Table des matières

1 Introduction **4**

2 Le gestionnaire de packages **6**

2.1 Une interface multi pages 6

2.1.1 Le principe 7

2.1.2 Les avantages 7

2.1.3 La gestions des menus 8

2.1.4 La barre d'état 8

2.2 Les différentes pages 8

2.2.1 Répertoire courant 8

2.2.2 Liste des packages 9

2.2.3 Informations du package 10

2.2.4 Buffer de sélection 11

2.3 Recherche et tri 13

2.4 Gestion des tâches longues 14

2.5 Expérience personnelle 15

2.5.1 Apprentissage GTK 15

2.5.2 Organisation d'un programme 16

3 Le module de recherche **18**

3.1 Parcours de l'arborescence du système de fichier . . . 18

3.1.1 Sauvegarde des chemins des sous-répertoires . 19
3.1.2 Pseudo-code 19
3.2 "Recherche dans" 20
3.2.1 principe 20
3.2.2 Fonction is_text_file 20
3.2.3 Reconaissance des mots dans un fichiers texte 21

4 Le module de gestion des extentions 22
4.1 Les types de données 22
4.2 Sauvegarde et récupération des listes 23
4.3 Edition de la base de données 24
4.4 Conclusion 27

5 Le module de gestions des fichiers 29
5.1 La suppression 31
5.1.1 La suppression d'un répertoire 31
5.1.2 La suppression d'un fichier 31
5.2 La copie 32
5.2.1 La création d'un répertoire 32
5.2.2 La copie d'un fichier 32
5.2.3 Le copier/coller et le couper/coller 32
5.3 Le renommage 33
5.4 La gestion des droits 33

6 Une application multi-tâches grâce au thread 35
6.1 Initialisation 36
6.1.1 Les attributs d'un thread 36
6.1.2 La création d'un thread 38
6.2 Le déroulement normal d'un thread 38
6.3 L'annulation d'un thread 38

7 L'explorateur de fichiers **40**
7.1 Les améliorations 40
7.1.1 La gestion des accents pour les noms des fichiers 40
7.1.2 Execution avancée à partir shell avec des options à la manière Posix.2 42
7.1.3 Intégration de plusieurs modules à l'explorateur 43
7.1.4 Extention des signaux pour le listage de fichiers 43
7.2 Un noyau pour une meilleure gestion de l'application 46
7.2.1 Qu'est ce notre noyau 46
7.2.2 Pourquoi 47
7.2.3 Comment ça marche 47
7.3 Les outils utilisés pour la fiabilisation de l'explorateur 48
7.3.1 Débogeur 49
7.3.2 Profileur 49
7.3.3 La gestion de la mémoire alloué 50

8 Le site WEB **51**
8.1 Elaboration du site 51
8.2 Les rapports du projet 52
8.3 Le contenu du site 53

9 Conclusion **54**
9.1 Qui a fait quoi 54
9.2 Pour la troisième soutenance 55
9.3 A la prochaine fois 55

Chapitre 1

Introduction

Avant de vous présenter l'avancement de notre projet pour cette deuxième soutenance, on voulait vous rappeler que le projet ExploreurMaker, consiste en la réalisation d'un explorateur de fichiers sur les systèmes Unix/Linux doté de plusieurs fonctionnalités comme la gestion des packages, recherche de fichiers, d'expressions régulières...

Pour notre deuxième soutenance, nous y présentons les nouveaux aspects de la réalisation de ce projet, c'est-à-dire les différentes tâches faites depuis la première soutenance, mais aussi les améliorations des tâches réalisées pour la première soutenance. Voici donc un plan de ce rapport :

- Le gestionnaire de packages
- Le module de recherche
- Le gestionnaire d'applications
- Le gestionnaire de fichiers
- Une application multi-tâches grâce aux threads
- Les améliorations de l'explorateur de fichiers

Maintenant que nous avons une plus grande maîtrise d'Unix et du C nous commençons à faire des choses plus intéressantes. C'est pourquoi ce projet avance encore et encore jusqu'à tendre à la troisième soutenance vers un produit déjà bien fini. Mais ceci étant il ne faut pas s'arrêter en si bon chemin.

Chapitre 2

Le gestionnaire de packages

Le but fixé pour cette deuxième soutenance était de créer une interface graphique en GTK+ pour le gestionnaire de package. Cet objectif est atteint.

Lors de la soutenance précédente, nous avions présenté une ébauche d'interface. C'était une fenêtre qui affichait la liste des packages installés sur la machine avec leurs fichiers respectifs sous forme de liste et d'arborescence. En fait cette fenêtre avait principalement pour but de montrer l'avancement de la base de données qui à été implémentée lors de la première soutenance. Pour cette soutenance, l'interface a été entièrement réorganisée.

2.1 Une interface multi pages

Il y a tellement d'informations à afficher sur les packages qu'il fallait une interface plus puissante. Nous avons donc crée une fenêtre qui peut contenir plusieurs pages différentes et les afficher à tour de rôle. Chaque page possède des fonctions précises ce qui permet de

mieux répartir les tâches à l'intérieur du logiciel. Nous avons aussi rajouté quelques composants utiles comme une barre d'état et une barre de progression pour indiquer l'avancement d'une tâche longue.

2.1.1 Le principe

On crée une fenêtre GTK qui va servir de base et contenir tous les objets communs à toutes les pages. Ces objets sont : la barre de menus, la barre d'état et la barre de progression. A l'intérieur de la fenêtre, on ajoute 5 boites (on a 5 pages à gérer mais on peut tous aussi bien en mettre n). Chacune de ces boites va contenir une page. Au départ elles sont invisibles. Par la suite, elles s'afficheront à tour de rôle (toujours une à la fois) selon les actions de l'utilisateur.

2.1.2 Les avantages

Le première avantage est que ici, on ne s'occupe que d'afficher des boites. On peut ensuite se concentrer sur la création et le fonctionnement d'une page sans avoir à se préoccuper de la fenêtre de base ou des autres pages.

Le deuxième avantage est au niveau de la gestions des ressources système. Au départ, les pages ne sont pas crées en mémoire (on a juste les boites). Les pages ne sont crées qu'à partir du moment ou une page est requise par l'utilisateur. De plus lorsque l'on a plus besoin d'une page, on peut la fermer. Elle sera alors détruite.

Ce principe permet donc de charger en mémoire uniquement les composants dont on a besoin.De plus le temps de chargement de l'application au départ est réduit puisque l'on construit que la fenêtre

de base.

2.1.3 La gestions des menus

La barre de menus est commune à toutes les pages. Les titres des menus ne changent pas. Certaines actions sont communes à toutes les pages. Soit elles sont indépendantes de la page courante (comme par exemple ouvrir une nouvelle page), soient elles s'adaptent à la page courante (comme par exemple rechercher une chaîne de caractères). D'autres actions sont spécifiques à certaines pages et ne fonctionnent que lorsque la page appropriée est affichée.

2.1.4 La barre d'état

La barre d'état est également commune à toutes les pages. Elle affiche des informations sur la page courante. Elle peut enregistrer autant de messages différents qu'il n'y a de pages. Cela signifie que une fois que l'on a définit un message pour une page il reste en mémoire. On peut ensuite changer de page sans se préoccuper de mettre à jour les messages puisque la barre le fait automatiquement.

Nous allons maintenant détailler les différentes pages que nous avons crée ainsi que leurs fonctionnalités.

2.2 Les différentes pages

2.2.1 Répertoire courant

Cette page affiche le contenu d'un répertoire. C'est un mini explorateur de fichiers. Il est destiné par la suite à repérer les fichiers

.TGZ afin de les ouvrir. Cette page est constituée d'une une liste de trois colonnes qui affiche pour chaque fichier du répertoire :

- le nom
- la taille en octets
- un commentaire qui est fonction de l'extension du fichier

Cette page possède également deux champs d'entrée texte. Le premier champ permet de sélectionner un répertoire différent. Le deuxième permet de définir un masque pour afficher uniquement les fichiers d'un certain type. Par exemple *.TGZ. Pour cela on réutilise l'algorithme du pattern matching qui avait été mis au point lors de la soutenance précédente.

2.2.2 Liste des packages

Cette page affiche la liste des packages actuellement installés sur la machine. Cette page est relativement simple puisqu'elle n'est constituée que d'une liste qui affiche pour chaque package :

- le nom
- la catégorie (éditeur, composant Xwindow, archiveur, www, etc.)
- une descriptions courte (une phrase maximum)

En fait cette page correspond un peu à la commande pkg info. La nouveauté est que ici, on affiche en plus la catégorie du package. Le but de cette page est de pouvoir visualiser rapidement la liste des packages présents ainsi que leur description.

