République Algérienne Démocratique et Populaire
الجمهورية الجزائرية الديمقراطية الشعبية
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
وزارة التعليم العالي و البحث العلمي

ECOLE NATIONALE SUPÉRIEURE D'INFORMATIQUE

المدرسة الوطنية العليا للإعلام الآلي
(المعهد الوطني للتكوين في الإعلام الآلي سابقا)
Ecole nationale Supérieure d' Informatique
ex. INI (Institut National de formation en Informatique)

Création d'une commande de recherche de fichier sous LINUX

TP DE SYSTEME D'EXPLOITATION

Réalisé par :

→ BELFODIL Adnane
→ MEDKOUNE Nawel

April 29, 2014

Sommaire

Introduction :

Présentation du système de fichier UNIX (Linux)

Définition d'un système de fichier :

Le système de gestion de fichiers (SGF) est la partie la plus visible d'un système d'exploitation qui se charge de gérer le stockage et la manipulation de fichiers (sur une unité de stockage : partition, disque, CD, disquette. Un SGF a pour principal rôle de gérer les fichiers et d'offrir les primitives pour manipuler ces fichiers.

Les systèmes de gestion de fichier diffèrent les uns des autres de part la taille maximale qu'un fichier peut avoir, ml taille maximale d'une partition sur un disque dur, la gestion des droits d'accès aux fichiers et répertoires et la journalisation.

A noter que la journalisation est l'utilisation d'un fichier journal qui trace les dernières modifications à effectuer sur un système de fichiers afin de récupérer les données en cas d'arrêt brutal du système d'exploitation.

Quelques systèmes de fichiers Unix :

Nom du système de fichiers	Taille maximale d'un fichier	Taille maximale d'une partition	Journalisée ou non ?	Gestion des droits d'accès?
ext2fs (Extended File System)	2 TiB	4 TiB	Non	Oui
ext3fs	2 TiB	4 TiB	Oui	Oui
ext4fs	16 TiB	1 EiB	Oui	Oui
ReiserFS	8 TiB	16 TiB	Oui	Oui

Types de fichiers :

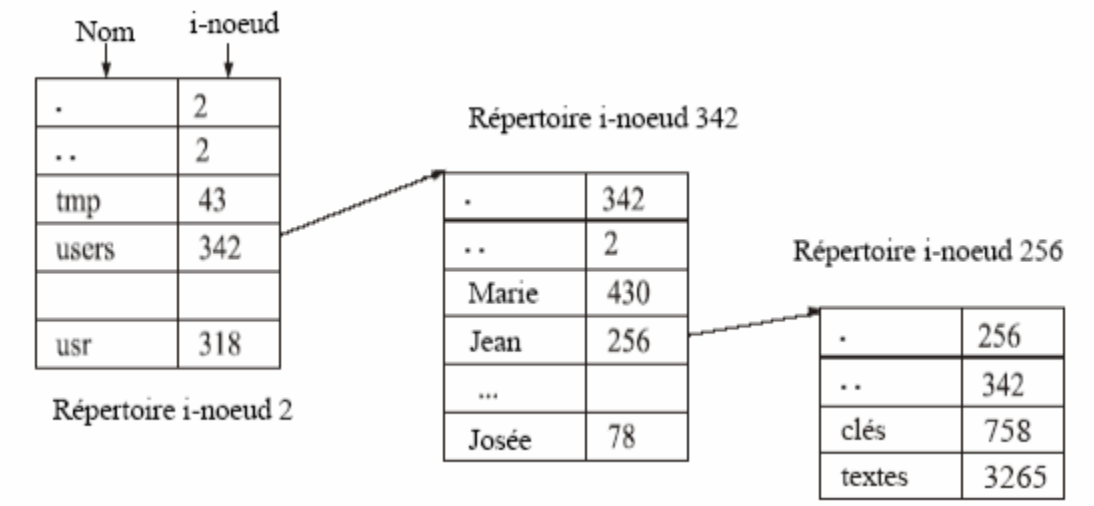
Fichier normal:

Un fichier est l'unité de stockage logique mise à la disposition des utilisateurs pour l'enregistrement de leurs données : c'est l'unité d'allocation.

Sous les systèmes UNIX tout élément est représenté sous forme de fichier. Chaque fichier est identifié par un numéro appelé numéro d'inode.

Répertoire :

Du point de vue SGF, un répertoire est un fichier qui dispose d'une structure logique : il est considéré comme un tableau qui contient une entrée par fichier. Ce tableau est appelé Tableau des Inodes. Chaque entrée pointe vers le numéro d'inode du fichier contenu dans ce répertoire.



Il est important de noter que l'entrée d'un répertoire est différente d'une version à une autre du SGF.

Liens :

Les liens sont des fichiers spéciaux permettant d'associer plusieurs noms (liens) à un seul et même fichier. Ce dispositif permet d'avoir plusieurs instances d'un même fichier en plusieurs endroits de l'arborescence sans nécessiter de copie, ce qui permet notamment

d'assurer un maximum de cohérence et d'économiser de l'espace disque. On distingue deux types de liens :

- Liens symboliques : représentent des pointeurs virtuels vers des fichiers réel (i.e. raccourcis)
- Liens physiques : ou hardlinks, représentent un nom alternatif pour un fichier. Pour comprendre, on va prendre le cas de la suppression d'un fichier. Tant que ce fichier possède au moins un lien physique, il n'est pas effacé.

