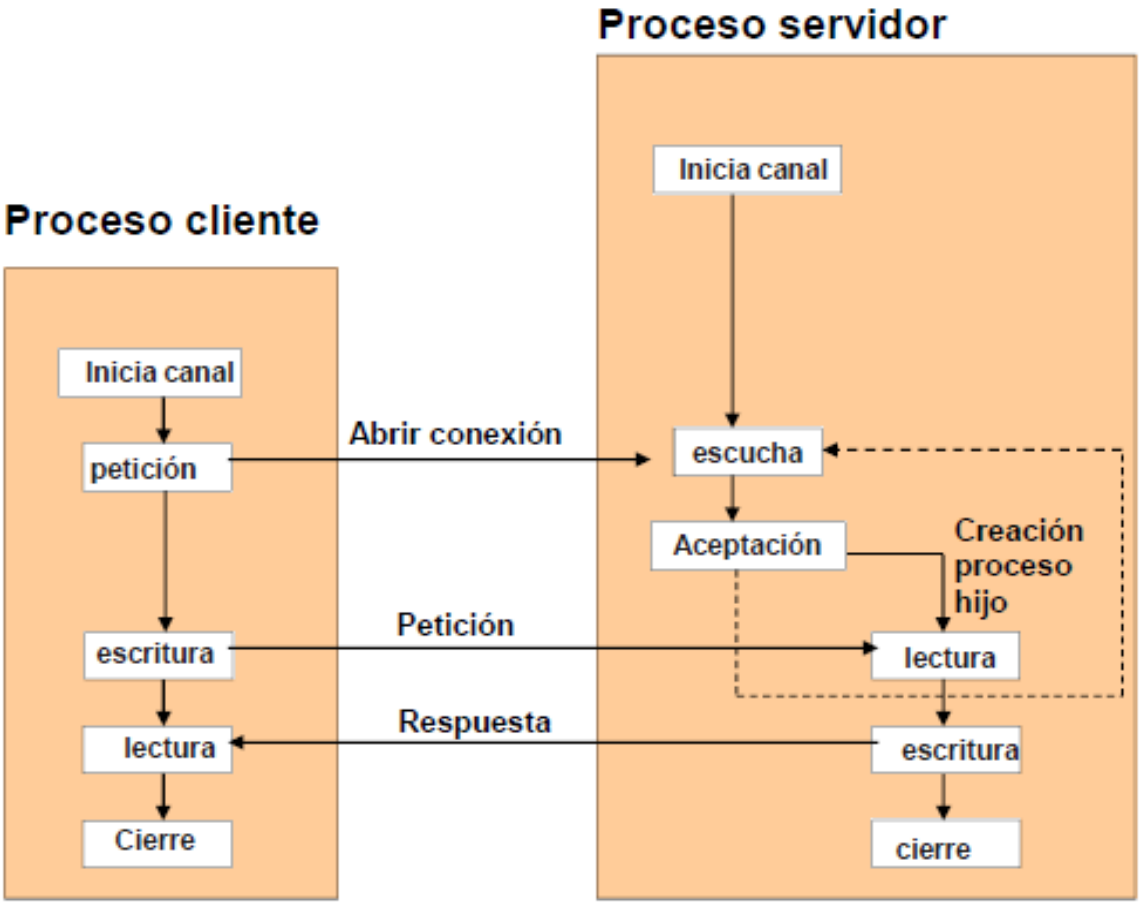
• Administrar la interfaz de usuario.
• Interactuar con el usuario.
• Procesar la lógica de la aplicación y hacer validaciones locales.
• Recibir resultados del servidor.

Las funciones que lleva a cabo el proceso servidor se resumen en los siguientes puntos:

- Administración de transacciones distribuida.
- Comunicación a sincrónica de programa, normalmente a través de colas de mensajes .
- Un modelo de activación de objetos oportuno.

- Servicios de detección de errores y estado de las aplicaciones .

Diagrama cliente servidor



El programa cliente/servidor utilizando el módulo socket en python. El módulo socket (canal de comunicación) es utilizado para comunicar un programa cliente con un programa servidor en una red (también se puede utilizar en el mismo equipo).

