Projet "find the cat"

RS : Réseaux et Systèmes

Deuxième année

L'objectif de ce projet est d'écrire un programme qui permette de retrouver un fichier dans une arborescence.

1 Logistique

Groupes. Apprendre à travailler en groupe fait partie des objectifs de ce projet. Il vous est donc demandé de travailler en **binôme**. Il est **interdit** de travailler seul, si vous êtes un nombre impair, un groupe pourra travailler à trois. Un ingénieur travaille rarement seul.

Langage. Le projet devra être codé en langage C et devra compiler sans intervention extérieure à l'aide de `make` (et d'un `Makefile` correspondant).

Tests automatiques. Une part importante de l'évaluation du projet sera faite à l'aide d'une batterie de tests exécutés sur une distribution GNU/Linux (Ubuntu précisément). Il est donc très important, au cours de votre travail, d'accorder une large part à vos propres tests et de vérifier leur bonne exécution dans le même environnement que celui d'évaluation. 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, et les résultats de ces tests vous seront communiqués. **Il est donc indispensable de commencer à travailler tôt pour bénéficier de ces «tests blancs» et avoir la garantie que votre projet fonctionne correctement.**
Par ailleurs, dans l'objectif de rendre le développement plus facile, des tests d'intégration continue seront disponibles sur git et indiqueront si les résultats sont conformes aux attentes à chaque push. Il est toutefois à noter que ces tests ne seront pas les mêmes que ceux de l'évaluation, simplement un indicateur. Réussir à passer les test d'intégration devrait néanmoins garantir que le format des réponses devrait passer les tests finaux.

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 extensions que vous aurez développées, 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, implémentation, tests, rédaction du rapport) par membre du groupe.

Évaluation

- **12 points** seront attribués de manière automatique par les tests. Après compilation, nous utiliserons votre programme pour exécuter différentes suites de tests. Vous perdrez des points si votre programme ne se comporte pas comme prévu. Votre programme doit donc bien respecter les codes de retour qui sont détaillés plus loin dans ce document.
- **1 points** sera attribué en fonction de la qualité subjective du code source de votre programme
- **3 points** seront attribués en fonction de la qualité du mini-rapport.
- Des points supplémentaires seront attribués pour chacune des extensions implémentées.

Soutenances. Des soutenances pourront être organisées (éventuellement seulement pour certains groupes, par exemple s'il y a des doutes sur l'originalité du travail rendu). Vous devrez nous faire une démonstration de votre projet et être prêts à répondre à toutes les questions techniques sur l'implémentation de l'application.

Plagiat et aide extérieure au binôme. Si, pour réaliser le projet, vous utilisez des ressources externes, votre rapport doit les lister (en expliquant brièvement les informations que vous y avez obtenus). Un détecteur de plagiat [1] sera utilisé pour tester l'originalité de votre travail (en le comparant notamment aux projets rendus par les autres groupes). Toute triche sera sévèrement punie.

Concrètement, il n'est pas interdit de discuter du projet avec d'autres groupes, y compris de détails techniques. Mais il est interdit de partager, ou copier du code, ou encore de lire le code de quelqu'un d'autre pour s'en inspirer.

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

Questions. Vos questions éventuelles peuvent être adressées à `maiwenn.racouchot@inria.fr`. Les réponses (et les questions correspondantes) pourront être publiées dans la FAQ sur la page du projet. En cas de problème inextricable sur le projet, de besoin de conseil urgent ou simplement pour un bonjour de courtoisie, je rappelle également que le bureau 1.31 de M.Bouthier se trouve au service informatique, 1er étage à côté de l'escalier principal. (La légalité de cette recherche de conseil a été vérifiée auprès de Mme.Heurtel qui approuve ce recours).

2 Description du projet "Find the cat"

Vous devez écrire un programme `ftc` dont l'objectif est de permettre à un utilisateur de trouver le chemin de fichiers dans une arborescence.

Votre programme `ftc` devra accepter en ligne de commande un dossier de base à partir duquel réaliser la recherche et des options additionnelles spécifiant les fichiers à trouver, détaillées ci-dessous. Le format attendu de la commande est le suivant : `ftc starting-point [-option [paramètre]]`.