Fichiers spéciaux :

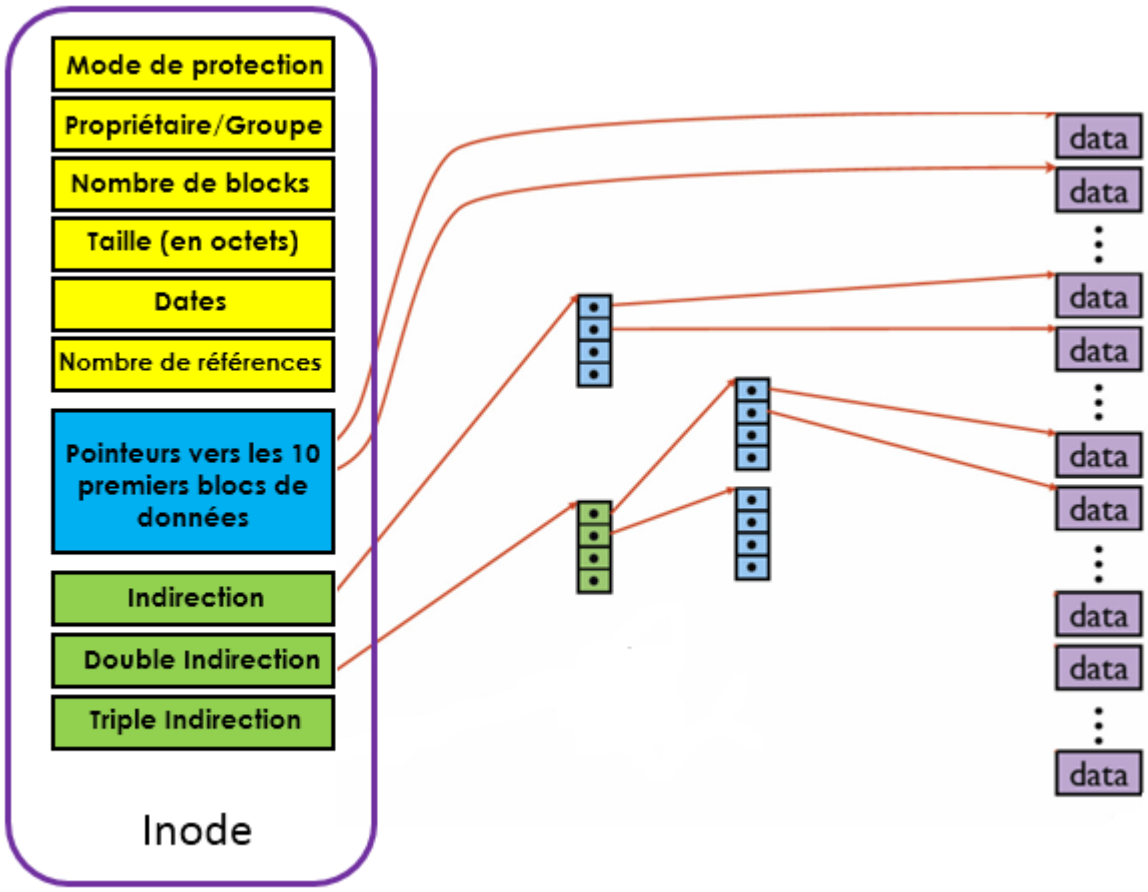
Situés dans /dev, ce sont les points d'accès préparés par le système aux périphériques. Le montage va réaliser une correspondance de ces fichiers spéciaux vers leur répertoire "point de montage". Par exemple, le fichier /dev/hda permet l'accès et le chargement du 1er disque IDE

Les i-nodes :

Structure Inode

Un nœud d'index est constitué d'attributs décrivant le fichier ou le répertoire et d'adresses de blocs contenant des données. Cette structure possède plusieurs entrées, elle permet au système de disposer d'un certain nombre de données sur le fichier :

- Taille du fichier.
- Propriétaire et groupe.
- Les droits d'accès : pour chaque fichier, Unix définit trois droits d'accès (lecture (r), écriture (w) et exécution (x)) pour chaque classe d'utilisateurs (trois types d'utilisateur {propriétaire, membre du même groupe que le propriétaire, autres}). Donc à chaque fichier, Unix associe neuf droits.
- Les dates de création, de dernière consultation et de dernière modification,
- Nombre de références,
- Les dix premiers blocs de données.
- Trois entrées contiennent l'adresse d'autres blocs (bloc d'indirection) :
    - Indirection
    - Double indirection
    - Triple Indirection



Gestion des Inodes :

Quand un processus se réfère à un fichier par son nom, il va le décomposer en répertoires et en nom de fichier. Il va ensuite vérifier que le processus a le droit d'accéder à chacun des répertoires, et éventuellement récupérer l'inode du fichier.

Quand un processus crée un nouveau fichier, le noyau lui assigne donc un inode libre (non utilisé).

Pour ce faire, il regarde sa table des inodes libres, en choisit un, l'enlève de la table, ajoute une référence à cet inode, et remplit les informations contenues dans l'inode.

Particularité des systèmes de fichier Unix (Linux) :

Sur Microsoft Windows, il est impossible de modifier les propriétés d'un fichier quand celui-ci est ouvert. Cette restriction n'existe pas sur les systèmes de fichiers de type Unix (ext2, ext3, ReiserFS…). La raison est que sur les systèmes de fichiers *nix, les fichiers sont selon l' inode. Lorsqu'on supprime un fichier par exemple, l'inode est délié du SGF, c'est-à-dire qu'il n'est plus indexé. Mais si ce fichier est ouvert en même temps, il se trouve lié au programme qui l'a ouvert. Il continue donc d'être existé. Un fichier ne peut être supprimé que si tous les liens avec cet inode sont coupés.

