



BUAP



Benemérita Universidad Autónoma De Puebla

Facultad De Ciencias De La Computación

Periodo: Otoño 2020. **Nrc:** 15700. **Sección:** 001.

Materia: Sistemas Operativos I.

Docente: Dr. Luis Enrique Colmenares Guillen.

Actividad: Práctica 11 protocolos TCP y UDP (sockets).

Fecha: 16/Noviembre/2020.

Integrantes:

Hernandez Paredes Alejandro	201740063.
Lara Reyes Cristhian Yair	201730849.
Moctezuma Vazquez Eva	201758475.

Realizar la comunicación remota utilizando un chat con el uso de sockets (No JAVA, utilizar un lenguaje scripting):

1. Crear un servidor que acepte n conexiones clientes, los n clientes van a ser : orientado a conexión y no orientado a conexión.
2. Utilizar protocolos de transporte TCP y UDP.
3. Crear una interfaz gráfica por cada cliente se debe distinguir la diferencia entre clientes utilizando ip cliente, etiqueta de cliente, puertos. (es decir que realice la distinción entre cada cliente) .
4. Generar conexiones punto a punto, utilizando el servidor, un cliente se podra conextra con otro cliente utilizando el paradigma cliente-servidor, utilizando una direccion IP realizando conexiones locales y remotas
5. Empaquetar el codigo dividido en carpetas para el cliente y servidor, generar un reporte y subirlo a BB.
6. Generar un video con pantallas de salida mostrando el funcionamiento del chat (puntos 1,2,3 y 4).

Introduccion.

En este reporte se podra ver la salida y el funcionamiento de la aplicación “chat cliente servidor” elaborada en python con sockets y la interfaz elaborada con la librería Tkinter.

Conceptos basicos.

Las aplicaciones RPC, por naturaleza, tienen una gran cantidad de archivos auxiliares cuyos contenidos y relaciones a veces pueden oscurecer su funcionalidad. En una aplicación basada en RPC, es fácil perder el contacto y el control de la mecánica del proceso de comunicación. Parecería que lo que se necesita es una extensión del paradigma básico de lectura / escritura con la inclusión de suficiente semántica de red para permitir que procesos no relacionados en diferentes hosts se comuniquen como si estuvieran leyendo y escribiendo en un archivo local.

Para comprender cómo funcionan los sockets, se necesita una comprensión básica de algunos de los detalles de las comunicaciones de proceso en un entorno de red y su terminología asociada.

Cada host en una red tiene, como mínimo, dos direcciones únicas

- MAC 48 bits ejemplo 00: B0: D0: AB: 7C: 96
- IP=protocolo de internet (existen IPV4=32 bits O IPV6=128bits) ejemplo ipv4 127.0.0.1
Máscara: 255.0.0.0

Si bien las direcciones IP son una forma práctica de hacer referencia a un host específico, a menudo asignamos una dirección IP con puntos a una notación simbólica más fácil de entender utilizando el Sistema de nombres de dominio (DNS). server DNS{2607:f8b0:4012:805::200e==google.com}

Algunos protocolos, como TCP, precedieron al modelo OSI y, por lo tanto, no se asignan claramente a sus capas. TCP / IP logra la misma funcionalidad con cuatro capas conceptuales: aplicación, transporte, Internet e interfaz de red (enlace de datos). Las capas de transporte y red de TCP / IP son aproximadamente equivalentes a las capas de transporte y acceso a la red del modelo OSI, excepto que TCP / IP admite UDP, un protocolo poco confiable.

- Protocolo de control de transmisión TCP. TCP es confiable, full duplex y orientado a la conexión. Los datos se transmiten como flujo de bytes.
- Protocolo de Internet IP. Proporciona entrega de paquetes. TCP, UDP e ICMP suelen llamar IP.
- Protocolo de resolución de dirección ARP / RARP / dirección inversa. Estos protocolos se utilizan para resolver el direccionamiento de Internet / hardware.
- Protocolo de datagramas de usuario UDP. UDP no es confiable, dúplex completo y sin conexión. Los datos se transmiten como una serie de paquetes.
- Protocolo de mensajes de control de Internet ICMP. Utilizado para manejo de errores y control de flujo.

Para que los procesos se comuniquen en un entorno de red, los datos deben transmitirse y recibirse. Podemos considerar que los datos comunicados están en una secuencia (es decir, una secuencia de bytes) o en formato de datagrama. Los datagramas son paquetes pequeños y discretos que, en un nivel bruto, contienen información de encabezado (como direcciones), datos e información de cola (corrección de errores, etc.). Como los datagramas son de tamaño pequeño, las comunicaciones entre procesos pueden consistir en una serie de datagramas.

Cuando creamos un socket, su tipo determinará cómo se llevarán a cabo las comunicaciones entre los procesos que utilizan el socket. Los enchufes deben ser del mismo tipo para comunicarse

TCP Stream enchufes. Estos enchufes son confiables. Cuando se utilizan estos sockets, los datos se entregan en orden, en la misma secuencia en la que se enviaron. No hay duplicación de datos y, por lo general, existe alguna forma de verificación de errores y control de flujo. Los enchufes de flujo permiten la comunicación bidireccional (dúplex completo).

UDP Datagram sockets potencialmente poco fiables. Los datos recibidos pueden estar desordenados, admiten comunicaciones bidireccionales, se consideran sin conexión, no existe una conexión lógica entre los procesos de envío y recepción. Cada datagrama se envía y procesa de forma independiente, no hay control de flujo, el control de errores, los paquetes de datagramas son normalmente pequeños y de tamaño fijo.

Funcionamiento chat cliente-servidor.


Para el correcto funcionamiento de las aplicaciones es necesario tener instalado python 3 en nuestro equipo así como tener la librería de Tkinter.

Imagen 1. Estando posicionado en nuestra carpeta donde tenemos nuestro archivo de servidor con extensión “.py” que es nuestro caso se llama “server.py” tendremos que ejecutar nuestro scrip en una terminal con el comando “Python3 server.py” como se puede ver a continuacion.

A screenshot of a terminal window with a black background and red, yellow, and green window control buttons at the top left. The title bar is red and contains the text 'aw@aw: ~/Documentos/S'. The terminal shows the command 'python3 server.py' being entered at the prompt 'aw@aw: ~/Documentos/S.O I/Unidad IV/actividad_11 70x35\$'. The cursor is at the end of the command line.

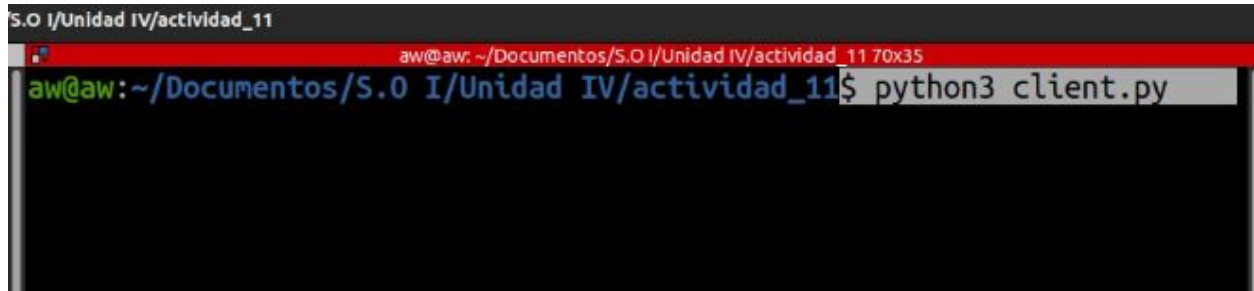
```
aw@aw: ~/Documentos/S.O I/Unidad IV/actividad_11 70x35$ python3 server.py
```

Imagen 2. Una vez ejecutado el scrip de server.py podremos ver que nos imprime en terminal “Esperando conexiones” y con esto ya tendremos el servidor listo para que los clientes se puedan conectar.

A screenshot of a terminal window with a black background and red, yellow, and green window control buttons at the top left. The title bar is red and contains the text 'aw@aw: ~/Documentos/S'. The terminal shows the command 'python3 server.py' being entered at the prompt 'aw@aw: ~/Documentos/S.O I/Unidad IV/actividad_11 70x35\$'. Below the command, the output 'Esperando conexiones' is displayed. The cursor is at the end of the command line.

```
aw@aw: ~/Documentos/S.O I/Unidad IV/actividad_11 70x35$ python3 server.py
Esperando conexiones
```

Imagen 3. Estando en una terminal distinta a la de nuestro servidor nos moveremos a la ruta donde se encuentre nuestra aplicación de nuestro que en nuestro caso se llama “client.py”, una vez estando en la ruta correcta procederemos a ejecutar nuestro scrip con el comando “python3 client.py”.

A terminal window with a black background and a red title bar. The title bar contains the text "S.O I/Unidad IV/actividad_11" on the left and "aw@aw: ~/Documentos/S.O I/Unidad IV/actividad_11 70x35" on the right. The terminal prompt is "aw@aw:~/Documentos/S.O I/Unidad IV/actividad_11\$". The command "python3 client.py" has been entered and is highlighted in grey.

```
S.O I/Unidad IV/actividad_11
aw@aw: ~/Documentos/S.O I/Unidad IV/actividad_11 70x35
aw@aw:~/Documentos/S.O I/Unidad IV/actividad_11$ python3 client.py
```

Imagen 4. Podremos observar que se nos abrirá una interfaz donde se nos solicitará un nombre de usuario y contraseña, tendremos la opción de iniciar sesión en el botón de “Login” en caso de que ya no hayamos registrado y en caso contrario nos registramos en el botón de “Register”.