Python proporciona dos niveles de acceso a los servicios de red. En un nivel bajo, puede acceder a la ayuda básica de conectores en el sistema operativo subyacente, lo que le permite implementar clientes y servidores para ambos protocolos de conexión y orientados a conexión. Python también tiene bibliotecas que proporcionan mayor nivel de acceso a los protocolos de red de nivel de aplicación específicos.

Los enchufes son los extremos de un canal de comunicación bidireccional. Zócalos pueden comunicarse dentro de un proceso, entre los procesos en la misma máquina, o

entre procesos en diferentes continentes.

Zócalos pueden ser implementados en un número de diferentes tipos de canales: **sockets** de dominio Unix, TCP, UDP, y así sucesivamente. La biblioteca *socket* proporciona clases específicas para el manejo de los medios de transporte comunes, así como una interfaz genérica para manejar el resto.

Los **socket** se pueden configurar para que actúen como un servidor y así poder escuchar los mensajes entrantes, o conectarse a otras aplicaciones como clientes. Luego de que ambos extremos de un socket TCP/IP están conectados, la comunicación es bidireccional. Para la implementación de nuestro cliente servidor utilizaremos un lenguaje de programación determinado que nos permita desarrollar las dos aplicaciones las cuales son la del cliente y la del servidor, pero para ser esto posible utilizaremos la tecnología socket que nos permite comunicarnos de una maquina distinta.

Socket
Los ***sockets*** son mecanismos de comunicación entre procesos que permiten que un proceso hable (emita o reciba información) con otro proceso incluso estando en distintas máquinas.

*Una forma de conseguir que dos programas se transmitan datos.

* Un **socket** no es más que un "canal de comunicación" entre dos programas que corren sobre ordenadores distintos o incluso en el mismo ordenador.

* Desde el punto de vista de programación, un **socket** no es más que un "fichero" que se abre de una manera especial.

Tipos de Sockets

Existen básicamente dos tipos:

Los no orientados a conexión

- El programa de aplicación da la fiabilidad

Los orientados a conexión.

- Comunicaciones fiables
- Circuito Virtual

Tipos de Sockets

*Un socket queda definido por una dirección IP, un protocolo y un número de puerto.
* En el caso concreto de TCP-IP, un socket se define por una dupla Origen-Destino.
*Tanto el origen como el destino vienen indicados por un par (IP, PUERTO).

Los métodos de conexión de cliente-servidor que se utilizan para creación y la aplicación y conexión.

Métodos de servicios socket

Método	Descripción
s.bind()	Esta dirección método de enlace (nombre de host, el par número de puerto), al enchufe.
s.listen()	Este método establece y se inicia escucha TCP lista de cliente.
s.accept()	Esta aceptar pasivamente conexión de cliente TCP, esperando a que llegue la conexión (bloqueo).