Aussi, à chaque création du système de fichiers, le système réserve une certaine quantité d'espace pour le super-utilisateur (root), en général 5%. Ceci permet au super-utilisateur de pouvoir se logger sur le système, et de faire des tâches administratives quand le système de fichiers est plein pour les utilisateurs.

Structure physique d'un fichier :

La structure d'un fichier sur le disque est décomposée en plusieurs blocs.

L'inode précédemment décrit contient une sorte de table des matières permettant de localiser les données d'un fichier sur le disque.

Les structures de fichiers sur les systèmes de fichiers linux actuels ont été largement inspirées par le système de fichiers présent sur les systèmes BSD (FFS).

Le système est constitué de N groupes de blocs, présents après le secteur de boot du système de fichiers.

Chacun de ces groupes possède la même structure, et possède une copie redondante des informations vitales du système de fichiers. Ces informations sont stockées dans les blocs intitulés « superblock » et « FS Descriptors ».

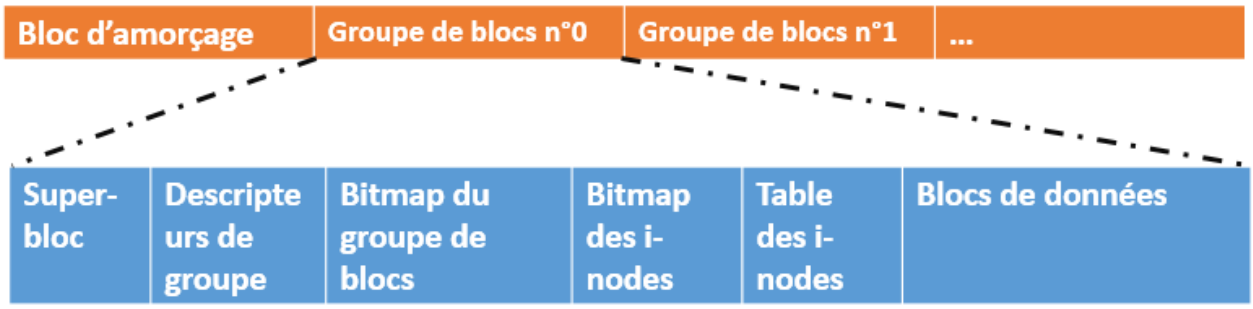
Parmi les informations les plus importantes que contient le superbloc, on retiendra :

- **s_isize**: la taille en blocs de la liste des inodes (i_list).
- **s_fsize:** la taille en blocs du système de fichiers.
- **s_fname:** le nom externe du système de fichiers.
- **s_free:** la liste des blocs libres.
- **s_inode:** la liste des inodes libres
- **s_tfree:** le nombre de blocs libres
- **s_tinode:** le nombre d'inodes libres

FS Descriptors c'est la table des descripteurs, elle localise les différentes tables du groupe.

Ces blocs sont suivis par les blocs suivants :

- **Block Bitmap :** c'est la carte des blocs du groupe.
- **Inode Bitmap :** c'est la carte des inodes du groupe.
- **Inode Table :** la table des inodes du groupe
- **Data blocks :** ce sont les blocs de données dans lesquels est stocké le contenu des fichiers.



La table des inodes est l'élément fondamental du système de fichier. Tout dommage à cette structure détruit irrémédiablement les liens donc les fichiers qui deviennent difficilement récupérables.

Avantages de cette structure :

- Permet de récupérer un système de fichiers dont le super-bloc a été corrompu.
- La redondance des informations vitales du système permet de repérer une corruption de ces informations.
- Permet d'avoir une proximité permanente des informations vitales du système de fichiers, ce qui raccourcit le déplacement nécessaire des têtes de lectures pour y accéder

L'Arborescence :

Les fichiers d'une machine Unix sont organisés en une arborescence dont la base, appelée *racine*, est notée «/». Les feuilles sont des fichiers ordinaires (textes, programmes, ...). Les noeuds internes sont les répertoires (ou directories).

Le nom complet d'un fichier est formé d'une liste des répertoires qu'il faut traverser à partir du haut de la hiérarchie (le répertoire racine (root directory)) plus le nom_du_fichier. Les répertoires sont séparés par « / » slash

La norme selon FHS :

La norme sur la hiérarchie des systèmes de fichiers définit une organisation logique standard concernant l'organisation de ces répertoires.

/bin commandes binaires utilisateur essentielles (pour tous les utilisateurs)
/boot fichiers statiques du chargeur de lancement
/dev fichiers de périphériques
/etc configuration système spécifique à la machine
/home répertoires personnels des utilisateurs
/lib bibliothèques partagées essentielles et modules du noyau
/mnt point de montage pour les systèmes de fichiers montés temporairement
/proc système de fichiers virtuel d'information du noyau et des processus
/root répertoire personnel de root (optionnel)
/sbin binaires système (binaires auparavant mis dans /etc)
/sys état des périphériques (model device) et sous-systèmes (subsystems)
/tmp fichiers temporaires

Montage d'un système de fichier :

Les fichiers d'un système UNIX sont organisés dans une arborescence unique. L'accès et l'utilisation de systèmes extérieurs doit s'effectuer par intégration de ces systèmes de fichiers dans la racine « / ». Ce mécanisme d'intégration, souple et paramétrable, s'appelle le montage.

