# RAPPORT DE PROJET RESEAUX ET SYSTEMES


# REMERCIEMENTS

Avant tout développement du travail effectué pour la réalisation du projet, nous tenions à remercier notre professeur de Réseaux et Systèmes, Lucas Nussbaum, qui nous a enseigné les bases nécéssaires pour la réalisation de ce projet et qui a répondu par mail à de nombreuses reprises à nos intérrogations.

Nos remerciements s'adressent également à Damien Picard pour son aide dans la recherche d'une erreur de ségementation.

Et enfin, merci aux nombreux sites internet qui nous ont permis de réaliser notre projet (liste détaillée dans la partie "Références").

# I. Introduction

Dans le cadre du projet Réseaux et Systèmes, nous devions développer une application permettant d'effectuer des recherches multi-critères de fichiers.
Le sujet étant plutôt intéréssant d'un point de vu conception et développement, nous avons été motivé pour travailler et produire un outil fonctionnel qui répond à toutes les attentes du sujet.

Nous avons également eu le temps d'implémenter toutes les extensions demandées : utilisation de libdl pour charger libmagic uniquement lors des recherches d'images, la recherche par expressions régulères (en utilisant libpcre), le démarage de l'examen des fichiers en même temps que l'exploration recursive des répertoires et l'affichage des fichiers au fur et à mesure de leur découverte.
Nous développerons leur implémentation dans la suite de ce rapport.

# II. Choix de conception

Dans l'optique d'avoir un exécutable fonctionnant avec plusieurs threads, nous avons pensé avant d'écrire les premières lignes de code à utiliser une structure permettant un accès multithreads.
Même si l'étape où l'implémentation des threads était la dernière étape, nous nous sommes dit que si l'on prévoyait déjà les choses en conséquence, il serait plus facile d'utiliser des threads plus tard.

Le sujet proposait d'utiliser un tableau contenant les éléments trouvés lors du listing afin d'effectuer les recherches de textes via des threads (en "coupant" le tableau en n parties pour n threads). Nous nous sommes dit qu'avec cette implémentation, les threads n'auraient pas forcément un travail équitable.
En effet, une partie du tableau peut contenir un ensemble de fichiers légers tandis qu'une autre peut contenir un ensemble de fichiers plus importants. Ainsi, certains threads peuvent se finir plus vite, et au contraire, d'autres peuvent durer plus de temps...
Nous avons donc décidé d'utiliser une structure de type **liste simplement chainée** (La liste fut d'abord doublement chainée car nous pensions que nous ferions des insertions triées au fur et mesure du listing, cependant l'utilisation de "scandir" a simplifié le problème en nous fournissant des résultats déjà triés lors du listing).

```
struct Liste
{
        Fichier * fichier;          // Structure représente un fichier (cf. ci-dessous)
        struct Liste * suivant;     // Fichier suivant dans l'arborescence
};
```

La liste simplement chainée utilisée représente donc l'arborescence triée d'un chemin. Ainsi avec cette implémentation, dès lors qu'un fichier était ajouté à la liste, il pouvait automatiquement être analysé lors d'une recherche de texte ou d'image par un thread en attente. Pour savoir quel fichier un thread devait analysé, nous avons utilisé une variable qui contenait le prochain élément à analyser (protégée par un mutex).

L'objectif étant de retourner des chemins relatifs au chemin passé (ou pas) en argument, avec (ou pas) des informations de droits, nous avons utilisé la structure suivante pour représenter un fichier :

```
struct Fichier
{
        char * chemin;              // représente le chemin absolu d'un fichier
        char * chemin_relatif;      // représente le chemin relatif d'un fichier
        struct stat * infos;        // représente les infos (droits, taille) d'un fichier
};
```

## III. Extensions

### a) Utilisation de libdl pour ne charger libmagic que si la bibliothèque est nécessaire, au lieu de se lier dynamiquer avec.

Cette extension a été assez simple à ajouter puisqu'elle ne représente pas beaucoup de ligne de code et qu'une documentation en français était disponible sur internet (voir Références). Cependant, il a été compliqué de trouver le fichier "libmagic.so" (travaillant sur mac, ce fichier n'était pas présent), il a donc fallut implémenter l'extension sur une plateforme linux (ubuntu) et s'assurer que le chemin utilisé ("/usr/lib/libmagic.so") était bien le même que celui utilisé sur la machine de test du loria.

### b) Utilisation de libpcre pour rajouter la recherche d'expressions régulières Perl.

Pour cette libraire, nous avons trouvé un exemple de son utilisation dans la documentation dans l'archive de l'installation de PCRE. Le traitement étant assez similaire à la recherche de texte, l'implémentation s'est également faite facilement.

### c) Amélioration de la parallélisation pour démarrer l'examen des fichiers en même temps que l'exploration récursive des répertoires.

