SIMONE RIZZO
24/6/20

MATRICOLA 562131 CLASSE B

Università di Pisa

Dipartimento di Informatica

DOCUMENTAZIONE PROGETTO – SAM

Sommario

SIMONE RIZZO
24/6/20

MATRICOLA 562131 CLASSE B

SVILUPPO APPLICAZIONI MOBILI

Documentazione (MediaMe) di Simone Rizzo

1 Introduzione:

Il progetto **MediaMe** consiste nella realizzazione di un'applicazione Android in grado di far **condividere foto** in modo **semplice** e veloce, utilizzando lo standard **Wifi Direc**t.

2 Scelte progettuali:

La scelta di far connettere i due dispositivi tramite tecnologia **Wifi** è dovuta principalmente ad una questione di velocità di trasferimento dei dati, poiché l'app invia una grande quantità di foto. L'applicazione è implementata secondo una architettura client-server che sfrutta il protocollo TCP-IP. L'accoppiamento dei due dispositivi viene effettuato tramite riconoscimento di **codice QR,** semplice e largamente utilizzato. Per convenzione utilizzeremo la parola "**server**" per indicare Il dispositivo che intende condividere i propri media, mentre utilizzeremo il termine "**client**" per indicare il dispositivo che li riceverà.

3 Architettura del sistema:

Il sistema è composto da due dispositivi che comunicano tramite protocollo **TCP-IP** in cui, una volta stabilita la connessione, entrambi avranno un thread in background di lettura e scrittura sul socket che funge da servizio. L'intera architettura si può descrivere in tre fasi principali: **connessione**, **invio ricezione dati**, **disconnessione**.

3.1 Connessione:

La connessione avviene tramite l'utilizzo della classe WifiP2pManager, che fornisce le funzionalità per avviare la ricerca dei dispositivi circostanti e instaurare la connessione. Vi è poi la necessità di monitorare lo stato del dispositivo mediante un **BroadcastReceiver** dichiarato dinamicamente. Esso ci permette di leggere sia i messaggi di sistema riguardanti il wifi che lo stato della connessione.

- WIFI_P2P_THIS_DEVICE_CHANGED_ACTION: ci permette di ottenere il mac address del dispositivo.
- WIFI_P2P_CONNECTION_CHANGED_ACTION: ci indica se la connessione è stabilita o meno.
- WIFI_P2P_PEERS_CHANGED_ACTION: ci indica se i peers locali sono cambiati.
- WIFI_P2P_STATE_CHANGED_ACTION: ci indica se il wifi è in funzione o meno.

L'accoppiamento dei dispositivi viene effettuato mediante riconoscimento di codice QR dove, al suo interno, è contenuto il MAC address del dispositivo.

La generazione e lettura del codice QR viene effettuata mediante librerie: "**com.budiyev.android:code-scanner:2.1.0", "androidmads.library.qrgenearator:QRGenearator:1.0.4"**. Dopo aver generato il codice, il server riceverà la notifica di un eventuale connessione tramite il **ConnectionInfoListener.** Il client invece leggerà il codice QR tramite camera e proverà a instaurare una connessione. Una volta stabilita la connessione, si creerà un Gruppo in cui: uno dei due dispositivi sarà il proprietario mentre l'altro sarà il cliente. Il proprietario del gruppo aprirà un *SocketServer* sul proprio indirizzo, mentre il cliente aprirà un *socket* che si collegherà all'indirizzo del proprietario del gruppo. Dato che si tratta di una connessione 1:1, i due dispositivi si possono comportare sia da client che da server.

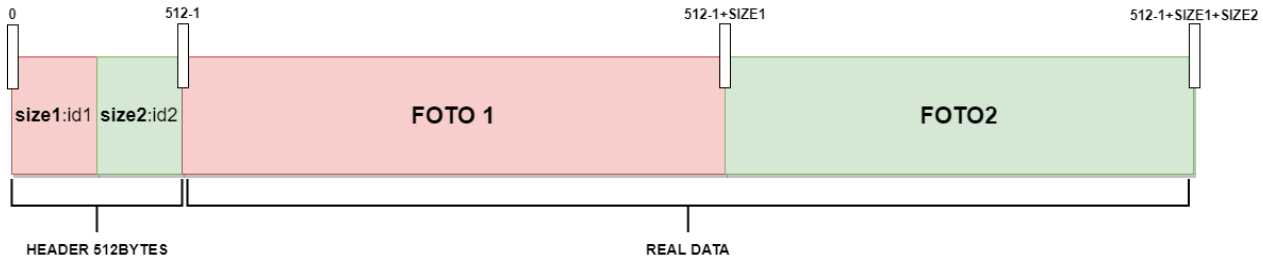
3.2 Invio ricezione dati:

Una volta stabilita la connessione, i due hosts eseguiranno il Thread dell'oggetto **SendReceive**, che si occuperà di ricevere e rispondere ai comandi grazie al socket aperto. Per leggere i dati ricevuti sul socket si è utilizzata la classe **DataInputStream.** Quest'ultima ci permette di leggere prima la dimensione del messaggio ricevuto chiamando ".readInt()" e poi leggere tutti i bytes del body del messaggio con il metodo ".readFully()". I primi 512 bytes del messaggio sono riservati all'header contenente i comandi, mentre i restanti sono riservati ai dati. Per scrivere i dati sul socket si utilizza la classe **DataOutputStream**. Ogni comando ricevuto viene decodificato, elaborato e la risposta viene successivamente inviata al mittente. Tutti e due i dispositivi in modalità client-server avranno in esecuzione lo stesso thread che offre le medesime funzionalità per ambo le parti. Questa scelta è dovuta al fatto che in futuro questa applicazione sarà predisposta a supportare la condivisione contemporanea delle gallerie da ambo i dispositivi.

3.2.1 Server:

Il server offre due funzionalità:

1. La prima permette di inviare gruppi di foto dalla propria galleria. Una volta ricevuto il comando, si esegue una query sul content provider (Vedi punto 8) che ci fornisce il path delle foto. Successivamente si leggono i bytes di queste foto che vengono inserite all'interno di un unico array in modo sequenziale e, al tempo stesso, si genera l'intestazione. Quest'ultima contiene una lista ordinata di coppie <**dimensione_foto**, **id_foto**> che viene inviata insieme ai dati stessi in un unico pacchetto.



2. La seconda funzionalità, invece, permette di inviare una singola foto con la massima qualità (conoscendo il suo id). Inoltre, viene implementata utilizzando il content provider (Vedi punto 8), lanciando una differente query sull'id ricevuto che ci restituirà il path della foto. Questa volta però, a differenza della prima funzionalità, non è necessario creare un header complesso ma basta scrivere l'intestazione con "path:nomefoto," e inviare l'array di bytes della foto.

3.2.2 Client:

Il client usufruisce dei servizi del server in due casi:

- Scrollando la RecyclerView si invierà la richiesta di ricevere altri blocchi di foto da visualizzare. Nel momento in cui il RecyclerView necessita la visualizzazione di nuove foto, viene spawnato un thread che invia sul socket il comando "get_media". Una volta ricevuta risposta sarà compito del thread SendReceive ricevere il messaggio, leggere tutte le foto sullo stream e salvarle nella memoria cache. In seguito, le aggiungerà nel recyclerview adapter notificando l'evento.
- Aprendo una singola immagine il client invierà al server la richiesta di avere la foto con massima qualità. Nel momento in cui viene cliccata una foto nella RecyclerView il client apre la nuova activiy