Structures utilisées :

Dirent :

Cette structure représente une entrée du répertoire avec les informations nécessaires pour retrouver le fichier contenu dans le répertoire

```
struct dirent
               {
                   long d_ino;                 /* numéro de l'inode*/
                   off_t d_off;                /* offset à cette entrée */
                   unsigned short d_reclen;    /* longueur de d_name */
                   char d_name [NAME_MAX+1];   /* nom du fichier */
               }
```

Stat :

Cette structure contient les informations d'un fichier donné.

```
struct stat
               {
                   dev_t         st_dev;      /* Périphérique                */
                   ino_t         st_ino;      /* Numéro i-noeud              */
                   mode_t        st_mode;     /* Protection                  */
                   nlink_t       st_nlink;    /* Nb liens matériels          */
                   uid_t         st_uid;      /* UID propriétaire            */
                   gid_t         st_gid;      /* GID propriétaire            */
                   dev_t         st_rdev;     /* Type périphérique           */
                   off_t         st_size;     /* Taille totale en octets     */
                   unsigned long st_blksize;  /* Taille de bloc pour E/S     */
                   unsigned long st_blocks;   /* Nombre de blocs alloués     */
                   time_t        st_atime;    /* Heure dernier accès         */
                   time_t        st_mtime;    /* Heure dernière modification */
                   time_t        st_ctime;    /* Heure dernier changement    */
               };
```

Tm

Contient des informations sur le temps.

```
struct tm
               {
                       int     tm_sec;         /* secondes           */
                       int     tm_min;         /* minutes            */
                       int     tm_hour;        /* heures             */
                       int     tm_mday;        /* quantième du mois  */
                       int     tm_mon;         /* mois (0 à 11 !)    */
                       int     tm_year;        /* année              */
                       int     tm_wday;        /* jour de la semaine */
```

```
                    int     tm_yday;        /* jour de l'année    */
                    int     tm_isdst;       /* décalage horaire   */
            };
```

Fonctions utilisées

Ces fonctions sont présentes dans la bibliothèque système : dirent.h

Opendir: Permet d'ouvrir un répertoire

```
DIR *opendir(const char *dirname);
```

Closedir: Permet de fermer un répertoire.

```
int closedir(DIR *dirp);
```

Readir :

Cette fonction retourne un pointeur vers une structure dirent qui représente une entrée du répertoire dirp à la position actuelle du pointeur dans le répertoire. Après la lecture, il positionne le pointeur vers la prochaine entrée.

```
struct dirent *readdir(DIR *dirp);
```

Stat :

Cette fonction renvoi les informations à propos du fichier indiqué dans file_name et le remplit dans le buffer qui a une structure de stat

```
int stat(const char *file_name, struct stat *buf);
```

Strftime:

La fonction strftime() formate les divers champs de la structure tm en fonction de la chaîne de spécification format, puis place le résultat dans la chaîne s de taille max.

Conception et réalisation du TP :

Vue générale :

Pour Réaliser la commande de recherche de fichiers, on a choisi de diviser le problème en quatre modules principaux, chacun à son tour est composé d'une ou plusieurs fonctions, l'exécution de la commande passe par deux étapes, chaque étape est assuré par un ensemble de modules, les deux étapes sont trivialement ceux suivants :

1. **Lecture de la commande :** la lecture de la commande consiste à récupérer les différents arguments depuis l'expression de la commande, cela impliquerait la récupération du répertoire de départ s'il existe, les différentes options données et l'expression régulière qui définit les noms de fichiers souhaités, cette étape est assuré par le module « Parse Arguments ».
2. **Le parcours de fichiers :** Si aucune erreur n'est signalée par l'étape précédente, on lance le parcours à partir du répertoire initial, durant cette étape on parcourt l'arborescence du répertoire initiale et ceci en prenant compte de niveau de profondeur donné en argument, et pour chaque fichier trouvé on fait appel au module « Match Strings », si le fichier est conforme à l'expression régulière annoncé par la commande on fait appel au module « Display Informations » qui a son tour selon les options affiche les informations du fichier en question.

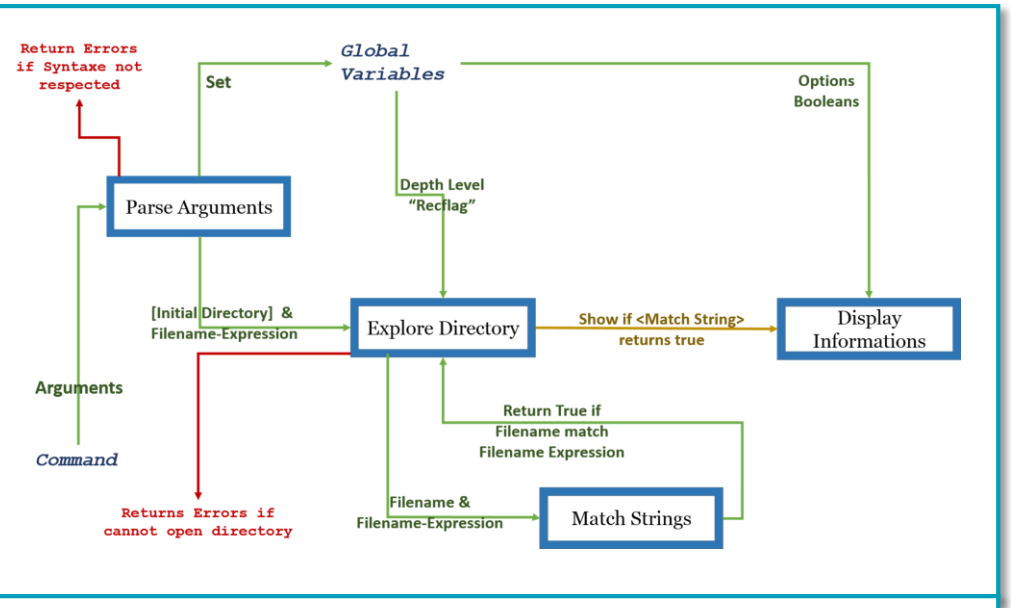


Fig.1 – Vue générale du processus de l'exécution de la commande

Les Variables Globaux :

