cherche – recherche multi-critères de fichiers

RS : Réseaux et Systèmes
Deuxième année

L'objectif de ce projet est de réaliser un outil de recherche multi-critères de fichiers.

1 Logistique

1.1 Comment travailler

Apprendre à travailler en groupe fait partie des objectifs pédagogiques de ce projet. Il vous est donc demandé de travailler en binôme. Vous pouvez travailler par groupe de trois si vous le souhaitez (mais sachez que nous tiendrons compte de la taille des groupes lors de la notation – regardez du côté des extensions pour compenser). Il est **interdit** de travailler seul. Un ingénieur travaille rarement seul.

La suite du sujet contient une stratégie possible que vous êtes libre de suivre, ou non.

1.2 Évaluation de ce projet

Tests automatiques. Une part importante de l'évaluation du projet sera faite à l'aide d'une batterie de tests. Il est donc très important, au cours de votre travail, d'accorder une large part à vos propres tests. Plusieurs «tests blancs» seront également organisés au cours du projet : votre code sera récupéré, compilé, et testé sur des cas de tests qui feront partie de ceux utilisés pour l'évaluation finale. Il est donc indispensable de commencer à travailler tôt pour bénéficier de ces «tests blancs».

Rapport. Vous devez rendre un mini-rapport de projet (5 pages maximum hors annexes et page de garde, format pdf). Vous y détaillerez vos choix de conception, les difficultés auxquelles vous avez été confronté, et comment vous les avez résolues. Vous indiquerez également le nombre d'heures passées sur les différentes étapes de ce projet (conception, codage, tests, rédaction du rapport) par membre du groupe.

Section «Remerciements» du rapport. Votre rapport doit contenir quelque chose comme la mention suivante : «Nous avons réalisé ce projet sans aucune forme d'aide extérieure» ou bien «Nous avons réalisé ce projet en nous aidant des sites webs suivants (avec une liste de sites, et les informations que vous avez obtenues de chaque endroit)» ou encore «Nous avons réalisé ce projet avec l'aide de (nommer les personnes qui vous ont aidé, et indiquez ce qu'elles ont fait pour vous)». Si la liste est trop longue, vous pouvez la déplacer en annexe (pour ne pas empiéter sur vos 5 pages de rapport). Rien ne vous empêche de vous faire aider, mais il vous est demandé un minimum d'honnêteté académique en listant vos aides. Tout manquement à cette règle sera sanctionné comme il se doit.

La tricherie sera **sévèrement punie**. Par se faire aider, on entend : avoir une discussion sur le design du code, y compris sur des détails techniques et/ou les structures de données à mettre en œuvre. Par tricher, on entend : copier ou recopier du code ou encore lire le code de quelqu'un d'autre pour s'en inspirer. Nous testerons l'originalité de votre travail par rapport aux autres projets rendus [1].

«Rendu» du projet. Vous devez utiliser un dépôt SVN sur la forge de l'ESIAL (`https://redmine.esial.uhp-nancy.fr/`). Créez votre projet comme un sous-projet de *RS 2012* (dans *Projets 2A*). Votre projet doit être privé pour ne pas que les autres binômes puissent y accéder. L'identifiant de votre projet doit être de la forme `rs2012-login1-login2` (où `login1` et `login2` sont les deux logins des membres du binôme). Ajoutez *Lucas Nussbaum* (login : `lnussbau`) aux développeurs de votre projet afin qu'il puisse récupérer votre code.

Le projet sera récupéré directement sur votre dépôt SVN à la date de fin du projet. Une fois votre binôme constitué et votre projet créé, envoyez un mail à `lucas.nussbaum@univ-lorraine.fr` pour indiquer :

- vos noms, prénoms, logins ;
- l'identifiant de votre projet ;
- le chemin d'accès SVN à votre projet, utilisable par l'utilisateur `lnussbau` (de la forme `http://redmine.esial.uhp-nancy.fr/svn/rs2012-login1-login2/`).