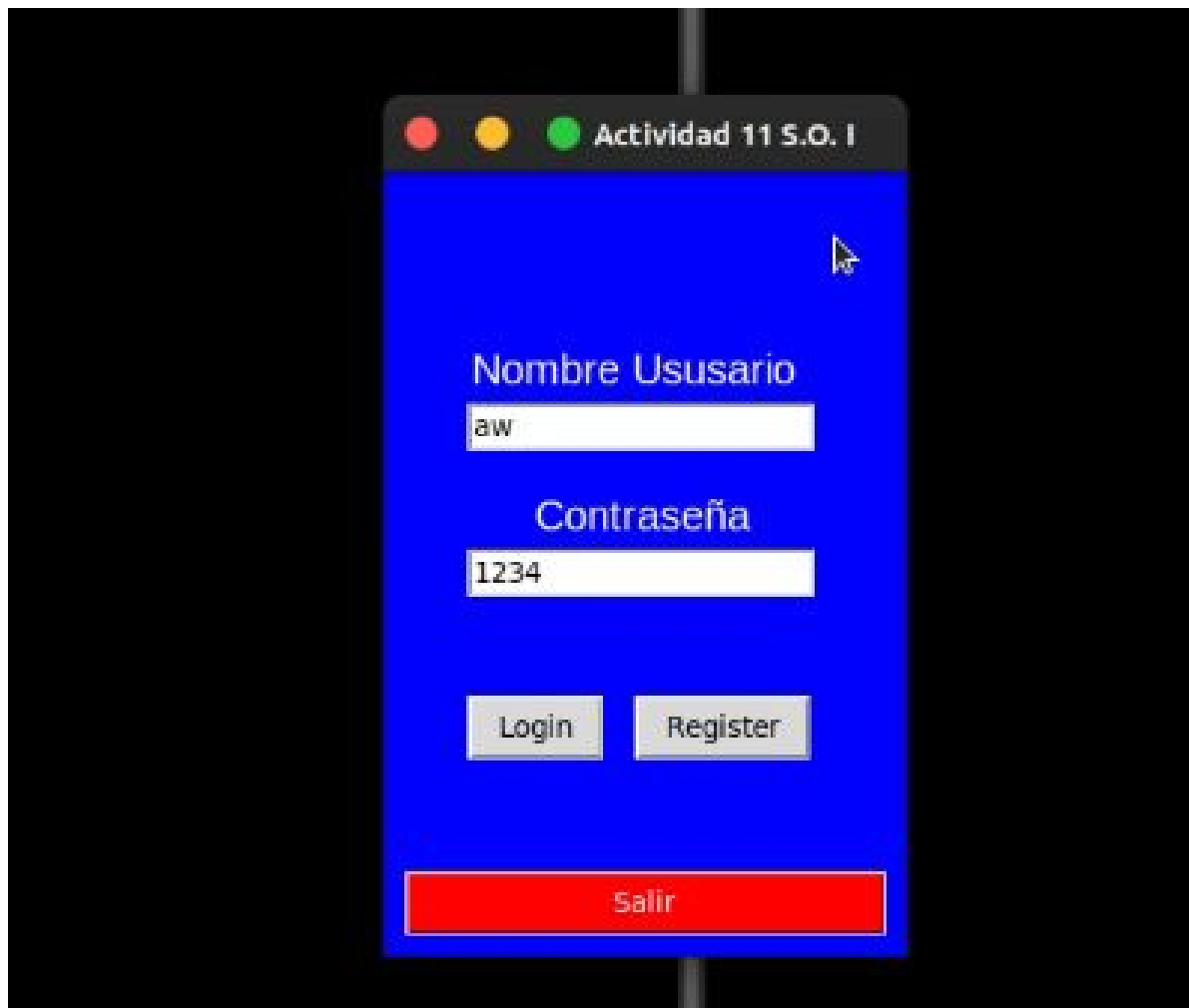


Imagen 5. Estando en otra máquina, en nuestro caso al no tener otra computadora físicamente hemos tenido que virtualizar un nuevo sistema operativo para poder hacer la prueba. En una máquina virtual instalamos la distribución de lubuntu al ser muy ligera. En la máquina virtual solo tenemos el archivo de nuestro cliente y configurado el archivo como la máquina respectivamente solo ejecutamos nuestro scrip en la terminal.

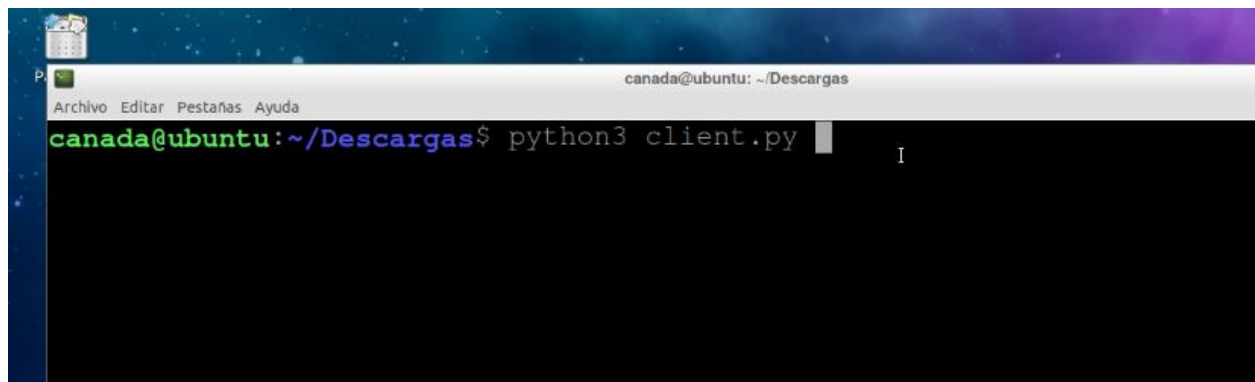


Imagen 6. Como podemos ver se nos ha abierto una interfaz de nuestro cliente y nos registramos como pepe.

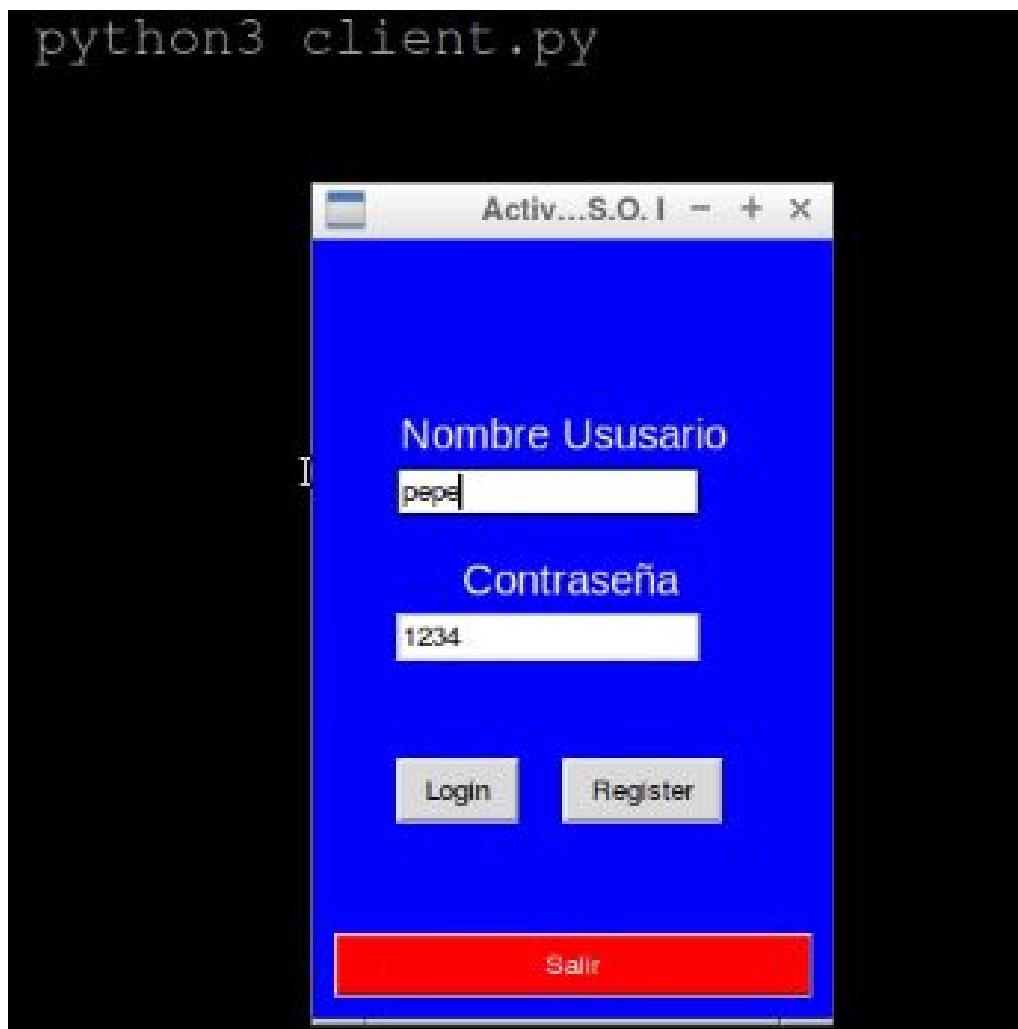


Imagen 7. A continuación se muestra una captura de la dos interfaces de cliente, tanto de la máquina virtual como la nuestro sistema.