Pour réaliser la commande, on a utilisé les variables globaux suivants, celle-ci permettent de stocker en générale les options récupéré depuis la commande :

1. ***Recflag*** – entier qui stocke la valeur de profondeur de recherche, positive dans le cas où la profondeur est donnée en argument, -1 sinon, ainsi la valeur -1 représente l'infini et donc dans le cas où on a -1 on recherche dans toute l'arborescence du répertoire de départ.
2. ***dflag*** – entier qui stocke 1 dans le cas où l'option -d ou -a est donné en option, 0 sinon.
3. ***mflag*** – entier qui stocke 1 dans le cas où l'option -m ou -a est donné en option, 0 sinon.
4. ***sflag*** – entier qui stocke 1 dans le cas où l'option -s ou -a est donné en option, 0 sinon.
5. ***tflag*** – entier qui stocke 1 dans le cas où l'option -t ou -a est donné en option, 0 sinon.
6. ***pflag*** – entier qui stocke 1 dans le cas où l'option –p ou -a est donné en option, 0 sinon.
7. ***entete*** – entier qui stocke 1 dans le cas où l'entête de l'affichage est affiché, 0 sinon.
8. ***nbres*** – entier qui stocke le nombre de fichiers trouvés correspondant à l'expression régulière donné en argument.
9. ***regexp*** – chain de de caractère qui stocke l'expression régulière donné en argument.
10. ***divideresult*** – tableau de chaines de de caractères qui stocke les chaines de caractères résultantes après la décomposition de l'expression régulière en petites chaines séparés par le caractère '*'.
11. ***sizeofdivideresult*** – entier qui stocke la taille du tableau `divideresult`.
12. ***startwith*** – entier qui stocke '1' si l'expression régulière commence par le caractère '*', 0 sinon.
13. ***endwith*** – entier qui stocke '1' si l'expression régulière termine par le caractère '*', 0 sinon.

Module Parse Arguments :

Le module « Parse Arguments » a pour rôle la décomposition de la commande, et la lecture de ses arguments, c'est le module qui assure l'étape de la lecture de la commande, il procède à la récupération du répertoire de départ s'il existe, les différentes options et l'expression régulière du nom de fichier, dans le cas où il y a une anomalie (cas d'une option inexistante), une erreur est signalé, et l'exécution de la commande s'arrête au niveau de la lecture et ne passe pas vers la prochaine étape, ainsi le module Parse Arguments peut être considéré comme le validateur de l'expression de la commande.

Ce module est composé de deux fonctions, la première `parse_arguments` étant celle qui récupère les arguments, et la deuxiéme `afficher_erreur_cmd` est celle qui affiche l'erreur signalé par `parse_arguments`.

Fonction parse_arguments :

Entête :

```
int parse_arguments(int argc, char * argv[], char* path).
```

Rôle :

La fonction parse_arguments permet à partir d'`argc` et `argv` récupéré à partir de la ligne de commande, de récupérer l'expression régulière et de la mettre dans la variable globale `regexp`, le chemin du répertoire de départ s'il existe et de le renvoyer dans la chaine de retour `path`, les différentes options.

Cette fonction retourne un entier qui est à 0 si aucune anomalie n'est signalée, négatif dans le cas où une option inexistante est donnée, l'indice de l'option erroné dans `argv` est décrit dans la valeur de retour.

Analyse :

Pour réaliser la fonction, on vérifie d'abord si le premier argument n'est pas une option, si c'est le cas on déduit que le chemin du répertoire de départ est donné en argument sinon la valeur de `path` serait «.» pour dire que le chemin de départ est le répertoire actuel.

On récupère ensuite le dernier argument qui représente l'expression régulière recherché, puis on parcourt le reste d'`argv` pour récupérer les différentes options, pour chaque `argv[i]` on s'assure si la longueur de l'option est strictement égale à 1, si c'est le cas on s'assure d'abord si le caractère correspond vraiment à une option qui existe ou une valeur d'un entier , dans le cas échéant on renvoie –1-1 pour dire que l'option `argv[i]` est inexistante.

Dans le cas où la longueur d'`argv[i]` est strictement supérieure à 1, on vérifie d'abord si c'est une valeur d'entier, si c'est le cas on met à jour l'entier `recflag` qui représente la profondeur exigé, sinon une erreur est signalé par le retour de –1-1.

Fonction afficher_erreur_cmd :

Entête :

```
void afficher_erreur_cmd(int err,char *argv[]).
```

Rôle :

Cette fonction permet d'afficher l'option erronée à partir de l'entier `err` et `argv` , la valeur de l'entier err étant celle de l'entier retourné par parse_arguments est utilisé pour afficher l'option inexistante.

Module Match String :

Le module « Match String » est celui qui compare l'expression régulière donné en argument avec le nom du fichier envoyé par le module « Explore Directory », il retourne **VRAI** dans le cas ou les deux chaines de caractères sont équivalentes dans le sens de du modèle de l'expression régulière énoncé dans le TP, **FAUX** sinon.

