Projet Programmation des mobiles

Projet Connexion Bluetooth

Département Sciences du Numérique
Parcours Réseaux
2ème Année

Yassir BOUISSE
Ladislas BISCHOFF

07 juin 2021

Table des matières

1 Objectif du projet

La plate-forme **Android** prend en charge la pile réseau **Bluetooth**, qui permet à un appareil d'échanger des données sans fil avec d'autres appareils Bluetooth.
Dans notre projet, l'application permet d'accéder à la fonctionnalité Bluetooth via les **API Bluetooth Android** qui permettent aux applications de se connecter sans fil à d'autres appareils Bluetooth, permettant des fonctionnalités sans fil point à point et multi-point.

L'objectif principal du notre projet est de bien faire communiquer deux smartphones Android pour qu'ils soient en mesure d'échanger des informations via une communication Bluetooth.
On mettra alors en place le socketBluetooth pour l'échange des données de monitoring.

2 Monitoring de processus à distance

Nous allons dans un premier temps créer une première Activité affichant un bouton.

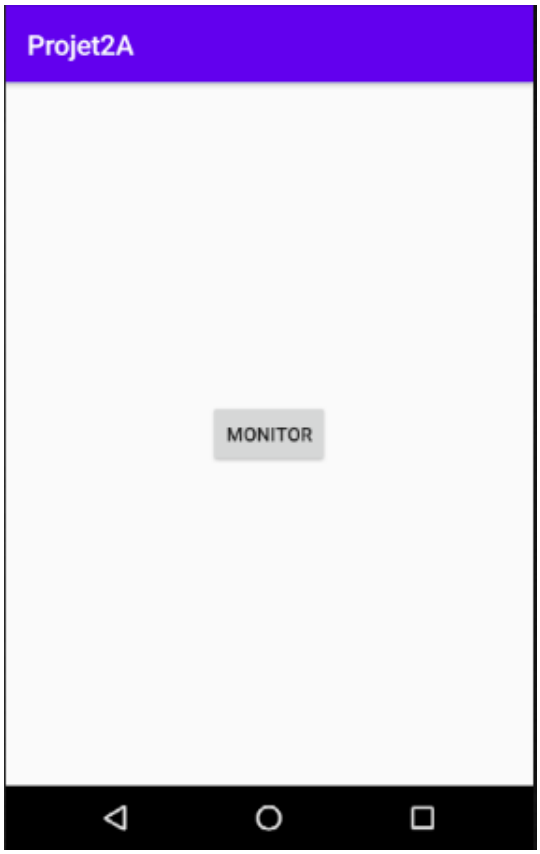

FIGURE 1 – Une première activité

Notre objectif ici est de lancer une nouvelle activité lors du clique de l'utilisateur sur le bouton. Il nous faut donc réaliser les activités suivantes :

— Creation de la nouvelle activité à lancer.
— Capture du clique sur le bouton.
— Assignation d'un comportement lors du clique sur le bouton.

Nous allons commencer par créer les différents éléments graphiques et par la suite nous y ajouterons des éléments de comportement. Nous proposons d'obtenir un affichage des applications actives (i.e processus systèmes) ayant la forme proposée dans la figure 2.

FIGURE 2 – L'affichage des informations relatives aux processus actifs

Une fois l'affichage mis en place, il nous faut maintenant peupler les widgets avec les informations relatives aux processus actifs du système. Nous devons donc récupérer le nom des applications en cours d'execution ainsi que leur consommation mémoire. Nous avons donc rusé un peu et utilisé les appels systèmes pour récupérer les informations relatives aux processus. Une solution est donc d'utiliser l'appel système *ps*.

Nous avons vu que nous pouvons créer des vues directement par programmation. Nous allons maintenant nous atteler à lancer un monitoring de l'utilisation de mémoire de chaque processus lors de l'appui sur le bouton MONITOR correspondant à chaque application affichée (cf. figure 2).

