Moteur de recherche

Maréchal Anthony et Benoît

5 juin 2005

Table des matières

Chapitre 1

Cahier des charges

1.1 Analyse du sujet

Dans le cadre de notre module "Systèmes d'exploitation" nous avions à mettre en place la création d'un moteur de recherche et d'indexation de pages HTML afin de mettre en évidence quelques principes des systèmes d'exploitation. Ainsi nous avons pu aborder les notions de :

- Processus (programmation multi-threadé)
- Hashtable (organisation des fichiers)
- Réseaux (communication par socket)

Voici le fonctionnement général du moteur de recherche que nous devions réaliser : tout d'abord notre moteur devait permettre la gestion d'une liste d'url qu'il devait alors distribuer aux robots analyseurs de pages web (ces robots peuvent être appelés des *crawlers*). Les robots, une fois leur url récupérée, vont l'analyser pour en extraire les mots clés ce qui permettra de construire un index. Ils doivent également trouver les liens qu'elle contient et les ajouter dans la liste d'url du moteur et ainsi recommencer leur travail jusqu'à ce qu'il n'y ait plus d'url dans la liste, ou que le temps qui leur était alloué soit écoulé ou encore lorsqu'ils auront atteint une certaine profondeur. Le serveur doit également écouter les connexions des clients. Ces clients sont en réalité tout les utilisateurs se connectant au serveur ; ils vont formuler une demande de recherche en fournissant une liste de mots-clés. Le résultat de la recherche dans l'index sera une liste d'url classée par pertinence d'après les mots entrés par l'utilisateur.

1.2 Objectifs de réalisations

Le moteur de recherche que nous devions réaliser devait comporter les éléments suivants :

- Un serveur : Celui-ci doit écouté les connexions des clients et des robots

- Un index : Il doit permettre d'associer une liste d'url à chaque mots clés
- Une applet JAVA stand alone : Qui permettra d'obtenir une liste d'url de l'index d'après une liste de mots clés
- Des robots de recherche : Fonctionnant sur n'importe quelle machine du réseau, ils analysent les pages web en extrayant les mots clés et les liens.

1.3 Objectifs supplémentaires

Les points suivants sont les fonctionnalités que nous avons décidé d'ajouter au moteur :

- Une interface cliente (applet Java)
- Une interface cliente utilisant la technologie Flash
- La sérialisation de l'index (possibilité de dépassement de la mémoire)
- Un système de points pour associer une importance aux mots clés indexés
- L'utilisation de moteur de recherche yahoo.fr pour l'initialisation de la file d'attente d'url
- Une série d'optimisations
  - Décomposition de l'index en 28 hashtables
  - Utilisation de regex pour la requête utilisateur

Chapitre 2

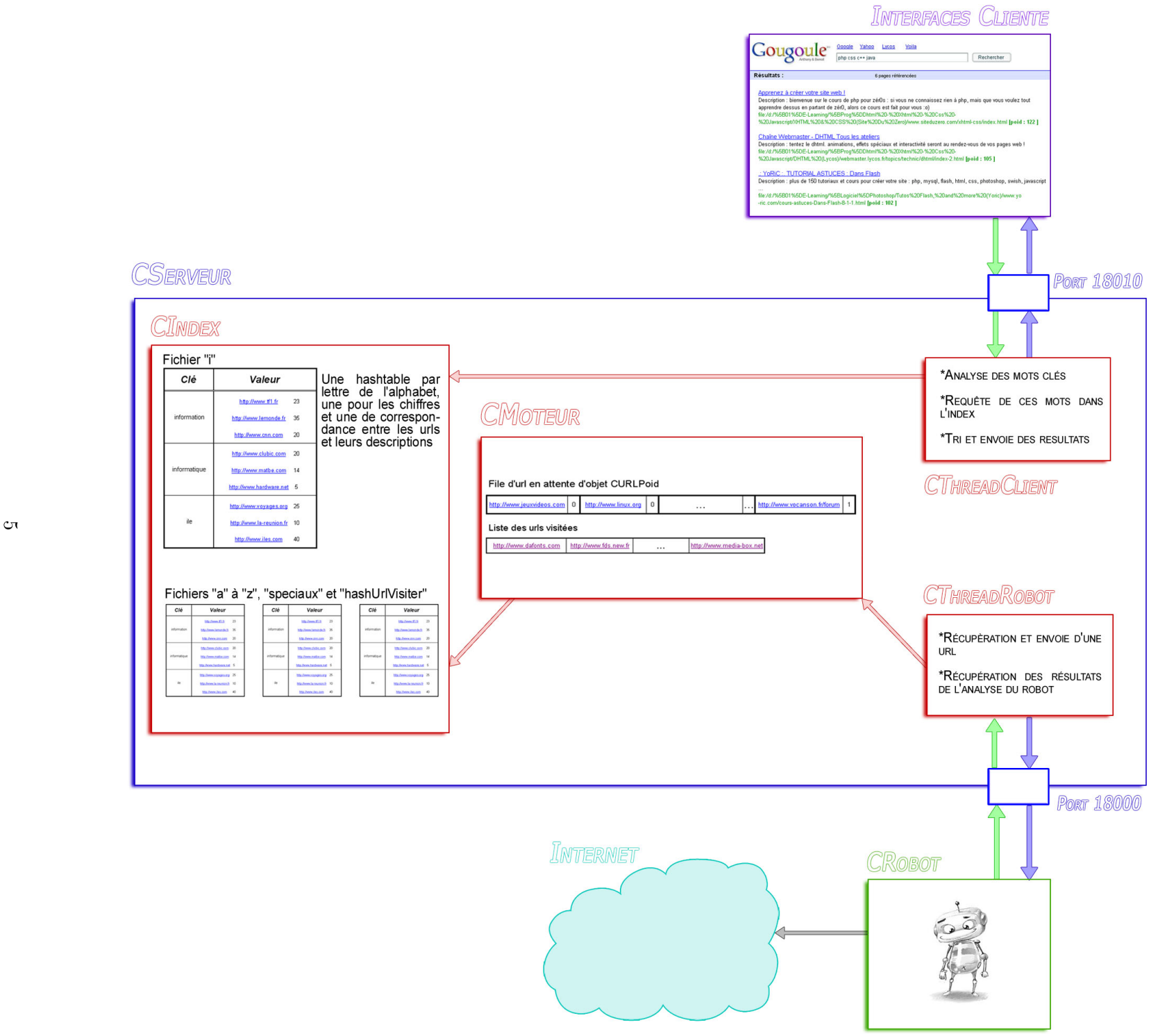
Analyse

2.1 Introduction

Nous détaillerons dans cette section le fonctionnement des différentes classes, leurs rôles et les interactions qu'elles peuvent avoir les unes envers les autres sans faire appel au code, puisque celui-ci sera expliqué dans le prochain chapitre. Pour l'implémentation des différents concepts et la réalisation de toutes les parties composant notre moteur de recherche nous avons choisi le langage JAVA.