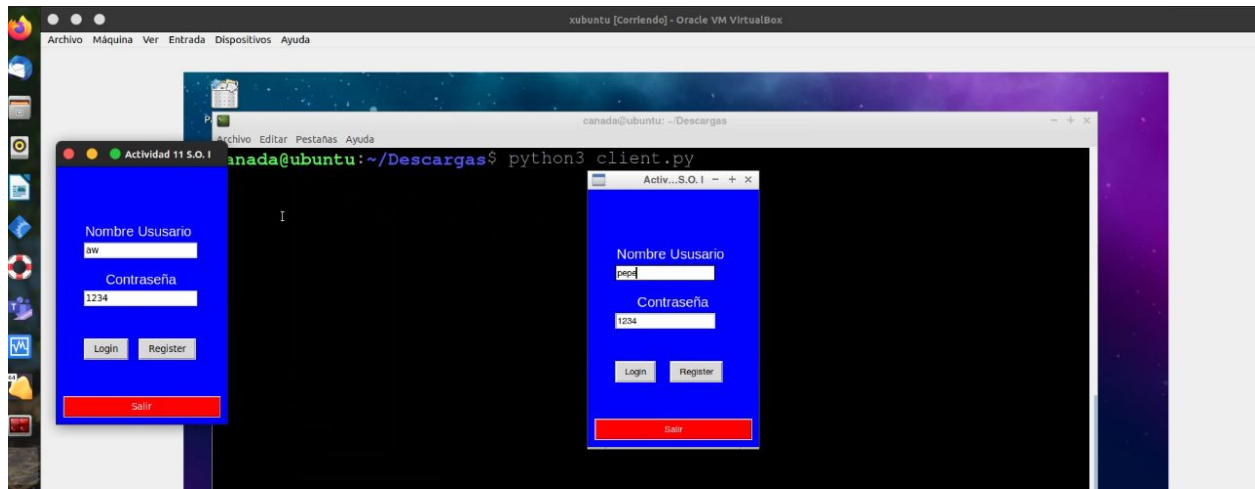


Imagen 8. Una vez iniciada la sesión el equipo físico podremos ver que es una interfaz bastante amigable donde a simple vista tendremos la opción de elegir el protocolo “TCP” o “UDP”, al usuario con el que haremos la conexión y nos indica el usuario que inicio session, el puerto y la ip de equipo.

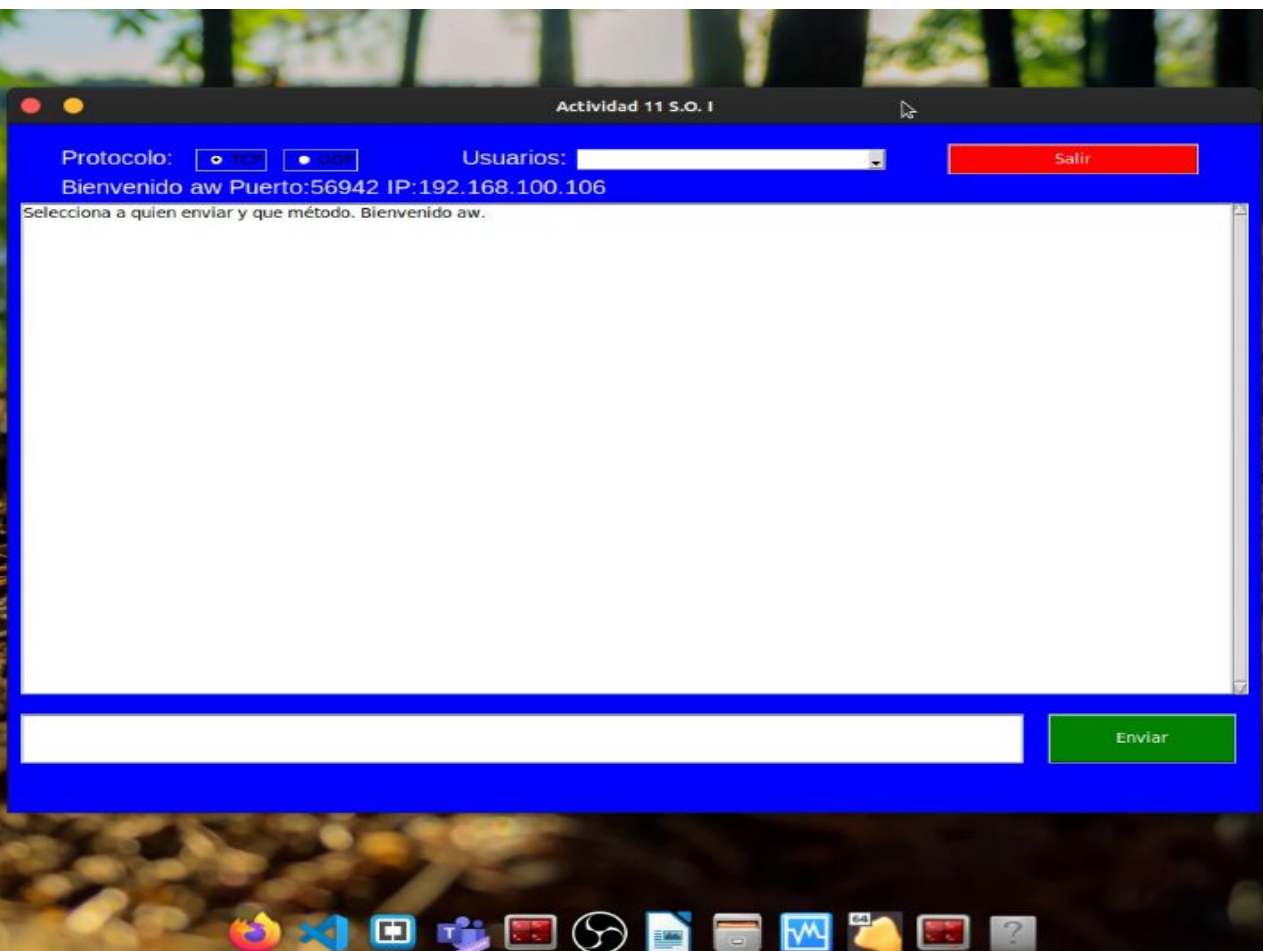


Imagen 9. Del lado de la máquina virtual tendremos la interfaz con la sesión iniciada de pepe.