En effet certains nom de packages ne sont pas très explicites. La catégorie et la descriptions courte permettent alors de se faire

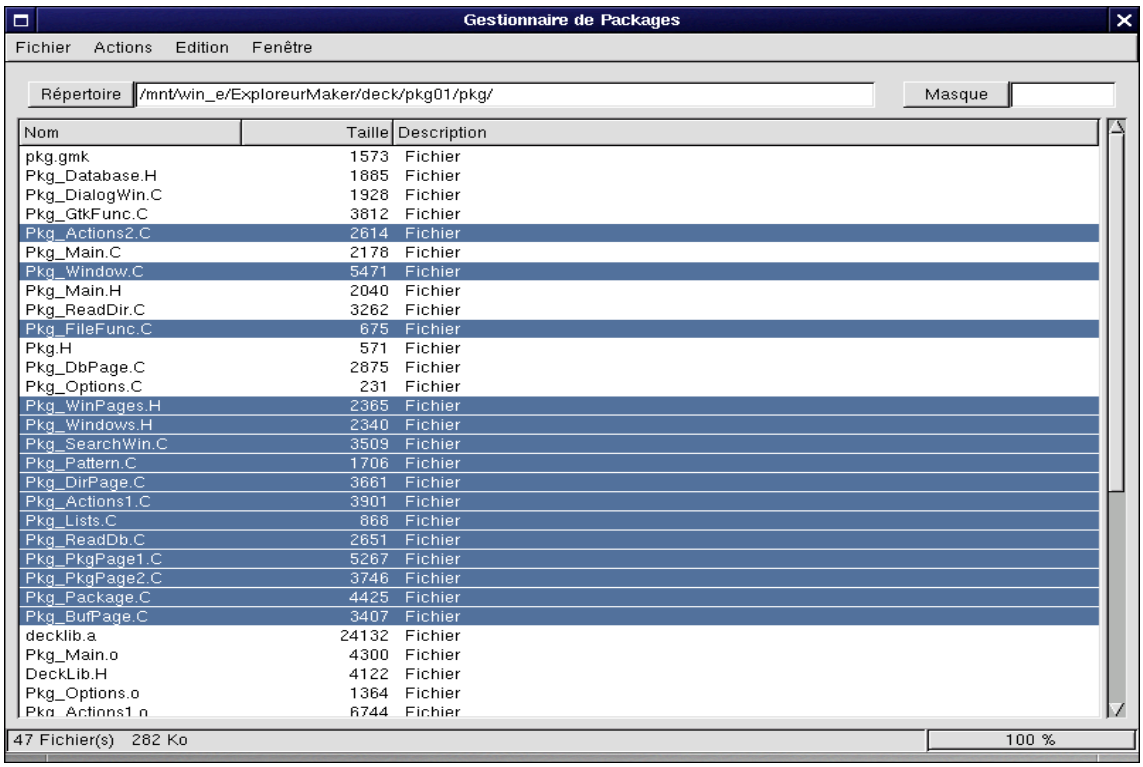


FIG. 2.1 – *La page affichant le répertoire courant*

rapidement une idée. A partir de cette page, on peut sélectionner un package afin de l'éditer pour avoir plus d'informations. On accède alors la page Package.

2.2.3 Informations du package

Les informations concernant un package on étés divisées sur deux pages.

La première page regroupe les informations générales :

– le nom

- la description courte
- la catégorie
- la description longue
- le nombre de fichiers
- la taille totale
- la liste des packages qui sont requis.
- la liste des packages qui sont dépendants.

La liste des packages requis correspond à la liste des packages qui doivent êtres installés au préalable pour que ce package puisse fonctionner. Ce sont généralement des bibliothèques dynamiques ou des composants partagés. La listes des packages dépendants correspond à la liste des packages qui ont besoin de la présence de ce package pour fonctionner. Par exemple, pour un composant partagé, on aura la liste des tous les packages qui utilisent ce composant.

Ces deux listes permettent de visualiser les relations établies entre les packages. Lorsque l'on installe un package, on peut vérifier si il a besoin d'autres composants. Inversement, lorsque l'on veut désinstaller un package, on peut vérifier que sa suppression ne perturbera pas les autres packages.

La deuxième page affiche une arborescence correspondant aux répertoires du packages ainsi qu'une liste qui indique les fichiers présents dans le répertoire sélectionné. Cette page à la même forme que l'explorateur de fichiers. Au lieu d'afficher les répertoires du disque dur ou d'un quelconque périphérique, on affiche les fichiers et les répertoires qui appartiennent au package.

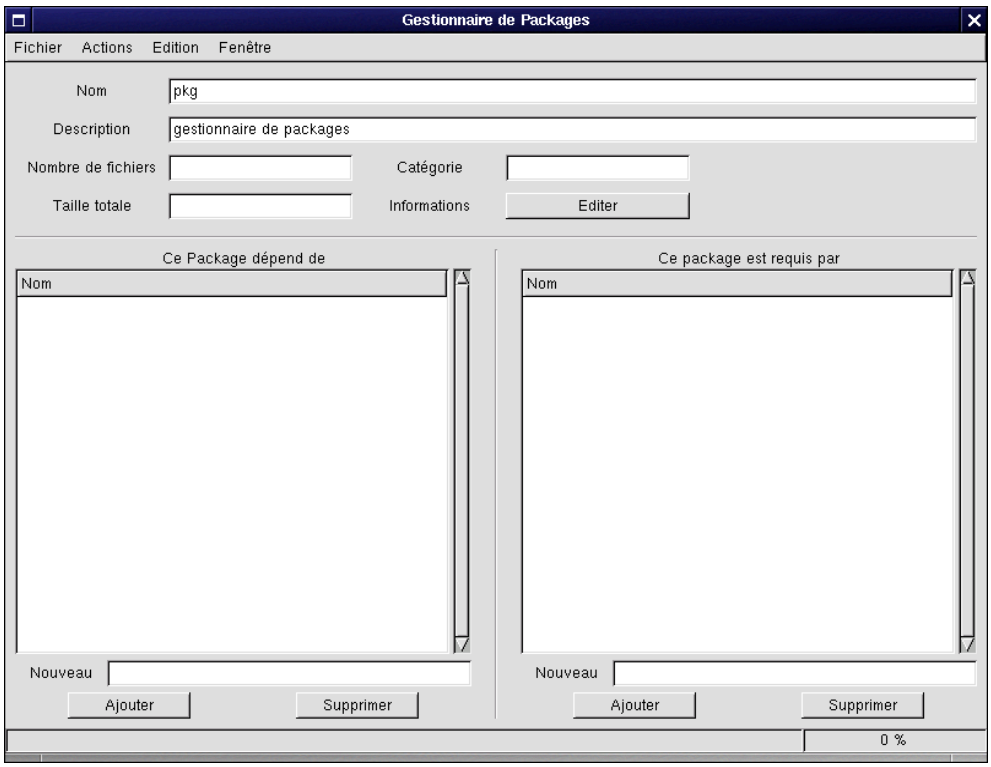


FIG. 2.2 – *Les informations concernant un package*

2.2.4 Buffer de sélection

Le buffer de sélection fait office de presse papier. Il s'agit d'une liste qui enregistre en mémoire les sélections de fichiers de l'utilisateur. Pour enregistrer une sélection, on sélectionne des fichiers puis on fait Menu Edition Copier. Jusque là classique. Habituellement, lorsque l'on fait plusieurs fois Copier à la suite, on efface la sélection précédente.

Avec ce buffer de sélection, on peut enregistrer plusieurs sélections, pour être plus précis, on peut sélectionner un ensemble de fichiers en plusieurs fois.On peut également visualiser cette sélections

en ouvrant la page buffer. La page affiche alors tous les fichiers que l'utilisateur à mis en mémoire. L'utilisateur peut ensuite sélectionner des fichiers à l'intérieur même du buffer pour ne Coller qu'une partie ou tout le contenu du buffer.

Ce système est destiné à faciliter la gestion des sélections. Sont utilisation principale dans le gestionnaire de package est de pouvoir sélectionner un ensemble de fichiers ou de répertoires à partir de la fenêtre Répertoire pour ensuite coller cette sélection dans un package.

2.3 Recherche et tri

Comme vous pouvez le constater, toutes les pages contiennent au moins une liste. Ces listes peuvent parfois devenir très longues. Par exemple à EPITA, il y a plus de 250 packages sur chaque machine. Pour faciliter la recherche d'une informations nous avons mis au point deux méthodes.

Premièrement, toutes les listes peuvent êtres triées. Il suffit pour cela de cliquer sur le titre de la colonne que l'on veut trier.

Deuxièmement on dispose d'une petite fenêtre de recherche que l'on peut ouvrir à tout moment. Cette fenêtre contient un champ pour rentrer le mot clé que l'on souhaiter rechercher dans la page courante.

On dispose de trois modes de recherche :

- rechercher un mot clé
- rechercher un mot clé en respectant les majuscules et les minuscules

– rechercher une expression régulière en utilisant le pattern matching.

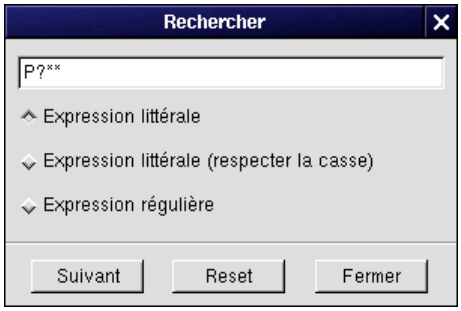


FIG. 2.3 – *Pour effectuer une recherche dans le nom ou la description des packages*

Lorsqu'on lance la recherche, la liste se positionne et sélectionne l'élément recherché si il existe. Sinon on affiche une boite de dialogue indiquant une recherche négative. La recherche s'arrête automatiquement lorsque la première occurrence de l'élément est trouvé. Pour chercher d'autres occurrences, on clique sur suivant et on relance la recherche à partir de l'élément courant.

2.4 Gestion des tâches longues

Certaines tâches comme par exemple le chargement de la liste des packages installés ne s'effectuent pas en 1/10ème de secondes. En effet il faut parcourir un répertoire, ouvrir et lire un certain nombre de fichiers. Cela peut prendre plusieurs secondes notamment pour les machines qui ont beaucoup de packages.