2.2 Vue d'ensemble

Voici un schéma du fonctionnement de notre moteur, avec le nom des classes que nous avons utilisées pour repérer les différents éléments. On peut également voir les principales interactions entre ceux-ci. Nous utilisons une structure à base de référence comme l'énoncé le prévoyait. Notez qu'il ne figure pas sur ce schéma les threads d'écoute des ports permettant de créer autant de ***CThreadRobot*** ou ***CThreadClient*** que de clients ou robots se connectant.


INTERFACES CLIENTE
Gougoule
Google Yahoo Lycos Voila
php css c++ java
Rechercher
Résultats :
6 pages référencées
Apprenez à créer votre site web !
Description : bienvenue sur le cours de php pour zéros : si vous ne connaissez rien à php, mais que vous voulez tout apprendre dessus en partant de zéro, alors ce cours est fait pour vous :o)
file:/d:/%5B01%5DE-Learning/%5BProg%5DDhtml%2D-%20Html%20-%20Css%20-%20Javascript/9HTML%20&%20CSS%20(Site%20Du%20Zero)/www.siteduzero.com/xhtml-css/index.html [poid : 122]
Chaîne Webmaster - DHTML Tous les ateliers
Description : tentez le dhtml. animations, effets spéciaux et interactivité seront au rendez-vous de vos pages web !
file:/d:/%5B01%5DE-Learning/%5BProg%5DDhtml%2D-%20Html%20-%20Css%20-%20Javascript/DHTML%20(Lycos)/webmaster.lycos.fr/topics/technic/dhtml/index-2.html [poid : 105]
:: YoRiC :: TUTORIAL ASTUCES : Dans Flash
Description : plus de 150 tutoriaux et cours pour créer votre site : php, mysql, flash, html, css, photoshop, swish, javascript ...
file:/d:/%5B01%5DE-Learning/%5BLogiciel%5DPhotoshop/Tutos%20Flash,%20and%20more%20(Yoric)/www.yoric.com/cours-astuces-Dans-Flash-8-1-1.html [poid : 102]
PORT 18010
CSERVEUR
CINDEX
Fichier "i"
Clé
Valeur
information
http://www.tf1.fr 23
http://www.lemonde.fr 35
http://www.cnn.com 20
informatique
http://www.clubic.com 20
http://www.matbe.com 14
http://www.hardware.net 5
île
http://www.voyages.org 25
http://www.la-reunion.fr 10
http://www.iles.com 40
Une hashtable par lettre de l'alphabet, une pour les chiffres et une de correspondance entre les urls et leurs descriptions
Fichiers "a" à "z", "speciaux" et "hashUrlVisiter"
CMOTEUR
File d'url en attente d'objet CURLPoid
http://www.jeuxvideos.com 0
http://www.linux.org 0
...
http://www.vocanson.fr/forum 1
Liste des urls visitées
http://www.dafonts.com
http://www.fds_new.fr
...
http://www.media-box.net
*ANALYSE DES MOTS CLÉS
*REQUÊTE DE CES MOTS DANS L'INDEX
*TRI ET ENVOIE DES RESULTATS
CTHREADCLIENT
CTHREADROBOT
*RÉCUPÉRATION ET ENVOIE D'UNE URL
*RÉCUPÉRATION DES RÉSULTATS DE L'ANALYSE DU ROBOT
PORT 18000
INTERNET
CROBOT

2.3 Robots de recherche

2.3.1 Principe de la recherche

Les robots analyseurs de pages HTML (*crawlers*) ont, conformément à l'énoncer du sujet, été implémentés dans un premier temps sur une seule machine en mode multi-thread, puis modifiés afin qu'ils puissent fonctionner en parallèle sur plusieurs machines. Les robots sont donc maintenant des clients du moteur d'indexation : celui-ci leur fournit l'url dont ils ont besoin pour fonctionner.

Les robots vont donc commencer par faire la demande d'une url contenue dans la liste du moteur d'indexation au moyen d'une communication par socket. S'il n'y a plus d'url dans la file alors ils seront mis en attente jusqu'à ce que de nouvelles url soient ajoutées. Une fois cette url obtenue, ils vont commencer leur analyse des mots clés. Il s'agit de tous les mots qui sont contenus entre les balises HTML, qui comportent plus de deux lettres et qui ne font pas partie d'une liste de mots interdits (car trop redondants). Un système de points permet d'attribuer une importance aux mots clés dans la page. Ce système de points est le suivant :

- 20 points pour les mots présents dans le titre de la page
- 10 points pour les mots présents dans la balise *<meta name=keywords>*
- 5 points pour les mots présents entre les balises *<H1>*
- 1 point pour les autres mots

Lors de l'analyse de ces différents mots clés nous procédons à plusieurs modifications de ceux-ci pour améliorer l'homogénéité des mots indexés. Ils figurent ainsi en caractère minuscule, sans accents, et les différents codes HTML (caractères spéciaux) sont remplacés par leur correspondant. Tous les mots clés de la page se trouvent finalement dans un *Vector* contenant des objets ***CClePoid***, ceux-ci comprennent un mot clé et le poids qui lui est associé. Avant d'indexer cette liste de mots nous supprimons les doublons qu'elle contient (et ajoutons les points du doublons au mot d'origine afin d'obtenir un score adéquat, représentatif de la fréquence du mot et de son importance). A la fin de l'analyse d'une page nous possédons également une liste d'url (pouvant être soit absolues, soit relatives), qui a été extraite des balises *<a href=...>*. Si elles sont relatives nous reconstruisons l'url complète en nous servant de l'url de la page web en cours de traitement afin d'obtenir une adresse complète. Au final, le robot doit envoyer un *Vector* de ***CClePoid***, un tableau de liens, un *String* contenant le titre ainsi qu'un autre contenant la description de la page analysée.