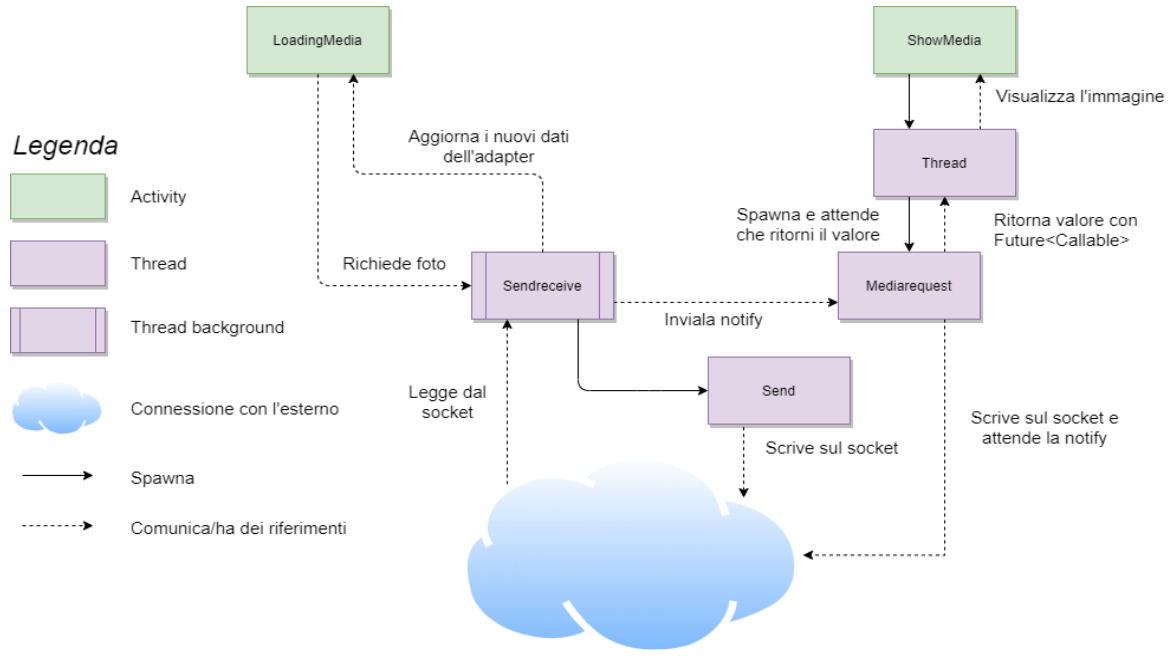
ShowMedia. Con l'avvio di questa nuova activity si lancerà thread di tipo **Callable** che invierà la richiesta sul socket e, successivamente, si bloccherà in attesa di notifica del risultato (Descritto nel punto 5). Ricevuta la foto, il thread SendReceive la scaricherà in memoria cache notificando e sbloccando il thread callable e facendogli così ritornare il path della foto da visualizzare.

3.3 Disconnessione:

La disconnessione dei dispositivi può avvenire in due modi: in modo volontario, cioè premendo il bottone Disconnect oppure in modo involontario (ad esempio perdita ricezione segnale, wifi off, IO Exception, chiusura dell'app, etc). In entrambi i modi si ritornerà nella schermata home.

4 Schema dei thread (caso significativo):

Questo schema mostra come i thread comunicano e da chi vengono generati nel caso più significativo del client.



5 Controllo concorrenza:

Per gestire la concorrenza nel progetto, si è scelto di utilizzare la funzione *synchronized* e variabili atomiche. Nella classe oggetto MediaRequest si utilizza la funzione *synchronized* sull'oggetto *fotoDownloaded* mettendosi in attesa che venga scaricata la foto e notificata dalla classe Sendreceive che legge dal socket la foto e notifica il suo percorso all'interno della variabile fotoDownloaded.

```java
public String call() throws Exception {
        try {
            byte[] headerBytes = header.getBytes();
            byte[] total = new byte[bytes.length+HEADER_SIZE];
            System.arraycopy(headerBytes,0,total,0,headerBytes.length);
            System.arraycopy(bytes,0,total,HEADER_SIZE,bytes.length);
            outputStream.writeInt(total.length);
            outputStream.write(total);
            outputStream.flush();
            String res="";
            /**
             * Il thread attende finchè la foto non è stata scaricata e settata all'interno di fotoDownloaded.
             */
            synchronized (fotoDownloaded)
            {
                while (fotoDownloaded.length()==0)
                    fotoDownloaded.wait();
                res=fotoDownloaded.toString();
                fotoDownloaded.setLength(0);
            }
            return res;
        } catch (IOException e) {
            e.printStackTrace();
            return null;
        }
    }
}
```

```java
else if(header.contains("path")) {
                byte[] message = new byte[lenght-HEADER_SIZE];
                String name = header.split(",")[0].split(":")[1];
                inputStream.readFully(message, 0, lenght-HEADER_SIZE);
                File downloadingMediaFile = new File(chacheDir, name + ".jpg");
                FileOutputStream out = new FileOutputStream(downloadingMediaFile);
                out.write(message, 0, message.length);
                out.close();
                synchronized (fotoDownloaded) {
                    fotoDownloaded.append(downloadingMediaFile.getAbsolutePath());
                    fotoDownloaded.notify();
                }
                Log.v("Connessione","Foto arrivata:"+fotoDownloaded);
            }
```

6 Descrizione sintetica classi definite:

App.java: classe che estende Application, serve ad implementare i NotificationChannel

BitmapUtils.java: classe di utilità, ci permette di comprimere, decodificare, ridimensionare le immagini in Bitmap.