Si on lance une fonction qui tourne en boucle jusqu'à temps

d'avoir fini sa tâche, c'est toute l'application qui reste bloquée pendant ce temps. Cela peut être désagréable pour l'utilisateur qui a l'impression d'avoir une application qui "rame".

Pour remédier à cela, on lance les tâches longues en tâche de fond. Une application GTK peut gérer une ou plusieurs fonctions qu'elle relance en permanence. Ces fonction tournent en boucle sauf que à chaque itération, elle redonnent la main à l'application GTK. En fait, ces fonctions sont exécutées en permanence lorsque l'application n'a rien à faire. On se sert donc de se principe pour exécuter une tâche longue sans bloquer l'application.

On divise la fonction initiale en 3 fonctions :

- Une fonction qui démarre la tâche
- Une fonction qui constitue la boucle principale (mais qui ne fait en réalité qu'une seule itération)
- Une fonction qui termine la tâche

Ces trois fonctions seront ensuite appelées par une 4ème fonction qui déterminera à chaque appel laquelle des trois fonction il faut lancer en fonction du résultat de l'appel précédent.

Ce principe à également pour avantage, le fait que l'utilisateur puisse interrompre une tâche longue même si elle n'est pas terminée. Pour cela, on lance directement la fonction 3 qui termine la tâche (cette fonction étant indépendante du nombre d'itérations effectué par la fonction 2).

2.5 Expérience personnelle

2.5.1 Apprentissage GTK

En créant les différentes pages du gestionnaire d'application, j'ai pu me familiariser avec de nombreux objets GTK+. J'ai aussi pu remarquer que certaines lignes de code se répètent fréquemment lorsque l'on crée plusieurs fois les mêmes objets. J'ai donc implémenté des petites fonctions pour chaque objets GTK que j'utilise en permanence, notamment les boites, les labels, les entrées, les listes etc.

Ces fonctions permettent de recréer facilement et plus rapidement les objets GTK avec les paramètres les plus courants. Cela évite également de recopier sans arrêt des paragraphes de code qui ont seulement 2 ou 3 variantes. De plus ces fonctions sont indépendantes du gestionnaire de packages et peuvent êtres réutilisées dans n'importe quel autre programme.

2.5.2 Organisation d'un programme

Depuis que je suis à EPITA, et que j'ai commencé à faire des programmes consistants notamment les projets, je cherche comment organiser correctement mes programmes. En effet l'année dernière en SUP, il m'est arrivé à deux reprises de réorganiser l'intégralité du code de mon projet afin de pouvoir avancer correctement. J'aimerais ne pas avoir à le refaire cette année. Par conséquent cette année, j'ai mis au point la méthode suivante et je l'applique depuis le début de l'année.

Toutes les fonctions qui ne sont pas spécifique à l'application que je suis en train de programmer sont implémentées dans le cas gé-

néral de manière à être réutilisables dans d'autres projets. Ce sont par exemple les fonctions GTK que j'ai présenté dans le paragraphe précédent.

Le programme doit être décomposé en deux grandes parties. La première partie est la partie principale, celle qui gère l'ensemble du programme et qui coordonne ses différents modules. La deuxième partie regroupe l'ensemble des modules du programmes.

La partie principale du programme est divisée en 3 parties. Une partie initialisation qui doit mettre en marche le programme et initialiser tous ses modules. Une partie boucle principale. C'est une fonction qui boucle en permanence. Elle appelle les fonctions des différents modules et analyse leurs résultats. Elle contrôle le déroulement générale de l'application. Et Enfin une partie terminaison du programme qui libère les ressources système et arrête correctement tous les composants du programme.

Les modules doivent êtres indépendant les uns des autres. C'est à dire que l'ajout ou la suppression d'un module ne doit pas entraîner une restructuration complète du programme. Cette année, je programme en C++. Chaque module de mon programme correspond à une classe. Chaque classe est prévue pour fonctionner de manière la plus autonome possible, c'est à dire en évitant d'utiliser les variables globales. Au niveau de la compilation séparé, chaque fichier .C correspond à une classe.

Pour conclure là dessus, je ne sais pas si c'est comme ca qu'il faut faire, mais il me semble que c'est une bonne méthode. En tous cas pour l'instant (et depuis l'année dernière que je cherche) je n'ai pas

trouvé mieux.

Chapitre 3

Le module de recherche

Pour la recherche on doit pouvoir chercher un nom de fichier ou répertoire, cependant nous avons jugé pratique d'intégré aussi le mode "recherche dans" pour que l'utilisateur puisse trouver le nom du fichier (texte) qui contient le mot qu'il cherche.

3.1 Parcours de l'arborescence du système de fichier

Pour faire une recherche à partir d'un répertoire il faut biensur parcourir toutes les entrées de celui-ci, puis récursivement dans ses sous-répertoires. Pour cela, il faut pouvoir sauvegarder les chemins et noms des sous-répertoires pour pouvoir les ouvrir par la suite et y relancer la recherche. On a donc besoin des types "struct dirent" et "DIR" pour le parcours des entrées et de "struct stat" pour tester si l'entrée en cours est un répertoire.

Ensuite il faut adapter ce principe suivant l'utilisation dont on a besoin. Pour la recherche par exemple il suffit de comparer chaque

entrée parcourue avec le pattern.

3.1.1 Sauvegarde des chemins des sous-répertoires

Pour sauvegarder les chemins des sous-répetoires j'utilise une liste chaînée avec ajout en queue et suppression en tête (donc utilisée comme une file FIFO). L'élement que contient un noeud de la liste est de type "char *" car un noeud représente le chemin complet d'une entrée ayant été identifiée comme répertoire.

Le Parcours de l'arborescence s'arrête lorsque la liste est vide (c'est à dire lorsque tous les sous-répertoires ont été parcourus).

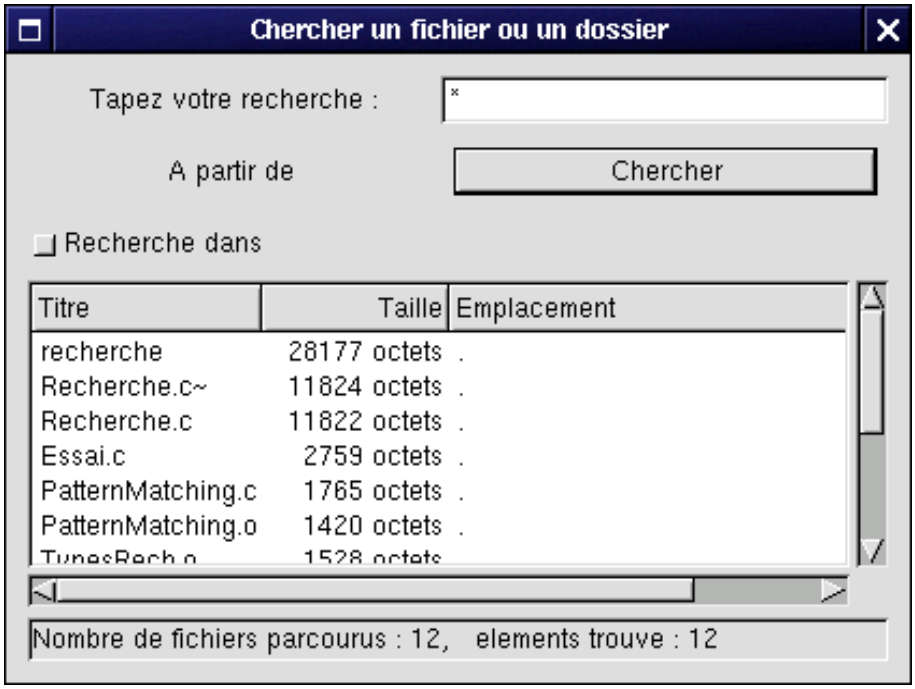
3.1.2 Pseudo-code

```
faire
    si la liste est non vide alors
        on defile un chemin complet
    on ouvre le repertoire désigné par ce chemin
    fin si
    tant qu'il y a des entrées
        si c'est un repertoire alors
        on enfile son chemin complet dans la liste
    fin si
    entrée suivante
    fin tant que
tant que la liste est non vide
```

3.2 "Recherche dans"

3.2.1 principe

Pour la "recherche dans" on applique le même principe de parcours de l'arborescence mais au lieu de comparer le pattern avec le nom de l'entrée courante on doit l'ouvrir et parcourir les mots qu'elle contient si c'est un fichier texte. J'ai donc fait une fonction is_text_file pour savoir si l'entrée courante est un fichier texte ou non.



3.2.2 Fonction is_text_file

Cette fonction sert donc à reconnaître les fichiers texte. Elle prend en paramètre le nom du fichier et renvoit 1 si c'est un fichier texte 0

sinon.

Cette fonction fait une statistique : Pour cela on lit les 100 premiers caractères (ou jusqu'à EOF s'il y en a moins) qui ne sont ni des '\n' ni des espaces, une variable compte les nombre de caractères lus au total et un autre compte le nombre de caractères alphanumériques. Si le rapport est inférieur à 0.80 alors on considère que ce n'est pas un fichier texte et la fonction renvoie 0 sinon elle envoie 1.