2.3.2 Envoie des données

Au terme de l'analyse le robot envoi donc les résultats. Pour ce faire il va sérialiser les objets et les envoyer au ***CThreadRobot*** par socket. La classe ***CThreadRobot*** reçoit donc les différents objets envoyés par le ***CRobot***.

C'est aussi elle qui récupère une url dans la liste du moteur, vérifie si la profondeur associée à cette url est conforme et que le temps qui lui été alloué n'est pas écoulé et la retourne au ***CRobot*** pour qu'il (re)commence sont travail.

2.4 Index

2.4.1 Les différentes hashtables

L'index est un ensemble de fichiers dans lesquels des mots clés sont associés aux url analysées par les *crawlers*. C'est grâce à 27 hashtables (une pour chaque lettre de l'alphabet, puis une pour les nombres et mots commençant par des caractères spéciaux) que nous stockons ces données. Ces hashtables nous permettent en effet d'associer une clé (un *String*) à une valeur (Un *Objet*). Ici la clé est un mot trouvé dans une page et sa valeur est un (*Vector*) d'objets ***CURLPoid***. Cet objet comporte un champ contenant le nom de l'url dans laquelle le mot a été trouvé et un deuxième champ représentant le nombre de points que le mots a obtenus lors de l'analyse de cette url. Ainsi, dans les hashtables, les url peuvent être classées par ordre d'importance grâce au poids qui leur à été attribué.

Exemple : Pour le mot "maison" il pourrait y avoir une url "http ://www.hotes.fr" associée à un poids de 38 car l'analyse de cette page aurait montré que le mot "maison" figurait une fois dans le titre (+20 points) une fois dans la liste des keywords (+10 points) et huit fois dans la page (+8 points). Une autre url telle que "http ://www.meubles.com" serait associée à un poids de 2 car le mot "maison" n'aurait été vu que deux fois dans toute la page, etc.

La dernière hashtable de l'index nous sert à associer aux url un objet ***CTitreDesc*** qui contient le titre et la description de l'url. Cette hashtable pourrait sembler inutile car nous aurions pu ajouter à l'objet ***CURLPoid***, deux champs : titre et description. Nous aurions eu ainsi un accès direct à ces deux champs lorsque, pour un mot donné, nous récupérions l'url qui lui était associée et son poids. Le problème est que pour chaque mot trouvé à l'intérieur d'une page web nous aurions du ainsi créer un objet que nous aurions pu appeler ***CURLPoidTitreDesc*** contenant en plus ces deux champs. Pour chaque mot d'une page, le titre et la description de la page auraient donc été enregistrés et seraient contenus dans les objets ***CURLPoidTitreDesc*** ! Ceci nous aurait conduit à une augmentation du poids des fichiers de notre index inutilement. C'est pourquoi nous avons créé cette hashtable de correspondance entre une url et l'objet ***CTitreDesc***.

2.4.2 La sérialisation

Afin de pouvoir dépasser la capacité de la mémoire volatile nous sérialisons les hashtables de l'index. Les fichiers de sérialisations sont initialisés à vide s'ils n'existent pas, et lors de l'ajout d'une clé dans l'index nous désérialisons le fichier concerné (en effet, chaque nom de fichier sérialisé étant une lettre de l'alphabet, si on prend comme exemple le mot informatique, sa première lettre sera celle qui nous servira a déssérialisé le bon fichier, donc le fichier " i ") puis nous ajoutons la clé, créons un *Vector* d'objet ***CURLPoid*** et insérons le premier objet dans cette liste. Si la clé existe déjà nous nous contentons d'ajouter l'objet ***CURLPoid*** dans le *Vector* existant. Puis nous sérialisons à nouveau le fichier. Pour augmenter la rapidité d'indexation, qui nécessiterait autant de déssérialisation/sérialisation des fichiers que de mots clés trouvés dans la page, nous classons la liste de mots clés, déssérialisons le premier fichier, ajoutons tout les mots clés commençant par sa première lettre, puis refermons le fichier. Ceci nous fait donc passer de n déssérialisation/sérialisation à 27 au maximum (n étant égal au nombre de mots-clés trouvés dans la page). Cette amélioration a permis d'augmenter la vitesse d'indexation de manière très significative.

2.5 Requêtes utilisateurs

2.5.1 Interfaces

Les clients ont la possibilité de choisir parmi deux interfaces, l'une utilisant le format swf de ©Flash Macromédia et l'autre étant un applet JAVA.

L'interface en Flash : Elle a été conçue avec Flash MX dont le format d'exportation (swf) est libre. Tout d'abord l'utilisateur a la possibilité d'entrer un numéro de port et l'adresse du serveur. Le numéro de port correspond à celui choisit dans le serveur (il s'agit du 18000 normalement), et l'adresse du serveur est l'adresse IP de la machine sur laquelle le serveur est exécuté. Ensuite il doit simplement taper les mots clés qu'il souhaite et cliquer sur *Rechercher*. Cette action entraîne l'envoie de la chaîne de caractère qu'il a donné sous la forme d'une chaîne XML, c'est grâce à l'objet XMLConnector de l'action script 2 (langage de programmation de Flash) que l'envoi se déroule jusqu'au CThreadClient. Lors de la reception de la réponse du serveur plusieurs informations sont recus par le client. il apparait tout d'abord le nombre total de pages référencées par le serveur, ainsi que le nombre de page trouvées pour la requête du client. Enfin, chaque résultat comporte trois informations, le titre du site (sous forme de lien cliquable), sa description, son URL et son score.