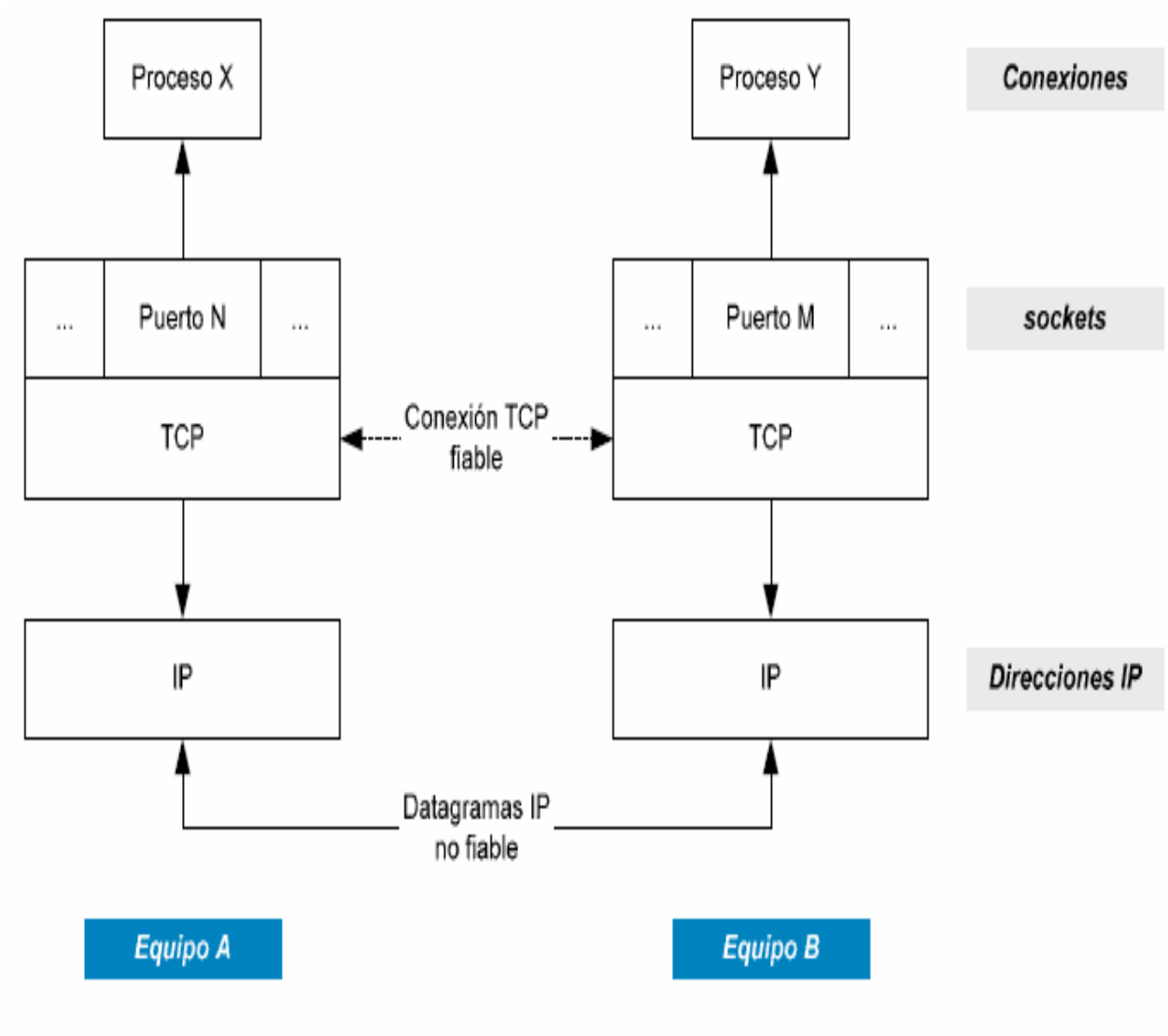
Método del cliente socket

Método	Descripción
s.connect()	Este método inicia de forma activa la conexión del servidor TCP.

General método de socket

Método	Descripción
s.recv()	Este método recibe el mensaje TCP
s.send()	Este método transmite el mensaje TCP
s.recvfrom()	Este método recibe el mensaje UDP
s.sendto()	Este método transmite el mensaje UDP
s.close()	Este método cierra socket

Diagrama de socke



Arquitectura de la conexión.

Servidor:
• Está ejecutándose y esperando a que otro quiera conectarse a él.
• Nunca da "el primer paso" en la conexión.
• Es el que "sirve" información al que se la pida.

Cliente:
• Es el programa que da el “primer paso” en la conexión.
• En el momento de ejecutarlo o cuando lo necesite, intenta conectarse al servidor.
• Es el que solicita información al **servidor**.

Conexión
Para poder realizar la conexión entre ambos programas (cliente y servidor) es necesario conocer:

• **Dirección IP** del servidor.
• **Servicio** que queremos crear / utilizar.

Programación de Sockets

En primera instancia, para programar un socket, escogimos un lenguaje que este caso fue el lenguaje de Python por ser unos de los lenguajes de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible. Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma.

Cliente

Vamos a escribir un programa cliente muy simple que se abre una conexión a un puerto determinado y equipo dado
El Socket.connect (hosname, puerto) abre una conexión TCP con hostname en el puerto. Una vez que tenga una toma abierta, se puede leer de él como cualquier otro objeto. Cuando haya terminado, no olvide cerrarlo, como era de cerrar un archivo.