La figure suivante permet d'expliquer le schéma d'exécution d'une comparaison entre deux chaines et de mettre en évidence les différentes fonctions qui composent le module « Match String » :

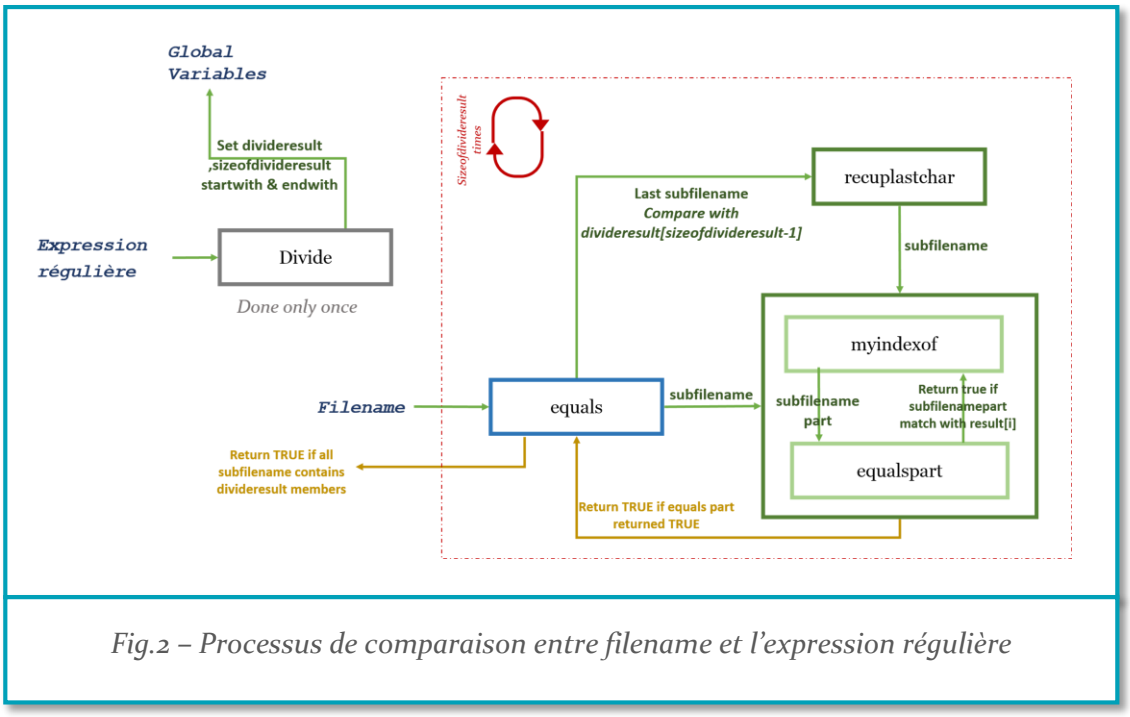


Fig.2 – Processus de comparaison entre filename et l'expression régulière

Remarque :

La figure ci-dessus a pour but d'expliquer en global le processus de comparaison entre le nom du fichier et l'expression régulière, vue que ce n'est pas l'objet du TP, on n'a pas détaillé les différentes fonctions de ce module, pour une meilleure compréhension consulter le code source de la fonction `Search`.

Module Explore Directory :

Le module « Explore Directory », permet de traverser l'arborescence du répertoire de départ tout en prenant compte de la profondeur limite signalé dans la commande, son rôle se résume dans le parcours des différents fichiers, et pour chaque fichier on récupère son nom puis on le compare avec l'expression régulière en faisant appel au module « Match

Strings » , dans le cas où le nom de fichier correspond à l'expression régulière, on fait appel au module « Display informations » pour afficher les différentes information du fichier en question.

Ce module est assuré par la fonction récursive `parcourir`, qui a comme paramètres le chemin du répertoire ou il faut explorer et le niveau de récursivité (profondeur courante) qui sont nécessaires pour l'appel récursive de la fonction.

Fonction parcourir:

Entête :

```
int parcourir(char * path, int recniv).
```

Rôle :

La fonction `parcourir` comme décrit auparavant a deux paramètres qui sont : `path` qui est une chaine de caractère qui décrit le chemin du répertoire ou il faut explorer, et l'entier recniv qui décrit la profondeur courante, elle a pour rôle de parcourir les différents fichiers, puis dans le cas où le nom de fichier corresponds à l'expression régulière `regexp` on affiche les informations du fichier.

Cette fonction retourne un entier, qui est à 0 dans le cas où le chemin `path` ne peut pas être ouvert comme répertoire, 1 sinon .

Analyse :

Pour réaliser la fonction, on ouvre d'abord le répertoire de chemin `path` dans une variable de type `DIR`, ici deux cas de figures se présentent, soit la fonction `opendir` retourne `NULL` et donc le répertoire ne peut pas être ouvert, on retourne dans ce cas la valeur 0, soit le répertoire peut être ouvert, dans ce cas on réalise une boucle, et pour chaque itération on récupère une entrée qui représente un fichier depuis le répertoire ouvert en utilisant la fonction `readdir` dans une variable de type `dirent`, après la récupération du nom du fichier depuis le fichier, on le concatène avec le chemin du répertoire actuel, puis on essaye de lire les informations du fichier en utilisant `stat` , ici aussi deux cas de figures se présentent, soit on peut lire les informations, soit non, dans le cas où c'est possible on vérifie d'abord si le fichier est un répertoire, si c'est le cas, on s'assure avant que le fichier n'est pas un «.» ou «..» pour éviter une boucle infini sur le même répertoire, si c'est le cas et qu'on a pas encore atteint à la limite de profondeur, on appelle récursivement la fonction parcourir avec en paramètre, le `path` du répertoire trouvé et une diminution de 1 de la valeur de `recniv` (dans le cas où on a la profondeur différente de -1) .