L'applet Java : Plusieurs modifications par rapport à une applet Java "normale" on dû être éffectués afin de la rendre compatible avec le système de formatage des données imposé par Flash. Premièrement, pour la reception

des réponses du serveur, il a fallut parcourir la chaîne caractère par caractère jusqu'à ce que le caractère de fin de chaîne (" \ u0000 ") soit trouvé (Flash n'utilisant pas de méthode readLine() pour la lecture des données). Nous avons également dû créer une fonction capable de récupérer la valeur des attributs présents dans les balises XML. Enfin, les informations apparaissant lors de la réponse à une requête cliente, sont en tout point similaire aux informations présente sur l'interface Flash.

2.5.2 CThreadClient

Cette classe permet l'analyse de la chaîne de caractère donnée par l'utilisateur afin d'obtenir des mots en adéquation avec ceux enregistrés dans l'index. Pour ce faire, la classe ***CThreadClient*** va séparer les différents mots-clés de la chaîne et les mettre dans un tableau grâce à une regex. Ensuite les mêmes traitements que ceux appliqués par les mots clés indexés seront effectués à savoir : mise en minuscule et suppression des accents. Ainsi tous les mots traités auront le même "format" que ceux indexés. Nous possédons donc une liste de mots-clés compatible avec les clés indexées, nous allons donc récupérer les *Vector* associés aux clés de l'index. Nous créons pour cela une ***Arraylist*** qui contiendra les objets ***CURLPoid*** contenus pour chaque clé. Cette liste ainsi créée, nous possédons alors toutes les url qui peuvent potentiellement intéresser l'utilisateur. Pour garantir la pertinence des résultats nous supprimons les url en double, et modifions leur poids en conséquence. Notez qu'il ne peut pas y avoir deux liens identiques que si l'utilisateur a fourni au minimum deux mots (puisque le moteur possèdent une liste d'url déjà visitées empêchant ainsi toute redondance). Si l'utilisateur fournit trois mots clé et que deux de ces mots possèdent une url en commun alors ce site est sûrement bien plus intéressant qu'un mot clé pris à lui seul (même si son score est très grand). C'est pourquoi nous ajoutons 10^n (n est ici égale au nombre de liens identiques présents dans la liste), avant de faire la somme des points des deux url.

Exemple : L'utilisateur fournit la chaîne "belles maisons d'extérieur", les différents traitements que nous apportons à cette chaîne entraîne la recherche des mots "belles", "maisons", "exterieur" dans l'index. Imaginons maintenant que pour le mot "belles" il y ait les adresses "http ://www.belles-filles.com" avec un poids de 35 et "http ://www.immobilier.fr" associé à un poids de 3, pour le mot "maisons" il y ait les adresses "http ://www.cabane.fr" et un poids de 11 et "http ://www.immobiliers.fr" associés à un poids de 12, et enfin qu'il n'y ait aucun résultats pour le mot "exterieur". Les résultats de la recherche une fois les urls classées par score seront donc belles-filles.com (35 points), puis immobiliers.fr (15 points, car 3+12) et enfin cabane.fr (11 points). On constate que malgré l'addition des points de l'adresse en double (immobilier.fr), le premier résultat n'est sûrement pas ce que l'utilisateur

attendait. Ainsi, avec notre système, les points attribués à l'adresse immobilier.fr serait de 100 (car 10 exposant 2) + 15 (nb de points cumulés des deux adresses). Ce système permet donc de ranger par "classes" les url selon le nombre de mots clés commun à une page, s'il y a deux mots identiques le nombre de points minimum est de 100 et s'il y en a trois : 1000 etc.

Ce système est celui qui nous a permis d'avoir des résultats les plus pertinent selon la chaîne donné et c'est pourquoi il a été retenu. Nous recherchons également dans la hashtable des url associés à leur titre/description, le titre et la description de chaque url récupérée. Maintenant il nous faut envoyer les résultats à l'interface cliente. Ceci se passe sur le port 18000 grâce au socket et à l'envoi d'un *String* écrit dans un format XML. Voici à quoi ressemble cette chaîne : <reponse value="URL" titre="TITRE" poid="POID" description="DESCRIPTION" />. Le client Flash extrait les informations automatiquement grâce à l'objet *XMLConnector*, l'applet JAVA récupérera ces informations grâce à une méthode déjà créées dans la classe ***CRobot***.

2.6 Serveur

La classe ***CServeur*** est la classe principale car elle possède la fonction *main()*. Elle va créer les éléments nécessaires au fonctionnement du programme. Elle va donc commencer par créer l'index (classe ***CIndex***) puis le moteur d'indexation (***CMoteur***), nous ajoutons ensuite les liens d'amorce dans la file d'url que possède la classe ***CMoteur***, mais nous avons la possibilité d'initialisé cette file grâce à une classe d'amorce qui à partir d'un mot clé va récupéré les résultats du moteur de recherche *Yahoo*. Et enfin nous créons deux threads d'écoute : l'un pour l'écoute du port permettant les communications avec la classe ***CRobot*** et l'autre pour l'écoute des connexions des l'interfaces clientes.

Les threads d'écoute nous permettent de surveiller deux ports simultanément. Ces threads en attente d'acceptation de connexions, une fois celle-ci établit, ils vont créer les threads de traitements correspondant. Ainsi lorsque le thread d'écoute du robot (classe ***CThreadEcouteRobot***), recevra une requête de connexion il créera une nouvelle instance de la classe ***CThreadRobot*** et ainsi de suite pour chaque connexion. Il en sera de même pour le thread à l'écoute des connexions d'interfaces clientes (classe ***CThreadEcouteClient*** qui créera une classe ***CThreadClient***).

La classe ***CMoteur*** quand à elle, gère la file d'attente d'url. Lorsque que la classe ***CRobot*** a fini son travail la classe ***CThreadRobot*** reçois les résultats de l'analyse puis fait la demande d'une url (si le temps alloué au robot le permet) à la classe ***CMoteur***, celle-ci va donc prendre le premier objet ***CURLProfondeur*** de se file d'attente et le transmettre au ***CThreadRobot*** qui va vérifié si la profondeur de l'url est atteinte ou non avant d'envoyé

l'url au ***CRobot***. Le moteur possède également une *Arraylist* d'url déjà visité afin d'évité toute analyse de pages qui ont déjà été indéxés et enfin, le moteur peut mettre en attente les ***CThreadRobot*** qui font la demande d'une url alors que la liste est vide (ils seront notifiés lorsque de nouvelles url seront ajoutées).