3.2.3 Reconaissance des mots dans un fichiers texte

Il faut parcourir tous les caractères du fichier jusqu'à EOF ou la première occurrence du pattern. Un mot commence à être identifié à partir du premier caractère alphanumérique qui suit le mot précédent ou du debut du fichier. Ensuite on sauvegarde chaque caractère jusqu'à tomber sur un espace ou sur '\n' ou EOF ce qui arrête l'identification du mot. Puis on compare le mot obtenu avec le pattern de la recherche. Comme la "recherche dans" se fait plutôt sur des textes que du code on à adapté la reconnaissance de mot pour les textes c'est à dire que si le dernier caractère sauvegardé n'est pas alphanumérique alors on l'efface ce qui résoud les problèmes de ponctuation dans un texte (comme le point ou la virgule).

Chapitre 4

Le module de gestion des extentions

L'explorateur ExplorerMaker doit permettre d'éxécuter, comme sous Window, une application juste en cliquant sur un fichier. Ainsi il doit reconnaître le type du fichier pour déterminer le(s) programme(s) à lancer. Il faut donc stocker ces informations sous forme de fichiers pour pouvoir les récupérer à chaque démarrage de l'explorateur. La gestion de cette base de données sera faite par l'utilisateur, il déterminera quelles applications doivent être démarrées quand il clique sur un fichier de type connu. Il remplira donc la base de données comme il le souhaite.

4.1 Les types de données

L'explorateur utilise deux listes chaînées. Une liste déterminant quel type de fichier (image, texte, ...) doit être associé à chaque extension. L'autre liste déterminant quel(s) type(s) de fichiers peut ouvrir chaque application.

Pour notre utilisation, nous n'avons besoin de seulement trois fonctions sur ces listes. Il en faut deux permettant d'insérer un élément dans et de supprimer un élément dans une liste. Puis il faut une autre fonction pour détruire une liste, libérant ainsi la mémoire.

Les éléments des listes sont classés par ordre alphabétique. La liste déterminant le type d'une extension est triée suivant les extensions. Et la liste déterminant le type de fichiers utilisés par les applications est triée suivant les type. On retrouve ainsi facilement toutes les applications possibles pour un certain type de fichier, et donc pour une certaine extension.

Les trois fonctions ainsi définies sont très simples. La difficulté a plutôt été de trouver quel était la meilleure façon de faire dans notre cas. Au début, il y avait une seule liste associant chaque extension à un type et à une application. Mais dans l'utilisation, cette liste devient rapidement imposante. Si un type regroupe plusieurs extensions et également plusieurs programmes (ce qui est souvant le cas), on va beaucoup répéter une même application, or on est obligé de stocker le chemin complet d'un application, ce qui prendrait donc beaucoup de mémoire.

4.2 Sauvegarde et récupération des listes

L'explorateur doit donc récupérer ces liste à chaque démarrage. De plus il faut pouvoir les sauvegarder quand l'utilisateur les modifient. Les deux listes chainées ont donc deux fonctions permettant ceci prenant en paramètre la liste à sauvegarder, et le nom du fichier source ou de destination.

4.3 Edition de la base de données

Pour le moment la base de données n'est pas encore liée à l'explorateur. Une fois le type de donnée définit, il fallait un outil pour la modifiée. Il y a alors une interface faite avec GTK pour cela. Elle est composée d'un tableau (de type GtkCList) à trois colonnes indiquant à chaque ligne une extension, son type et les applications qui lui sont associées. Cette liste est bien évidemment chargée à partir des deux listes chaînées définies précédemment.

Pour le moment la base de données n'est pas encore liée à l'explorateur. Une fois le type de donnée définit, il fallait un outil pour la modifiée. Il y a alors une interface faite avec GTK pour cela. Elle est composée d'un tableau (de type GtkCList) à trois colonnes indiquant à chaque ligne une extension, son type et les applications qui lui sont associées. Cette liste est bien évidemment chargée à partir des deux listes chaînées définies précédemment. Au démarrage de la fenêtre d'édition de la base de donnée, le programme parcourt la liste des extensions, récupérant le type de l'extension pour le mettre dans la deuxième colonne. Et pour chaque extension, il faut aussi récupérer les applications liées par rapport à son type. Le programme parcourt donc la liste des applications, triée alphabétiquement par type. L'ensemble des programmes permettant de lancer une extension est affiché dans la troisième colonne. Une fonction extrait uniquement le nom du programme sans le chemin pour plus de lisibilité. Les colonnes étant limitées en nombre de caractères, on marque trois points ('...') s'il y'a trop d'applications associées (cf figure suivante).

Un texte (GtkLabel), sous le tableau, indique les informations extraites de la ligne séléctionnée par l'utilisateur pour plus de lisibilité.

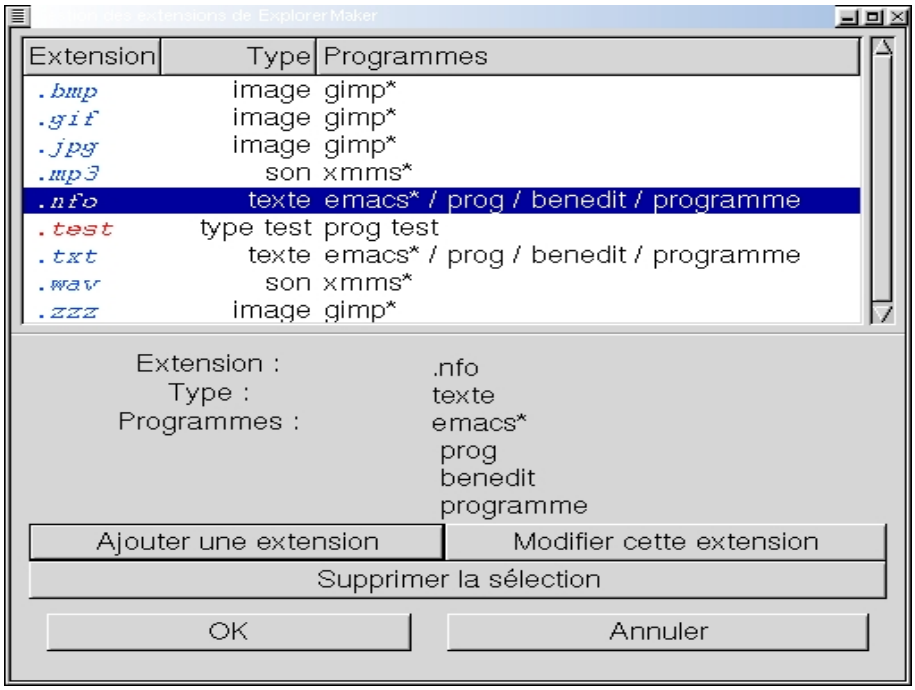


FIG. 4.1 – *Les extensions en rouge sont celles qui ont été ajoutées. Les noms de programmes suivis d'une astérisque désigne l'application par défaut associée au type de fichier.*

A chaque fois qu'il change de ligne, ce texte est actualisé. Il est également possible, en cliquant en haut d'une colonne de retriée le tableau suivant la colonne cliquée. Ainsi on peut classer le tableau par type de fichiers, et non par extension si on le souhaite. Les boutons ajou-

ter et modifier une extension démarrent une nouvelle fenêtre pour éditer les propriétés d'une extension précise. S'il s'agit d'ajout, tous les champs sont éditables. On peut définir une nouvelle extension. A celle-ci on peut choisir son type parmis ceux connus ou en ajouter un. En choisissant un type, l'ensemble des applications associées est éga-

lement modifié, celui-ci dépendant du type de fichier. Si l'utilisateur veut ajouter ou modifier une application associée à cette extension (et donc à ce type de fichier), il le peut également.

S'il s'agit de modifier une extension préexistante, on ne peut éditer le nom de l'extension, mais on peut agir sur le type et les applications associées comme précédemment. La liste des type est affichée par un menu défilant. Et la liste des programmes est affichées sous forme de liste éditable grace aux boutons associés (cf figure suivante).

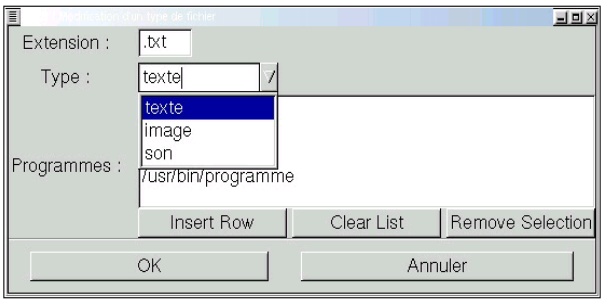


FIG. 4.2 – *Identification d'un type de fichier*

Quand on change le type de fichier, la liste des applications est actualisée en fonction de ce type (à condition qu'il soit connu dans la base de donnée).

Le bouton pour insérer dans une application dans la liste des applications ouvre une fenêtre de sélection de fichiers GTK. Ainsi on choisit dans l'arborescence l'application à ajouter.

En cliquant sur 'OK', l'utilisateur sauvegarde l'extension et donc modifie les listes chargées dans la fenêtre de gestion des extensions. S'il clique sur annuler, aucune modification n'est prise en compte.

Pour le moment, quelques problèmes avec GTK empêche la sauvegarde d'une extension. Il est possible de supprimer simplement une ou plusieurs extensions. Le bouton 'Supprimer la sélection' supprime toutes les lignes séléctionnées et les données qu'elles contiennent dans la liste des extensions.

En cas d'erreur d'utilisation, le bouton 'Annuler' permet de fermer la fenêtre sans modifier quoiquecesoit. En revanche le bouton 'OK' sauvegarde les nouvelles listes.

