Práctica 3.2.2: Implementación (parcial) de un servidor TFTP

Descripción:

En esta práctica el/la estudiante tendrá que implementar en Python un servidor (parcial) de la aplicación TFTP. En concreto tendrá que implementar la opción de bajar un fichero del servidor (opción *Read Request* o *RRQ*) desde un cliente TFTP.

Material:

Para hacer esta práctica se dispone de los siguientes recursos:

1. Fichero con el código fuente del servidor parcialmente desarrollado (*tftp_ser_rrq.py*). Éste es el fichero que habrá que modificar para desarrollar la práctica.
2. Una carpeta con el nombre *data* con varios ficheros con un tamaño concreto y contenido indiferente que podremos utilizar para hacer pruebas. Ya mencionados en el enunciado de la práctica del cliente TFTP.
3. Un cliente TFTP para probar el servidor que desarrolles. Puedes usar el propuesto como resolución de la práctica anterior “Implementación (parcial) de un cliente TFTP”.

Protocolo de aplicación

El protocolo de la aplicación TFTP es bastante sencillo y ya ha sido explicado en clase. Todos los detalles los podemos encontrar en el RFC 1350. De todas formas, a continuación se resume el procedimiento RRQ:

1. El cliente envía el comando RRQ, el nombre del fichero a descargar e indica que lo quiere hacer en modo binario. Formato: El valor 1 en binario en 2 bytes para codificar el comando, el nombre del fichero, el valor 0 en binario en un byte para indicar el fin del nombre del fichero, la palabra “*octet*” (modo de descarga en binario) segida de un 0 en binario en un byte. También se puede usar la palabra “*binary*”, en lugar de “*octect*”, para indicar lo mismo, además ambas escritas en letras mayúscula o minúscula indistintamente.
2. Si todo va bien, el servidor debería responder con un paquete DATA: Código 3 en binario en 2 bytes, el número de bloque en binario en 2 bytes (empezando por 1) y el siguiente bloque de datos del fichero de 512 bytes.
3. Para confirmar la recepción de datos, el cliente responde con un ACK: Código 4 en binario en 2 bytes y el número de bloque recibido en binario y en 2 bytes.

Los pasos 2 y 3 se repetirán hasta que el servidor envíe el último bloque de datos, que siempre será de menos de 512 bytes (incluído el tamaño 0).

En esta práctica, puesto que tanto cliente como servidor estarán en la misma red local, la probabilidad de pérdida de paquetes es muy baja. Debido a esto, y para simplificar el trabajo, supondremos que no se perderán paquetes y, por tanto, no tomaremos ninguna medida para afrontar este caso.

Servidor

El/la estudiante tendrá que implementar un servidor TFTP que atienda peticiones RRQ. Para simplificar el trabajo tendremos en cuenta las siguientes consideraciones:

- A pesar de que en el paso 2 del protocolo de aplicación se puede usar otro puerto diferente, en esta práctica usaremos el mismo.
- Si mientras el servidor está atendiendo una petición RRQ de un cliente, recibiera otra petición de otro cliente, a éste le responderá con un mensaje de error y seguirá atendiendo al cliente anterior.

Cliente

El programa cliente a usar para probar el correcto funcionamiento del servidor a implementar puede ser el desarrollado en la práctica anterior “Implementación (parcial) de un cliente TFTP” o bien el propuesto como resolución de esta práctica que encontrarás disponible en el aula virtual de eGela.

Dinámica de trabajo

1. Terminar de implementar el servidor y probarlo con el cliente disponible.

Fichero *“tftp_ser_rrq.py”*:

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

import sys
import os
import socket

NULL  = b'\x00'
RRQ   = b'\x00\x01'
WRQ   = b'\x00\x02'
DATA  = b'\x00\x03'
ACK   = b'\x00\x04'
ERROR = b'\x00\x05'

PORT = 50069
BLOCK_SIZE = 512
FILES_PATH ='./data/'

def send_error(s, addr, code, message):
      resp  = ERROR
      resp += code.to_bytes(2, 'big')
      resp += message.encode()
      resp += NULL
      s.sendto(resp, addr)

def send_file(s, addr, filename):
      try:
            f = open(os.path.join(FILES_PATH, filename), 'rb')
      except:
            send_error(s, addr, 1, 'File not found.')
            exit(1)

      data = f.read(BLOCK_SIZE)
      """A COMPLETAR POR EL/LA ESTUDIANTE:
      Enviar al cliente el primer bloque de datos (DATA)
      """

      block_num = 1
      last = False if len(data) == BLOCK_SIZE else True
      while True:
            """A COMPLETAR POR EL/LA ESTUDIANTE:
            Recibir el mensaje del cliente.
```

```
            Si es un mensaje de error, terminar.
            Comprobar que es un ACK, si no, volver al comienzo del bucle.
            Comprobar que el número de bloque es el correcto (block_num), si no,
volver al comienzo del bucle.
            """
            if last:
                  break
            block_num += 1
            data = f.read(BLOCK_SIZE)
            """A COMPLETAR POR EL/LA ESTUDIANTE:
            Enviar al cliente el siguiente bloque de datos (DATA)
            """
            if len(data) < BLOCK_SIZE:
                  last = True

      print('{}: {} bytes sent.'.format(filename, (block_num - 1) * BLOCK_SIZE +
len(data)))
      f.close()

if __name__ == '__main__':
      """A COMPLETAR POR EL/LA ESTUDIANTE:
      Crear un socket UDP en s y asociarle el puerto PORT.
      """

      while True:
            req, cli_addr = s.recvfrom(64)

            opcode = req[:2]
            if opcode != RRQ:
                  send_error(s, cli_addr, 5, 'Unexpected opcode.')
            else:
                  filename, mode, _ = req[2:].split(b'\x00')
                  if mode.decode().lower() not in ('octet', 'binary'):
                        send_error(s, cli_addr, 0, 'Mode unkown or not
implemented')
                        continue
            filename = os.path.basename(filename.decode()) # For security,
filter possible paths.
            send_file(s, cli_addr, filename)
```

Posibles mejoras

1. Mostrar en pantalla la dirección IP y número de puerto del cliente que envió el comando RRQ y asegurar que todos los mensajes recibidos fueron enviados de esta misma dirección y puerto. Si no es así, descartar ese mensaje.
2. Si en algún momento, el mensaje recibido del cliente es de tipo ERROR, mostrar en pantalla el código y el mensaje de error.
3. Implementar la opción de subir un fichero al servidor (opción *Write Request* o *WRQ*).