Chapitre 3

Programmation

3.1 Introduction

Nous allons maintenant voir comment s'est déroulée l'implémentation des différents concepts et des différentes classes en générale. L'énoncé du sujet nous imposait de justifier l'emploi des membres static, nous vous informons que mise a part les fonctions *main()*, aucun membre de ce type n'as été utilisé. Dans un premier temps nous verrons les principales méthodes permettant aux robots d'analyser les pages web, puis nous regarderons comment nous indexons et sauvegardons les informations de la classe ***CIndex***. Nous verrons ensuite les moyens de communications que l'on a du mettre en place, les aspects de la programmation multi-threadé et enfin nous détaillerons séquentiellement ce qui se passe dans la classe ***CServeur***.

3.2 Robots de recherche - *CRobot*

Voici la liste des principales méthodes de la classe ***CRobot*** qui nous permettent de traiter les mots-clés des pages web, d'extraire les liens et d'envoyer ces résultats à la classe ***CThreadRobot***.

Méthodes :

- *void afficherInfoURL()* : Affiche les infos relatives à l'url
- *void chercherURLCle(String)* : Méthode de traitement de la page web afin de trouver les mots-clés, leur attribuer un poids, et trouver le lien de la page
- *void enregistrerURL()* : Stocke la page web en mémoire et lance la méthode pour effectuer les traitements sur cette chaîne
- *void etatFinal()* : Quand on a finit les traitements sur la page et donc qu'on dispose d'un tableau de liens et d'une liste de mots-clés traités alors nous sérialisons et envoyés ces objets.
- *void main(String[])* : Fonction principale

- *boolean motInterdit(java.lang.String mot)* : Méthode permettant de ne pas indexer les mots trop récurrents de plus de deux lettres
- *void recupererURL()* : Récupération d'une url dans la file d'attente du serveur
- *String recupererValeurAttribut(String , String)* : Récupération de la valeur d'un attribut d'une balise HTML
- *String remplacementAccentsParHTML(String)* : Méthode permettant de remplacer les caractères accentués par leur code HTML
- *String suppressionAccents(String)* : suppression des accents et caractères non-mots
- *String suppressionCaracteresSpeciaux(String)* : Méthode traitant les caractères spéciaux à l'intérieur des chaînes du titre et/ou de la description
- *String suppressionEspaces(String)* : Suppression des espaces à l'intérieur des mots
- *void traiterCle()* : Traitements des mots clés trouvés dans la page : mise en minuscule, suppression des doublons et suppression des mots de moins de 3 lettres
- *void traiterKeywords(String)* : Méthode utilisant les StringTokenizer pour extraire les mots du "content" de la balise
- *void traiterLien(String)* : Traitement des liens trouver dans la page, vérifie leur validité et reconstruit les liens relatifs
- *void trierListeClePoid()* : Tri par ordre alphabétique de la liste de clé/poids (pour un ajout plus rapide dans l'index)

Détaillons les différentes instructions présentes dans le constructeur de cette classe :

```
 public CRobot(Socket sock) throws IOException,UnknownHostException,
ClassNotFoundException {
// initialisation de la variable de classe _socket
        _socket = sock ;
       try {
// connexion du flux d'entrée au socket
        _ois = new ObjectInputStream( _socket.getInputStream() );
// connexion du flux de sortie au socket
        _oos = new ObjectOutputStream( _socket.getOutputStream() );
       }
       catch(Exception e) { System.out.println("CRobot> EXCEPTION : "+e); }
        while(true) {
            _desc ="";
            _titre = "";
// nombre de liens dans le tableau _liens[]
            _nbLiens=0;
// tableau contenant tout les liens trouvés dans la page
```

```
            _liens = new String[250];
// ArrayList contenant tout les objets CClePoid
            _listeClePoid = new Vector();
// méthode qui va récupérer une url dans la file d'attente du serveur
            recupererURL();
// méthode d'information sur l'url
            afficherInfoURL();
// Mise en mémoire de la page web
            enregistrerURL();
// traite les mots clés trouvés dans la page
            traiterCle();
// envoie toutes les données trouvées au serveur.
            etatFinal();
        }
// fermeture des flux
        _oos.close();
        _ois.close();
        _socket.close();
    }
```

Et voici un extrait de code de la fonction *etatFinal()* permettant l'envoie des données par socket :

```
public void etatFinal()  throws IOException , UnknownHostException{
  /* Affichage */
  System.out.println("CRobot> 5/8 Nombre de clés à envoyer au
  Serveur ajouter : "+_listeClePoid.size());
  System.out.println("CRobot> 6/8 Nombre de Liens à envoyer au
  Serveur : "+_nbLiens);
  System.out.println("CRobot> 7/8 Envoie des objets...");

  // le nom de l'url en cours
  String url = _url.toExternalForm();

  // tri alphabétique des clés
   trierListeClePoid();
  /* Envoie des objets */
  // envoie de la liste de Cle/Poid
  _oos.writeObject(_listeClePoid);
  // envoie du tableau de liens
  _oos.writeObject(_liens);
  // envoie du nom de l'url que l'on vient d'analyser
  _oos.writeObject(url);
  // envoie du titre de la page
  _oos.writeObject(_titre);
```

```
  // envoie de la description de la page
  _oos.writeObject(_desc);
  // envoie le nb de liens contenu dans le tableau
  _oos.writeObject(new Integer(_nbLiens));
  // envoie de la profondeur de l'url analysée
  _oos.writeObject(new Integer(_prof));
  System.out.println("CRobot> 8/8 Envoie reussit.");
}
```