4.4 Conclusion

La gestion des extensions est pour le moment encore indépendante de l'explorateur. Mais son interface est terminée. Elle doit encore être optimisée, suivant le besoin que l'on en aura par la suite. L'important était de se mettre tous d'accord sur un type de donnée et sa gestion via les fichiers. L'interface a été très lourde à implémenter avec GTK,

c'est ce qui prend le plus de temps. Elle n'est pas tout à fait finit, certains alignements de textes restent presque incompréhensibles pour nous même. Pour la prochaine soutenance, il s'agira de finir ceci, de permettre l'ajout d'un type de fichier (ce qui ne devrait plus prendre beacoup de temps, les algorithmes étant fait) et de relier cette partie à l'explorateur et d'optimiser certaines fonctions. Grace à la gestion

des packages implémentée, nous pouvons savoir quelles applications sont présentes sur l'ordinateur sur lequel est installé ExplorerMaker. Ainsi on pourrait imaginer qu'à l'installation de l'explorateur, on vérifie si certains programmes courants sont installés (emacs, gimp) pour les ajouter directement dans la base de données avec les types que

l'on connaît. On s'est aperçu qu'à partir des informations fournises avec un package, on pouvait connaître le type des fichiers qu'il l'application contenue pouvait exécuter ; mais ce serait trop lourd pour l'utilisateur si toutes les applications répertoriées étaient ajouter à la base de données. On réfléchit donc encore pour le moment au contenu initial de la base de données. Evidemment l'éditeur de texte que l'on a conçu sera intégrer initialement.

Chapitre 5

Le module de gestions des fichiers

Ce module est destiné à la gestion des emplacements des fichiers et de leurs caractèristiques. Cette gestion peut être très fasticieuse et peu maniable à partir d'un shell. Il y a, en fait, deux grands ensembles de fichiers qui peuvent être traités sèparement : les répertoires et les autres fichiers (fichiers réguliers, liens, ...). On remarque dès lors qu'une fois les méthodes implèmentées sur l'ensemble des fichiers, on les répète pour le contenu du répertoire afin de l'étendre à tous les types de fichiers.

Ainsi on obtient différentes opérations sur un fichier :

- Suppression
- Copie vers un répertoire source
- Copier en mémoire (pour être après coller)
- Couper en mémoire (pour être après coller)
- Coller (ce qui enregistrer par le Coller ou le Couper)
- Déplacer vers un répertoire source

- Renommer le nom du fichier
- Modifier les droits d'accès
- Création d'un nouveau fichier en utilisant les deux dernières méthodes ou par défaut

Puis on peut aussi sur un répertoire :
- Suppression du répertoire et de son contenu
- Copie du répertoire et de son contenu
- Copier du répertoire et de son contenu
- Couper du répertoire et de son contenu
- Coller du répertoire et de son contenu
- Déplacer du répertoire et de son contenu
- Renommer le répertoire
- Modifier les droits d'accés du répertoire et/ou de son contenu
- Création d'un répertoire en utilisant les deux dernières méthodes ou par défaut

Il y a les mêmes méthodes entre les deux ensembles, à l'exception près que pour les répertoires on les parcours et on applique la méthode demandée à tous ses fichiers. Ensuite, on poursuit dans ses sous répertoires jusqu'à avoir traité tous les fichiers de tous les sous répertoires.

En fait il y a seulement deux méthodes de base pour la plupart des méthodes : la copie et la suppression. Cependant la bibliothèque du langage C nous fournit des fonctions réalisant les différentes opérations énumérées précédement. Donc dans la plupart des opérations, notre travail consiste, d'une part à gérer toutes les possibilités (surtout les erreurs pour qu'il quitte proprement ce module afin d'assurer

la stabilité du logiciel), et de décoder les informations de notre listage de fichiers pour les fournir à notre module de gestion de fichier. Ceci est aisément réalisé car il suffit de concaténer la chaîne du label indiquant le répertoire listé avec le nom du fichier traité.

5.1 La suppression

Les commandes d'Unix font la différence entre la suppression d'un répertoire (*rmdir*) et la suppression d'un fichier (*rm*). Donc en fait pour nous les deux méthodes décrites ci dessous sont des méthodes privées. C'est à dire que ce sera un mécanisme interne du module afin de faire la suppression d'un répertoire et implicitement de tous son contenu.

5.1.1 La suppression d'un répertoire

Il existe la fonction de la librairie *rmdir()* afin de supprimer le répertoire. En effet cette fonction ne peut qu'être une sous routine pour la suppression globale, car elle renvoit une erreur si le répertoire n'est pas vide. Donc cela suppose la destruction des fichiers qu'il contient.

5.1.2 La suppression d'un fichier

La suppression d'un fichier ce fait simplement pas la fonction *unlink()*. Dans notre cas même si c'est un lien on ne supprimera le lien et pas ce qui est pointé par le lien.

Ces deux fonctions gère automatiquement les droits et la propriété du fichiers. C'est à dire que si l'utilisateur n'a ni les droits ni les

fichiers, il ne peut donc pas les supprimer : ce qui est très prèférable.

5.2 La copie

Là encore les deux premières méthodes sont des méthodes privées servant surtout le copier/coller et le couper/coller.

5.2.1 La création d'un répertoire

Elle se fait simplement avec la fonction *mkdir()*. Il est aussi possible de communiquer à cette fonction les droits du répertoire créer.

5.2.2 La copie d'un fichier

Pour la copie, on crée un fichier en faisant un appel à l'appel système *open()*. On ouvre aussi la source. On fait une copie de bloc à bloc jusqu'à ce qu'on pointe à fin du fichier source. La taille des blocs est détermine par le champs de la structure *stat* qui donne le flux optimale pour la gestion d'un fichier.

5.2.3 Le copier/coller et le couper/coller

Pour garder le lien avec les objets on peut dire que le couper/coller hérite du copier/coller.

Le copier/coller se fait en deux étape :

- Le copier
- Le coller

Le copier se résume à inscrire les fichiers et les répertoires à copier. Pour limiter les calculs de cette étape on inscrit juste les fichiers et les

répertoires sélectionnés. On ne se charge pas encore d'inscrire dans le fichier le contenu des répertoires. Ceci nous facilitera la tâche d'une part pour la programmation des threads car ceci evite en générale la création d'un thread pour le copier. Ainsi, le thread du coller n'aura pas à attendre la mort du thread du copier. Mais aussi il n'y aura pas de demande de coller pendant l'exécution du copier (même si cela est bien controlé par le noyau, on verra pourquoi dans la section consacré au noyau).

Le couper/coller effectue exactement les mêmes opérations mais il supprime les fichiers à la fin de ces opérations avec les méthodes vues précédement.

5.3 Le renommage

Là encore pour renommer un fichier la librarie C nous offre une fonction pour réaliser ce changement : *rename()*. On affiche dans un label le fichier traité, la boite de saisie est initialisée avec le nom actuel du fichier. Pour annuler la modification, il suffit juste d'annuler avec le bouton. En cas de validation on teste seulement si le nouveau nom est disponible c'est à dire on vérifié qu'il n'a aucun fichier qui se nomme ainsi dans le répertoire.

5.4 La gestion des droits

Pour la définition des droit il existe bien sur une fonction : *chmod()*. Il faut être le propriètaire du fichier. De toute façon on verra plus que si l'utilisateur n'est pas propriétaire de tous les fichiers sélectionnés il ne pourra pas atteindre ce menu. Même système que pour le renom-

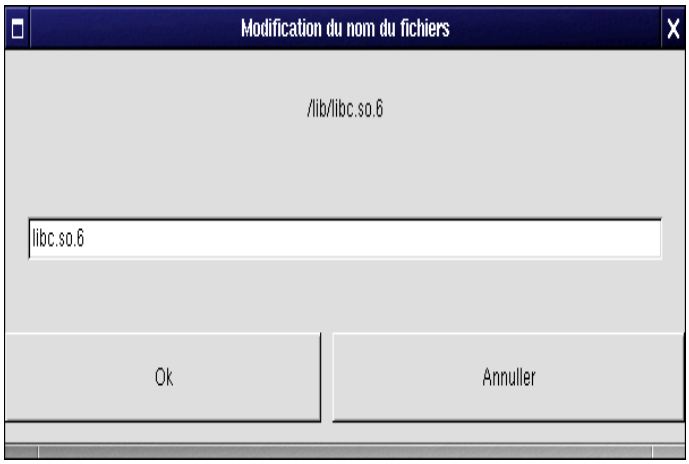


FIG. 5.1 – *Le renommage d'un fichier sélectionné*

mage, on initialise avec les droits du premier fichier sélectionné (dans ce cas on ne considère par le type du fichier) et l'utilisateur modifit à sa guise. Pour les répertoires, leur contenu en est aussi affecté.