El código del programa Cliente:

-HOST y PORT: Se inicializan estas variables con valores fijos, ya que para fines prácticos, lo conectaremos al servidor con un IP y puerto prefijado (dentro de un red doméstica).

-socket(AF_INET, SOCK_STREAM): función que crea el socket. AF_INET hace alusión a la comunicación en máquinas distintas. SOCK_STREAM indica que se transportaran los datos vía TCP.
-cliente.connect(HOST, PORT): conecta al servidor prefijado.

-cliente.send(message): envía el mensaje escrito por el cliente.

-cliente.recv(): recibe un mensaje proveniente del servidor.

-cliente.close(): Cierra la conexión.

servidor.py | cliente.py

```
1  #https://translate.google.com.sv/translate?hl=es-419&sl=en&u=ht
2  #https://underc0de.org/foro/python/tutorial-envio-de-archivos-u
3  from socket import*
4  HOST = 'localhost'
5  PORT =6973
6  cliente=socket(AF_INET, SOCK_STREAM)
7
8  ARCHIVO="al.pdf"
9
10 cliente.connect((HOST, PORT))
11
12 with open(ARCHIVO, "rb") as archivo:
13     buffer = archivo.read()
14
15 while True:
16     message=raw_input("ESCRIBIR EL MENSAJE AL SERVIDOR: ")
17     cliente.send(message)
18     print "ESPERANDO REPUESTA DEL SERVIDOR: "
19     reply=cliente.recv(1624)
20     print"RECIBIENDO MENSAJE DEL SERVIDOR:  ",repr(reply)
21
22
23     print"ENVIANDO BUFFER:..."
24     cliente.send(str(len(buffer)))
25
26     recibido=cliente.recv(10)
27     if recibido=="OK":
28         for byte in buffer:
29             cliente.send(byte)
30         break
31 cliente.close()
32
```

Servidor
Para escribir los servidores, utilizamos la función de socket disponibles en el módulo de tubo para crear un objeto socket. Un objeto socket se utiliza para llamar a otras funciones para configurar un servidor de socket.

Ahora llamar a la función bind (nombre de host, puerto) para especificar un puerto para su servicio en el equipo dado.

A continuación, llamar al método accept del objeto devuelto. Este método espera hasta que un cliente se conecta al puerto que ha especificado, y luego devuelve un objeto conexión que representa la conexión a ese cliente.

El código del programa Servidor:

socket(AF_INET, SOCK_STREAM): Lo mismo que en el cliente, AF_INET para conectarse a otra máquina, y SOCK_STREAM para usar transporte vía TCP.

-**servidor.bind(HOST, PORT):** Se le asigna una dirección y un puerto al servidor. Al igual que antes estos ya están prefijados, sin embargo, HOST no es necesario en este caso, solo el puerto.

-servidor.listen(1): Indica cuantas conexiones aceptara. En este caso aceptaría una sola conexión.

-servidor.accept(): Acepta la conexión entrante. Esta variable es una tupla y es igualada a las variables **conn** y **addr**.

