Delia Clar Ramírez
4ºGITT

PRÁCTICA FINAL

IMPLEMENTACIÓN DE UN CHAT MULTICAST

SISTEMAS DISTRIBUIDOS 4ºGITT

En esta práctica se ha desarrollado un chat multicast. Para ello, se ha creado una interfaz gráfica.

Los mensajes multicast son mensajes que se envían a distintos puntos simultáneamente. Se definen unas direcciones IP reservadas para este tipo de mensajes, las cuales pertenecen al siguiente rango [224.0.0.0 a 239.255.255.255]. Un host que se une a una de estas direcciones recibe todos los mensajes procedentes del grupo.

El código se ha basado en un fichero escrito en Python. El cual hace uso de una clase con distintas funciones. A continuación, se explicará el código y la implementación de las distintas funciones.

Para empezar, se importan las librerías necesarias para el desarrollo del código

```
from tkinter import *
from tkinter import ttk

import socket
import struct
import sys
import pickle
import traceback
from threading import Thread
import threading
```

A continuación, se define la clase Aplicación, esta será la encargada de definir la interfaz gráfica, en la cual se encuentra toda la lógica de la aplicación. Dicha clase, cuenta con distintos métodos.

El primero, llamado __init__, hace referencia a la inicialización de la clase, los parámetros que recibe son la dirección ip de grupo multicast, el puerto y la dirección de enlace. Todos ellos vienen definidos por defecto, en caso de que no se indique ningún valor de entrada.

A continuación, observamos el desarrollo del código:

Delia Clar Ramírez
4ºGITT

```python
# MULTICAST
# Los mensajes multicast son mensajes enviados a múltiples puntos simultaneamente
# Mensajes multicast se envian haciendo uso del protocolo UDP
# Las direcciones multicast estan definidas en un rango [224.0.0.0 a 239.255.255.255]
# Los routers de la red tratan estas direcciones de diferente manera ya que están reservadas para grupos multicast
# Mensajes enviados a estas direcciones se distribuyen a todos los clientes pertenecientes al grupo



#Debinimos una clase Aplicacion
# Será la encargada de definir la interfaz gráfica
class Aplicacion():
    #Función de creación de la clase
    #Los parametros de la clase son:
    # 1. Dirección ip de grupo multicast -> por defecto será : 224.0.0.32
    # 2. El puerto del grupo multicast -> por defecto será: 1234
    # 3. Dirección de enlace -> vacío
    def __init__(self,multicast='224.0.0.32',port=1234,bind_addr=''):

        #Definimos las variables
        #Será necesario un nombre para identificar al usuario
        self.usuario=input('Introduzca el nombre de usuario que desee para el chat: ')
        self.multicast=multicast #Dirección ip de grupo multicast
        self.port =port #Puerto a usar para recibir y enviar mensajes
        self.bind_addr=bind_addr #Dirección de enlace
        self.raiz=Tk() #Definimos la ventana principal de la aplicación

        # Diccionario que almacenará todos los usuarios y sus mensajes
        # La clave del diccionario esta compuesta por el usuario y el orden de los mensajes del mismo usuario
        self.nicks={'':0}

        self.raiz.resizable(False,False) # El usuario no puede redimensionar en ninguna dirección
        #Definimos las dimensiones de la ventana
        self.raiz.geometry('300x400')
        self.raiz.configure(bg='#C7E7F6')  # le añadimos un color a la ventana
        self.raiz.title('Chat Multicast') # Asignamos un título


        #Creamos la variable que almacenará el texto del entry
        self.text=StringVar(value='')

        #Creamos otra variable que serán los mensajes recibidos del grupo
        self.mensajes=StringVar(value='Mensajes Recibidos: \n')

        #Creamos un label para los mensajes y otro para escribirlos
        self.label1 = Label(self.raiz,bg='#C7E7F6',textvariable=self.mensajes)
        #Situamos el primer label en el inicio del mensaje
        self.label1.grid(row=0,column=0,columnspan=3,rowspan=2)
        #Creamos el segundo label, en el que se encontrara el boton y el entry
        self.label2 = Label(self.raiz,bg='#D4DADD')
        self.label2.grid(row=2,column=0,columnspan=4)
        self.label2.config(padx=20,pady=5)


        # Definimos el boton y el cuadro de texto dentro del label2
        self.txt = ttk.Entry(self.label2, textvariable=self.text)
        self.txt.grid(row=0,column=0,columnspan=2)
        self.btn=ttk.Button(self.label2,text='Enviar',command=self.enviar)
        self.btn.grid(row=0,column=3)
        self.txt.bind('<Return>', self.onEnter)
        #Expandimos horizontalmente la columna 0
        self.raiz.columnconfigure(0,weight=3)
        #Expandir verticalmente la fila 0
        self.raiz.rowconfigure(0,weight=3)
        self.raiz.rowconfigure(2,weight=1)
```

```
#--------------CONEXIÓN DEL SOCKET---------------#

# Creamos el socket -> enlace de comunicación entre dos puntos
# Funcion socket crea una nueva conexión dada una familia de dirección y el tipo de socket a usar
# AF_INET -> IPv4 para direcciones de red
# SOCK_DGRAM -> mensajes estan orientados a transporte de datagramas
self.sock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
# Función socket define un único puerto para el grupo, así pueden escucharse varios clientes que
# envian y reciben en la misma dirección ip y puerto
self.sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEPORT, 1)
#TTL controla el tiempo de vida del mensaje, controla cuantas redes lo recibiran
# Lo asignamos a uno para que los paquetes no sean reenviados por el enrutador mas alla de la red actual
# Debe estar empaquetado en 1 byte unico -> por ello escribimos la funcion struct.pack
ttl=struct.pack('b',1)
# Asignamos la opcion ttl de multicast al creado recientemente
self.sock.setsockopt(socket.IPPROTO_IP,socket.IP_MULTICAST_TTL,ttl)
# Lo enlazamos a la dirección del servidor (dirección ip '' (string vacio), puerto)
self.sock.bind((bind_addr,port))
# Añadimos el socket a un grupo multicast en todas las intefaces

group = socket.inet_aton(multicast)#convierte el grupo multicast de dirección IPv4
# del formato string a un formato de 32-bit binario empaquetado
mreq = struct.pack('4sL', group, socket.INADDR_ANY)
# Configuramos los parametros del grupo -> protocolo ip
# Añadimos al socket el grupo multicast
self.sock.setsockopt(socket.IPPROTO_IP,socket.IP_ADD_MEMBERSHIP,mreq)
```