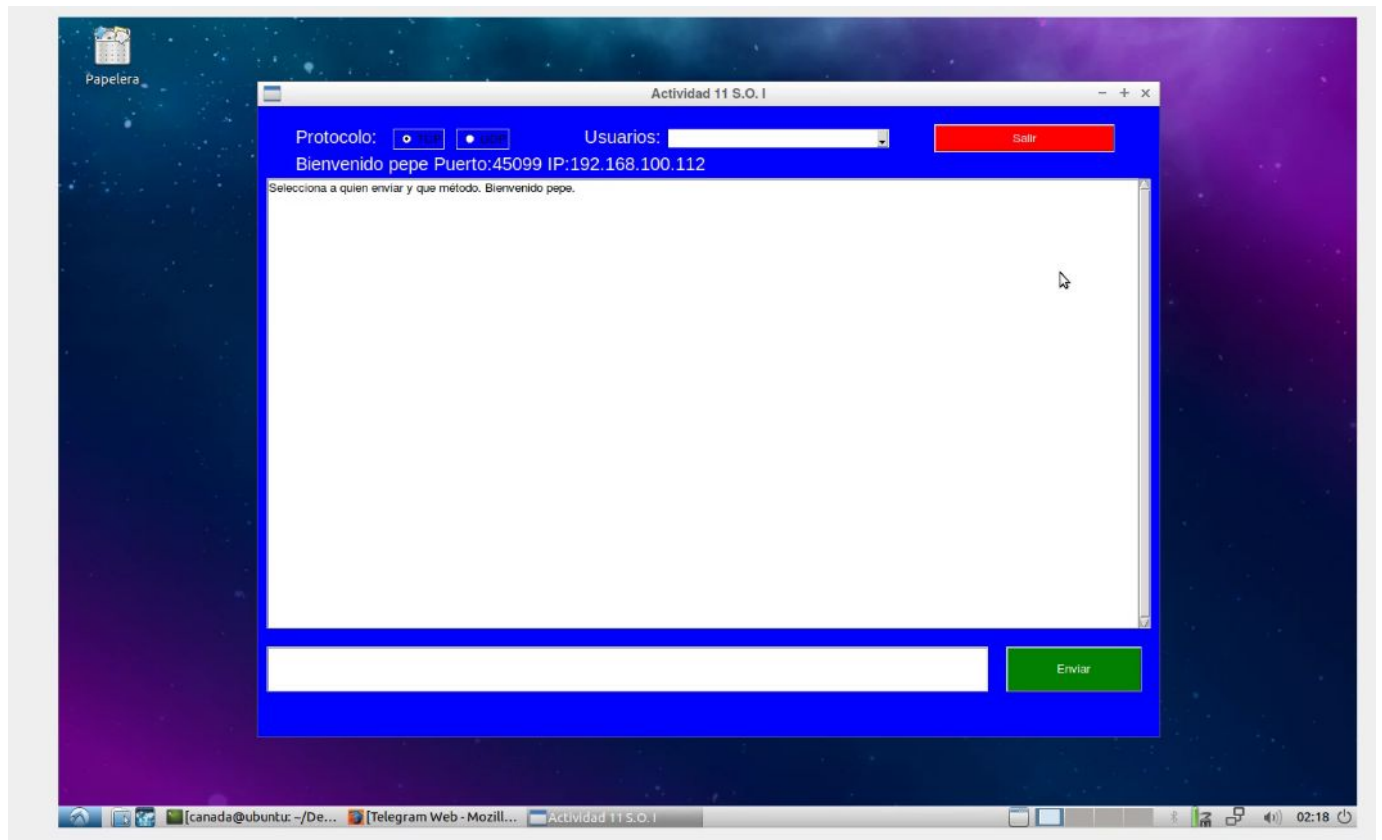


Imagen 10. Captura de la dos interfaces maquina virtual y equipo fisico,

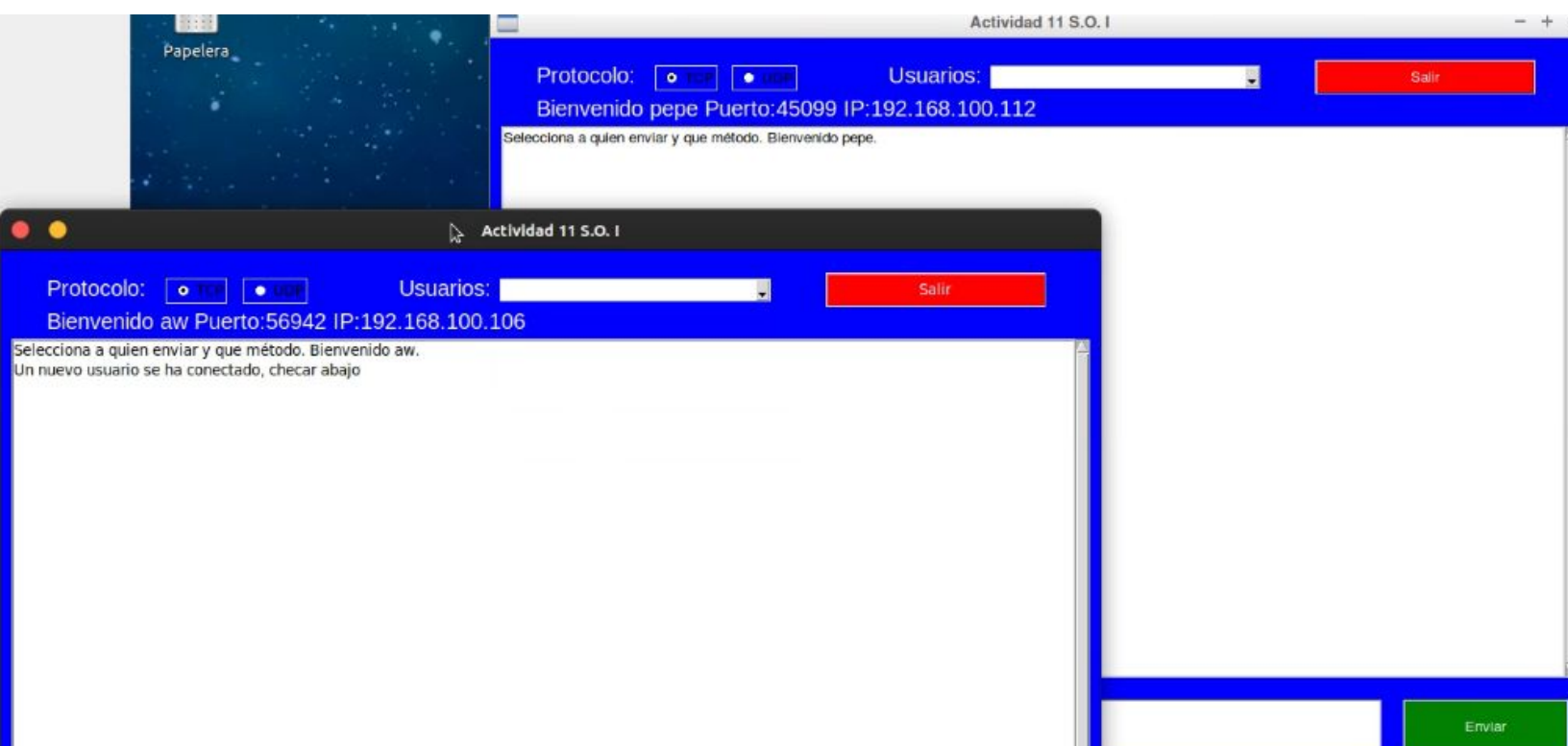


Imagen 11. En interface del equipo físico que tiene por usuario “aw” ha salido un mensaje que se ha conectado un nuevo usuario entonces lo seleccionamos que será el usuario “pepe” y en el fondo se puede ver la interfaz de la máquina virtual donde ya está seleccionado al usuario “aw”.



Imagen 12. En la interfaz del usuario de “aw”. Seleccionaremos el protocolo que en este caso será “TCP” aunque puede ser también “UDP”.



Imagen 13. Estando en la interfaz de la máquina física y siendo el usuario “aw” enviaremos un mensaje de hola a la máquina virtual con el usuario “pepe”.

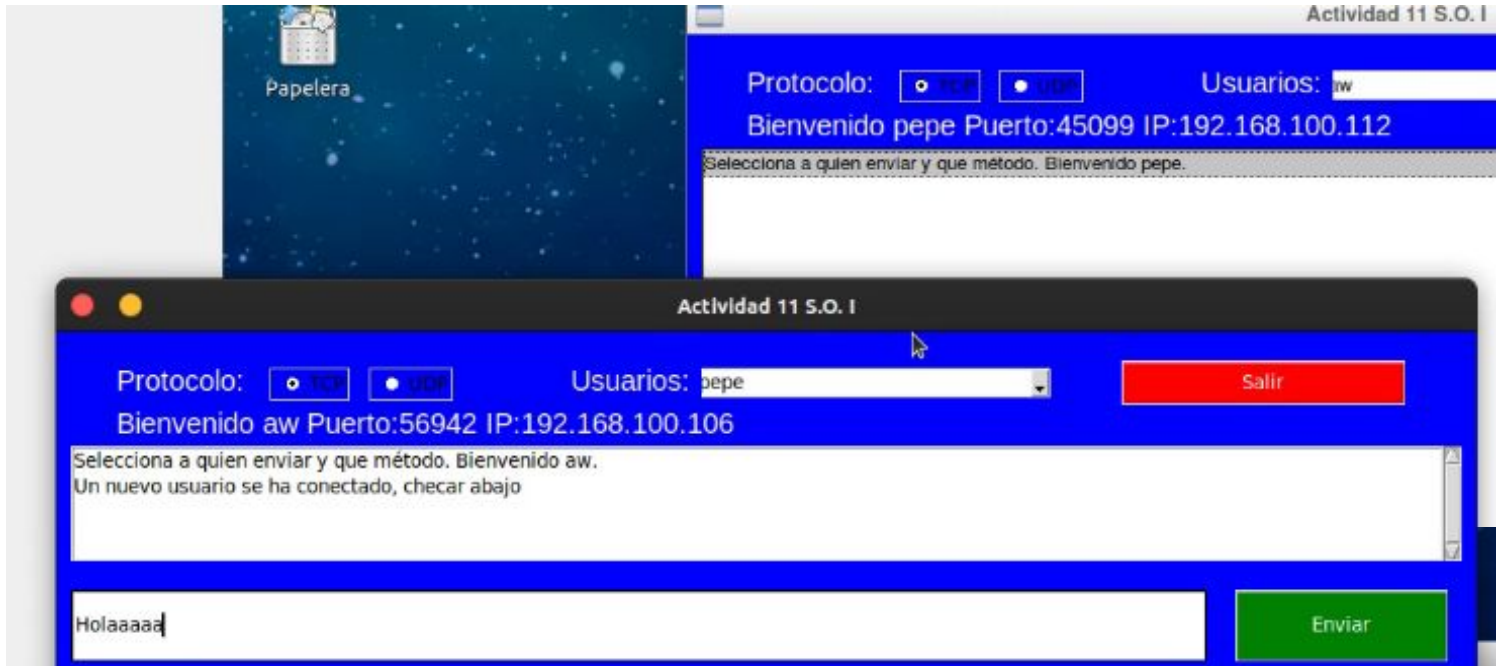


Imagen 14. Podremos ver en la interfaz de “aw” que el mensaje se ha enviado al usuario “pepe”.

