Lab.04 Network operations

Objetivo: Crear una aplicación con un servidor ECHO y tres tipos de clientes diferentes.

1. Gestión de fragmentos en un TabLayout

Vamos a gestionar los fragmentos con un SwipePage y Tab adapter dinámico (ver teoría).

1. Crear un nuevo proyecto *Lab05.Apellido1* empleando la plantilla *Tabbed Activity* con *AcitivityName* NetActiv y *Navigation Style*: *Action Bar Tabs (with ViewPager).*
2. Eliminar el Floating Action Button del código Java y del layout principal y la definición del fragmento *PlaceholderFragment* (ya crearemos los nuestros).
3. Realizar las modificaciones pertinentes para tener un número variable de fragmentos.

2. Cliente echo

Crear un fragmento *EchoFragm* que envíe y reciba mensajes a través de un socket [1]

1. Crear un layout: con 4 EditText (*et_name, et_host, et_port* y *et_msg*), 2 botones (*but_send* y *but_cancel*) y 1 ListView *lv_display.*
2. Comprobar permiso de acceso a internet en el fichero de manifiesto.
3. Si aún no existe, crear un socket que se conecte al servidor con IP *et_host* y puerto *et_port.* Incluir valores por defecto: IP '"193.144.50.23" y el puerto "12321" y emplear hilos.
4. Una vez creado el socket, el primer mensaje a enviar es el nombre del cliente (*et_name*). Deshabilitar los campos *et_name, et_host, et_port.*
5. En un bucle infinito esperar para enviar mensajes al servidor:
   a) Al pulsar el botón *but_send* recoger el texto de *et_msg* (puede tener varias líneas).
   b) Enviar el texto[2], recoger la respuesta y mostrarla en *lv_display.*
6. Al pulsar el botón *but_cancel* cerrar el socket, limpiar *lv_display* y habilitar de nuevo los campos *et_name, et_host, et_port.*

[1] Un **servidor ECHO** devuelve exactamente el mismo mensaje que ha recibido. Se puede implementar de varias maneras (apéndice A). En el mismo repositorio hay versiones para otros lenguajes. **Notas:**

- Si se emplea un servidor propio, conviene utilizar una red wifi sin "capar".
- Se puede usar `netcat` para implementar servidor (`nc` $-l$ 12321) o cliente (`nc` $<$ *host* $><$ *port* $>$) local.

[2] Si se usa *writeUTF*, no olvidar añadir fin de línea "\n".

3. Cliente Post

Realizar una conexión Post contra un servidor para validar un usuario y una contraseña. Se podría usar cualquier implementación[3]. Para las pruebas, se puede usar el servicio implementado en `"http://gac.udc.es/~cvazquez/psi/param.php"`.

1. Crear un nuev fragmento *PostFragm*.
2. Incluir en el layout 3 TextView (con textos "Username", "Passwd" e "IP"), 3 EditText (*et_username*, *et_passwd* y *et_ip*, con valores válidos por defecto) y 1 botón (*but_send*).
3. Recoger los parámetros y lanzar la consulta POST en un hilo secundario.
4. Añadir un cuarto TextView al layout (*tv_result*) para mostrar el resultado obtenido.

4. Cliente protocolo JSON

1. Añadir un nuevo fragmento *JsonFragm* para realizar consultas JSON.
2. Adaptar los ejemplos de la teoría para leer y mostrar los datos de algún servidor público. Hay ciudades (pocas) con servicios públicos: Gijón, Santander, Zaragoza, etc.

   En nuestro caso obtendremos la posición en tiempo real de los autobuses de Gijón (buscar en `http://datos.gijon.es/set.html`).
3. En el layout de este cliente JSON indicar la dirección del servicio (con un valor por defecto), un botón para enviar un ListView para mostrar los resultados obtenidos.

5. Servidor echo básico en Android

Crear un servidor ECHO en Android, es decir, que espere las conexiones de los clientes y que ejecute un hilo en segundo plano para gestionar cada cliente. El hilo será el mismo para todos los clientes y todos los puertos e implementará la tarea ECHO [4].

