Sistemas Distribuidos

VirtualDiskGII

– Grupo 1 –

Grado en Ingeniería Informática

7 de junio de 2015

Índice general

1. Miembros del grupo

Los miembros del grupo 1, ordenados alfabéticamente por el primer apellido, son:

- Emilio Bello Villanueva
- Juan Carlos de la Torre Macías
- Manuel Francisco Aparicio
- José Manuel Vidal Jiménez

2. Aportación de VirtualDiskGII a la comunidad

VirutalDiskGII es un sistema implementado en Python 2.7 que, usando librerías como ZeroMQ, permite a un ordenador compartir ficheros con varios ordenadores.

2.1. Justificación

Una de las peticiones más concurrentes entre alumnos de tercero y cuarto curso del grado de ingeniería informática es la de documentación de una especialidad diferente a la que estás cursando. Queramos o no, son muchas las ocasiones en las que las menciones se entrelazan, siendo necesario tener conocimientos de unas para elaborar trabajos de calidad en otras.

En primer y segundo curso es habitual el uso de una carpeta en dropbox para compartir todo tipo de información, convirtiéndose en un servidor de experiencias de años anteriores donde algunos aportan y todos consumen. En primero, el que aporta lo hace de buena fé, sin poner en cuestión el feedback de sus aportaciones. En segundo, la situación es similar, pero ya los aportadores empiezan a balancear lo aportado frente a lo recibido, balance que suele salir negativo y que provoca el abandono de dicha actividad aportadora.

En tercero y cuarto, el celo por lo propio es el gran dominante. Es muy complicado encontrar a alguien dispuesto a compartir. O por lo menos gratuitamente. Si deseas información, debes pagar con información.

Y es en esta situación donde hemos puesto nuestro punto de mira. La pregunta es: ¿Qué podemos ofrecer a los alumnos de tercero y cuarto que facilite el intercambio de información?

La respuesta es clara. Deberíamos ofrecer a los alumnos un sistema cliente-servidor. Dicho sistema debe permitir a un usuario ofrecer sus ficheros a los demás a través de la red de la UCA. Pero, para poder salvaguardar el celo existente, el que ofrece los ficheros debe poder controlar quiénes acceden al mismo.

2.2. Premisas

VirtualDiskGII debe dar respuesta a las siguientes premisas:

- Ante el dinamismo en la obtención de IPs, los clientes deben poder introducir manualmente la dirección IP del servidor.
- Cuando un cliente quiera un fichero, el propietario del servidor deberá facilitarle las credenciales para que pueda acceder al espacio compartido.
- Un cliente autorizado debe poder enviar un fichero al servidor.
- Un mismo servidor debe poder dar servicio a varios clientes.
- En el servidor no podrán alojarse dos ficheros con el mismo nombre.
- Un cliente no puede subir un fichero existente en el servidor, a menos que se indique explícitamente.

3. Código fuente.

3.1. Servicio servidor

El servicio servidor deberá estar corriendo, en primer o segundo plano, en cada ordenador que desee actuar como servidor. En dicho código se deberá configurar las credenciales de los clientes autorizados.

A continuación se muestra el código fuente del fichero **server.py**:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import zmq
import zmq.auth
from zmq.auth.thread import ThreadAuthenticator
import os
import sha
import sys

context = zmq.Context()

auth = ThreadAuthenticator(context)
auth.start()
auth.configure_plain(domain='*', passwords=
{
  'admin': 'admin',
  'user': 'user'
})
socket = context.socket(zmq.REP)
socket.plain_server = True
socket.bind("tcp://*:5059")

while True:
  try:
    #Leer comando del cliente
    comando = socket.recv()

    if comando == 'LS':
      #Listar.
      resultado = ''
      listado = os.listdir('Compartir/')

      #Para cada fichero en Compartir/, calcular su hash y enviar
```

```
    #nombre y hash
    for fichero in listado:
      s = sha.new()
      s.update(fichero)
      flujo = open('Compartir/' + fichero, 'rb')

      leido = flujo.read(1024)
      while len(leido) != 0:
        s.update(leido)
        leido = flujo.read(1024)

      resultado += fichero + '\t' + s.hexdigest() + '\n'

    #Enviar listado
    socket.send(resultado)

  elif 'GET' in comando:
    #El cliente quiere que enviemos un fichero
    nombre = comando.replace('GET ', '', 1)

    if os.path.isfile('Compartir/' + nombre):
      #El fichero existe.
      socket.send('OK')
      respuesta = socket.recv()

      if respuesta == 'READY':
        #Abrimos el fichero y lo enviamos
        flujo = open('Compartir/' + nombre, 'rb')
        socket.send(flujo.read())
        flujo.close()
      else:
        socket.send('NOT FOUND')

  elif 'PUT' in comando:
    #El cliente quiere enviarnos un fichero
    nombre = comando.replace('PUT ', '', 1)
    if os.path.isfile('Compartir/' + nombre):
      #El fichero existe. Lo abrimos y enviamos el hash para ver
      #si es el mismo.
      flujo = open('Compartir/' + nombre, 'rb')
      s = sha.new()
      s.update(nombre)

      leido = flujo.read(1024)
      while len(leido) != 0:
        s.update(leido)
        leido = flujo.read(1024)
```

