Android
Threading Model

C. Raïevsky

UNIVERSITÉ Grenoble Alpes

IUT Valence

Département Informatique

Processus - Single thread

Par défaut, une application Android possède

- un seul processus.
- Un thread principal (“*UI Thread*”).

Ce thread principal est responsable de :

- la création des éléments graphiques,
- la propagation des événements utilisateurs.

Chaque thread possède une “file d’attente”

- Contient tous les événements non encore traités
- Il est possible d’ajouter des éléments dans cette file
- de préférence via un `Handler`
- Un `Looper` s’occupe de traiter les événements

Callbacks

Tous les callbacks sont exécutés par le thread UI

- `onCreate()`
- `onKeyPressed()`
- `onClick()`
- etc.

Pendant l'exécution de ces méthodes

AUCUN ÉVÉNEMENT NE PEUT ÊTRE TRAITÉ

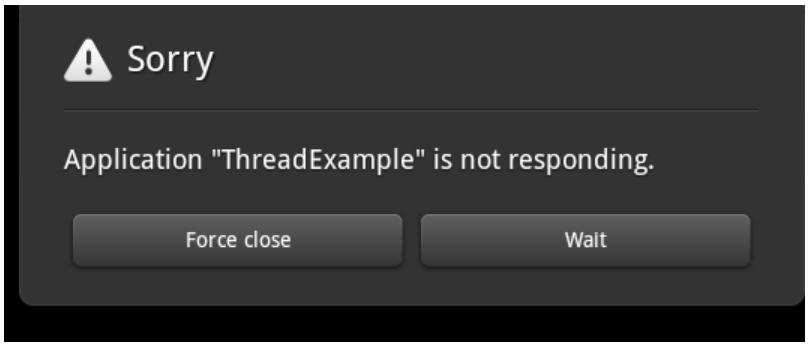
Plus de

- rafraichissement de l'interface,
- traitement des actions utilisateur.

ANR

Traitement long dans une de ces méthodes

- L'application apparaît comme bloquée à l'utilisateur,
- Provoque l'affichage par le système du fameux : **Application Not Responding**



- Objectivement MAL
- Donne une envie plus ou moins immédiate à l'utilisateur de désinstaller l'application...
- Pas de millions.

Comment éviter les ANR ?

Ne jamais bloquer le thread UI.

- Pas de traitements/calcul long,
- Pas d'opérations réseau,
- Pas de requête à des bases de données,
- Pas d'accès "disque"

dans une méthode exécutée dans ce thread.

Patience limitée

Un utilisateur met 200 millisecondes pour percevoir un délai. . .

Exemple

Charger une image sur click

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:orientation="vertical"
              android:layout_width="match_parent"
              android:layout_height="match_parent">
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="New Button"
        android:id="@+id/loadImageBtn"
        android:onClick="onLoadImageClick"/>

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/imageView3"
        android:layout_gravity="center_horizontal"/>
</LinearLayout>
```

source : developer.android.com

Exemple

Charger une image sur click

Première idée : création d'un thread.

```
public void onLoadImageClick(View v) {
    new Thread(new Runnable() {
        public void run() {
            Bitmap b = loadImageFromNetwork("http://example.com/image.png");
            mImageView.setImageBitmap(b);
        }
    }).start();
}
```

source : developer.android.com

Exemple

Charger une image sur click

Première idée : création d'un thread.

```
public void onLoadImageClick(View v) {
    new Thread(new Runnable() {
        public void run() {
            Bitmap b = loadImageFromNetwork("http://example.com/image.png");
            mImageView.setImageBitmap(b);
        }
    }).start();
}
```

MAL !

Piège !

Les méthodes du Andoid UI toolkit ne sont

PAS thread safe !

Règles fondamentales

1 - Do NOT block the UI Thread

2 - Do NOT access the Android UI toolkit from outside the UI thread

Exemple

Charger une image sur click

Mauvaise idée

```
public void onLoadImageClick(View v) {
    new Thread(new Runnable() {
        public void run() {
            Bitmap b = loadImageFromNetwork("http://example.com/image.png");
            mImageView.setImageBitmap(b);
        }
    }).start();
}
```

Accès au thread UI depuis un autre thread $\rightarrow$ MAL.

Exemple

Charger une image sur click

Solution : runOnUiThread

```
public void onClick(View v) {
    new Thread(new Runnable() {
        public void run() {
            final Bitmap bitmap =
                    loadImageFromNetwork("http://example.com/image.png");

            runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    mImageView.setImageBitmap(bitmap);
                }
            });
        }
    }).start();
}
```

Handler

Lorsque `runOnUiThread` ne suffit pas

- Lorsque les traitements sont plus complexes
- Lorsque vous voulez un contrôle plus fin de l'exécution

Un Handler a deux fonctions principales :

- Appeler des méthodes dans un autre thread
- Lancer des actions après un certain délai

- Un Handler est lié au thread qui l'a créé
- Il peut être passé en paramètre

Handler : méthodes

Les principales méthodes d'un Handler :

- `post` : Ajoute une action à la file d'attente du thread associé au Handler
- `postDelayed` : comme `post` mais après un délai
- `sendMessage` : Ajoute un message dans la file des messages du thread associé au Handler

Pour traiter les messages

- nécessité de sous-classer `Handler` et de
- redéfinir `handleMessage`

Cas exotiques

Possibilité d'exécuter deux applications dans le même processus

À condition :

- d'avoir été signées par la même clef et
- d'avoir le même User ID Linux (*cf.* attribut sharedUserId dans le manifest)

Possibilité de préciser la répartition des éléments dans des processus spécifiques

Là encore dans le manifeste (*cf.* attribut android :process)

developer.android.com