1. `http://theory.stanford.edu/~aiken/moss/`

Votre projet doit contenir, dans le répertoire racine (pas dans un sous-répertoire) :
- Un ficher `AUTHORS` listant les noms, prénoms et logins des membres du groupe (une personne par ligne) ;
- Un `Makefile` compilant votre projet en créant un fichier exécutable nommé `cherche`, également à la racine du projet ;
- Un fichier `rapport.pdf` contenant votre rapport au format PDF.

Voir `http://www.loria.fr/~lnussbau/rs2012.html` pour d'éventuelles informations complémentaires.

Vos questions éventuelles peuvent être adressées à `lucas.nussbaum@univ-lorraine.fr`. Les réponses (et les questions correspondantes) pourront être publiées sur la page du projet.

2 Description du projet

Le projet de cette année vise à développer un outil de recherche multi-critères de fichiers. Il pourrait être vu comme une combinaison (intégrée) des commandes `find`, `grep` et `file` du shell.

Le programme à réaliser est décrit par sa page de manuel ci-dessous.

```
CHERCHE(1)                      Commandes                      CHERCHE(1)

NOM
       cherche - Recherche multi-critères de fichiers

SYNOPSIS
       cherche [OPTION]... [CHEMIN]

DESCRIPTION
       Recherche des fichiers dans CHEMIN (dans le répertoire courant par
       défaut) et ses sous-répertoires et affiche éventuellement des
       informations. Les fichiers trouvés sont affichés dans l'ordre
       alphabétique.

OPTIONS
       Par défaut, cherche affiche tous les fichiers, répertoires, liens
       symboliques, etc. trouvés.  Il est possible d'utiliser cherche pour
       n'afficher que les fichiers contenant une chaîne et/ou étant détectés
       comme des images par la bibliothèque libmagic.

       Aucune option n'est obligatoire.

       Si un chemin n'est pas précisé, le répertoire courant est utilisé.

       -l
            listing long (détaillé). Affichage des permissions, de la
            taille, et de la cible des liens symboliques.

       -t CHAINE
            Restreint la sortie aux fichiers contenant la chaîne CHAINE.

       -i
            Restreint la sortie aux fichiers détectés comme des images par
            la bibliothèque libmagic (ceux dont le type MIME
            commence par image/).

       -p NBTHREADS
            Utilise NBTHREADS threads pour accélérer l'analyse des fichiers
            pour la recherche de texte ou d'images.

       Les options -i et -t peuvent être combinées. Dans ce cas, les
```

```
       fichiers affichés sont ceux correspondant aux deux critères. C'est
       par exemple utile pour analyser des images vectorielles SVG, dont le
       format est basé sur XML.

VALEUR DE RETOUR
       cherche s'arrête en renvoyant la valeur de retour 0 si tous les
       traitements se sont déroulés avec succès. En cas d'erreur (répertoire
       inaccessible, fichier ne pouvant être analysé car non-lisible par
       l'utilisateur, etc.), cherche s'arrête en renvoyant la valeur de
       retour 1.

EXEMPLES
       cherche
            Liste les fichiers (et répertoires) trouvés dans le répertoire
            courant et ses sous-répertoires.

       cherche -l /usr/bin
            Liste de manière détaillée le contenu du répertoire /usr/bin
            (et ses sous-répertoires)

       cherche -l -p 4 -t human -i /usr/share/openclipart/svg/
            Cherche les images contenant la chaîne de caractères "human"
            dans /usr/share/openclipart/svg/, et les affiche sous la forme
            d'un listing détaillé. Pour accélérer la recherche, 4 threads
            sont utilisés.

VOIR AUSSI
       find(1), file(1), libmagic(3), magic(5)
```

2.1 Conseils de réalisation

La quantité de code à produire est relativement faible (quelques centaines de lignes de code tout au plus). Mais le niveau de difficulté du code à produire est très important. Il est crucial de programmer de manière prudente, réfléchie, claire, en testant bien les différents cas d'erreur.