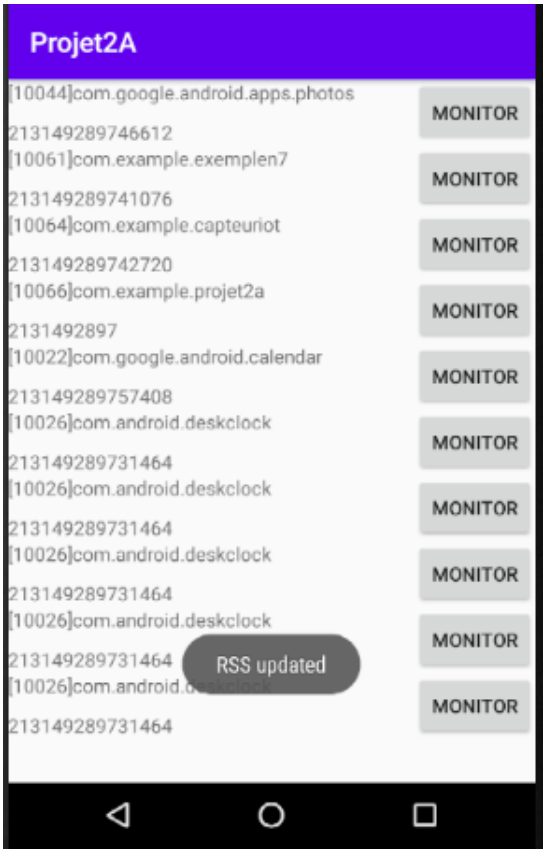

FIGURE 3 – La vue affichant les informations pour un process

Nous mettons à jour les informations de chaque processus à chaque fois que nous appuyons sur le bouton correspondant. L'utilisateur peut être aussi intéressé de savoir que le thread as bien été lancé. Nous avons ajouté un élément d'interface *popup*, permettant de visualiser cette information(cf. figure 3).

3 Socket Bluetooth : connexion

Il est demandé de créer une application qui s'installera sur deux smartphones Android et qui permettra de présenter l'état des processus du téléphone distant. Le premier écran de l'application permet de réaliser la connexion Bluetooth. Un des deux smartphones joue le rôle du **serveur Bluetooth** et l'autre celui du **client Bluetooth**. Une fois la connexion établie, il faut échanger des informations entre les téléphones. Cet échange se fait par l'ouverture d'un socket Bluetooth. L'ouverture du socket de communication passe par une étape de connexion.

Le client affiche une activité qui liste l'ensemble des applications installées sur le smartphone du serveur avec l'interface présentée dans la figure 2. Quand le client demande de monitorer un processus particulier, une requête est envoyée périodiquement via Bluetooth pour mettre à jour la valeur **RSS** monitorée.

Notre application démarre par un écran d'accueil où l'utilisateur initialise la connexion Bluetooth. Sur cet écran, on a positionné deux boutons, l'un pour choisir le mode **Serveur Bluetooth** et l'autre le mode **Client Bluetooth**.

FIGURE 4 – Ecran d'invite de connexion

Il faut cliquer sur CONNECTER SERVEUR sur le smartphone 'serveur', puis sur CONNECTER CLIENT sur le smartphone 'client'. Une fois que la connexion est établie, on pourra utiliser le socket Bluetooth pour communiquer,lire ou écrire des données dans l'objet socket.

