Práctica 1.2

Silviu Constantin Sofrone

Modelo cliente-servidor

- **servidor.py**

```
#!/usr/bin/env python3
# -*- coding: cp1252 -*-

import sys
import socket
import select

if len(sys.argv) != 2:
  print('Usage:', sys.argv[0], '<Server Port>\n')

ServPort = sys.argv[1]

# Socket TCP del servidor
ServSock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# Configurarlo para que no se bloquee
ServSock.setblocking(0)

# Unimos (bind) socket al puerto
ServSock.bind(('', int(ServPort)))

# Escucha conexiones entrantes
ServSock.listen(10) # Nº de conexiones posibles

# Sockets que van a ready_to_read
entrantes = [ServSock]
```

```
# Sockets que van a ready_to_write
salientes = []


try:
  while entrantes:
    ready_to_read, ready_to_write, error = select.select(entrantes, salientes, entrantes)
    for s in ready_to_read:
      if s is ServSock: #Si s es el socket del servidor el que está listo aceptamos las conexiones de los clientes que entran al chat
        connection, clntAddr = ServSock.accept()
        print( 'Conexión con:', clntAddr)
        entrantes.append(connection) # Añadimos a la lista de entrantes el nuevo cliente
      else:
        datos = s.recv(1024) # Cuando se reciba un mensaje
        if datos:
          for cliente in entrantes: # Para todos los clientes en la lista de entrantes
            if cliente != ServSock and cliente != s: # Excepto el servidor y el cliente que ha mandado el mensaje
              cliente.sendall(datos) #El servidor reenvía el mensaje de s a cada cliente
        else:
          print('Ya no hay conexión con:',s.getpeername())
          entrantes.remove(s) #En caso de perderse la conexión con algún cliente se le expulsa de la lista
          s.close() #Y se cierra su socket asociado
except KeyboardInterrupt:
  for s in ready_to_read: #Se cierran todas las conexiones en caso de cerrar el servidor
    if s != ServSock:
      entrantes.remove(s)
      s.close()
```

- **cliente.py**

```
#!/usr/bin/env python3
# -*- coding: cp1252 -*-

import sys
import socket
import select
import re #expresiones regulares

if len(sys.argv) < 2 or len(sys.argv) > 3:
  print('Usage:', sys.argv[0], '<Server IP> [<PING Port>]')

#Definición de variables
servIP = sys.argv[1]
pingServPort = sys.argv[2]

#Crear socket TCP
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

#Conexión
sock.connect((servIP, int(pingServPort)))

#En el caso de los clientes, la lista estará formada solo por el socket conectado con el servidor (para recibir los mensajes que nos reenvía) y sys.stdin (para leer nuestro teclado)
lectura = [sock, sys.stdin]

try:
  while lectura:
    read, write, error = select.select(lectura, [], [])
```

```
    for s in read:
      if s is sock: #Si es el socket conectado al servidor el que está listo
        datos = sock.recv(1024) #Recibimos el mensaje reenviado por el servidor
        if datos:
          print('\n< ',datos.decode("utf-8")) #Decodificamos la información y la mostramos por pantalla en el chat
          try:
            f = open('recv.txt', 'w')
            f.write(datos.decode("utf-8")) #Almacenamos los mensajes en un archivo de texto
          except IOError:
            print('El fichero no es accesible')
          finally:
            f.close()
        if not datos:
          sock.close() #En caso de que se pierda la conexión con el servidor
          print('Conexión cerrada')
          break
      if s is sys.stdin: #Si s es sys.stdin leemos de teclado
        mensaje = input('> ')
        if re.findall('[.]txt$', mensaje): #En caso de que el mensaje escrito termine en ".txt", identificamos que se trata de un archivo de texto
          try: #Por tanto, intentamos abrirlo y enviar su infromación contenida
            f = open(mensaje, 'r')
            bytes_f = f.read().encode() #Leemos el archivo y codificamos su información para ser enviada
            sock.sendall(bytes_f)
          except IOError:
            print('El fichero no es accesible')
            sock.sendall(bytes(mensaje, 'utf-8')) #En caso de que no se trate de un archivo de texto como tal, enviamos el nombre del archivo que hemos tratado de abrir
          finally:
            f.close()
```

```
        else:
            sock.sendall(bytes(mensaje, 'utf-8')) #Si no intentamos abrir un archivo de texto se trata de un mensaje normal que debe ir codificado
except KeyboardInterrupt:
    sock.close()
```