N'hésitez pas à utiliser `strace`, `gdb`, `valgrind`, etc. pour déboguer vos programmes.

Il est conseillé de réaliser votre projet dans l'ordre qui suit.

Étape 1 : analyse des options de la ligne de commande

Il est possible de programmer manuellement l'analyse des options passées sur la ligne de commande, mais il est bien plus commode d'utiliser la fonction `getopt` pour cela.

Étape 2 : listing du contenu d'un répertoire

Étape 3 : listing des sous-répertoires

Il peut être pratique d'utiliser de la récursivité.

Étape 4 : listing détaillé (`-l`)

Les champs à afficher sont :

- les permissions (comme dans `ls -l`, en gérant les cas des fichiers, répertoires, liens symboliques) ;
- la taille (en octets, décimal), en réservant de la place pour 8 caractères afin d'obtenir un formatage en colonnes ;
- le nom ;
- dans le cas d'un lien symbolique, la cible, avec la syntaxe `-> cible`.

La cible des liens symboliques n'est affichée que dans le listing détaillé.

Étape 5 : recherche de texte (`-t`)

Il faut chercher le texte passé en paramètre dans les fichiers (pas les liens symboliques), tout en continuant d'explorer les sous-répertoires. Seuls les fichiers contenant la chaîne sont listés.

Étape 6 : recherche d'images (`-i`)

Il faut détecter les fichiers contenant une image. Regarder leur extension ne suffit pas, puisqu'on

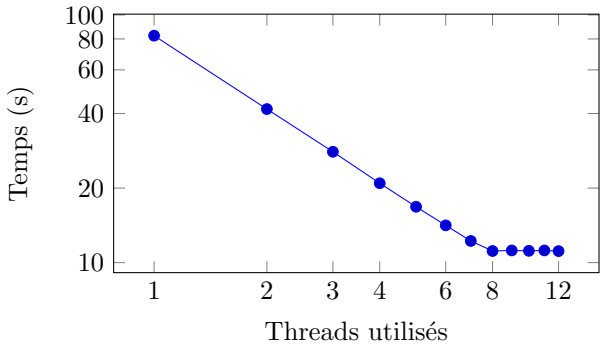

FIGURE 1 – Temps d'exécution avec 30962 fichiers sur une machine équipée de deux processeurs Intel Xeon L5420 (4 coeurs chacun). L'accélération est bien linéaire jusqu'à 8 threads (échelle log-log).

peut très bien changer l'extension du fichier. Une bonne manière de procéder est donc d'utiliser la bibliothèque `libmagic`, qui est fournie avec l'utilitaire `file`[2], et qui, à l'aide d'une base de données de *valeurs magiques*, permet d'identifier le contenu d'un fichier. Consultez la page de manuel `libmagic(3)`[3] pour plus de détails sur son utilisation. Une bonne manière de traiter son résultat est de demander le type MIME, et de vérifier qu'il commence par `image/`.

Étape 7 : parallélisation

La recherche d'images (et dans une moindre mesure de texte) sont assez coûteuses en temps. Sur une machine multi-cœurs, il est utile (voir figure 1) de paralléliser l'analyse en modifiant l'application pour déléguer l'examen des fichiers à différents threads. Le schéma d'exécution suivant est suggéré :

1. Dans un premier temps, les répertoires à analyser sont parcourus récursivement, et tous les fichiers à examiner sont stockés dans un tableau ;
2. Ensuite, des threads sont démarrés, et accèdent de manière concurrente au tableau, pour y chercher les fichiers à examiner avec *libmagic* ;
3. Quand tous les threads ont terminé (et donc, que tous les fichiers ont été analysés), on parcourt le tableau pour afficher les fichiers dont l'examen aura été positif.