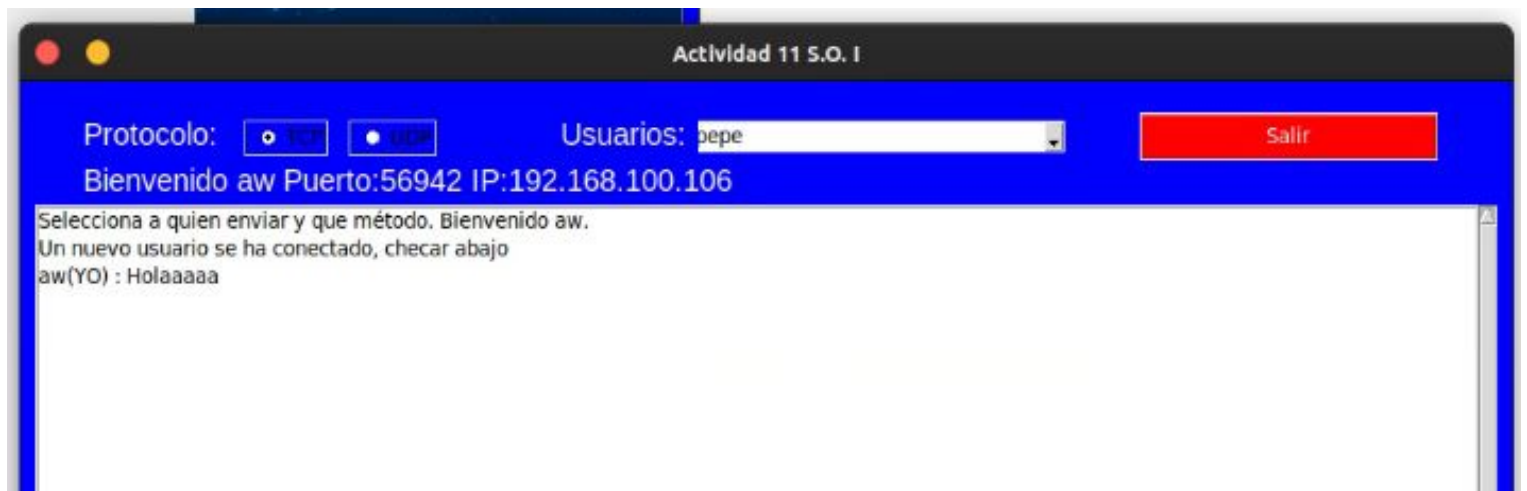


Imagen 15. Estando en la máquina virtual como el usuario “pepe” podemos ver que nos ha llegado un mensaje del usuario “aw”, el mensaje dice “Holaaaaa” y contestando desde la máquina virtual le responderemos “hey! estoy en maquina virtual :D”

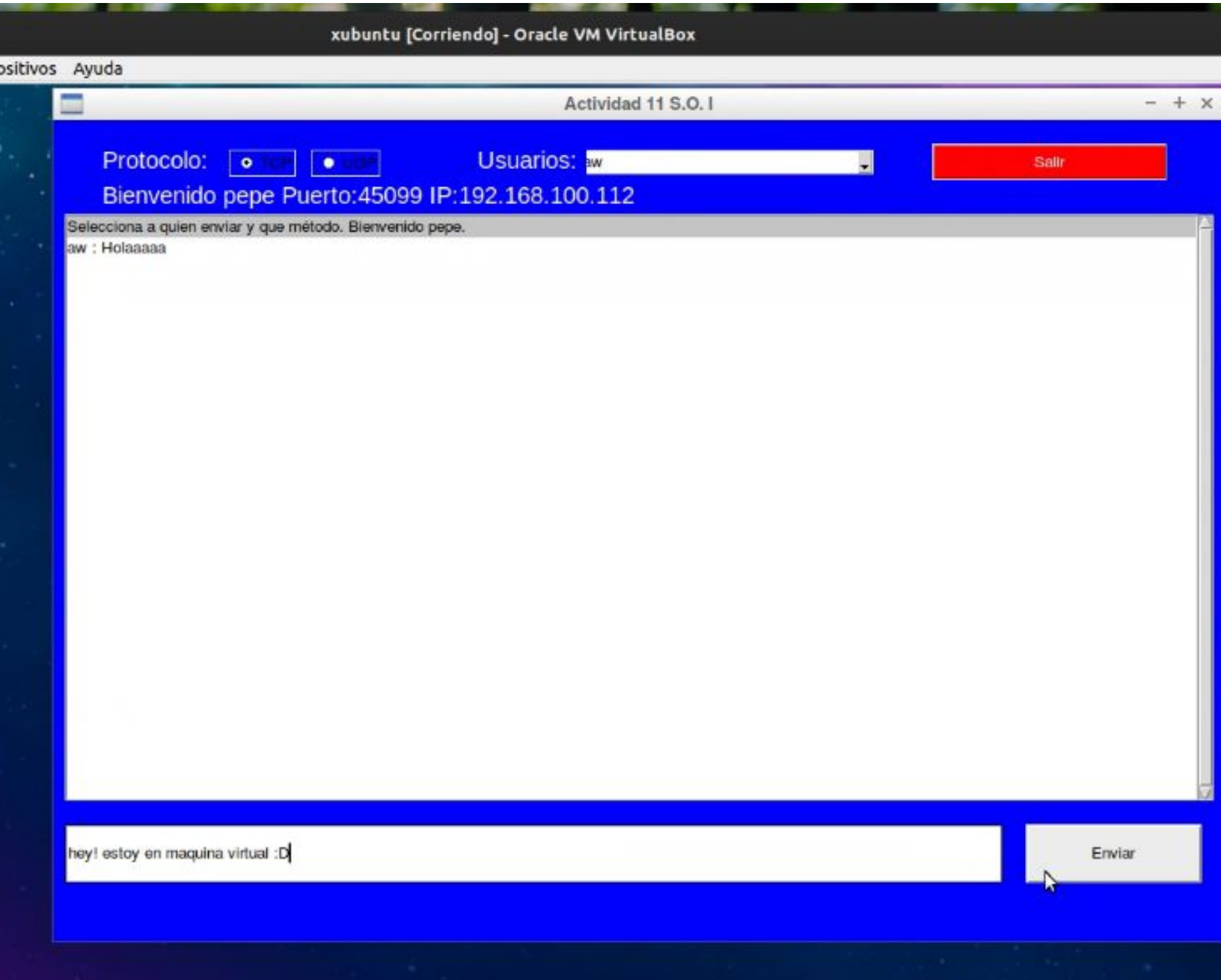


Imagen 16. Podremos observar que el mensaje del usuario “pepe” desde la máquina virtual se ha enviado correctamente y le ha llegado al usuario “aw”.

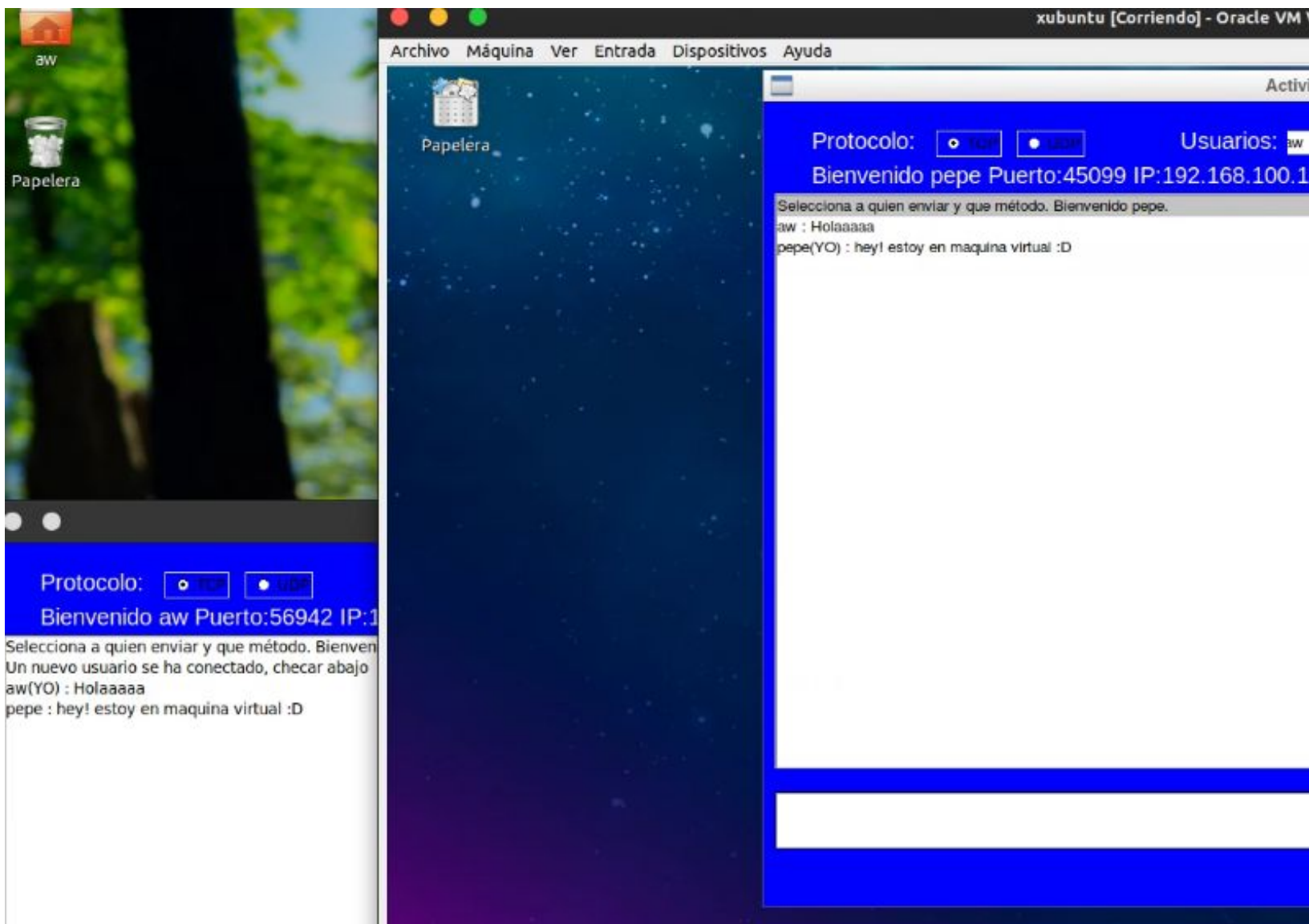
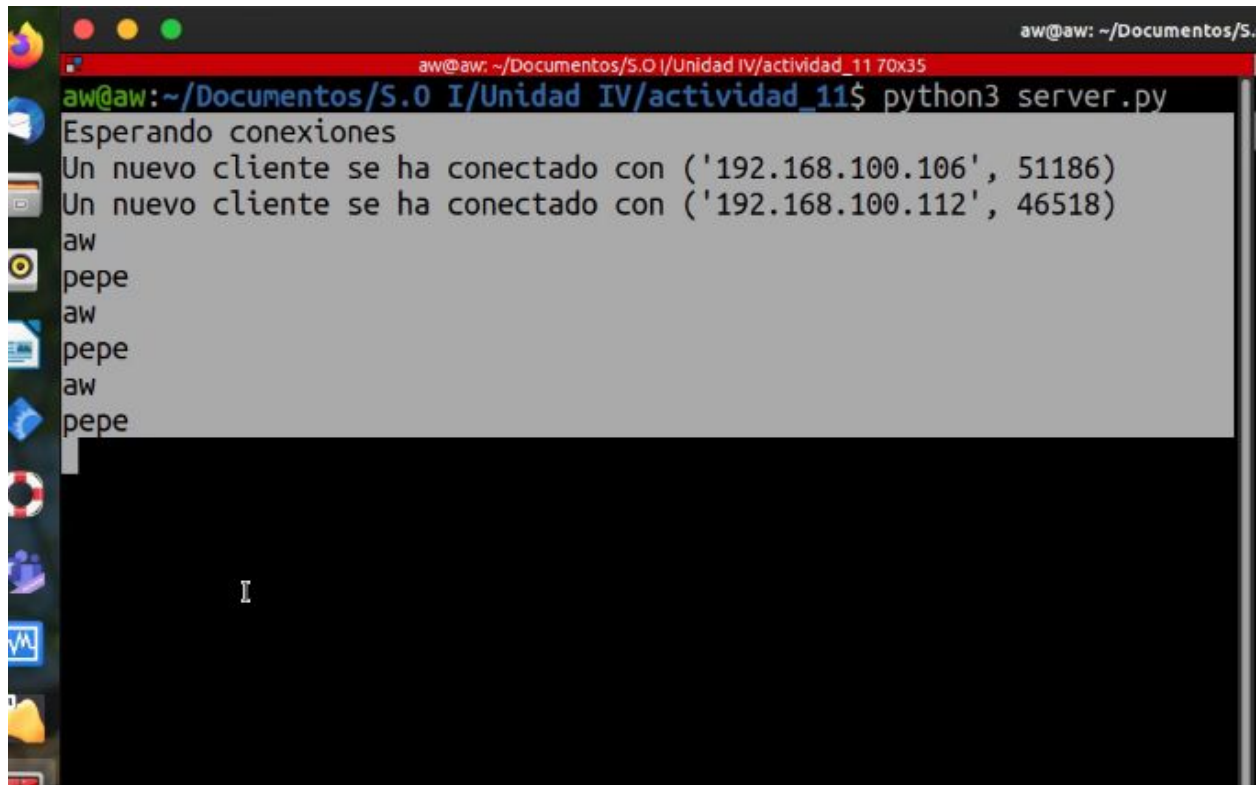