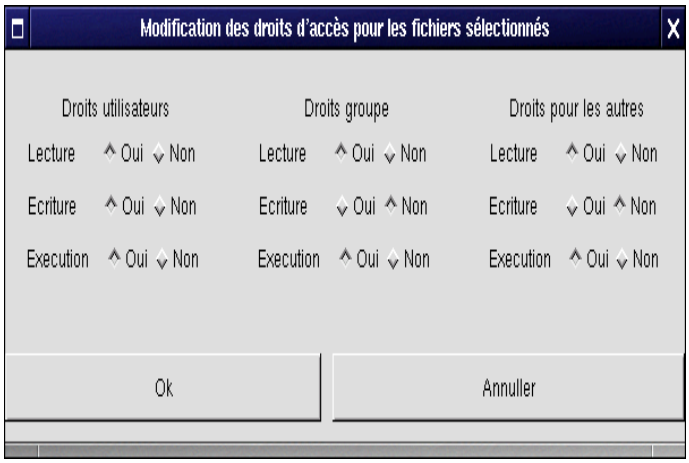


FIG. 5.2 – *L'édition des droits des fichiers sélectionnés*

Chapitre 6

Une application multi-tâches grâce au thread

A la première soutenance nous avons déjà bien étayé pourquoi nous avons choisi les threads plutôt que les processus pour avoir une application mulitâches. En résumé nous les avons choisi pour :

- Leur performance (demande moins de ressources)
- Leur simplicité d'utilisation
- Leur portabilité (fonctionnement indépendant du noyau contrairement au processus)

On distingue trois moment lors de l'implémentation :

- l'initialisation et définition des attributs du thread
- leur déroulement et leur destruction automatique
- leur annulation pendant leur déroulement

Avec les threads, il n'y a pas de hiéarchie (contrairement au processus) mais il y a un toujours dans chaque programme un thread principal. Dans notre programme C, la fonction qu'éxécute ce thread

est la fonction *main()*. Chaque thread d'un programme est comparable à un sous programme. Dans notre cas ce sera encore plus simple car, on effectura une fonction (comme la copie, la suppression ou la recherche...) sans que l'explorateur n'ait à s'en soucier de l'état de ces opérations. Donc on négligera les retours de valeur du thread car ce thread sera capable de gère les erreurs tout seul (si entre un copier et un coller on supprime les fichiers mémorisés, ce sera le thread qui gèra cette erreur et pas le thread principale).

6.1 Initialisation

Chaque thread a des attributs spécifiques déterminés lors de son initialisation et ne peut être modifié lors de son déroulement. Donc la phase d'initialisation consiste en deux étapes :

- La définition des attributs du thread à l'aide d'une structure de la librairie **pthread.h**
- La création avec les attributs et la fonction que le thread doit éxécuter

6.1.1 Les attributs d'un thread

Par exemple comme on a dit plus haut, on n'avait pas besoin qu'un thread ait la valeur de retour d'un autre thread. Ainsi dans les attributs on peut indiquer au thread de se libérer tout seul et tout suite.

On avait exposé dans le rapport de la première soutenance les trois types d'ordonnancements possibles pour les threads et les processus, ainsi on peut choisir le type ordonnancement des threads :

- Ordonnancement classique

- Ordonnancement temps-réel avec l'algo Round Robin
- Ordonnancement temps-réel FIFO

Par contre on peut se poser la question suivante: "Un processus éxécute plusieurs threads, le processus est ordonné par le noyau mais est ce que les threads sont eux aussi ordonnancés par le noyau, ou par l'application?". En fait il est possible d'effectué les deux possibilités c'est à dire :

- On mets en concurrence les threads avec tous les autres processus
- Dans le processus, on mets seulement les threads en concurrence avec eux mêmes

Il est aussi possible de choisir une possibilité pour un thread et l'autre pour un autre thread de la même application. Cependant pour respecter le temps d'éxecution de notre logiciel vis à vis des autres processus du système, on utilisera de préférence la deuxième possibilité.

On peut aussi configurer le fait que le thread peut être annulé (on verra explicitement ce que ça veut dire dans la section suivante). Ceci nous est fort utile lors que par exemple, on effectue une copie mais pendant la copie on décide l'arrêt alors on pourra instaurer des méthodes pour quitter le modules dans les régles de l'art (on nettoie la mémoire, on ferme les fichiers ouverts, on remets le système dans l'état auquel on l'a trouvé). Tout ceci évitera d'avoir des erreurs qui ne seront pas géré par notre programme, lorsque par exemple on va réutiliser un fichier à moitié copié.

6.1.2 La création d'un thread

Ainsi une fois qu'on a fait tous ces choix on peut créer le thread. Pour créer un thread il faut :

- L'adresse d'un thread
- L'adresse d'une structure indiquant les attributs
- Un pointeur de fonction (renvoie un *void **)
- L'argument de la fonction

On voit tout de suite qu'il demande une fonction générique, ce qui n'est pas une restriction bien au contraire car ce problème se résout très simplement à l'aide de conversion explicite.

6.2 Le déroulement normal d'un thread

Le déroulement normal d'un thread se fini toujours par un appel à `pthread_exit`. Dans ce cas la fonction principale du thread est éxécuté entierement donc le nettoyage à lieu dans cette fonction et on a rien d'autre à s'occuper pour la libération du thread.

6.3 L'annulation d'un thread

C'est ici qui faut qu'on soit rigoureux pour avoir une implémentation propre des threads. En effet si on annule un thread en plein milieu de son éxécution on peut très bien avoir des fichiers ouverts et des variables allouées. Cependant le problème peut être très bien le suivant : le nombre de fichiers ouverts ou de variables allouées est variable pendant l'éxécution du thread. Ceci implique que l'on sait pas qu'est qui faut fermer. Cependant la librairie des threads nous offre deux fonctions pour résoudre facilement ce problème il s'agit de

`pthread_cleanup_push` et `pthread_cleanup_pop`. Ceci nous permet d'empiler des pointeurs de fonction dans un pile propre au thread lors de l'éxécution de la fonction du thread et lorsque l'on quitte, il dépile toutes les fonctions (en les éxécutant) ce qui nous permet de libérer seulement ce qui à été alloué.

Chapitre 7

L'explorateur de fichiers

A la première soutenance on avait déjà un explorateur pouvant parcourir toute l'arborescence des disques et aussi afficher le contenu tous de les répertoires. Cependant nous avons rajouté plusieurs options à cette application, en dehors des modules plus importants.

7.1 Les améliorations

7.1.1 La gestion des accents pour les noms des fichiers

En règle générale, sur le système de fichiers d'Unix il n'y a pas d'accent (de caractère latin) dans les noms de fichiers. Chaque lettre du fichiers est à la norme américaine ASCII[1] c'est à dire codé sur 7 bits, alors que pour les caractères latin il faut un bit de plus, c'est la norme ISO-8859-1 ou latin-1 (pour coder en char les caractères è é à...). En fait le type **char** est codé sur 8 bit mais en norme ASCII le bit de signe n'est pas utilisé pour le codage, contrairement à la

1. ASCII : American Standard Code for Information Interchange

norme ISO-latin-1. Cependant, Unix peut avoir des partition monté provenant des OS Windows qui eux par contre, considère les accents dans les noms de fichiers.

C'est pourquoi, si l'on ne transforme par les caractères noms ASCII d'une chaine de caractère, l'affichage de cette chaine est alors indeterminé. Notre tâche est simplement de transformer une tablé ASCII étendu OEM[2] en une table ASCII étendue ANSI[3]

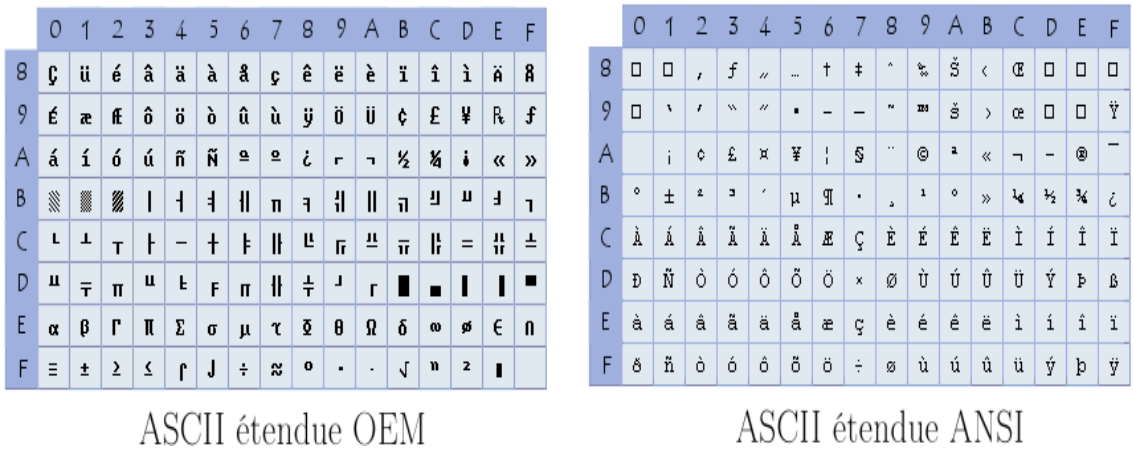


FIG. 7.1 – *Voici Exploreur Maker avec ses différentes parties*

Ainsi quand on lit dans le flux de répertoire, ces chaines sont dans la table ASCII étendu OEM alors que la plupart des systèmes pour la fonction *printf()* considèrent la table ASCII étendue ANSI, suivant le packages installés sur le linux (par exemple le labelFR). Donc il y une opposition entre les deux tables ont effectue une transformation.

2. ASCII étendu OEM : utilisé par les premières machines de type IBM PC
3. ASCII étendue ANSI : utilsé par les sytèmes d'exploitation récents

Cependant il faut garder une trace du nom d'origine ou refaire la transformation inverse sur le nom accentué afin de pouvoir par exemple utilisé la fonction *stat()* de la bibliothèque C qui ne considère par les accents. (Si il y a un accent cette fonction s'arrête un inscrivant une erreur dans la variable globale *errno*)

7.1.2 Execution avancée à partir shell avec des options à la manière Posix.2

Ces options, à la manière Posix.2, sont précédées d'un tiret et sont représentées par un caractère alphanumérique. On peut aussi avoir un tiret et plusieurs caractères alphanumériques. On peut donc avoir ces deux syntaxes :

```
ps -ax
ps -a -x
```

Il y a plusieurs régles comme un tiret seul est un argument et non une option. Deux tirets consécutifs est considérés comme la fin de la liste des options et les options se trouvant après ces deux tirets seront considérées comme des arguments.

