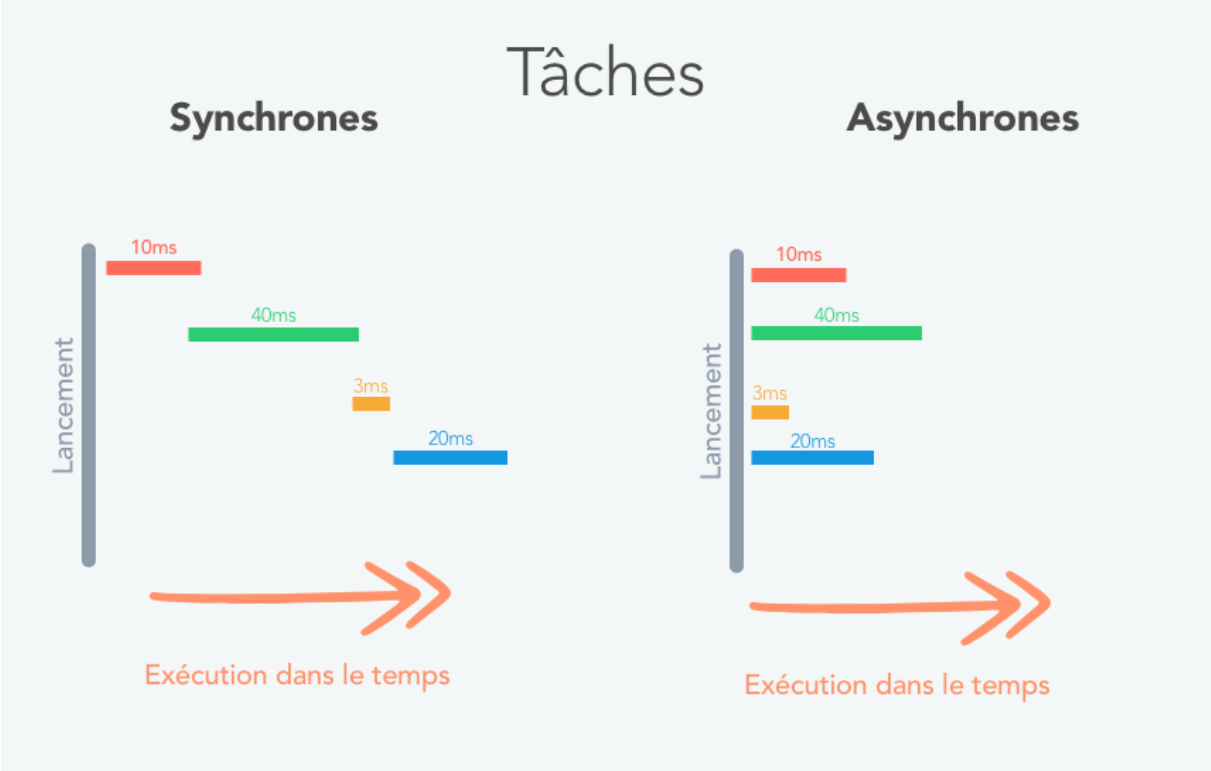
Thread Android

1. Introduction

Chaque application Android a un thread principal nommé UIThread (User Interface Thread) qui est en charge de : la gestion de l'interface utilisateur (y compris la mesure et le dessin des vues), la coordination des interactions des utilisateurs et la réception des événements du cycle de vie. S'il y a trop de travail sur ce fil, l'application semble se bloquer ou ralentir, conduisant à une utilisation indésirable. Tous les calculs et opérations de longue durée tels que le décodage d'une image bitmap, l'accès au disque ou l'exécution de requêtes réseau doivent être effectués sur un thread d'arrière-plan distinct. En général, tout ce qui prend plus de quelques millisecondes doit être délégué à un thread d'arrière-plan. Certaines de ces tâches peuvent être nécessaires pendant que l'utilisateur interagit activement avec l'application.

Chaque application a son propre thread spécial UIT qui exécute des objets d'interface utilisateur tels que les objets View. Seuls les objets exécutés sur le thread d'interface utilisateur ont accès à d'autres objets sur ce thread. Si vous lancer d'autres thread, elles n'ont pas accès aux objets d'interface utilisateur. Pour déplacer des données d'un thread d'arrière-plan vers le thread d'interface utilisateur, utilisez un gestionnaire Handler qui s'exécute sur le thread d'interface utilisateur.



2. Implémentation des Thread :

Il existe deux façons de créer un nouveau thread d'exécution :

1. Methode1

```
class PrimeThread extends Thread {
    long minPrime;
    PrimeThread(long minPrime) {
        this.minPrime = minPrime;
    }

    public void run() {
        // compute primes larger than minPrime
         . . .
    }
}
```

The following code would then create a thread and start it running:

```
PrimeThread p = new PrimeThread(143);
p.start();
```

2. Methode 2 :

```
class PrimeRun implements Runnable {
    long minPrime;
    PrimeRun(long minPrime) {
        this.minPrime = minPrime;
    }

    public void run() {
        // compute primes larger than minPrime
         . . .
    }
}
```

The following code would then create a thread and start it running:

```
PrimeRun p = new PrimeRun(143);
new Thread(p).start();
```

---

3. Communication avec l'UIThread

Problème :

La manipulation des ressources utilisées par l'UIT peut provoquer des comportements inattendus ou impossibles à prédire.