Imagen 17. Del lado nuestro servidor podremos observar las conexiones que se han hecho durante la prueba, tenemos la dirección “192.168.100.106” como la dirección del usuario “aw” y la dirección “192.168.100.112” como la dirección ip de la máquina virtual que pertenece al usuario “pepe”.

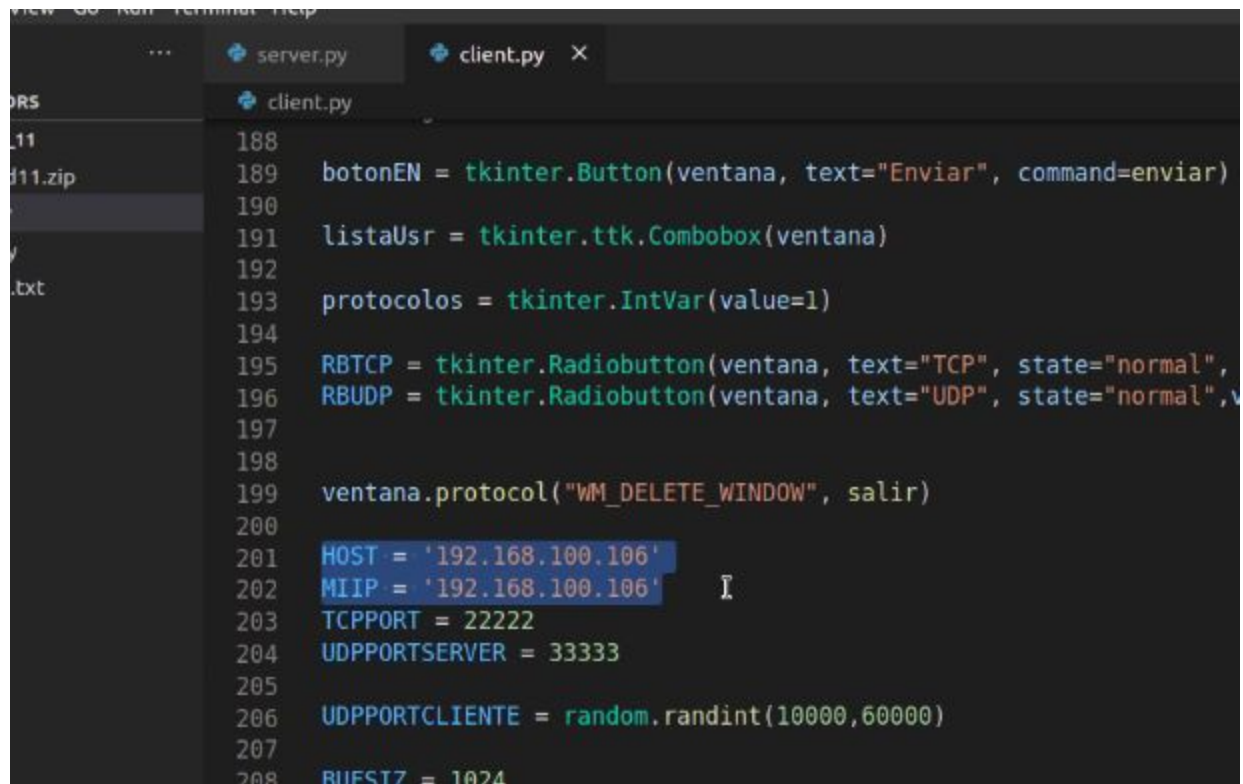


```
aw@aw: ~/Documentos/S.O I/Unidad IV/actividad_11 70x35
aw@aw:~/Documentos/S.O I/Unidad IV/actividad_11$ python3 server.py
Esperando conexiones
Un nuevo cliente se ha conectado con ('192.168.100.106', 51186)
Un nuevo cliente se ha conectado con ('192.168.100.112', 46518)
aw
pepe
aw
pepe
aw
pepe
```

Imagen 18. Para finalizar la sesión solo es cuestión de pulsar el botón de “salir” que se encuentra en rojo y al pasar el cursor sobre él, cambiará a gris .

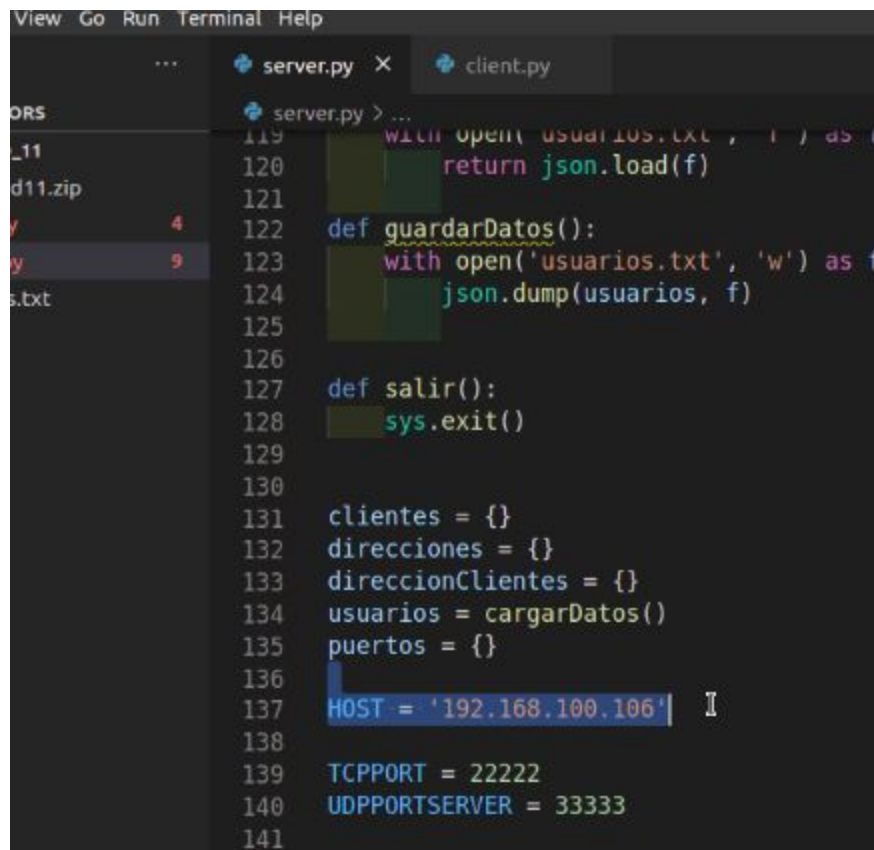


Imagen 19. Código del cliente en la máquina física veremos que en la variable HOST tendremos que poner la dirección ip de nuestro servidor que en este caso será la misma pc y nuestra variable MIIP será para colocar la ip de nuestro equipo.