La siguiente función de la clase que nos encontramos es onEnter, dicha función únicamente llama a la función enviar. Se ha definido para poder enviar los mensajes con el teclado.

La función enviar se encarga de recoger el mensaje escrito, pasarlo a la función actualizar y limpiar la caja de texto.

En la función actualizar se crea una hebra, para poder recibir los mensajes mientras se desarrolla la aplicación. Además, enviamos el mensaje al grupo y actualizamos todos los mensajes llamando a la función actualizoLabel.

La función clientThread, implementa el desarrollo de la hebra. Se crea un bucle infinito para recibir todos los mensajes del grupo en tiempo real. Una vez recibido se actualizan todos los mensajes llamando a la función actualizoLabel

Finalmente, la función actualizoLabel, simplemente escribe en la interfaz todos los mensajes de los usuarios recibidos hasta el momento.

```python
def onEnter(self,event):
    self.enviar()

def enviar(self): #Esta función será la encargada de enviar el mensaje al pulsar el boton
    self.actualizar(self.text.get()) # Almacenamos el mensaje del entry para enviarlo
    self.text.set('') # Borramos el mensaje que hemos escrito

def actualizar(self,mensaje):
    #self.sock.sendto(mensaje.encode('utf-8'),(self.multicast,self.port)) #Enviamos mensaje al servidor
    try:
        Thread(target=self.clientThread).start()
    except:
        print("Thread did not start")

    lon=len(self.nicks) # Almacenamos el tamaño para añadir el mensaje en la última posición
    self.nicks[(self.usuario,lon)]=mensaje #Añadimos el mensaje al último usuario
    self.sock.sendto(pickle.dumps(self.nicks),(self.multicast, self.port)) # Enviamos mensaje
    self.actualizoLabel() #En cuanto añadimos un nuevo mensaje hemos de actualizar el Label

#Creamos una hebra para poder actualizar los mensajes recibidos
def clientThread(self):
    while True:
        receptor = self.sock.recv(1024) #Recibimos los mensajes
        msg2 = pickle.loads(receptor) #Recibimos su variable nicks en el cual su mensaje es el últimno
        #actualizamos nicks
        self.nicks=msg2
        # Para no perder información
        # Las claves de un diccionario no pueden ser iguales, si un cliente escribiese un mensaje dos veces, si no lo ordenamos,
        # se perderían los mensajes anteriores
        self.actualizoLabel()

# Creamos una función que actualice el Label de los mensajes a medida que enviamos y recibimos mensajes
def actualizoLabel(self):
    #Convertimos el diccionario en un string
    msg_completos='Mensajes Recibidos: \n' #Definimos un string donde se encadenarán todos los mensajes recibidos hasta el momento
    for user,ms in self.nicks.items(): #recorremos el diccionario
        if user=='': #Pasamos del primer usuario
            continue #Continuamos a la siguiente iteración
        #Vamos almacenando en la variable los distintos mensajes con el nombre del usuario asociado
        msg_completos = msg_completos + '<' + str(user[0]) + '>: ' + str(ms) + '\n'
    #Actualizamos el label con los mensajes del grupo
    self.mensajes=StringVar(value=msg_completos) #En el servidor se almacenan los mensajes de los participantes del grupo
    self.label1.configure(textvariable=self.mensajes) #La añadimos al label1 para que salgan los mensajes
```

Finalmente, nos encontramos con la función main, la cual es la principal, únicamente llama a la clase.

```python
def main(): #Función principal donde se ejecuta el desarrollo del código
    mi_app=Aplicacion() #Llamamos a la clase aplicación que se encarga de crear la interfaz
    return 0

#Si ejecutamos desde la consola python pract.py entonces __name__ será igual a __main___
if __name__=='__main__':
    main()
```

Un ejemplo del desarrollo de la práctica es el siguiente, desde la terminal en la línea de comandos ejecutamos el código.

Primero se nos solicitará que introduzcamos un usuario el cual identificará nuestros mensajes, una vez escrito, aparecerá la interfaz gráfica.

Para ver un ejemplo de conversación entre dos usuarios, abrimos otra ventana de la terminal y volvemos a ejecutar el código.