Ce schéma ressemble à un schéma producteur-consommateur, mais les étapes de production et de consommation sont séparées pour simplifier le code. De même, l'affichage des résultats se fait à la fin, et non au fur et à mesure de la découverte de fichiers-images.

2.2 Extensions possibles

Les extensions réalisées seront valorisées (mais il faut d'abord réaliser correctement tous les points décrits ci-dessus). Elles doivent être décrites dans le rapport.

- Utilisation de `libdl` pour ne charger `libmagic` que si la bibliothèque est nécessaire, au lieu de se lier dynamiquer avec.
- Utilisation de `libpcre` pour rajouter la recherche d'expressions régulières *Perl* (en rajoutant une option -T pour la différencier de la recherche de chaînes).
- Amélioration de la parallélisation :
  - pour démarrer l'examen des fichiers en même temps que l'exploration récursive des répertoires ;
  - pour afficher les fichiers-images au fur et à mesure de leur découverte (par exemple en *signalant* au thread-afficheur l'arrivée de nouveaux résultats).

  Dans tous les cas, il est crucial (notamment pour l'évaluation) que l'affichage des fichiers se fasse dans le même ordre que lors d'une exécution séquentielle.

Si vous souhaitez vous lancer dans une autre extension, n'hésitez pas à demander conseil à Lucas Nussbaum.

2.3 Exemple de sorties

Attention, le respect du format des sorties est crucial pour permettre le test de votre programme. Une sortie différente sera considérée comme fausse.

2. `http://www.darwinsys.com/file/`
3. `http://manpages.debian.net/cgi-bin/man.cgi?query=libmagic`

En particulier :
- Les entrées sont affichées dans l'ordre alphabétique
- Elles ne sont pas préfixées par le répertoire à analyser
- Les entrées de répertoires finissent par '/'
- Les entrées pour '.' et '..' ne sont pas affichées
- En mode parallèle, les entrées restent affichées dans l'ordre

Seule la sortie standard (*stdout*) est utilisée pour l'évaluation. Il est donc utile d'envoyer vos messages de debug éventuels vers *stderr*.

2.3.1 Listing complet (sans recherche), long

```
./cherche -l test/
-rw-------         0 a
-rw-------        42 b
-rw-------    123456 c
lrwxrwxrwx         1 d -> /
drwx------      4096 da/
drwx------      4096 da/db/
drwx------      4096 da/db/dc/
lrwxrwxrwx        54 da/db/dc/f -> /un_chemin/vraiment_tres/tres_tres/long
-rw-------         0 da/db/dc/f2
-rw-------         0 da/db/f1
lrwxrwxrwx         1 e -> c
-rw-------         9 matches1
-rw-------         7 matches2
-rw-------         7 matches3
```

2.3.2 Recherche d'images et parallélisation

```
$ ./cherche -p 10 -i /usr/src/linux-source-3.2/
Documentation/blockdev/drbd/DRBD-8.3-data-packets.svg
Documentation/blockdev/drbd/DRBD-data-packets.svg
Documentation/logo.gif
drivers/video/logo/clut_vga16.ppm
drivers/video/logo/logo_blackfin_clut224.ppm
drivers/video/logo/logo_blackfin_vga16.ppm
drivers/video/logo/logo_dec_clut224.ppm
drivers/video/logo/logo_linux_clut224.ppm
drivers/video/logo/logo_linux_mono.pbm
drivers/video/logo/logo_linux_vga16.ppm
drivers/video/logo/logo_m32r_clut224.ppm
drivers/video/logo/logo_mac_clut224.ppm
drivers/video/logo/logo_parisc_clut224.ppm
drivers/video/logo/logo_sgi_clut224.ppm
drivers/video/logo/logo_spe_clut224.ppm
drivers/video/logo/logo_sun_clut224.ppm
drivers/video/logo/logo_superh_clut224.ppm
drivers/video/logo/logo_superh_mono.pbm
drivers/video/logo/logo_superh_vga16.ppm
```