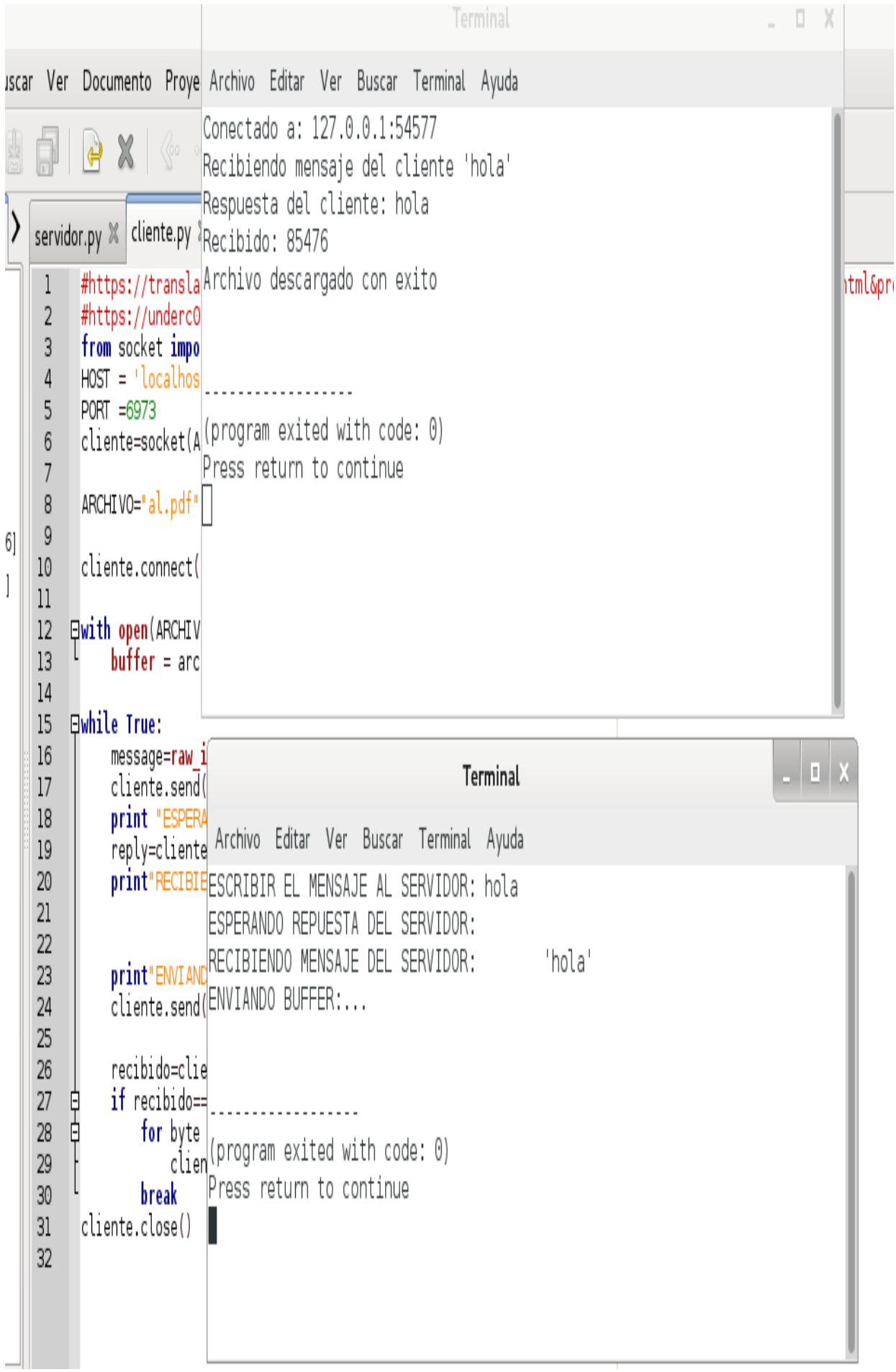
-conn.recv(1024),Conn.sendall(reply), con.close(): .**recv** recibe lo que le envío el **cliente**, **.sendall(reply)** envía a todos lo que está en **reply.**

-conn.close() cierra la conexión.

ervidor.py × cliente.py ×

```
1  from socket import*
2  HOST = ''
3  PORT = 6973
4  servidor= socket(AF_INET, SOCK_STREAM)
5
6
7  servidor.bind((HOST, PORT))
8  servidor.listen(5)
9
10 # Aceptamos conexiones
11 sck, addr = servidor.accept()
12 print "Conectado a: {0}:{1}".format(*addr)
13 while 1:
14     data=sck.recv(1024)
15     print"Recibiendo mensaje del cliente",repr(data)
16     reply = raw_input("Respuesta del cliente: ")
17     sck.sendall(reply)
18
19
20     # Recibimos la longitud que envia el cliente
21     recibido = sck.recv(1024).strip()
22     if recibido:
23         print "Recibido:", recibido
24     if recibido.isdigit():
25         sck.send("OK")
26         buffer = 0
27         with open("archivo", "wb") as archivo:
28             while (buffer <= int(recibido)):
29                 data = sck.recv(1)
30                 if not len(data):
31                     break
32                 archivo.write(data)
33                 buffer += 1
34             if buffer == int(recibido):
35                 print "Archivo descargado con exito"
36             else:
37                 print "Ocurrio un error/Archivo incompleto"
38         break
39 sck.close()
40
41
```