3.3 Index - *CIndex*

3.3.1 Principe du Hash-coding en JAVA

Le hash-coding permet de réorganiser les fichiers grâce à une fonction de *hachage*. Cette fonction associe à chaque valeur d'une clé un numéro de paquet. La fonction de hachage peut par exemple utiliser les lettres de la clé pour obtenir ce numéro. Ainsi le hash-coding permet d'accéder à n'importe quelle clé grâce au calcul de la fonction de hachage puis un petit accès séquentiel. Les tables de hachage (***Hashtable***) en JAVA font correspondre des objets clés à des objets valeur, et grâce à la structure de données utilisée cette classe permet une recherche efficace de la valeur asociée à une clé. Nous avons utilisé la classe ***Hashtable*** plutôt que***HashMap*** car elle à l'avantage d'être synchronisée. Voici comment nous créons nos Hashtable en JAVA :

```
// constructeur
Hashtable H = new Hashtable();
// Ajout d'une clé et sa valeur
H.put(clé,valeur);
// Retourne la valeur (objet \Classe{CURLPoid}) d'une clé
H.get(clé);
```

3.3.2 Principe de la sérialisation

La sérialisation nous permet de sauvegarder l'état d'un objet à un moment donné en l'écrivant sur le disque dur de l'ordinateur. Dans l'index cette sérialisation nous permet de dépasser la capacité de la mémoire volatile, grâce à la sérialisation/déssérialisation de nos 28 ***Hashtables***. Voici le code nous permettant de sérialisé un objet :

```
Object o; // objet à sérialisé
FileOutputStream fos; // fichier à écrire
try {
// définition du nom de fichier à écrire
fos = new FileOutputStream("nom_fichier");
// connexion d'un flux d'envoi d'objet au fichier de sortie
```

```
ObjectOutputStream oos = new ObjectOutputStream(fos);
// envoi de l'objet
    oos.writeObject(o);
// fermeture du flux de sortie
    oos.close();
}
catch(Exception e) {};
```

3.3.3 Implémentation

Voyons maintenant, comment nous avons implémenté ces deux concepts pour la gestion de l'index. Tout d'abord voici la liste des méthodes de l'index :

- *void ajouter(String, Vector)* : Ajoute une clé avec sa valeur (***Vector*** d'objet CClePoid) dans la hashtable
- *void ajouterUrl(String, CTitreDesc)* : Ajoute une clé (String correspondant à l'url) et sa valeur (titre et description de l'url) dans la Hashtable
- *Vector rechercher(String)* : Recherche la valeur correspondant à la clé donnée en paramètre
- *CTitreDesc rechercherUrl(String)* : Recherche la valeur à partir de plusieurs clé

Et voici l'extrait de code correspondant à la méthode *ajouterURL* :

```
public synchronized void ajouterUrl(String cle,CTitreDesc valeur) {
Hashtable hashDes=new Hashtable();

/** On désérialise(chargement)la Hashtable en restaurant le fichier
correspondant. Celui-ci ira donc en mémoire **/
try {
FileInputStream fis;
fis = new FileInputStream("hashUrl");
ObjectInputStream ois = new ObjectInputStream(fis);
hashDes = (Hashtable) ois.readObject();
hashDes.put(cle,valeur);
ois.close();
}
catch(Exception e) {
System.out.println("CIndex> EXCEPTION, pendant la désérialisation);
System.exit(0);
}

/** (Re) Sérialisation ***/
try {
FileOutputStream fos = new FileOutputStream("hashUrl");
```

```
ObjectOutputStream oos = new ObjectOutputStream(fos);
oos.writeObject(hashDes);
oos.flush();
oos.close();
}
catch(Exception e) {
System.out.println("CIndex> EXCEPTION, pendant la serialisation);
System.exit(0);}
}
```

Cette méthode (comme toute celle de la classe) va en effet désérialisé un fichier contenant une hashtable puis ajouter l'objet dans la *valeur* associé a la clé. Ici il s'agit d'un objet ***CTitreDesc*** et la clé est un *String* contenant une url. Une fois l'ajout dans la hashtable effectués on referme le fichier.

3.4 Serveur - *CServeur*

La classe CServeur est la classe principale du moteur de recherche. Elle va créer un objet ***CIndex*** puis un objet ***CMoteur***, nous ajoutons ensuite les liens d'amorce dans la file du ***CMoteur*** ou en utilisant un mots d'amorce pour initialisé la file avec les résultats de *Yahoo*, et enfin nous créons et lançons deux threads d'écoute de port.

Voici le code de la classe CServeur :

```
public class CServeur {
public static void main(String[] args) {
/* 1 - Initialisation des variables */
// numéro du port sur lequel se déroulerons les communications
//de l'applet
int portApplet=18002;
// numéro du port sur lequel se déroulerons les communications
//de la classe CRobot
int portRobot=18012;
// valeur de la profondeur d'arrêt
int profArret=0;
// temps attribuer aux robots en secondes
int temps=30;

System.out.println("CServeur> Donnez le(s) mot(s) d'amorce
(tapez 0 pour passe cette etape): ");
String amorce=Lire.S();

// Création de l'index ( CIndex )
CIndex index   = new CIndex() ;
```

```
// Création du moteur d'indexation avec l'index en référence
CMoteur Moteur = new CMoteur(index);

/* Ajout d'url dans la file d'attente de la classe CMoteur */
// si l'utilisateur veut amorcer la file d'attente avec yahoo.fr
if(!amorce.equals("0")) {
System.out.println("CServeur> Lancement de l'amorcage de la
file d'attente...\n");
new CThreadAmorce(amorce,Moteur);
}
else {
URL nvURL ;
try {
nvURL = new URL("http://www.siteduzero.com/php/index.html");
Moteur.ajouterURL(new CURLProfondeur(nvURL,0));

nvURL = new URL("http://www.u-bourgogne.fr");
Moteur.ajouterURL(new CURLProfondeur(nvURL,0));

} catch( MalformedURLException e) {
System.out.println("CServeur> EXCEPTION : "+e);
}
}

/* Création et lancement des threads d'écoute sur le port choisit */
CThreadEcouteRobot  EcouteRobot  =
new CThreadEcouteRobot(Moteur, portRobot,profArret,temps);
CThreadEcouteClient EcouteClient =
new CThreadEcouteClient(index, Moteur, portApplet);

// Lancement de l'écoute de connexion des Robots
EcouteRobot.start();
// Lancement de l'écoute de connexion des interfaces clientes
EcouteClient.start();
}
}
```