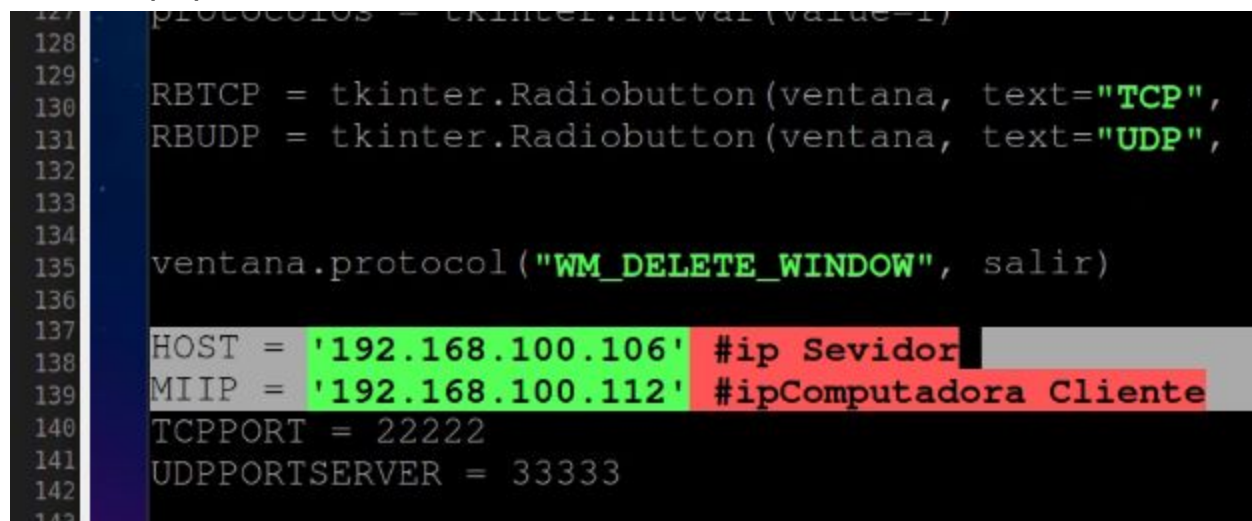
```
188
189 botonEN = tkinter.Button(ventana, text="Enviar", command=enviar)
190
191 listaUsr = tkinter.ttk.Combobox(ventana)
192
193 protocolos = tkinter.IntVar(value=1)
194
195 RBTCP = tkinter.Radiobutton(ventana, text="TCP", state="normal",
196 RBUDP = tkinter.Radiobutton(ventana, text="UDP", state="normal",
197
198
199 ventana.protocol("WM_DELETE_WINDOW", salir)
200
201 HOST = '192.168.100.106'
202 MIIP = '192.168.100.106'
203 TCPPOINT = 22222
204 UDPPORTSERVER = 33333
205
206 UDPPORTCLIENTE = random.randint(10000,60000)
207
208 BUFSIZE = 1024
```

Imagen 20. En el servidor solo tendremos que darle una ip que sera la nuestro equipo.



```
View Go Run Terminal Help
... server.py X client.py
server.py > ...
119 with open( 'usuarios.txt', 'r' ) as f:
120     return json.load(f)
121
122 def guardarDatos():
123     with open('usuarios.txt', 'w') as f:
124         json.dump(usuarios, f)
125
126
127 def salir():
128     sys.exit()
129
130
131 clientes = {}
132 direcciones = {}
133 direccionClientes = {}
134 usuarios = cargarDatos()
135 puertos = {}
136
137 HOST = '192.168.100.106'
138
139 TCPPORT = 22222
140 UDPPORTSERVER = 33333
141
```

Imagen 21 . Finalmente en el cliente de nuestra máquina virtual le asignaremos la direccion del servidor que sera la “192.168.100.106” y nuestra ip que sera “192.168.100.112”.



```
127 protocolos = tkinter.IntVar(value=1)
128
129 RBTCP = tkinter.Radiobutton(ventana, text="TCP",
130 RBUDP = tkinter.Radiobutton(ventana, text="UDP",
131
132
133
134
135 ventana.protocol("WM_DELETE_WINDOW", salir)
136
137 HOST = '192.168.100.106' #ip Sevidor
138 MIIP = '192.168.100.112' #ipComputadora Cliente
139
140 TCPPORT = 22222
141
142 UDPPORTSERVER = 33333
143
```


Codigos

Codigo "client.py"

```
from socket import AF_INET, socket, SOCK_STREAM, SOCK_DGRAM
from threading import Thread
import tkinter as tk
import tkinter
from tkinter import font
import tkinter.font as tkFont
from tkinter.constants import END, FALSE
import tkinter.ttk
import json
import time
import random
import sys
from tkinter.ttk import Label

def center(toplevel):
    toplevel.update_idletasks()
    w = toplevel.winfo_screenwidth()
    h = toplevel.winfo_screenheight()
    size = tuple(int(_) for _ in toplevel.geometry().split('+')[0].split('x'))
    x = w/2 - size[0]/2
    y = h/2 - size[1]/2
    toplevel.geometry("%dx%d+%d+%d" % (size + (x, y)))

def ocultar():
    botonIN.place_forget()
    botonRG.place_forget()
    usrtxt.place_forget()
    contratxt.place_forget()
    l1.place_forget()
    l2.place_forget()
    botonSal.pack_forget()

def iniciado():
    l3 = tk.Label(ventana, text="Protocolo: ", font=fontStyle, bg="blue", fg="white")
    l3.place(x=40, y=20)
    RBTCP.place(x=150, y=25)
    RBTCP.configure(bg="blue", fg="black")
    RBUDP.place(x=220, y=25)
    RBUDP.configure(bg="blue", fg="black")

    l4 = tk.Label(ventana, text="Usuarios: ", font=fontStyle, bg="blue", fg="white")
    l4.place(x=360, y=20)
```

```

l5=tk.Label(ventana, text='Bienvenido ' + usr.get() + " Puerto:" + str(UDPPORTCLIENTE) + " IP:"
+ MIIP ,font=fontStyle ,bg="blue", fg="white")
l5.place(x=40, y=50)

```

```

listaUsr.place(x=450, y=25, width=250)
botonSal.place(x=750, y=20, width=200)
scrollbar.pack(side=tkinter.RIGHT, fill=tkinter.Y)
listaMsgs.pack(fill=tk.X)
ventMsg.place(x=10, y=80, height=500,width=980)

```

```

textoMsg.place(x=10, y=600, width=800, height=50)
botonEN.place(x=830, y=600, width=150, height=50)
botonEN.configure(bg="Green", fg="white")
ventana.resizable(0,0)

```

```

def recibirTCP():
    while True:
        try:
            msg = client_socket.recv(BUFSIZ).decode("utf8")
            chat = json.loads(msg)
            if type(chat) is dict:
                conectados = chat.copy()
                listaUsr['values'] = list(conectados.keys())
            else:
                if 'Selecciona a quien enviar y que método. Bienvenido' == chat:
                    ocultar()
                    iniciado()
                    ventana.geometry("1000x700")
                    center(ventana)

                    chat = 'Selecciona a quien enviar y que método. Bienvenido ' + usr.get()+". "
                    listaMsgs.insert(tkinter.END, chat)
                if 'Inicia sesion o regístrate para chatear' == chat:
                    listaMsgs.insert(tkinter.END, chat)
                if 'ERROR: Ya existe usuario, intente de nuevo' == chat:
                    listaMsgs.insert(tkinter.END, chat)
                if 'ERROR: No existe usuario o contraseña incorrecta, intente de nuevo' == chat:
                    listaMsgs.insert(tkinter.END, chat)
                if chat == 'Un nuevo usuario se ha conectado, checar abajo':
                    listaMsgs.insert(tkinter.END, chat)
                if chat == 'Un usuario ha dejado el chat, checar abajo':
                    listaMsgs.insert(tkinter.END, chat)
                if chat[0] == 'Enviar a':
                    string = chat[2] + " : "+chat[3]
                    listaMsgs.insert(tkinter.END, string)
            except OSError:
                break

```

```
def recibirUDP():
    while True:
        data, direccion = UDPSEVER.recvfrom(BUFSIZ)
        chat = json.loads(data)
        string = chat[2] + " : "+chat[3]
        listaMsgs.insert(tkinter.END, string)
```