Pour afficher les fichiers correspondants à l'expression régulière `regexp` on compare le nom de chaque fichier avec `regexp` en utilisant la fonction `equals (filename)`, dans le cas où elle retourne `VRAI`, on affiche le fichier.

Module Display Informations:

Le module « Display Informations » a pour rôle d'afficher les différentes informations d'un fichier à partir de son chemin, tout en prenant compte des options données en arguments. Ce module est réalisé de la manière suivante :

- Une fonction principale `afficher` dont le but est d'afficher le chemin absolu du fichier en question, de plus elle fait appel aux fonctions secondaires dépendamment des options données en argument.
- Des fonctions secondaires, dont chacune s'occupe de la présentation d'un type d'information particulier, les fonctions secondaires sont ceux suivants :
    - `Printtime` – permet de formater l'affichage du temps de la dernière modification ou le temps de la dernière utilisation, les deux options –d et –m font appels à cette fonction.
    - `filetype` – permet de retourner un caractère décrivant le type du fichier, l'option –t fait appel à cette fonction.
    - `permission` – permet de donner en résultat une chaine de caractère contenant les permissions du fichier donné en paramètre, la chaine de caractère résultante à la même structure de ce qu'affiche la commande linux `ls` pour les permissions des fichiers.

Ce module permet aussi d'assurer l'affichage d'aide d'utilisateur, en utilisant la fonction `afficher_help`, l'affichage du help est provoqué quand la commande est appelée sans aucuns arguments.

Remarque – Fonction permission :

Pour le module Display Informations toutes les fonctions sont simple à réaliser, pour récupérer les différents informations d'un fichier, on utilise les deux lignes : `struct stat infofile; stat (filepath, &infofile);` et ceci pour récupérer le `stat` à partir du chemin de fichier donné en paramètre, pour récupérer les différentes informations on utilise les différents champs du `stat` et les différentes fonctions qui interagissent avec la structure `stat` expliqué dans la section « Présentation du système de fichier UNIX (Linux) » du rapport.

La seule fonction qui exige une explication est la fonction `permission` qui permet de donner en résultat une chaine de caractère décrivant les permissions du fichier. Pour cela il faut indiquer que le champ `mode_t` de la structure `stat` est un entier, pour récupérer une permission depuis `mode_t` d'un `stat` d'un fichier, on lui applique un masque en utilisant un « et logique - `& en c`», et on compare le résultat avec le masque, s'il y a égalité

alors on déduit que le fichier en question à la permission correspondante au masque, pour mieux mettre en évidence cela, voici un exemple :

`S_IRUSR` est le masque correspondant au droit de lecture pour l'utilisateur `(Read – User)`, les lignes de codes suivantes permettent de récupérer le mode_t depuis un chemin d'un fichier donné en paramètre « `filepath` » :

- `stat (filepath, &infofile)`
- `mode_t mode = infofile.st_mode;`

Maintenant pour récupérer s'assurer si il y'a la permission de la lecture par l'utilisateur on applique le masque sur mode et on vérifie l'égalité avec le masque, pour le faire on utilise la ligne de code suivante : `if ((mode & S_IRUSR) == S_IRUSR) usr ='r' else usr = '-'`, ainsi le caractère usr à la valeur `'r'` dans le cas ou le fichier du chemin filepath offre la permission de lecture pour l'utilisateur `'-'` sinon.

Jeu d'essais :

Pour une vision plus claire de la commande réalisée, on a choisi de mettre dans le rapport quelques prises d'écrans de la fonction `search` réalisés.