Cette extension posa un nouveau problème dans la gestion des threads : il fallait savoir quand est ce que les threads avaient terminé leur travail. Pour ce faire, après leur lancement, le listing s'exécute en incrémentant une variable à chaque fichier listé. Du côté des threads, une autre variable est incrémenté à chaque fois qu'un fichier est analysé. Ainsi lorsque les 2 variables sont égales (la comparaison a été effectué dans le père après le listing), on pouvait déterminer la fin de l'execution et killer les threads.
Pour synchroniser le lisiting et l'analyse des fichiers, nous avons utilisé un sémaphore initialisé à 0, c'est donc le listing (dans le père) qui informait les threads d'une analyse à effectuer.

### d) Amélioration de la parallélisation pour afficher les fichiers-images au fur et à mesure de leur découverte.

Pour ajouter cette extension, nous avons créé un thread d'affichage lancé en même temps que les threads d'analyses. Ce thread tournait en boucle tout en attendant un signal des threads d'analyses (pour ne pas boucler tout le temps, utilisation d'un sémaphore initialisé à 0 comme pour le sémaphore précedent). Lorsque le signal était reçu, le thread vérifiait que c'était bien le prochain fichier listé qui avait était analysé (et pas un autre, pour respecter l'affichage ordonnée), cela grace à une variable associé à chaque fichier indiquant si le fichier a été analysé ou non et si le résultat est positif).
Cette extension a également du nous amener à modifier la condition de fin des analyses, en effet, cette fois, tout le travail des threads était terminé non seulement quand les fichiers étaient analysés, mais aussi lorsque le thread d'affichage avait terminé son travail.

### e) Commande d'aide.

Ajout d'un fichier HELP qui est affiché dans le terminal lorsque l'on utilise l'option '-h'.

## IV. Difficultés rencontrées

Le temps de la recherche de texte était trop long lors des premiers tests blancs parce que dans la première méthode utilisée, nous découpions le fichier en motifs que l'on comparait caractère par caractère avec la chaine de caractères recherchée (utilisation de strcmp).
Dans la deuxième méthode de recherche (plus rapide), on lit le fichier ligne par ligne et on cherche la chaine de caractères recherchée grâce à la fonction strstr().

En plus des difficultés déjà mentionnées pour les extensions, nous avons rencontré un problème lors de la gestion des threads : souvent en fin d'analyse après l'affichage d'un résultat correct, un segfault survenait. Nous avons mis plusieurs heures à trouver ce problème ! Après avoir trouvé la cause : il semblerait que certaines fois, en faisant n fois un "v()" pour informer les threads d'un fichier à analyser, on rentrer n+1 fois dans le "p()". Cette erreur a été corrigé.

## V. Temps passé

On peut estimer notre temps de travail sur le projet à au moins 40h :

- Conception : 4 heures
- Codage : 35 heures
- Tests : 5 heures
- Rédaction du rapport : 3 heures.

## VI. Conclusion

Le projet RS nous permis de mieux nous familiariser avec les concepts abordés dans la première partie de la matière Réseaux et Systèmes et de les mettre en pratique.

Il nous a permis également d'entreprendre les démarches d'un informaticien, de la conception à la programmation en passant par de nombreuses phases de tests. Le travail en équipe a été également une bonne expérience car nous avons pu répartire équitablement les tâches et avancer rapidement dans le projet.

Malgré les difficultés rencontrées et les bogues découverts lors des phases de tests, le programme obtenu est bel et bien fonctionnel, sans bogue connu.

Pour conclure, le projet RS a été une bonne expérience pour nous, vues les nombreuses compétences acquises pour chacun de nous.

# RÉFÉRENCES

Utilisation de getopt() :
http://www.ai.univ-paris8.fr/~jk/C/Sys/td_3/node5.html

Utilisation de strstr() :
http://www.tutorialspoint.com/ansi_c/c_strstr.htm

Utilisation de libmagic :
http://vivithemage.co.uk/blog/?p=105

Utilisation de libdl :
http://www.unixgarden.com/index.php/gnu-linux-magazine/jouer-avec-la-libdl

Utilisation de libpcre dans la documentation du site :
http://www.pcre.org/

Utilisation des sémaphores :
http://www.commentcamarche.net/faq/11267-utilisation-des-semaphores-systeme-v#destruction-du-semaphore

Listing des fichiers :
http://www.gnu.org/software/libc/manual/html_node/Simple-Directory-Lister.html
http://www.siteduzero.com/tutoriel-3-178902-parcourir-les-dossiers-avec-dirent-h.html

Autres sites utlisés :
http://www.siteduzero.com/tutoriel-3-14189-apprenez-a-programmer-en-c.html
http://www.loria.fr/~lnussbau/RS/Intro-systemes-handout.4up.pdf
http://chgi.developpez.com/pile/
http://chgi.developpez.com/sortlist/
http://www.siteduzero.com/tutoriel-3-31992-compilez-sous-gnu-linux.html#ss_part_3