Pour simplifier cette nome la bibliothèque C nous fournit l'option *getopt()* et les varaibles globales *optind*, *opterr*, *optopt* et *optarg*. Ainsi en fournissant la liste des options possibles, tant que cette fonction de renvoit pas -1 on aura toute la liste des options valides passées par l'utilisateur à partir du shell. On peut même indiquer si une option prend en paramètre un argument, et ce dernier est contenu dans *optarg*.

Dans notre cas, il est tout à fait possible de voir la syntaxe de notre liste d'options à partir du shell en faisant :

```
./exploreur -h
```

Ainsi il nous affiche la syntaxe (cf ci-dessous) sans lancer l'application.

```
explorer -h          : aide
explorer -d [chemin] : execution avec chemin comme racine
explorer -D [chemin] : execution et se place à chemin
```

7.1.3 Intégration de plusieurs modules à l'explorateur

L'explorateur est le support à long terme de tous les autres modules. On a déjà commencé à intégrer quelques modules important à l'explorateur comme le module de gestion de fichiers, et le module de recherche de fichiers. Grâce au noyau ce regroupement est très rapidement mis en place, car on le verra un peu plus loin en détail, mais il ne faut que passer au noyau les données d'un module pour qu'un autre module puisse fonctionner. Donc il n'y a pas de modification à faire dans les modules mais seulement à faire des complétudes dans le noyau de l'application.

7.1.4 Extention des signaux pour le listage de fichiers

Afin d'intégrer plusieurs modules à l'explorateur on se doit d'augmenter les possibilités d'accès à ces modules. Ainsi on a un petit menu qui apparait sur le listage de fichier à chaque fois que l'on clique sur

le bouton droit. Ce menu est personnalisé suivant les fichiers sélectionnés.

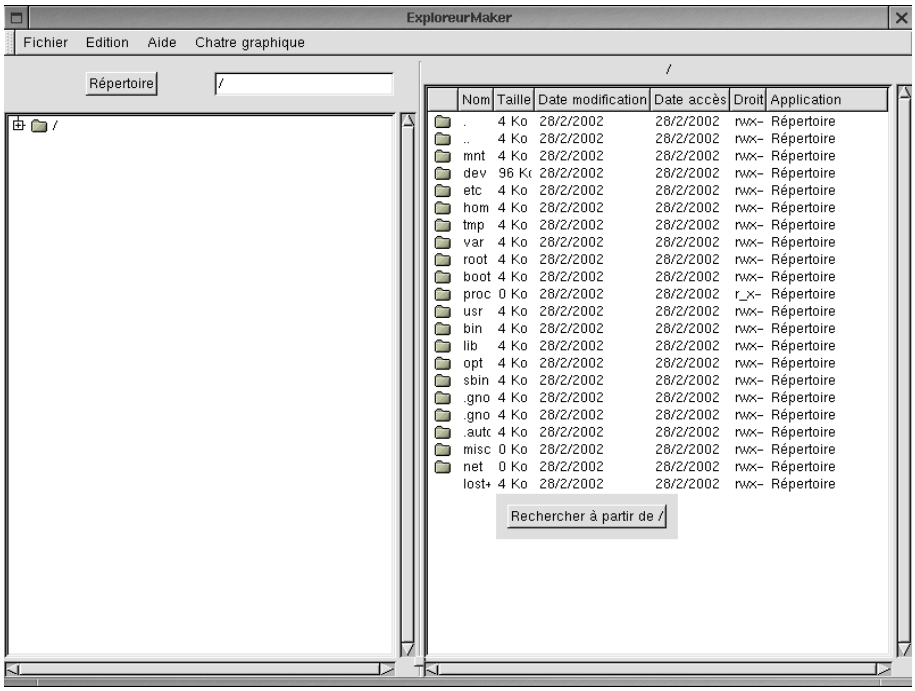


FIG. 7.2 – *Voici Exploreur Maker pour démarrer une recherche à partir du répertoire de travail*

Si aucun fichier n'est sélectionné on ne peut faire qu'une recherche dans le fichier à partir de ce répertoire ou un coller si il y a eut déjà avant soit un copier soit un coller.

Supposons maintenant qu'il y a des fichiers sélectionnés. Si on n'est pas propriètaires des fichiers on ne peut pas :

- modifier les droits
- modifier le propriètaire

– renommer

On peut aussi noter que l'on ne peut avoir le menu de renommage seulement si la sélection du listage contient un seul fichier régulier. Et bien sûr on ne peut faire une copie seulement si on à les droits de lecture (on peut noter déjà que c'est pas le menu qui le gére mais le listage).

Bien sur pour la copie il faut un système de copier coller à base de glisser déplacer : ce sont les événements *"drag and drop"*. Ainsi on peut déterminer quels fichiers ont été glissés, à quel moment mais aussi où et quand ils sont déplacés. Vu qu'on à le copier/coller il n'y a pas de nouvelle méthode à implémenter pour le module de gestion des fichiers.

Le listage acceuillera aussi le module de gestion d'application. Sur l'objet GTK pour le listage de fichiers il est possible de sélection une ligne de différentes manières. Donc on sait si il a été sélectionné par un simple clique ou un double clique. Mais on peut aussi déterminer quel ligne a été sélectionné donc pour le gestionnaire d'application on peut facilement déterminer quel fichier doit être lancé et avec quel application, il suffit juste de lire les informations de la ligne de la liste sélectionné par double clique.

Pour le confort de l'utilisateur on a aussi implèmenter des événements de plus bas niveau dans l'architecture GTK car se sont des signaux de l'environnement GDK. Ainsi lorsque l'on clique sur la touche CTRL on peut avoir une sélection multiple (c'est à dire de plusieurs à la fois). Mais aussi en utilisant le code ASCII on peut faire des bondkeys c'est à dire des combinaison entre plusieurs touches. Pour donner un exemple on eput tous sélectionnés en appuyant si-

multanément sur CTRL et A : c'est le code ASCII 1, et ainsi de suite....

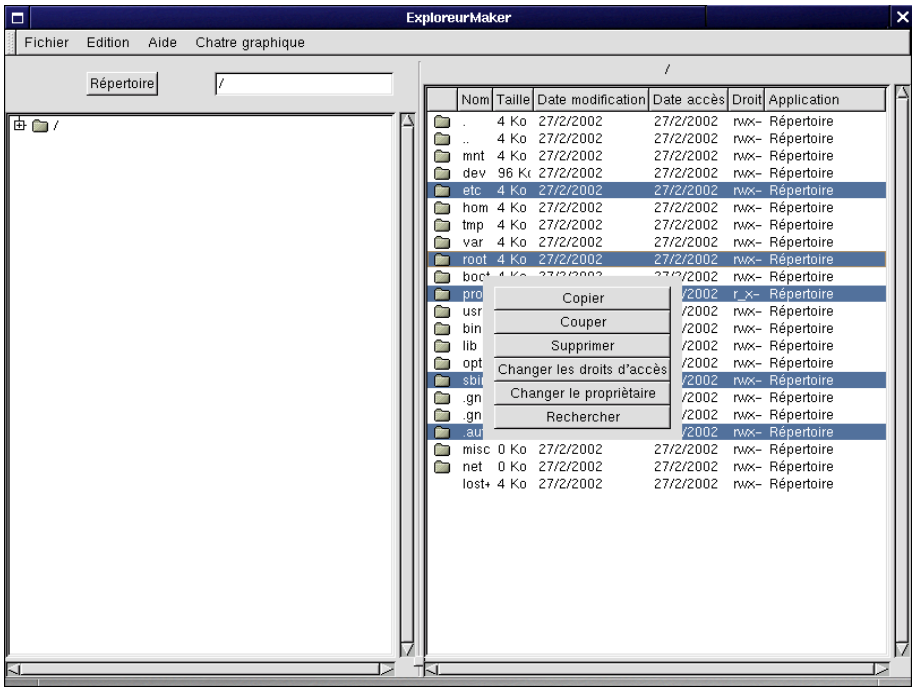


FIG. 7.3 – *Voici Exploreur Maker avec une sélection multiple et son menu*

7.2 Un noyau pour une meilleure gestion de l'application

7.2.1 Qu'est ce notre noyau

Notre noyau est une structure contenant toutes sortes de pointeurs sur des données importantes de différents modules. Mais le noyau comporte aussi des pointeurs de fonctions, un tube de com-

munications et un gestionnaire de signaux : c'est donc le moteur et le contrôleur de l'explorateur.

7.2.2 Pourquoi

Dans la vie de ce noyau qui à en fait la même durée de vie que le programme, il y a trois temps :

- L'initialisation : la naissance
- La gestion des signaux : la vie
- La libération : la mort

On peut comparer cette représentation comme un objet dans la programmation à objets.

Il a aussi des critères d'encapsulation, avec un tube de communication pour communiquer avec l'exterieur (le shell), il reçoit surtout des signaux. Mais il a aussi des données publiques (l'interface) de différents modules sans passer par des variables globales. Donc cela nous permet d'avoir aucunes variables globales dans le programme, on passe juste en paramètre de nos modules notre noyau.