2.3.3 Recherche d'images et parallélisation, listing long

```
./cherche -p 10 -i -l /usr/src/linux-source-3.2/
-rw-r--r--     22216 Documentation/blockdev/drbd/DRBD-8.3-data-packets.svg
-rw-r--r--     17433 Documentation/blockdev/drbd/DRBD-data-packets.svg
-rw-r--r--     16335 Documentation/logo.gif
```

```
-rw-r--r--   230 drivers/video/logo/clut_vga16.ppm
-rw-r--r--  55752 drivers/video/logo/logo_blackfin_clut224.ppm
-rw-r--r--  54088 drivers/video/logo/logo_blackfin_vga16.ppm
-rw-r--r--  76866 drivers/video/logo/logo_dec_clut224.ppm
-rw-r--r--  76845 drivers/video/logo/logo_linux_clut224.ppm
-rw-r--r--  12847 drivers/video/logo/logo_linux_mono.pbm
-rw-r--r--  76844 drivers/video/logo/logo_linux_vga16.ppm
-rw-r--r--  75502 drivers/video/logo/logo_m32r_clut224.ppm
-rw-r--r--  76846 drivers/video/logo/logo_mac_clut224.ppm
-rw-r--r--  78444 drivers/video/logo/logo_parisc_clut224.ppm
-rw-r--r--  76840 drivers/video/logo/logo_sgi_clut224.ppm
-rw-r--r--  14216 drivers/video/logo/logo_spe_clut224.ppm
-rw-r--r--  78440 drivers/video/logo/logo_sun_clut224.ppm
-rw-r--r--  76843 drivers/video/logo/logo_superh_clut224.ppm
-rw-r--r--  12845 drivers/video/logo/logo_superh_mono.pbm
-rw-r--r--  76842 drivers/video/logo/logo_superh_vga16.ppm
```

2.4 Règles du jeu

Seules les fonctions fournies par les bibliothèques suivantes peuvent être utilisées :
- bibliothèque C standard (libc) : `*dir`, `getopt`, pthreads, ...
- bibliothèque `libmagic`

Vous n'avez pas donc pas le droit d'utiliser de bibliothèque externe, de plus haut niveau, qui réaliserait une partie du travail à votre place. Si vous avez un doute sur le statut d'une fonction, posez la question.

3 Calendrier

- Des «tests blancs» seront effectués à au moins trois reprises, aux alentours du 07/10/12, du 22/10/12, et du 05/11/12. Votre code sera récupéré, compilé, et testé sur des cas de tests qui feront partie de ceux utilisés pour l'évaluation finale. Les résultats vous seront communiqués, mais ne seront pas pris en compte pour l'évaluation finale. L'expérience montre que l'environnement de développement et d'exécution (architecture, système, version du compilateur ...) peut influencer les résultats et mettre en évidence des bugs qui pourraient ne pas être visibles sur vos machines. Il est donc très utile de commencer à travailler tôt pour bénéficier de ces «tests blancs». Aucune réclamation ne pourra être acceptée si votre programme ne se comporte pas correctement dans l'environnement d'évaluation, puisque vous aurez pu bénéficier de plusieurs «tests blancs» avant le rendu du projet.
  Seule la sortie *standard* sera utilisée pour comparer la sortie de votre programme avec celle attendue. Il est donc conseillé d'utiliser la sortie *erreur* pour afficher vos messages de débogage.
- La version finale de votre projet est à rendre pour le **jeudi 15/11/2012 à 8h00**. Il sera récupéré directement sur vos dépôts SubVersion. Vous n'avez donc pas d'action particulière à effectuer pour *rendre* le projet, mais vous devez vous assurer que les fichiers requis sont bien présents. Une bonne manière de vérifier que tous les fichiers sont bien présents sur le dépôt est de réaliser un nouveau *checkout* et d'en vérifier le contenu. Les groupes dont le projet ne pourra pas être récupéré correctement seront évidemment sanctionnés.