```
        flujo.close()

        #Enviamos el hash calculado.
        socket.send(s.hexdigest())
        #Esperamos a la respuesta del cliente
        respuesta = socket.recv()

        if respuesta == 'OK':
          #El fichero es el mismo. No hacer nada.
          socket.send('')
        elif respuesta == 'OVERWRITE':
          #El cliente quiere sobreescribir.
          flujo = open('Compartir/' + nombre, 'wb')
          socket.send('READY')
          flujo.write(socket.recv())
          flujo.close()
          socket.send('DONE')
        else:
          #El fichero no existe. Abrimos el flujo y escribimos.
          flujo = open('Compartir/' + nombre, 'wb')
          socket.send('READY')
          flujo.write(socket.recv())
          flujo.close()
          socket.send('DONE')

    elif 'RM ' in comando:
      #El cliente quiere borrar un fichero
      nombre = comando.replace('RM ', '', 1)
      if os.path.isfile('Compartir/' + nombre):
        #El fichero existe. Lo borramos.
        os.remove('Compartir/' + nombre)
        socket.send('DONE')
      else:
        #El fichero no existe.
        socket.send('NOT FOUND')

    elif comando == 'PING!':
      socket.send('PONG!')
  except KeyboardInterrupt:
    auth.stop()
    print '\n Adios!'
    sys.exit()
```

3.2. Servicio cliente

El servicio cliente deberá estar corriendo en primer plano en cada ordenador que desee conectar con un servidor.

A continuación se muestra el código fuente del fichero **client.py**:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import sys
import zmq
import sha
import time

direccion = raw_input('Bienvenido. Introduce direccion y puerto del servidor.'
                      '(Por defecto localhost:5059)\n< ')
username = raw_input('Introduce nombre de usuario\n< ')
password = raw_input('Introduce contrasena\n< ')

if direccion == '':
  direccion = 'localhost:5059'

context = zmq.Context()
socket = context.socket(zmq.REQ)
socket.plain_username = username
socket.plain_password = password
socket.connect('tcp://' + direccion)

while True:
  try:
    #Prompt
    comando = raw_input('> ')

    #Procesar el comando introducido
    if comando == 'ls':
      #Listar ficheros en el servidor
      socket.send('LS')
      print socket.recv(),


    elif 'get ' in comando:
      #Obtener un fichero del servidor.
      #Enviamos la peticion al servidor: GET nombrefichero
      peticion = comando.replace('get ', 'GET ', 1)
      socket.send(peticion)
      respuesta = socket.recv()
```

```
    if respuesta == 'OK':
      #Abrimos el fichero donde vamos a guardar lo que nos envie el
      #servidor
      guardar = open('Recibido/' + comando.replace('get ', '', 1), 'wb')

      #Avisamos de que estamos preparados
      socket.send('READY')

      #Escribimos la respuesta
      guardar.write(socket.recv())
      guardar.close()

    elif respuesta == 'NOT FOUND':
      #El fichero no existe. Alertar y no hacer nada
      print 'El fichero no existe en el servidor. Abortado.'

    else:
      print 'Algo no ha ido bien...'

  elif 'put ' in comando:
    #Subir un fichero al servidor.
    #Enviamos la peticion al servidor: PUT nombrefichero
    peticion = comando.replace('put ', 'PUT ', 1)
    socket.send(peticion)
    #Abrimos el fichero a enviar
    fichero = comando.replace('put ', '', 1)
    flujo = open('Recibido/' + fichero, 'rb')

    #Leemos el estado del servidor
    respuesta = socket.recv()

    if respuesta == 'READY':
      #El servidor esta listo. Enviamos el fichero.
      socket.send(flujo.read())

      #Leemos la respuesta
      respuesta = socket.recv()

      if respuesta == 'DONE':
        #Todo correcto.
        print 'Fichero enviado.'
      elif respuesta == 'ERROR':
        #Algo ha ido mal
        print 'Algo ha ido mal... fichero no enviado.'

    elif 'READY' not in respuesta:
```

```
    #Si el servidor no esta listo, es porque ya existe un
    #fichero con el mismo nombre. Nos enviara por tanto el hash.
    #Comprobamos si es el mismo.
    s = sha.new()
    s.update(fichero)
    leido = flujo.read(1024)
    while len(leido) != 0:
      s.update(leido)
      leido = flujo.read(1024)

    if s.hexdigest() == respuesta:
      #El fichero es el mismo. No se hace nada.
      print 'El fichero ya esta en el servidor.'
    else:
      #El fichero no es el mismo. Preguntar si queremos
      #sobreescribir
      print 'El fichero ', fichero, ', ya existe en el servidor, '
                'con un contenido distinto. Continuar?'
      p = raw_input('(Y/N) ')

      if 'Y' == p or 'y' == p:
        #Sobreescribir.
        socket.send('OVERWRITE')
        respuesta = socket.recv()

        if respuesta == 'READY':
          flujo.seek(0, 0)
          socket.send(flujo.read())
          respuesta = socket.recv()
          if respuesta == 'DONE':
            print 'Fichero enviado.'
          elif respuesta == 'ERROR':
            print 'Algo ha ido mal... fichero no enviado.'
      elif 'N' == p or 'n' == p:
        #No hacer nada. Comunicar al servidor que todo
        #correcto y paramos.
        socket.send('OK')
        socket.recv()
      else:
        #El usuario ha introducido algo distinto a Y/y/N/n
        #No hacer nada. Comunicar al servidor que todo
        #correcto y paramos.
        print 'Respuesta incorrecta. Abortado.'
        socket.send('OK')
        socket.recv()
elif 'rm ' in comando:
  #Borrar un fichero del servidor.
```

```
      socket.send(comando.replace('rm ', 'RM ', 1))
      respuesta = socket.recv()

      if respuesta == 'NOT FOUND':
        #El fichero no existe
        print 'El fichero no existe en el servidor'
      elif 'DONE' not in respuesta:
        #La respuesta no ha sido ni NOT FOUND ni DONE.
        #Algo ha ido mal.
        print 'Algo ha ido mal...'

    elif comando == 'quit':
      print 'Adios!'
      break

    elif comando == 'ping':
      tic = time.time()
      socket.send('PING!')
      respuesta = socket.recv()
      toc = time.time()

      print respuesta, ' ', (toc-tic), ' s.'
    else:
      print 'Comando no encontrado.'
      print 'ls          \tLista los ficheros en el servidor remoto.'
      print 'get <nombre>\tDescarga el fichero <nombre> del servidor remoto.'
      print 'put <nombre>\tSube el fichero <nombre> al servidor remoto.'
      print 'rm  <nombre>\tElimina el fichero <nombre> del servidor remoto.'
      print 'ping        \tRealiza ping al servidor.'
  except KeyboardInterrupt:
        print('\n Adios!')
        sys.exit()
```