7.2.3 Comment ça marche

Ainsi comme on vient de le dire, on passe notre noyau en paramètre dans la plupart des modules. En fait ce sont les modules qui forment les grandes parties du programme, et qui suivant un état du programme ce module effectura une méthode d'une certaine manière. Par exemple notre module de copie n'aura pas le même comportement suivant le type des fichiers selectionnés. Ainsi notre module de copie peut travailler dans son coin sans se préoccuper de l'état du module de listage de fichier mais seulement des fichiers sélectionnés et notre

noyau fait le lien entre ces deux modules pour qu'il communique au module de copie les fichiers sélectionnés.

Mais aussi, il peut avoir des informations sur l'ensemble du programme et les programmes isssues de l'explorateur. Ce noyau contiendra tous les PID des processus créés pour lancer de nouveaux programmes et donc on pourra facilement savoir d'une part quel processus il faut tuer à la fermeture du programme, et d'autre part cela nous permettra d'afficher facilement dans notre menu les différents programmes et de les mettre en focalisation principale (mettre la fenêtre du programme lancé par dessus l'explorateur). Après dans ce menu, on peut facilement envisager d'autres possibilités comme fermer un programme à partir de l'explorateur.

7.3 Les outils utilisés pour la fiabilisation de l'explorateur

Bien entendu cette fiabilisation est présente du premier module au dernier module du projet. Cependant plus le projet avance et plus le débogage devient difficile. En premier lieu pour assurer un premier niveau de fiabilité, il faut que dans le code, on gère tous les cas même les cas d'erreur. Pour cela la librairie standard C nous fourni une librairie ***errno.h***. Ainsi lorsque l'on teste la valeur de retour d'une fonction on sait s'il y a une erreur ou pas. Si il n'y a pas d'erreur, on poursuit le programme normalement sinon on analyse la valeur de retour et le contenu de la variable globale *errno* qui nous indique le

pourquoi de l'echec. Ainsi si il y a une erreur, on a deux possibilité :

- il y a une erreur, on inscrit sur *stderr*[4] où il y a une erreur et laquelle, puis on quitte le programme
- il y a une erreur et on fait avec, si par exemple lors d'un d'échec avec *fork ()* selon la valeur de errno on peut envisager de recommencer l'opération jusqu'à la réussite mais dans un cas comme ça il faut mieux quitter directement le programme car une telle erreur indique clairement que le stabilité du noyau est dans un état critique.

7.3.1 Débogeur

Pour le débogage de l'explorateur on utilise **xxgdb** (version X-Window de **gdb**) sous Unix mais sur Linux il y encore un débogeur plus facile d'utilisation c'est **ddd**[5] reposant lui ausssi sur gdb c'est juste l'interface qui change, on peut mettre ainsi un point d'arrêt sur un ligne de code sans passer par les commandes de gdb. Cette utilisation nous permet tout d'abord de savoir où il y a une segmentation fault (écriture dans une zone non autorisé au processus). Mais aussi de savoir les différents messages destinés au processus par le noyau comme SIGSEGV ou SIGBUS (message entrainat la destruction du processus). Ces utilitaires permettent aussi d'exécuter le programme instruction en instrcution pour vérifier si le programme fait exactement ce qu'on voulait qu'il fasse.

4. stderr : sortie standard d'erreur sous Unix
5. ddd : Data Display Debugger

7.3.2 Profileur

Ce genre d'utilitaire est plus utilisé dans la phase de mise au point. Il permet d'observer le déroulement de l'application, et enregistre dans un fichier les temps de présence dans chaque méthode du programme. Bien sur, ceci ne concerne pas toutes les parties du code car de tel utilitaire ne permet que d'optimiser le code. Ceci nous a été par exemple utile lors du listage de fichier lorsque l'on sélection un répertoire ou la copie de fichier. Ainsi en modifiant les algorthimes et leurs implémantations cela nous permet d'avoir une application qui n'admet pas de latence visible à l'œil humain. Pour cela on a utilisé **gprof**. Il faut juste à la compilation ajouter l'option **-pg**, et pour analyser le fichier on fait à partir du shell :

```
gprof nomprogramme gmon.out | less
```

7.3.3 La gestion de la mémoire alloué

Pour limiter les erreurs de mémoire on s'efforce de libérer toutes les variables alloués par l'appel système *malloc()* avec l'appel système *free()*. Des outils nous permettent de savoir l'état de la mémoire du processus avant sa destruction (comme *mtrace()* déclaré dans la librairie standard ***mcheck.h***). Avec cette fonction et un petit script shell :

```
#bin/sh

export MALLOC_TRACE = "unfichier"
./programme
mtrace programme unfichier
```

on peut déterminer toutes les allocations présentes dans la mémoire du processus et qui n'ont pas été délivré. Ainsi on peut limiter les

éventuels débordements de mémoire. Et de plus, ceci nous évite d'agrandir la mémoire autorisé par le noyau au processus.

Chapitre 8

Le site WEB

La réalisation d'un site internet était quasiment une nouveauté. Aucun d'entre nous n'avez deja vraiment fait de site internet. Etant le seul (Benoit) à avoir quelques notions d'HTML, je m'en suis alors chargé.

8.1 Elaboration du site

Nous avons choisit de faire un site simple qui soit téléchargeable sans trop de diffcultés avec un modem classique. Pour commencer j'ai choisit d'utiliser Microsoft Frontpage 2000 pour faire le site. Ainsi j'ai obtenu sans difficulté une première version du site que j'ai mis en ligne. Mais j'ai rapidement eu des problèmes d'ordre logiciel avec Frontpage, détruisant le site (merci Microsoft ...). Pour ne pas tout recommencer, j'ai téléchargé le site depuis internet. Et je me suis mis à refaire le site à partir de ces fichiers le site avec Macromedia Dreamweaver 4. J'ai alors travaillé ligne par ligne, allégeant le plus possible tous les 'surplus' ajoutés par Frontpage (et il y en a beaucoup...).

2- Les graphismes Les graphismes n'ont rien d'extraordinaires sur

le site. J'ai laissé qulquesuns que Frontpage avait mis. Notamment les boutons de navigation utilisant un script en Java pour les eclaircir si la souris pointe dessus.

J'ai en revanche fait les graphismes de haut de page (indiquant le titre de la page) donnant un design un peu plus moderne au site.

8.2 Les rapports du projet

Les rapports étant fait avec LaTeX, je me suis servi du programme TTH qui permet de traduire un fichier .tex au format HTML. Il a fallut tout de même faire quelques modifications pour alléger les images utilisées avec LaTeX. Elles sont donc compressées au format JPEG en diminuant la taille et la qualité de façon à ce que la page reste légère. Les rapports sont égalements disponibles au format Adobe Acrobat (.PDF) grace une fois de plus à un traducteur LaTeXvers PDF.

8.3 Le contenu du site

En plus des rapports aux formats HTML et PDF, le site contient quelques autres informations. La page d'accueil contient une présentation succinte du projet et une liste des dernières nouveautés apportées au site. On peut également trouvé une page contenant quelques informations sur les membres de ce projet, il y aura même bientôt une photo de chacun de nous ! Une dernière page contient les informations sur tout ce qui nous a servi pour ce projet. On trouve des référence de livres ou de sites internet, ainsi qu'un lien pour récupérer la dernière version officielle de GTK, outils principal de ce projet. Cette page permmettrat également de télécharger ce que nous aurons réaliser. Pour le moment on peut déjà télécharger l'éditeur de texte (compressé au format .tgz). Cette page va bien évidemment se garnir dau fur et à mesure que le projet avancera, des références seront ajoutées.

La structure du site est donc faite. Elle ne devrait pas évoluer beaucoup d'ici la fin de la réalistation du projet. En revanche beaucoup d'informations y seront ajoutées : rapports, téléchargements et références.

Chapitre 9

Conclusion

9.1 Qui a fait quoi

Sébastien Deckmyn	Nicolas Le Coz	Benoit Merouze	Julier Perruche
Le gestionnaire de packages	Le gestionnaire de fichiers La finalisation et le débogage de l'explorateur Une application multi-tâches	Le site WEB Le gestionnaire d'extentions	Le gestionnaire de recherche

9.2 Pour la troisième soutenance

Sébastien Deckmyn	Nicolas Le Coz	Benoit Merouze	Julier Perruche
Finalisation du Le gestionnaire de packages Gestionnaire d'archives (tar,zip, tar.gz)	Gestionnaire d'application La finalisation et le débogage de l'explorateur	Le site WEB Finalisation du gestionnaire d'extentions	Finalisation du gestionnaire de recherche Gestionnaire d'applications

9.3 A la prochaine fois

A la fin de cette deuxième soutenance on peut constater que tous les modules importants sont finis ou en court de finalisation (le plus gros a été fais). On va rentrer dans un phase de mise en forme de tous les modules en un seul programme. Le travail a faire en plus de la finalisation est de faire coordonnées plusieurs modules à la fois comme par exemple le gestionnaire de packages avec le gestionnaire d'extentions. Ensuite le gestionnaire d'extentions va servir de support au module de gestion des applications. Comme le projet est en avance, on remarque que tous les modules commence déjà à s'imbriquer les uns aux autres très naturellement (sans avoir à en modifier leur structure). Comme par exemple le module de recherche a été intégré en cinq minutes à l'interface. Donc pour la troisième soutenance on essayera de vous présenter un seul logiciel regroupant tous les modules.