Et voici le code de la classe ***CThreadEcouteClient*** :

```
/* Importation des packages */
import java.net.*;
public class CThreadEcouteClient extends Thread {
    // référence vers l'index
CIndex  _index;
```

```
// référence vers le moteur pour connaitre le nb de liens visiter
    CMoteur _moteur;
// port à écouter
    int     _port;

    /* Constructeur */
    public CThreadEcouteClient(CIndex index,CMoteur moteur,int port) {
        this._index = index;
        this._port = port;
        this._moteur = moteur;
    }

    /* Lancement du Thread */
    public void run() {
        /* Ecoute du port de communication et
        gestion du port de connexion. */
          try  {
            ServerSocket serversocket = new ServerSocket(_port) ;
            while(true)
                new CThreadClient(serversocket.accept(),_index,
_moteur.getNbLiensVisiter());
        }
        catch(Exception e) {}
    }
}
}
```

3.5 Communications

3.5.1 Les sockets et les flux

La communciation réseaux en JAVA passe par l'utilisation d'une séries de classe du package *java.net*. Nous nous servons des classe ***ObjectOutputStream***, ***getOutputStream*** et ***PrintWriter*** ainsi que de la méthode *writeObject()*, pour l'envoie d'un flux ; et des classes ***BufferedReader***, ***InputStreamReader***, ***getInputStream*** ainsi que de la méthode *readObject()* pour la reception d'un flux.

Des communications réseaux ont été nécessaire entre les classes :

- CThreadEcouteClient - Interfaces cliente
- CThreadClient - Interfaces cliente
- CThreadEcouteRobot - CRobot
- CThreadRobot - CRobot

Toute ces communications utilisent les sockets et les classes énumérés plus haut. Elles reposent également toutes sur le même schéma et nous avons

déjà vu un extrait du code de la classe ***CRobot*** et comment celui-ci envoyait les données, c'est pourquoi nous vous donnons ici qu'un seul exemple, celui de la classe ***CThreadClient*** et des interfaces clientes.

3.5.2 Le *CThreadClient* et les interfaces cliente

Comme il a été vu dans la partie analyse, cette classe va récupéré et traité les mots clés envoyé par les interfaces clientes puis leur renvoyé un ***Vector*** d'url en résultat.

Méthodes :

- *String donnerValeurAttribut(String, String)* : Méthode qui renvoie la valeur d'un attribut d'une balise (XML ou HTML).
- *void envoyerResultats()* : Envoie des liens trouvés dans l'index selon les mots clé donnés par l'utilisateur.
- *void rechercherLiens(String)* : Méthode créant une ArrayList d'objet CURLPoid correspondant aux mots clés donnés par l'utilisateur puis traité par la regex
- *ArrayList regexMotsUtilisateur(String)* : Analyse des mots tapés par l'utilisateur pour en extraire de nouveau afin de mieux exploité l'index : délimitation des mots grâce à la regex, mise en minuscule et remplacement des caractères à accents par des caractères sans accents.
- *void run()* : Lancement du Thread : lancement effectué par le Client
- *void triBulle(int)* : Trie à bulle sur le tableau de liens constitué par la méthode *trierListeLiens()*
- *void trierListeLiens()* : Supprime les liens doublons de l'*Arraylist* généré par la méthode *rechercherLiens* et modifie le poid des urls en conséquence puis lance le tri a bulle sur le tableau qu'elle a créé.

Voici un extrait de code de la méthode *regexMotsUtilisateur()* :

```
import java.util.regex.*; //package nécessaire à l'utilisation des regex

// ...
// tout les caractères entrés dans la regex correspondent aux caractères
// "non-délimitateur de mot"
Pattern pat= Pattern.compile("[^a-zA-Z_0-9çñéèêöõôïîäâà'-]");
// mise en minuscule des chaines dans le tableau liste
String [] liste = pat.split(motsUtilisateur.toLowerCase());

// nous disposons maintenant dans liste, d'un tableau de mots issu du
// String envoyer par l'interface cliente
```

Comme énoncé dans la partie analyse, Flash n'utilise pas de méthodes évoluée pour l'envoi de chaines de caractères ainsi une méthode readLine() dans la classe ***CThreadClient*** serait inopérante. Nous nous sommes donc

résolut à utiliser la méthode *read* de la classe ***BufferedReader*** parcourant la chaîne caractère par caractère.

Extrait de code de la réception des données de la classe ***CThreadClient*** envoyé par les interfaces clientes (Flash ou applet) :

```
/* Dans le constructeur de CThreadClient */
 try {
 // connexion du flux de sortie vers le client
_out = new PrintWriter(_socket.getOutputStream());
// connexion du flux d'entré des requêtes du client
_in =new BufferedReader(new InputStreamReader(_socket.getInputStream()));
}
catch (IOException e) {}

...

/* Dans la méthode run() de CThreadClient */
char charCur[] = new char[1] ;

/* on traite la requête cliente caractère par caractère */
while(_in.read(charCur,0,1)!= -1) {
 // tant qu'on est pas en fin de chaîne
if ( (charCur[0] != '\u0000') && (charCur[0] != '\n')) {
// ...on concatène le  caractère à la chaîne déjà créé
_motsUtilisateur+=charCur[0] ;
}
else {
/* traitement de la chaîne (requête cliente) */
}
}
```

Envoi des données depuis l'applet Java : Pour simuler l'envoi d'une requete XML venant de Flash nous avons dû formater la requête cliente de la même manière et utiliser la méthode print d'un objet *PrintWriter* de bas niveau, car prenant un objet de type *OutputStream* en paramètre de son constructeur. L'envoi s'effectue alors avec la méthode *print()* suivit de la méthode *flush()* :

```
 // Lors du clique sur le bouton "Connexion" de l'applet java
_out = new PrintWriter(_socket.getOutputStream() );
//...
// Dans la méthode envoyerEtRecevoir de l'applet java
String contenu = jTextField1.getText() ;
```

```
// ecriture dans le flux du contenu
_out.print("<requete value=CONTENU />\u0000") ;
// envoie du flux vers le serveur
_out.flush();
```

La programmation ActionScript (langage de programmation de Flash), dépassant le cadre de ce rapport nous n'expliquerons pas ici les divers mécanismes de la communication réseau avec ce langage. Vous pourrez cependant trouver le code source commenté en Annexe.