FIGURE 5 – Ecran côté Serveur

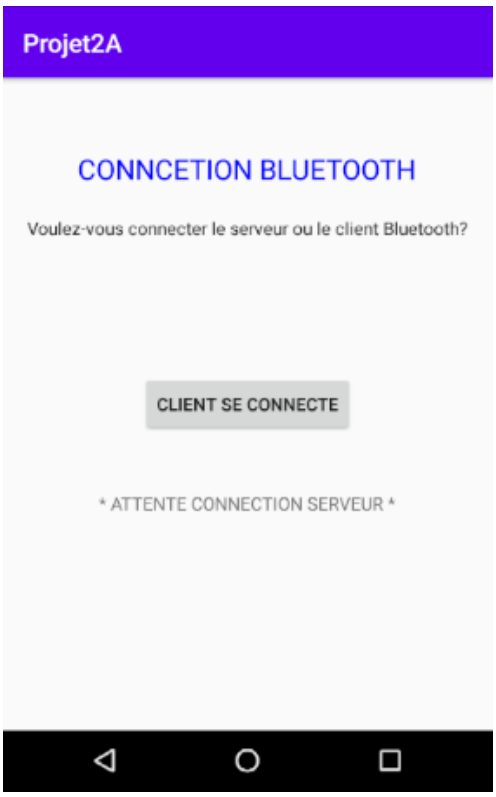

FIGURE 6 – Ecran côté Client

Pour obtenir le socket de communication, le serveur utilise un objet 'serveur de socket' *BluetoothServerSocket* qui attend la connexion d'un client pour retourner un socket Bluetooth. Ce serveur de socket est ensuite mis en écoute. Quand une requête de connexion arrive d'un client, ce 'serveur de socket' retourne un socket de communication de type *BluetoothSocket*. Ce socket est prêt à être manipulé pour échanger des messages avec le client.

4 Description de l'application générée

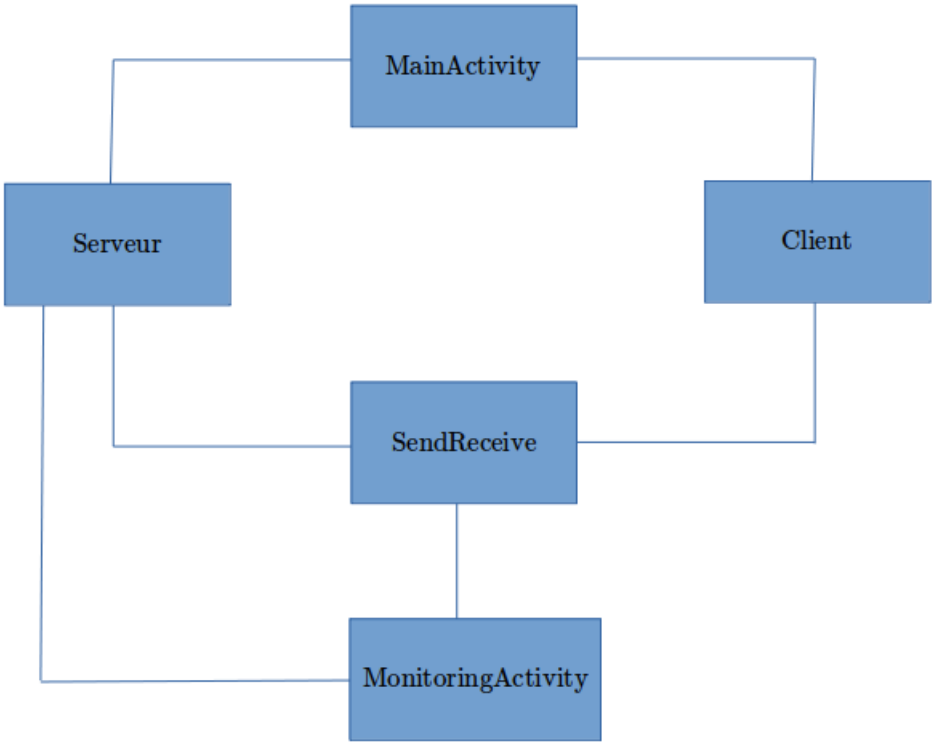

FIGURE 7 – Architecture de notre application

Notre aapplication sera répartie en deux activités : La première pour le choix du rôle Client/Serveur (MainActivity) et l'établissement de la connexion Bluetooth (Récupération du socket de communication). La deuxième se chargera d'extraire les informations de monitoring (MonitoringActivity), les envoyer vers l'autre bout de la connexion Bluetooth et enfin de récupérer les informations reçus et les afficher dans les deux côtés.