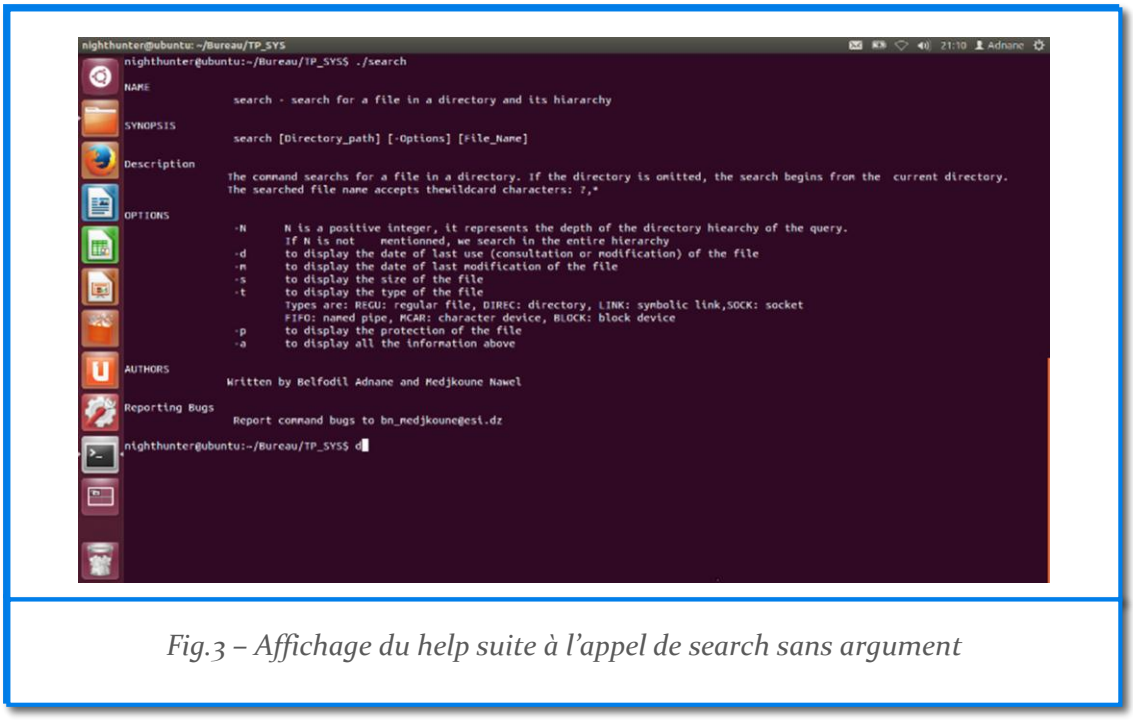


Fig.3 – Affichage du help suite à l'appel de search sans argument

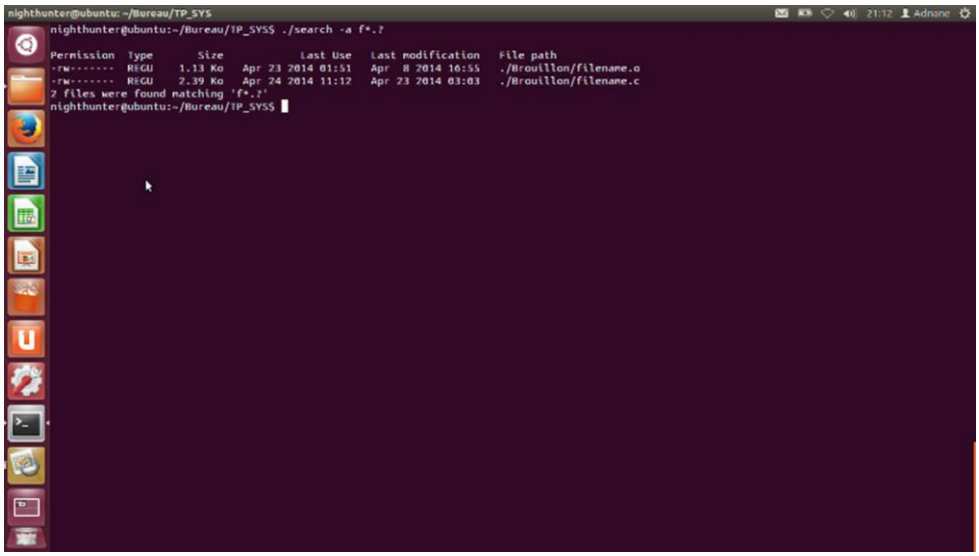


Fig.4 – Affichage des différents informations des fichiers correspondants à la recherche de f. ?*

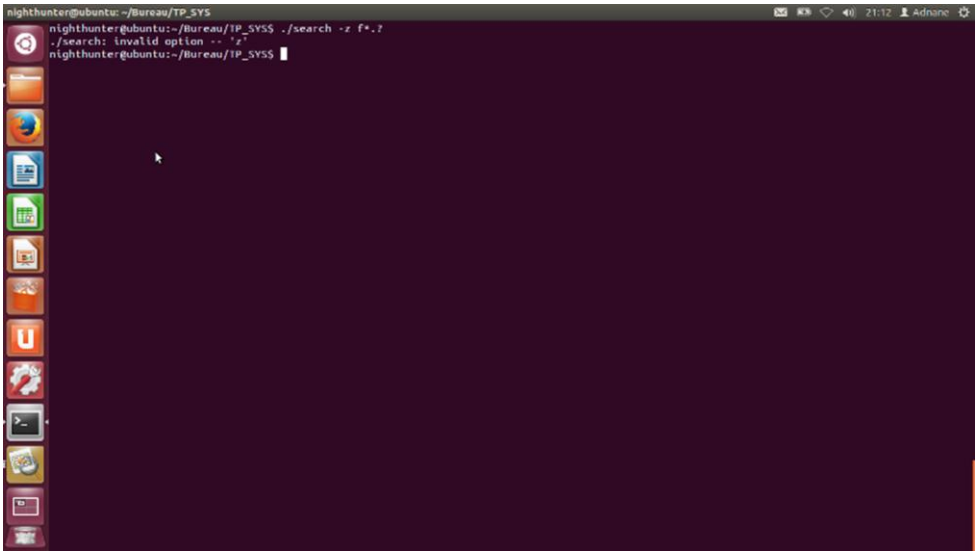


Fig.5 – Affichage d'une erreur suite à l'appel d' une option invalide

Conclusion

Références bibliographique

- http://www.int.impmc.upmc.fr/impmc/Enseignement/ye/informatique/systemes/chap4/441.html.
- http://contrib.xarli.net/secure-gnulinux/fs/.
- http://www.apprenti-developpeur.net/unix-et-os/systeme-de-fichiers-sous-linux/.
- Adnen . A – Institut Supérieur d'Informatique et des Technologies de Communication AU : 2009-2010.
- Documentation officielle d'ubuntu – Ubuntu community - http://doc.ubuntu-fr.org/systeme_de_fichiers.