#Funciones de Botones

```
def enviar(event=None):
    msg = msgToEnviar.get()
    listaMsgs.insert(tkinter.END, usr.get()+"(YO) : "+msg)
    listaSalida = ['Enviar a',listaUsr.get(),usr.get(),msg]
    msgToEnviar.set("")
    if protocolos.get() == 1:
        client_socket.send(bytes(json.dumps(listaSalida), "utf8"))
    else:
        UDPSEVER.sendto( bytes( json.dumps (listaSalida) ,"utf-8"), (HOST, UDPPORTSERVER))
```

```
def iniciar():
    miNombre=usr.get()
    listaSalida = ['Login',usr.get(), contra.get(),UDPPORTCLIENTE]
    client_socket.send(bytes(json.dumps(listaSalida), "utf8"))
    listaMsgs.delete(0,END)
```

```
def registrar():
    miNombre=usr.get()
    listaSalida = ['Register',usr.get(), contra.get(),UDPPORTCLIENTE]
    client_socket.send(bytes(json.dumps(listaSalida), "utf8"))
    listaMsgs.delete(0,END)
```

```
def salir(event=None):
    listaSalida = ['*Salir*',usr.get()]
    client_socket.send(bytes(json.dumps(listaSalida), "utf8"))
    ventana.destroy()
```

```
ventana = tkinter.Tk()
ventana.title("Actividad 11 S.O. I")
ventana.configure(background="blue")
ventana.geometry("250x375")
```

```
ventMsg = tkinter.Frame(ventana)
msgToEnviar = tkinter.StringVar()
msgToEnviar.set("")
usr = tkinter.StringVar()
usr.set("aw")
contra = tkinter.StringVar()
```

```
contra.set("1234")
```

```
scrollbar = tkinter.Scrollbar(ventMsg)
listaMsgs = tkinter.Listbox(ventMsg, height=30, width=80, yscrollcommand=scrollbar.set)
scrollbar.pack(side=tkinter.RIGHT, fill=tkinter.Y)
listaMsgs.pack(side=tkinter.LEFT, fill=tkinter.BOTH)
listaMsgs.pack(side="right")
ventMsg.pack(side="right", padx=10, pady=10)
```

```
scrollbar.pack_forget()
listaMsgs.pack_forget()
ventMsg.pack_forget()
```

```
fontStyle = tkFont.Font(family="Arial", size=15)
```

```
l1 =tk.Label(ventana, text="Nombre Usuario",font=fontStyle ,bg="blue", fg="white")
l1.place(x=40, y=80)
```

```
usrtxt = tkinter.Entry(ventana, textvariable=usr)
usrtxt.bind("<Return>", iniciar, registrar)
usrtxt.pack()
usrtxt.place(x=40, y=110)
```

```
l2 =tk.Label(ventana, text="Contraseña",font=fontStyle, bg="blue", fg="white")
l2.place(x=70, y=150)
```

```
contratxt = tkinter.Entry(ventana, textvariable=contra)
contratxt.bind("<Return>", iniciar, registrar)
contratxt.pack()
contratxt.place(x=40, y=180)
```

```
botonIN = tkinter.Button(ventana, text="Login", command=iniciar)
botonIN.pack(padx=10, pady=10)
botonIN.place(x=40, y=250)
```

```
botonRG = tkinter.Button(ventana, text="Register", command=registrar)
botonRG.pack(padx=0, pady=10)
botonRG.place(x=120, y=250)
```

```
botonSal = tkinter.Button(ventana, text="Salir", command=salir, bg="Red", fg="white")
botonSal.pack(padx=10, pady=10, side=tkinter.BOTTOM, fill=tk.X)
```

```
textoMsg = tkinter.Entry(ventana, textvariable=msgToEnviar)
textoMsg.bind("<Return>", enviar)
```

```
botonEN = tkinter.Button(ventana, text="Enviar", command=enviar)
```

```
listaUsr = tkinter.ttk.Combobox(ventana)
```

```
protocolos = tkinter.IntVar(value=1)
```

```
RBTCP = tkinter.Radiobutton(ventana, text="TCP", state="normal", value="1", variable=protocolos)
```

```
RBUDP = tkinter.Radiobutton(ventana, text="UDP", state="normal", value="2", variable=protocolos)
```

```
ventana.protocol("WM_DELETE_WINDOW", salir)
```

```
HOST = '192.168.100.106'
```

```
MIIP = '192.168.100.106'
```

```
TCPPORT = 22222
```

```
UDPPORTSERVER = 33333
```

```
UDPPORTCLIENTE = random.randint(10000,60000)
```

```
BUFSIZ = 1024
```

```
ADDR = (HOST, TCPPORT)
```

```
client_socket = socket(AF_INET, SOCK_STREAM)
```

```
client_socket.connect(ADDR)
```

```
ADDR = (MIIP, UDPPORTCLIENTE)
```

```
UDPSERVER = socket(AF_INET, SOCK_DGRAM)
```

```
UDPSERVER.bind(ADDR)
```

```
conectados = {}
```

```
recibirTCPHilo = Thread(target=recibirTCP)
```

```
recibirUDPHilo = Thread(target=recibirUDP)
```

```
recibirTCPHilo.start()
```

```
recibirUDPHilo.start()
```

```
tkinter.mainloop()
```

Codigo “server.py”

```
from socket import AF_INET, socket, SOCK_STREAM, SOCK_DGRAM
from threading import Thread
import json
import time
import sys
import pickle
```

```
def aceptarConexionesTCP():
    while True:
        client, client_address = TCPSERVER.accept()
        print("Un nuevo cliente se ha conectado con",client_address)
        salida = 'Inicia sesion o regístrate para chatear'
        listaSalida = salida
        client.send(bytes(json.dumps(listaSalida), "utf8"))
        direcciones[client] = client_address
        Thread(target=manejarClienteTCP, args=(client,)).start()
```

```
def manejarClienteTCP(client):
```

```
    listaEntrada = json.loads(client.recv(Buffer).decode("utf8"))
    name = listaEntrada[1]
    print(name)

    if listaEntrada[0] == 'Login':
        while True:
            if iniciarSesion(listaEntrada[1],listaEntrada[2]) == False:
                salida = 'ERROR: No existe usuario o contraseña incorrecta, intente de nuevo'
                listaSalida = salida
                client.send(bytes(json.dumps(listaSalida), "utf8"))
                listaEntrada = json.loads(client.recv(Buffer).decode("utf8"))
            else:
                break
        if listaEntrada[0] == 'Register':
            while True:
                if registrar(listaEntrada[1]) == True:
                    salida = 'ERROR: Ya existe usuario, intente de nuevo'
                    listaSalida = salida
                    client.send(bytes(json.dumps(listaSalida), "utf8"))
                    listaEntrada = json.loads(client.recv(Buffer).decode("utf8"))
                else:
                    break
            name = listaEntrada[1]
            usuarios[listaEntrada[1]]=listaEntrada[2]
            puertos[listaEntrada[1]]=listaEntrada[3]

    guardarDatos()
```

```

welcome = 'Selecciona a quien enviar y que método. Bienvenido'
listaSalida=welcome
client.send(bytes(json.dumps(listaSalida), "utf8"))

msg = 'Un nuevo usuario se ha conectado, checar abajo'
listaSalida=msg

direccionClientes[name] = direcciones[client]

broadcast(listaSalida)
clientes[client] = name

while True:

    msg = client.recv(Buffer)
    listaEntrada = json.loads(msg)

    if listaEntrada[0] == 'Enviar a':
        enviarTCP(listaEntrada,listaEntrada[1])

    if listaEntrada[0] == '*Salir*':
        listaSalida = 'Un usuario ha dejado el chat, checar abajo'
        del clientes[client]
        del direccionClientes[name]
        broadcast(listaSalida)
        break

def enviarTCP(lista,name):
    for sock in clientes:
        print(clientes[sock])
        if name == clientes[sock]:
            sock.send(bytes(json.dumps(lista), "utf8"))

def broadcast(listaSalida):
    for sock in clientes:
        sock.send(bytes(json.dumps(listaSalida), "utf8"))

def conectados():
    i=0
    while True:
        time.sleep(1)
        for sock in clientes:
            sock.send(bytes(json.dumps(direccionClientes), "utf8"))

def registrar(usuario):
    if usuario in usuarios:
        return True

```

```

        else:
            return False
def iniciarSesion(usuario, contraseña):
    if usuario in usuarios:
        if usuarios[usuario] == contraseña:
            return True
        else:
            return False
    else:
        return False

def manejarClientesUDP():
    while True:
        data, addr = UDPSERVER.recvfrom(Buffer)
        listaEntrada = json.loads(data)
        enviarUDP(listaEntrada, listaEntrada[1])

def enviarUDP(listaSalida, name):
    UDPSERVER.sendto( bytes( json.dumps( listaSalida ) , "utf-8" ), ( direccionClientes[name][0],
puertos[name] ))

def cargarDatos():
    with open('usuarios.txt', 'r') as f:
        return json.load(f)

def guardarDatos():
    with open('usuarios.txt', 'w') as f:
        json.dump(usuarios, f)

def salir():
    sys.exit()

clientes = {}
direcciones = {}
direccionClientes = {}
usuarios = cargarDatos()
puertos = {}

HOST = '192.168.100.106'

TCPPOINT = 22222
UDPPORTSERVER = 33333

Buffer = 1024

direccion = (HOST, TCPPOINT)

```



```
TCPSERVER = socket(AF_INET, SOCK_STREAM)
TCPSERVER.bind(direccion)

direccion = (HOST, UDPPORTSERVER)
UDPSERVER = socket(AF_INET, SOCK_DGRAM)
UDPSERVER.bind(direccion)

if __name__ == "__main__":

    TCPSERVER.listen(5)

    print("Esperando conexiones")

    aceptarTCPHilo = Thread(target=aceptarConexionesTCP)
    conectadosHilo = Thread(target=conectados)
    manejarUDPHilo = Thread(target=manejarClientesUDP)

    aceptarTCPHilo.start()
    conectadosHilo.start()
    manejarUDPHilo.start()

    aceptarTCPHilo.join()

    TCPSERVER.close()
    UDPSERVER.close()
```