Solution1 :

La méthode d'activité `void runOnUiThread(Runnable action)` spécifie qu'une action doit s'exécuter dans le thread UI. Si le thread actuel est le thread UI, alors l'action est exécutée immédiatement. Sinon, l'action est ajoutée à la pile des évènements du thread UI

Solution2 :

Utiliser la classe handle qui offre une méthode `handleMessage(Message msg)` qui s'exécute dans le UIThread. Puis invoquer cette méthode dans votre second thread.

```
1  public class MonHandler extends Handler {
2    @Override
3    public void handleMessage(Message msg) {
4      // Faire quelque chose avec le message
5    }
6  }
7
```

Méthode invoquant la méthode `handleMessage` :

- `boolean post(Runnable r)` pour ajouter `r` à la queue des messages. Il s'exécutera sur le `Thread` auquel est rattaché le `Handler`. La méthode renvoie `true` si l'objet a bien été rajouté. De manière alternative,
  `boolean postAtTime(Runnable r, long uptimeMillis)` permet de lancer un `Runnable` au moment `longMillis` et
  `boolean postDelayed(Runnable r, long delayMillis)` permet d'ajouter un `Runnable` à lancer après un délai de `delayMillis`.
- `boolean sendEmptyMessage(int what)` permet d'envoyer un `Message` simple qui ne contient que la valeur `what`, qu'on peut utiliser comme un identifiant. On trouve aussi les méthodes
  `boolean sendEmptyMessageAtTime(int what, long uptimeMillis)` et
  `boolean sendEmptyMessageDelayed(int what, long delayMillis)`.
- Pour pousser un `Message` complet à la fin de la file des messages, utilisez `boolean sendMessage(Message msg)`. On trouve aussi
  `boolean sendMessageAtTime(Message msg, long uptimeMillis)` et
  `boolean sendMessageDelayed(Message msg, long delayMillis)`.

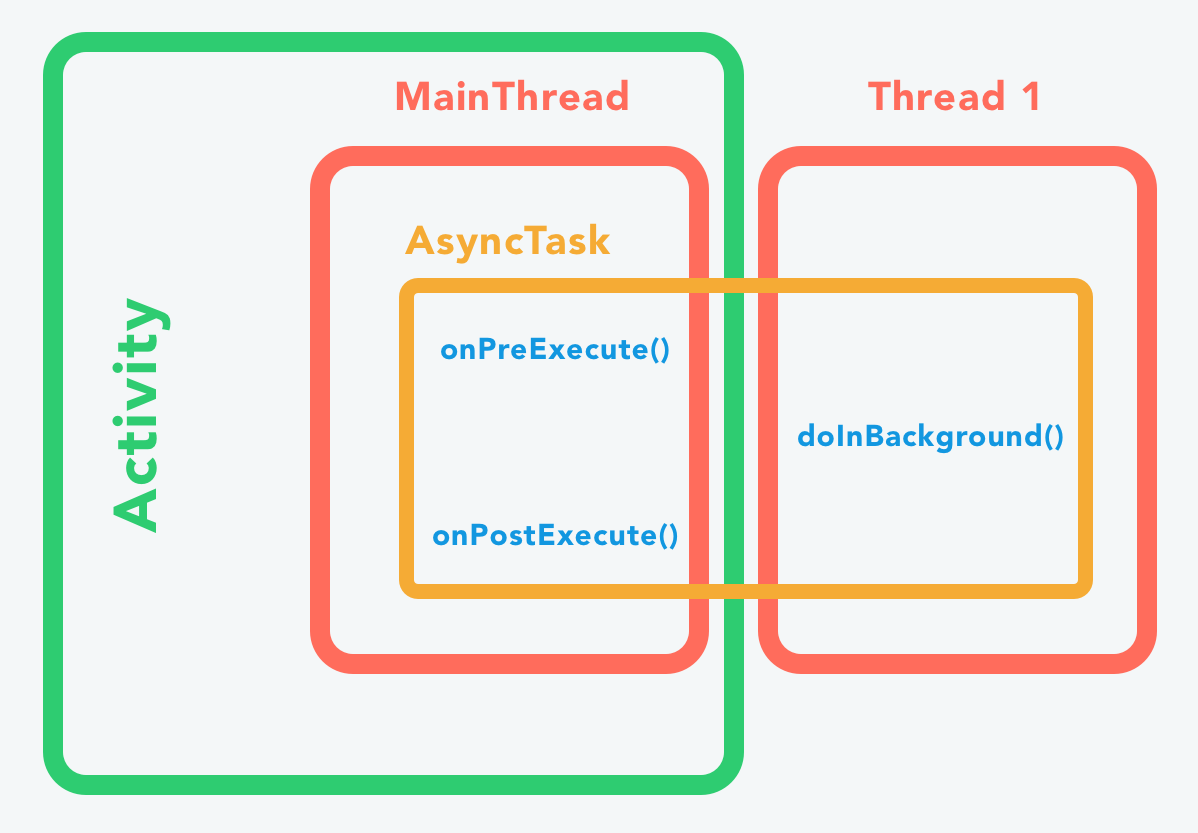
Exemple d'invocation :

```
class MonThread extends Thread
{
    MonHandler handler = new MonHandler();

    @override
    public void run()
    {
        Message msg = mHandler.obtainMessage();
        Bundle bundle = msg.getData();
        bundle.putInt(TestHandler.ACTION, action);
        bundle.putString(TestHandler.TEXT, text);
        msg.setData(bundle);
        mHandler.sendMessage(msg);
    }
}
```