1. Añadir un nuevo fragmento *ServerFragm*.
2. Crear en su layout: dos botones (*but_listen* y *but_close*, este último deshabilitado), un EditText (*et_port*), tres TextView (*tv_ip*, *tv_client* y *tv_mensajes*[5]). Añadir un ScrollView por si hay muchas líneas de información.
3. Mostrar la IP Wi-Fi en *tv_ip* (muy útil para el programa cliente).

   Importante: comprobar que sea una IP válida (no `0.0.0.0` o `127.0.0.1`).

[3]: Se propone utilizar el programa PHP del apéndice B, ejecutado desde un servidor web Apache (copiar el fichero al directorio raíz del servidor web). Se podrían añadir más variables, pares usuarios-contraseña y códigos de respuesta.

[4]Existen varias versiones en *http://rosettacode.org/wiki/Echo_server*, incluido Java.

[5]También se pueden usar ListView para clientes y mensajes.

4. Al pulsar el botón *but_listen* crear un hilo secundario para el puerto de escucha del servidor. Deshabilitar *but_listen* y *et_port*, y habilitar el botón *but_close*.

5. En ese hilo secundario, esperar la conexión de un cliente (método *accept()*): [6]

   a) Al conectarse un nuevo cliente, lanzar un nuevo hilo independiente que gestione dicho cliente. Se lanzará un nuevo hilo por cada cliente que se conecte y el servidor debe atenderlos a todos.

   b) Mostrar todos los datos que se consideren relevantes de cada cliente (p.e. IP, puerto) en el TextView de clientes (*tv_client*), incluido su nombre.

   c) Al desconectarse un cliente, indicarlo en el TextView de clientes.

   d) Para implementar una tarea ECHO: primero leer los datos del cliente y después reenviárselos. Para depuración, se puede usar el cliente diseñado en el punto 2.

   e) Mostrar (añadir) los mensajes recibidos del cliente en la cola de mensajes (*tv_mensajes*).

6. Al pulsar *but_close*, cerrar las conexiones activas con los clientes y el socket de escucha del puerto correspondiente. Borrar los TextView de clientes y mensajes. Habilitar *but_listen* y *et_port*, y deshabilitar el botón *but_close*.

7. Añadir nuevos fragmentos o eliminar los ya existentes:

   - Desde el servidor Echo lanzar nuevos fragmentos de clientes Echo.
   - Cada cliente Echo se podría eliminar a sí mismo de la aplicación.
   - Obviamente, habría que añadir sendos *interfaces* para gestionar ambos eventos, ya que **sólo la actividad** puede añadir y eliminar fragmentos a la vista.

[6] Se recomienda aplicar el método *setSoTime(int miliseconds)* a la conexión para evitar esperas infinitas y también detectar y gestionar los cambios de estado de la conexión.

A. Servidor de ECHO en Python

```
## http://rosettacode.org/wiki/Echo_server
import SocketServer
HOST = "192.168.1.102" #"localhost"    ===> change in each server !!!!
PORT = 12322                           ===> change in each server !!!!

  # this server uses ThreadingMixIn - one thread per connection
  # replace with ForkMixIn to spawn a new process per connection
class EchoServer(SocketServer.ThreadingMixIn, SocketServer.TCPServer):
    pass # no need to override anything - default behavior is just fine

class EchoRequestHandler(SocketServer.StreamRequestHandler):
    """
    Handles one connection to the client.
    """
    def handle(self):
        print "connection from %s" % self.client_address[0]
        while True:
            line = self.rfile.readline()
            if not line: break
            print "%s wrote: %s" % (self.client_address[0],line.rstrip())
            self.wfile.write(line)
        print "%s disconnected" % self.client_address[0]

  # Create the server
server = EchoServer((HOST, PORT), EchoRequestHandler)
  # Activate the server; keep running until interrupt with Ctrl-C
print "server listening on %s:%s" % server.server_address
server.serve_forever()
```

B. Servidor PHP `param.php`

```
<?php
    $username = $_POST["username"];
    $password = $_POST["password"];
    if( (strcmp($username,"Carlos")==0) &&
        (strcmp($password,"Vazquez")==0) ) echo 'OK';
    else if( (strcmp($username,"car")==0) &&
        (strcmp($password,"vaz")==0) ) echo 'OK1';
    else if( (strcmp($username,"Marga")==0) &&
        (strcmp($password,"Amor")==0) ) echo 'OK2';
        else echo 'NOK';
?>
```