Al ejecutar el servidor y el cliente podremos observar que el servidor esta a la espera del cliente que se se comunique, en esta imagen podremos observar que se pueden comunicar entre ellos localmente.


Terminal
Archivo Editar Ver Buscar Terminal Ayuda
Conectado a: 127.0.0.1:54577
Recibiendo mensaje del cliente 'hola'
Respuesta del cliente: hola
Recibido: 85476
Archivo descargado con exito
(program exited with code: 0)
Press return to continue
servidor.py
cliente.py
#https://transla
#https://underc0
from socket impo
HOST = 'localhos
PORT =6973
cliente=socket(A
ARCHIVO="al.pdf"
cliente.connect(
with open(ARCHIV
buffer = arc
while True:
message=raw_i
cliente.send(
print "ESPERA
reply=cliente
print"RECIBIE
print"ENVIAND
cliente.send(
recibido=clie
if recibido==
for byte
clien
break
cliente.close()
Terminal
Archivo Editar Ver Buscar Terminal Ayuda
ESCRIBIR EL MENSAJE AL SERVIDOR: hola
ESPERANDO REPUESTA DEL SERVIDOR:
RECIBIENDO MENSAJE DEL SERVIDOR: 'hola'
ENVIANDO BUFFER:...
(program exited with code: 0)
Press return to continue

Nota.

La comunicación que tiene el cliente-servidor es localmente, para poder establecer comunicación de un equipo a otro se de hacer unos cambios en el código del cliente, el código que hay que cambian es la variable **HOST=' '**, que esta ocasión es la dirección de localhost, esto es sustituido por una dirección IP que tiene el servidor donde esta montado. Ademas para poder establecer la comunicación de un equipo a a otro donde se encuentra el cliente y donde el servidor esta montado en otro equipo, necesitamos un switch cables UTP, y hacer una configuración de IP estática en el equipo donde estará montado el servidor, y de esta forma se estable la comunicación del cliente-servidor.

Recomendaciones

-para poder realizar una conexión entre el servidor y cliente hay tener en cuenta que se tiene que configurar una IP estática al servidores donde este estará montado y a la espera que el cliente se conecte a él .

-Utilizar el lenguaje de programación python y importar la librería socket.

-realizar una conexión utilizando la librería socket y realizar una previo programa que consiste en hacer una comunicación entre el cliente-servidor.

Conclusión.

Al finalizar este trabajo llegamos a la conclusión de que el modelo cliente servido nos puede permitir una comunicación de mensajería entre el cliente y el servidor y a la misma ves poderle mandar archivos que se le pueda enviar la servidor y el los pueda recibir y los pueda almacenar localmente.

Los servidores son equipos que proporcionan servicios y datos a los equipos cliente. Los servidores de una red realizan diversas tareas complejas.

Los servidores proporcionan recursos de compartición de archivos desde una ubicación centralizada. Cuando un cliente envía una solicitud de datos al servidor de archivos, se descarga en el equipo que realiza la petición.

Por ejemplo, cuando abrimos una aplicación de procesamiento de texto, ésta se ejecuta en nuestro equipo y el documento almacenado en el servidor de archivos se descarga en la memoria de nuestro equipo para que podamos editarlo o utilizarlo localmente.

Bibliografía.

https://es.wikipedia.org/wiki/Cliente-servidor

http://developeando.net/sockets-python/

https://underc0de.org/foro/python/tutorial-envio-de-archivos-usando-sockets/

http://welinforcto.blogspot.com/2011/09/caracteristicas-de-la-arquitectura.html

http://juanitocuirindichapio.blogspot.com/2011/09/diferentes-caracteristicas-de-la_09.html