ClientClass.java: classe che estende Thread, ci permette di connetterci al server

MyBroadcastReceiver.java: classe che estende BroadcastReceiver, ci permette di ricevere gli aggiornamenti sullo stato del wifi e wifi Direct.

Notificationreceiver.java: classe che estende Broadcastreceiver, utilizzata per gestire l'evento di volontà di disconnettersi dal servizio, effettuato attraverso le notifiche.

RecyclerViewAdapter.java: classe che estende RecyclerView.Adapter, ci permette di gestire la recyclerView contenente le foto scaricate.

Mynotification.java: classe Factory che genera oggetti Notification da utilizzare nei due canali per le notifiche.

SendReceive.java: classe che estende Thread, realizza effettivamente il servizio in background di invio e ricezione dei messaggi sul socket tcp.

ServerClass.java: classe che estende Thread, realizza il server ovvero apre la connessione in attesa di client.

7 Descrizione sintetica activity:

MainActivity.java: activity principale (Home)

Generaqr.java: activity che ci permette di generare il codice qr per effettuare la connessione da server.

Scannerizzaqr.java: activity che ci permette di leggere il codice qr per effettuare la connessione da client.

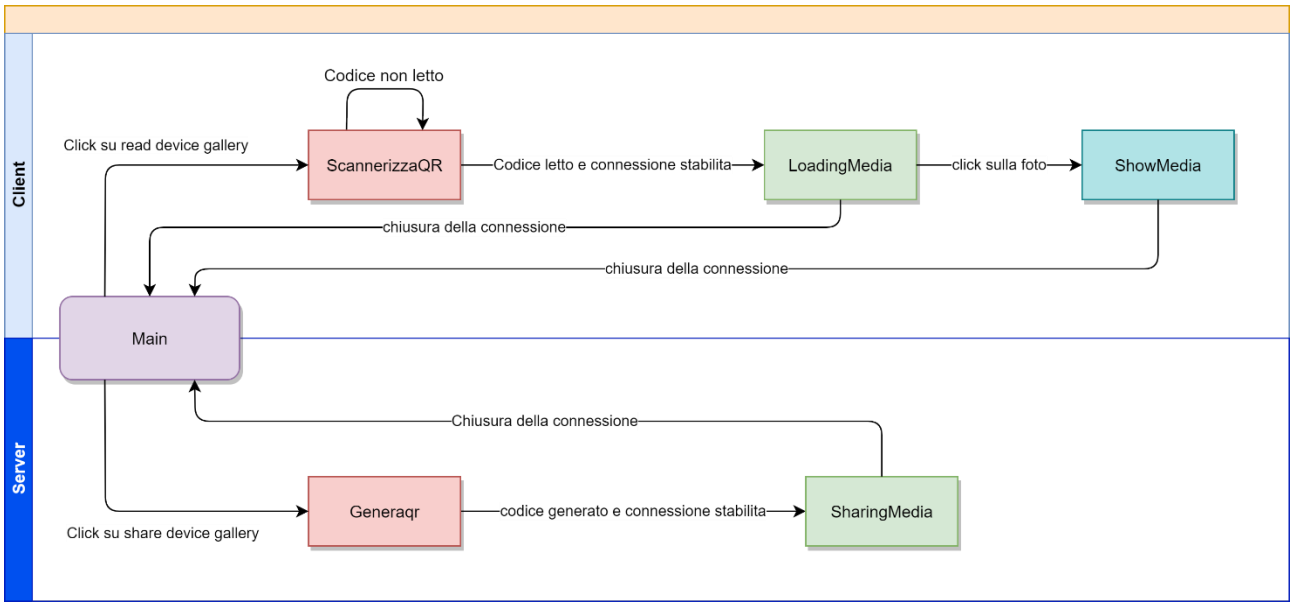
SharingMedia.java: activity che tiene in gestione lo share dei nostri media tramite il thread in background.

LoadingMedia.java: activity che ci permette di visualizzare le foto ricevute dal thread in background.

ShowMedia.java: activity che ci permette di visualizzare una foto al massimo della qualità e di scaricarla.