3.6 Multi-threading

Notre moteur de recherche est donc une application multi-threadé, en effet nous avons besoins de créer plusieurs processus afin de pouvoir écouter plusieurs ports (classes ***CThreadEcouteRobot*** et ***CThreadEcouteClient***), ou géré les réceptions des objets envoyés par chaque robot ***CThreadRobot***, ou de pouvoir traité les requêtes de plusieurs interfaces cliente(classe ***CThreadClient***), ou encore pour chronométré chaque robot (classe ***CChrono***).

A titre d'exemples nous montrerons seulement quelques interactions entre deux classes gérant les threads.

Le premier exemple est enclenché lorsque le ***CThreadRobot*** tente d'obtenir une url et que la file d'url dans la classe ***CMoteur*** est vide :

```
/*Méthode avoirURL de la classe CMoteur*/
public synchronized CURLProfondeur avoirURL() {
CURLProfondeur up ;
int i=0;
boolean donner=false;
// s'il n'y a pas d'url dans la file...
if (_laFile.size()==0) {
try {
// ...on fait attendre les threads
wait();
}
catch(InterruptedException e) {
System.out.println("CMoteur> EXCEPTION, le serveur a donnee
une url alors que la file d'attente est vide : "+e);
}
}
else{ // on donne l'url }
}
```

Et lorsque de nouvelles url sont ajouté dans la file on les notify() :

```
 /** Ajout d'une nouvelle URL dans la file  **/
public synchronized void ajouterURL(CURLProfondeur up) {
// ...
// si avant l'ajout il n'y avait pas d'url dans la file
// on débloque les threads en attente d'une url
if(_laFile.size()==1)
notify();
}
```

3.7 Problèmes rencontrés

Lors de l'élaboration du projet nous avons rencontré quelques problèmes. Tout d'abord dans l'établissement du rôle et des liens des différents composants, et ce malgré l'aide apportée par le schéma présent dans l'énoncé du sujet. Il nous a donc fallut quelques semaines avant de pouvoir structurer nos propre schémas, établir les liens entre les classes que nous devront créer et se répartir les tâches. Une fois cette phase terminée nous avons logiquement commencé par le robot analyseur de page HTML et nous nous sommes rapidement rendu compte que cette classe serait finalement plus complexe que prévu, dû fait que les pages HTML présentes sur le web ne respectent pas toujours les standards du *W3C*. De plus, nos premiers algorithmes ont rapidement démontrés que les mots clés trouvés dans la page étaient rarement de bonne qualité. D'où la série de traitements que nous avons décidé d'apporter à ceux-ci. Il en est allé de même avec les liens que nous devions récupérer dans les pages : il a fallut effectuée plusieurs séries de test pour comprendre pourquoi certains ne voulaient pas être "aspirés" et comment l'on pouvait faire pour reconstruire correctement les liens relatifs. Une fois cette classe achevé, nous avons mis en place le moteur avec les différents concepts que nous avions appris. Cette mise en place ne s'est pas faite directement, il a été necessaire de retranscrire en JAVA et dans le cadre de notre moteur de recherche ces différents concepts, ce qui n'a pas été très naturel. Malgré tout, le langage JAVA possède dans sa librairie les principaux éléments constituant ces concepts, et nous avons donc réussi à implémenter la notion de processus et la gestion de la communication de manière plutôt agréable ; seule l'indexation s'est révélée plus laborieuse, notamment la sérialisation. En effet nous n'avions pas de livre de référence du langage JAVA et nous nous sommes rapidement rendu compte de sa nécessité, afin de savoir par exemple si les objets de la librairie dont nous nous servions était sérialisables, et/ou synchronisés. Le dernier point concerne la transformation de nos robots pour qu'ils puissent fonctionner sur un réseau. Nous ne nous attendions pas à devoir changer autant nos classes et devoir repenser le projet comme cela ce qui nous a pris plus de temps que nous l'imaginions.

Conclusion

Ce projet nous a fait découvrir de nombreux concepts : gestions de fichiers, réseaux, swaping, et la notion de processus entre autres. Ces différents concepts ont été implémentés grâce au langage JAVA et celui-ci nous apparaît maintenant tout indiqué pour une programmation orienté réseaux (comme sa réputation le laissait présager). Malgré nos difficultés au début du projet sur le fonctionnement interne du moteur de recherche dû à notre relative récente découverte du langage JAVA, nous avons finalement réussi à manipuler la notion de programmation multi-threadé, de communication par socket et de sérialisation. Ce projet s'est ainsi révélé être particulièrement intéressant, notre plus gros problème a finalement été le temps nécessaire pour créer un moteur de recherche assez intelligent pour qu'il ne soit pas une simple démonstration des concepts des systèmes d'exploitation et pour qu'il puisse être réellement utilisable. Malgré tout il ne s'agit pas encore d'un assez bon concurrent d'un moteur comme Google ou Yahoo, c'est pourquoi nous eu finalement recours à une classe d'amorçage des liens. Notre moteur possède désormais toutes les fonctionnalités que nous étions en mesure de concevoir. Des tests sur notre serveur et celui de la Maison des Etudiants ont montré que notre projet fonctionnait parfaitement. Seul des problèmes de lenteur et déconnection ont été rencontré sur le réseau des machines des salles 104-105.

Le projet a été réalisé avec *netBeans 3.5.1*, *Flash MX* (pour l'interface cliente), *Photoshop CS* (schéma du chapitre 2 et images du mot Gougoule), *Cinéma4D* (image 3D des boules de couleurs) et enfin LaTeXpour la rédaction de ce rapport.