4.1 MainActivity

Au lancement de notre application on instancie les boutons pour inviter l'utilisateur de choisir entre Client et Serveur et on les mets sur écoute (cf. figure 4).
Attention : Il faut active le bluetooth s'il est désactivé.

```
// Partie traitement serveur
case R.id.btn_serveur:
    // Cacher le button du client
    btnClient.setVisibility(View.INVISIBLE);
    // Changer le texte affiché dans le button du servuer
    btnServeur.setText("SERVEUR EN ATTENTE");
    // Changer le texte view afficher dans l'application
    tv.setText("* ATTENTE CONNECTION CLIENT *");
```

FIGURE 8 – Bouton serveur

```
// Partie traitement client
case R.id.btn_client:
    // Cacher le button du serveur
    btnServeur.setVisibility(View.INVISIBLE);
    // Changer le texte affiché dans le button du client
    btnClient.setText("CLIENT SE CONNECTE");
    // Changer le texte view afficher dans l'application
    tv.setText("* ATTENTE CONNECTION SERVEUR *");
```

FIGURE 9 – Bouton client

On capture le clique sur le bouton et selon l'Id, on modifie l'affichage en attente d'établissement de connexion, du coté client on doit récupérer la liste des appareils appairer par Bluetooth et choisir le bon device et finalement on lance un **thread Client** et **thread Serveur**.

```
private class ServerClass extends Thread {
    private BluetoothServerSocket serverSocket;

    public ServerClass() {
        try {
            serverSocket = bluetoothAdapter.listenUsingRfcommWithServiceRecord(APP_NAME,MY_UUID);
        } ca
            com.example.projet2a.MainActivity.ServerClass
            private BluetoothServerSocket serverSocket
        }
    }

    public void run() {
        BluetoothSocket socket = null;
        while (socket == null) {
            try {
                // Retourner un socket si la connexion et établit avec le client.
                socket = serverSocket.accept();
                SocketHandler.setSocket(socket);
            } catch (IOException e) {
                e.printStackTrace();
            }
            if(socket != null) {
                // Envoyer-Recevoir
                sendReceive = new SendReceive(socket);
```

FIGURE 10 – Thread Serveur

Le thread Serveur a un attribut socket et à l'exécution lance un appel à la méthode accept et récupère le socket retourné pour le stocker dans un attribut d'une classe static SocketHandler.

```
private class ClientClass extends Thread {
    private BluetoothDevice device;
    private BluetoothSocket socket;

    public ClientClass (BluetoothDevice nv_device) {
        device = nv_device;
        // Obtenez un BluetoothSocket pour vous connecter avec le BluetoothDevice donné.
        try {
            // MY_UUID est la chaîne UUID de l'application,
            // également utilisée par le code serveur.
            socket = device.createRfcommSocketToServiceRecord(MY_UUID);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public void run() {
        bluetoothAdapter.cancelDiscovery();
        try {
            socket.connect();
            // Après réussite de la connexion.
            sendReceive = new SendReceive(socket);
            sendReceive.start();
        } catch (IOException e) {
            e.printStackTrace();
```

FIGURE 11 – Thread Client

Le thread Client de son coté fait un appel à la méthode connect sur le device choisi avant l'exécution du thread. De toute façons une fois la connexion dite établie on crée alors une intent pour lancer la deuxième activité et on l'envoie.

```
private class SendReceive extends Thread {
    private final BluetoothSocket bluetoothSocket;
    private final InputStream inputStream;
    private final OutputStream outputStream;

    public SendReceive (BluetoothSocket socket) {
        bluetoothSocket = socket;
        InputStream temponIn = null;
        OutputStream temponOut = null;

        try {
            temponIn = bluetoothSocket.getInputStream();
            temponOut = bluetoothSocket.getOutputStream();
        } catch (IOException e) {
            e.printStackTrace();
        }

        inputStream = temponIn;
        outputStream = temponOut;
    }

    public void run() {
        byte[] buffer = new byte[1024];
        int bytes;
```

FIGURE 12 – Thread SendReceive

Le thread SendReceive est alors instancié (et non lancé) par l'activité de connexion Bluetooth pour permettre les échanges de messages via le socket Bluetooth.

4.2 MonitoringActivity

Le traitement de cette activité est vu durant les deux seances du TP, aussi que dans la partie **Monitoring de processus à distance**.

On ce qui concerne la partie de mise à jour, lorsqu'on capture un click sur un bouton on extrait son Id par la methode View.getId() et on change la valeur du boolean correspondant dans la liste clicks. On lance par la suite un thread qui exécute le traitement nécessaire pour l'extraction des données et on stocke les données dans des variables List globales à l'activité, puis on envoie un message au handler qui récupere le textView correspondant aux boutons cliqué et met à jour le texte affiché à partir des variables globales.