8 Schema di Navigazione delle Activities:

Server e Client sono intesi come il dispositivo che offre il servizio di sharing dei media e quello che ne usufruisce.



7 Sintassi protocollo di comunicazione:

L'intestazione dei messaggi è di lunghezza fissa ovvero 512 byte, la sintassi dei comandi posti all'interno dell'header sono i seguenti:

- **id:{numero},**: indica la rischiesta dal client di voler vedere l'immagine con quello specifico id
- **path:{nomefile},:** indica la ricezione di un immagine ad alta qualità da scaricare di nome nomefile
- **send_media**: indica il voler ricevere foto dal server.

- **size1:id1,size2:id2.....size{n}:id{n},**: indica la ricezione di n foto con ognuna possiede size[n] e id[n].

8 Content Provider:

Per richiedere le foto del dispositivo è stato necessario utilizzare il content resolver sul ***MediaStore.Images.Media*** che ci permette di effettuare queries sui dati. Le query utilizzate sono due e sono implementate all'interno di *SendReceive.java.*

- **VediFotoMedia**: dato il numero di foto lette ed n, si effettua una query proiettando: {***Data, ID, Date_taken, Bucket_Display_name***} **raggruppando** per *Date_Taken* in ordine **decrescente** e ponendo un **LIMIT *{letti},{n}*** che ci permette di leggere n foto alla volta senza dover aspettare troppo tempo.
- **GetMediaFromID**: dato un id univoco di una foto, si effettua una query proiettando {Data,ID} e filtrando solamente sulla foto che presenta quell'id.

Per salvare l'immagine scaricata nella nostra galleria, si utilizza **MediaStore.Images.Media.insertImage** che passandogli come parametro il path della nostra foto provvederà ad inserire i dati all'interno del content provider.

9 Ottimizzazione trasferimento file:

In fase di lettura dei media della propria galleria, il server effettua una compressione del Bitmap della foto del -20% della qualità e fissa la dimensione a 40px X 40px diminuendo così di molto il peso in numero di bytes di ogni foto e rendendo l'invio più efficiente.

Immagine Originale

Immagine Compressa

Qualità -20%
ridimensionato 40x40

10 Notifiche:

L'applicazione implementa un sistema di notifiche su due canali: **service** e **download**. Questi canali sono stati definiti nella classe **App.java** che estende Application, il cui compito è quello di inizializzare i due channel: il primo con priorità alta e il secondo con priorità bassa. La classe **MyNotification** genera oggetti Notification da utilizzare nei due canali per le notifiche. La notifica di tipo Service viene generata nel momento in cui i due dispositivi si connettono e avviano il servizio correttamente. Questa notifica è fissa per tutta la durata di esecuzione del servizio e presenta un bottone che permette, tramite pending intent, di comunicare con l'applicazione. L'app riceve questo messaggio grazie al BroadcastReceiver

Notificationreceiver che eseguirà la disconnessione e cancellazione della notifica. Mentre la notifica generata sul secondo canale segnala il corretto salvataggio della foto nella galleria.

11 Permessi app:

L'applicazione per il corretto funzionamento necessita dei seguenti permessi:

- CAMERA: per poter scannerizzare il codice QR.
- READ_EXTERNAL_STORAGE: per leggere le foto contenute sul nostro dispositivo.
- WRITE_EXTERNAL_STORAGE: per poter scrivere e così salvare le foto.
- ACCESS_NETWORK_STATE: per leggere lo stato della connessione.
- ACCESS_FINE_LOCATION: per risolvere un eventuale bug sul discovery dei peer del wifi p2p.
- ACCESS_WIFI_STATE: per conoscere lo stato del nostro wifi
- CHANGE_WIFI_STATE: per conoscere se è cambiato lo stato del nostro wifi
- INTERNET: per utilizzare lo standard Java Sockets

12 Librerie esterne utilizzate:

Le librerie esterne utilizzate sono le seguenti:

com.budiyev.android:code-scanner:2.1.0: Libreria utilizzata per scannerizzare il codice QR.

com.karumi:dexter:6.0.0: Libreria utilizzata per gestire i permessi dell'app a runtime.

androidmads.library.qrgenearator:QRGenearator:1.0.4: Libreria utilizzata per generare il codice qr

pl.droidsonroids.gif:android-gif-drawable:1.2.18: Libreria utilizzata per visualizzare su una view un'immagine gif.