Modelo P2P

- **servidor_usuarios.py**

```
#!/usr/bin/env python3

import sys
import socket
import select
import pickle

if len(sys.argv) != 2:
        printf('Usage:', sys.argv[0], '<Server Port>\n')

ServPort = sys.argv[1]

ServSock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
ServSock.setblocking(0)

ServSock.bind(('', int(ServPort)))
ServSock.listen(10)

usuarios = [ServSock]
direcciones = []

try:
```

```
 while usuarios:
  ready_to_read, ready_to_write, error = select.select(usuarios,[] ,[] )
  for s in ready_to_read:
   if s is ServSock: #Si s es el socket del servidor aceptamos las conexiones de peers entrantes
    connection, clntAddr = ServSock.accept()
    port = connection.recv(1024) #Recibimos el puerto de escucha del nuevo peer
    port = pickle.loads(port) #Deserializamos la información con pickle.loads()
    print( 'Nuevo usuario:', clntAddr)
    usuarios.append(connection) #Añadimos el nuevo peer a la lista usuarios
    connection.sendall(pickle.dumps(direcciones)) #Enviamos al nuevo peer la lista de peers disponibles formada por duplas (Dirección IP, puerto de escucha)
    Dir = (clntAddr[0], port) #Creamos la dupla del nuevo peer con su dirección  IP a partir del primer elemento de clntAddr y su puerto de escucha
    print('Dirrecion de escucha:', Dir)
    direcciones.append(Dir) #Añadimos la nueva dupla a la lista correspondiente
   else:
    try:
     conn = ServSock.getpeername() #Obtenemos la dirreción de cada conexión
     s.connect(conn) #Y comprobamos si el peer correspondiente a tal conexión sigue conectado
    except: #En caso de que no siga conectado...
     print('Ya no hay conexión con: ', s.getpeername())
     i = usuarios.index(s) - 1 #La posición correspondiente en la lista de direcciones será la de usuarios - 1(ServSock)
     usuarios.remove(s) #Expulsamos el socket correspondiente de la lista usuarios
     s.close() #Cerramos su conexión
     direcciones.pop(i) #Expulsamos su dirección de escucha de la lista direcciones para que los nuevos peers no traten de conectarse a él

except KeyboardInterrupt: #En caso de cerrar el servidor se cierran todas las conexiones
 ServSock.close()
 for s in ready_to_read:
  if s != ServSock:
```

```
    s.close()
    usuarios.remove(s)
```

- **peer.py**

```
#!/usr/bin/env python3

import sys
import socket
import pickle
import select
import re

if len(sys.argv) != 3:
        print('Usage: {} <Server IP> [<PING PORT>]', sys.argv[0])

servIP = sys.argv[1]
ServPort = sys.argv[2]

ServSock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
ServSock.connect((servIP, int(ServPort))) #Primero nos conectaremos con el servidor, este, constatará que estamos conectados y le pasaremos nuestro puerto de escucha

#Socket de escucha
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.setblocking(0) #El socket de escucha se establece en no bloqueo
sock.bind(('', 0)) #Escuchamos en un puerto aleatorio
sock.listen(10)

print('Puerto de escucha: ', sock.getsockname()[1])
port = sock.getsockname()[1]
```

```
ServSock.sendall(pickle.dumps(port)) #Mandamos al servidor el puerto de escucha


lectura = [sock, sys.stdin] #Lista con el socket de escucha, sys.stdin y posteriormente los peers con los que nos vayamos conectando

datos = ServSock.recv(1024) #Recibimos del servidor una lista de duplas (dirrección IP, puerto) de los peers disponibles

if datos:

 lista = pickle.loads(datos) #Deserializamos el flujo de datos

 if lista: #Si la lista no está vacía

  for x in lista: #Nos conectamos con todos los peers disponibles

   print('Conectamos con:', x)

   soc = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

   soc.connect(x)

   lectura.append(soc) #Añadimos la conexión a la lista lectura

print()


try:

 while lectura:

  read, write, error = select.select(lectura, [], [])

  for s in read:

   if s is sock: #Si s es el socket de escucha aceptamos las nuevas conexiones

    connection, clntAddr = sock.accept()

    print('Conexión con peer:', clntAddr)

    lectura.append(connection) #Añadimos la nueva conexión a la lista lectura

   if s is sys.stdin: #Si s es sys.stdin leemos de teclado

    mensaje = input('> ')

    if mensaje:

     for peer in lectura: #Mandamos el mensaje a todos los usuarios conectados al chat con nosotros

      if peer != sock and peer != sys.stdin:

       peer.sendall(bytes(mensaje, 'utf-8'))
```

```
    if s != sock and s != sys.stdin: #El resto de sockets correspondientes a las conexiones con los demás usuarios reciben los mensajes y los muestran por pantalla. Entiendo que el primero en conectarse no va a pasar por el primer if y sock se comprobará aquí en tal caso
      datos = s.recv(1024)
      if datos:
        print('\n< ',datos.decode("utf-8"))
      if not datos:
        print('Ya no hay conexión con:', s.getpeername())
        lectura.remove(s) #En caso de desconexión se debe expulsar de la lista el socket correspondiente
        s.close() #Cierre del socket
    else:
      if s != sys.stdin and s != sock:
        print(s)
        try:
          conn = sock.getpeername() #Obtenemos la dirreción de cada conexión
          s.connect(conn) #Comprobamos que siga conectado
        except:
          print('Ya no hay conexión con: ', s.getpeername())
          lectura.remove(s)
          s.close()

except KeyboardInterrupt:
  sock.close()
  ServSock.close()
```