4. Instalación y funcionamiento.

4.1. Ejecución del servidor

En el directorio donde ejecutemos el servidor deberá contener una carpeta denominada **Compartir**. Si deseamos autorizar a un usuario, deberemos hacerlo en la estructura para tal fin existente en el código. En Linux, para ejecutarlo en segundo plano introduciremos:

$ python server.py &

4.2. Ejecución del cliente

En el directorio donde ejecutemos el cliente deberá existir una carpeta denominada **Recibido**. Ejecutaremos el programa introduciendo:

$ python client.py

Una vez lanzado el programa, nos solicitará la dirección ip del servidor y el puerto en el que éste está corriendo. Si pulsamos intro sin introducir nada, tomará el valor por defecto (localhost:5059).

A continuación, nos solicitará el usuario y la contraseña. Dichas credenciales estarán dadas de alta en el servidor y nos deben ser facilitadas por el administrador del mismo.

Por último, ya podemos hacer uso de VirtualDiskGII. Para una mayor usabilidad, se describen los comandos admitidos y su funcionamiento, aunque si introducimos un comando no válido, el mismo programa nos mostrará un mensaje con los comandos válidos:

- **ls**
  - El cliente envía LS al servidor
  - El servidor devuelve una lista de ficheros y sus hash
- **get**
  1. El cliente envía GET *nombre* al servidor
  2. El servidor:
     - Si el fichero existe:
       a) El servidor responde OK
       b) El cliente responde READY

c) El servidor envía el flujo de bytes del fichero

- Si el fichero no existe:

a) El servidor responde NOT FOUND

- **put**

1. El cliente envía PUT *nombre* al servidor
2. El servidor comprueba si existe el fichero:

- Si el fichero existe:

a) El servidor responde con el hash del fichero

b) El cliente comprueba si es igual al hash del que quiere enviar

- Si es igual. Acaba y no hace nada.
- Si es distinto. Pregunta al usuario si quiere sustituirlo.

1) El usuario no quiere sustituirlo.

- El cliente envía OK.
- El servidor prepara la nueva conexión enviando una cadena vacía.

2) El usuario quiere sustituirlo.

- El cliente envía OVERWRITE.
- El servidor responde READY.
- El cliente envía el flujo de bytes del fichero.
- El servidor responde DONE.

- Si el fichero no existe:

a) El servidor responde READY.

b) El cliente envía el flujo de bytes.

c) El servidor responde DONE.

- **rm**

1. El cliente envía RM *nombre* al servidor.
2. El servidor comprueba si existe el fichero:

- Si el fichero existe. El servidor elimina el fichero y responde con DONE.

        - Si el fichero no existe. El servidor responde con NOT FOUND.

- **ping**

    1. El cliente envía PING! al servidor.
    2. El servidor responde PONG! al cliente.

- **quit (o CTRL+C)**

    1. El cliente finaliza su ejecución.