4. Utiliser AsyncTask

AsyncTask est destiné à permettre une utilisation correcte et facile du thread d'interface utilisateur. Une tâche asynchrone est définie par un calcul qui s'exécute sur un thread d'arrière-plan et dont le résultat est publié sur le thread d'interface utilisateur. Une tâche asynchrone est définie par 3 types génériques appelés Params, Progress et Result, et 4 étapes, appelées onPreExecute, doInBackground, onProgressUpdate et onPostExecute.



- `onPreExecute()` : Cette méthode sera exécutée, sur **MainThread**, nous permettant ainsi par exemple, de mettre à jour notre UI en lançant une animation de chargement (ProgressBar).
- `doInBackground()` : Cette méthode sera en revanche exécutée dans un **Thread à part**, et vous vous en doutez, contiendra une **tâche longue** destinée à être lancée **en arrière-plan**.
- `onPostExecute()` : Cette méthode sera exécutée, sur **MainThread**, une fois que notre tâche longue soit terminée. On pourra ici arrêter l'animation de chargement par exemple (ProgressBar).

C’est dans la méthode doInBackGround où on fait notre traitement de second processus.

Exemple :

```
1.  public class AjoutTask extends AsyncTask<String,Integer,Boolean>
2.  {
3.      AlertDialog dialog;
4.      AlertDialog.Builder build;
5.      MainActivity con;
6.      public AjoutTask(MainActivity con) {
7.          this.con = con;
8.      }
9.
10.     @Override
11.     protected void onPreExecute() {
12.         // UIThread
13.         build=new AlertDialog.Builder(con);
14.         build.setTitle("Ajout");
15.         build.setMessage("Veuillez patientez...");
16.         dialog=build.create();
17.         dialog.show();
18.     }
19.     String IP="192.168.1.1";
20.     /**
21.      * avd: 10.0.2.2
22.      * reseaul local (pc +tel): IPv4
23.      * hebergement : www.nomdomaine.com
24.      */
25.     String url="http://"+IP+"/test_servicephp/ajout_profil.php";
26.
27.     @Override
28.     protected Boolean doInBackground(String... strings) {
29.         // recuperation des parametres du requetes
30.         String param1=strings[0];
31.         String param2=strings[1];
32.         String param3=strings[2];
33.         url+="?nom="+param1+"&prenom="+param2+"&pseudo="+param3;
34.         //Correspond à run dans la classe Thread
35.         //2eme thread: execution des fichiers php
36.         try {
37.             Thread.sleep(2000);
38.             JSONObject result=JSONParser.makeRequest(url);
39.             int s=result.getInt("success");
40.             if(s==1)
41.             {
42.                 /**  ajout avec success
43.                  *   MAJ de vos interfaces graphiques
44.                  *   par exmples si on met ici:
45.                  *   ednom_ajout.setText("");
46.                  *   ==> ça provoque une erreur et l'application se plante
47.                  *   le view ednom_ajout est bloqué par l'UIT
48.                  *   ==> Impossible accès
49.                  *   solution:
50.                  **/
51.                 publishProgress(1);
52.                 // Appel implicite à onprogressUpdate avec le parametre 1
53.                 return true;
54.             }
55.             else {
56.                 // ya pas de données
57.                 return false;
58.             }
59.         } catch (InterruptedException | JSONException e) {
60.             e.printStackTrace();
61.         }
62.         return false;
63.     }
64.     @Override
65.     protected void onProgressUpdate(Integer... values) {
66.         // UIThread
67.         int i=values[0];
68.         if(i==1) con.ednom_ajout.setText("");
69.     }
70.
71.     @Override
72.     protected void onPostExecute(Boolean aBoolean) {
73.         // UIThread
```

```
74.         if(aBoolean)
75.         {
76.             dialog.dismiss();
77.             // afficher vos données
78.         }
79.         else {
80.             dialog.setMessage("vs navez pas de resultat..");
81.             dialog.setIcon(android.R.drawable.ic_dialog_alert);
82.
83.         }
84.     }
85. }
```