# TME 8

**Objectifs pédagogiques :**

- *Executors*
- *Blocking Queue*

## 1 Web crawler

### 1.1 Introduction

Dans cet énoncé nous allons étudier le comportement d'un programme conçu pour crawler un site web et en extraire des pdf. Son objectif est de télécharger les pdf trouvés.

On suppose une url de départ "baseUrl" contenant le début du crawl et un dossier "outputFolder" où les pdf seront sauvegardés.

Voici le code actuel du main :

```java
package pc.crawler;
import java.nio.file.*;
import java.net.*;
import java.util.*;
import java.io.*;


public class WebCrawler {
   public static void main(String[] args) {
      // Hardcoded base URL to start crawling from
      String baseUrl = "https://www-licence.ufr-info-p6.jussieu.fr/lmd/licence/2023/ue/
          LU3IN001-2023oct/index.php";

      // Hardcoded output directory where downloaded pages will be saved
      Path outputDir = Paths.get("/tmp/crawler/");

      try {
         // Ensure the output directory exists; create it if it doesn't
         if (!Files.exists(outputDir)) {
            Files.createDirectories(outputDir);
            System.out.println("Created output directory: " + outputDir.toAbsolutePath());
         }

         // Process the initial URL (depth 0)
         System.out.println("Processing (Depth 0): " + baseUrl);
         List<String> extractedUrls = Collections.emptyList();
         try {
            extractedUrls = WebCrawlerUtils.processUrl(baseUrl, baseUrl, outputDir);
         } catch (URISyntaxException|IOException e) {
            System.err.println("Error during crawling: " + e.getMessage());
         }

         // Check if there are URLs extracted to process at depth 1
         if (extractedUrls.isEmpty()) {
            System.out.println("No URLs found to process at depth 1.");
         } else {
            // Process each extracted URL (depth 1)
            for (String url : extractedUrls) {
               System.out.println("Processing (Depth 1): " + url);
               try {
                  WebCrawlerUtils.processUrl(url, baseUrl, outputDir);
```

```
                } catch (URISyntaxException|IOException e) {
                    System.err.println("Error during crawling: " + e.getMessage());
                }
            }
        }

        System.out.println("Sequential crawling completed successfully.");

    } catch (IOException e) {
        // Handle exceptions that may occur during crawling
        System.err.println("Error during crawling: " + e.getMessage());
        e.printStackTrace();
    }
  }
}
```

Il utilise une classe de support `WebCrawlerUtils` pour identifier les liens et télécharger les pdf.

Actuellement il a plusieurs limitations :

- Il ne gère qu'un seul niveau de profondeur, i.e. il ramasse les pdf de la page de départ et ceux des pages référencées par la page de départ, mais c'est tout.
- Il ne gère pas les cycles de liens; en effet il est important de ne pas crawler plusieurs fois la même page.
- Il est séquentiel

L'objectif est de paralléliser le programme.

## 1.2 Parallélisation

Dans cet objectif :

- On va créer une file de paires (url, profondeur) à traiter. Cette file est partagée par plusieurs threads et bloquante si vide.
- Un thread donné doit : 1. piocher une paire dans la file, 2. scanner et télécharger les pdf dedans et extraire les liens de la page (fonctionnalité offerte pas Utils), 3. ajouter les liens à la file avec une profondeur diminuée de 1 sauf si on est déjà à profondeur 0.

On utilisera un pool de threads pour traiter les pages.

**Question 1.** Implanter WebCrawlerParallel en respectant les consignes ci-dessus.

**Question 2.** Ajouter une ConcurrentHashMap pour gérer les urls déjà visités.

## 1.3 Terminaison

**Question 3.** Implanter une classe ActivityMonitor; elle dispose de l'API suivante :

- Possède en attribut un compteur atomique (AtomicInteger)
- Constructeur sans argument, initialise le compteur à 0.
- `taskStarted()` incrémente le compteur
- `taskCompleted()` décrémente le compteur; notifie les threads en attente si le compteur atteint 0.
- `awaitCompletion()` bloque le thread appelant tant que le compteur n'est pas nul.

**Question 4.** Modifier WebCrawlerParallel pour utiliser ActivityMonitor :

- Créer un ActivityMonitor dans le main
- Appeler `taskStarted()` juste avant chaque insertion d'une url dans la file (y compris l'url de départ)

- Appeler `taskCompleted()` juste après avoir fini de traiter entièrement une url (y compris l'insertion des liens trouvés dans la file)
- Le main attend la fin de tous les threads avec `awaitCompletion()`
- Quand il détecte cette situation, il insère des "poison pills" dans la file pour débloquer les threads bloqués.
- Les threads workers doivent s'arrêter s'ils trouvent une "poison pill".

**Question 5.** Mesurer le temps de traitement pour le site fourni en exemple, en faisant varier la profondeur et les réglages choisis pour le ExecutorService.

## 2 Thumbnail

**Question 1.** Selon le temps disponible, proposer une version parallèle du programme de "thumbnail" qui suit la ligne d'implantation proposée en TD avec un pipeline.