Si les paramètres passés sont incorrects, votre programme doit s'arrêter avec un message expliquant l'usage correct de la commande.

3 Travail à réaliser

Le cœur du projet est de manipuler l'arborescence et les fichiers sous Linux. Pour la première phase de ce projet, on ne considérera que des fichiers encodés en ASCII. Bien qu'il ne soit pas obligatoire d'avoir valider les tests pour toutes ces fonctionnalités avant de se mettre aux options, il est **très vivement** conseiller de commencer par celles-ci. Voici les fonctionalités de bases qui sont attendues de votre projet :

1. Écrire le code pour parser la ligne de commande et analyser les paramètres. Le nom des options devra suivre la nomenclature détaillée dans le tableau 3. Pour cette étape, il nécessaire de pouvoir reconnaître toutes les options et de lire leur paramètre. On attend du programme qu'il affiche "La valeur du flag -xxxx est yyyy.". Pour ne pas interférer avec les étapes suivantes, cette étape sera lancée avec un flag -test qui lancera ce comportement. En son absence, le programme exécutera la recherche selon l'option passée en paramètre. On attend en particulier la sortie : `La valeur du flag -xxxx est yyyy`. On n'attend pas à cette étape que -test gère plus d'un flag supplémentaire.
2. Écrire le code permettant de parcourir l'arborescence de fichier (de manière récursive). Pour vérifier le bon fonctionnement de cette étape, le programme devra afficher le chemin de tous les dossiers et fichiers présents depuis la racine. Il se comportera donc comme un 'ls' récursif.
3. Écrire le code reconnaissant les fichiers par leur nom ou par leur taille. Pour cette étape on attend que `ftc` soit capable de localiser un fichier d'après son 'nom.extension' avec l'option '-name' ou sa taille avec l'option '-size'. Les tailles seront spécifiées avec les même paramètres que find (voir tab 1). Pour localiser un fichier selon sa taille, vous pourrez commencer par chercher les fichiers par

Symbole	Taille
c	bytes (par défaut si aucune unité n'est spécifiée)
k	kibibytes (KiB, unité of 1024 bytes)
M	mebibytes (MiB, unité de 1024 * 1024 = 1048576 bytes)
G	gibibytes (GiB, unité de 1024 * 1024 * 1024 = 1073741824 bytes)

TABLE 1 – Paramètre pour l'option -size

leur taille exacte puis ajouter les options '+' et '-'. Elles fonctionneront selon la syntaxe suivante : `ftc . -size +9k` et permettront de trouver les fichiers dont la taille est strictement supérieure ou inférieure à la borne donnée.
4. Implémenter une fonctionnalité 'ET'. Elle ne requiert pas de flag supplémentaire mais permettra de gérer la présence de plusieurs flags en ligne de commande et en ne sélectionnant que les fichiers répondant à tous les critères passés.
5. Écrire le code permettant de récupérer les fichiers par date de dernier accès (option -date). Le programme devra renvoyer les fichiers dont le dernier accès a eu lieu au moment de la période

passée en paramètre. Celle-ci pourra être définie de plusieurs manière (cf tab 2). Par défault, on considérera les fichiers accédés au cours de la période passée en paramètre. Par exemple, pour lister les fichiers modifiés il y a moins 3 heures on écrira : `ftc . -date 3h` (cela inclus les fichiers modifiés il y 3h mais pas 3h01). Pour cela, il sera nécessaire de commencer par récupérer la date actuelle en plus de celle de dernier accès du fichier pour déduire la période écoulée. Pour compléter cette option, vous pourrez ajouter le '+' comme pour la taille. Il permettra d'inverser le fonctionnement de la commande et de considérer les fichiers modifiés depuis plus de temps que la période précisée.

Symbole	Durée
m	minutes
h	heures
j	jours

TABLE 2 – Paramètre pour l'option -date

6. Écrire le code permettant de chercher les fichiers en utilisant des regex. Il est conseillé de trouver une librairie permettant de gérer les regex plutôt que d'implémenter tout à la main si vous ne souhaitez pas y dédier votre année. Cette option ne requiert pas de flag supplémentaire puisqu'elle étend le comportement initial de -name.
7. Écrire le code permettant de gérer les types mime (option '-mime'). On voudra être en mesure de chercher selon les types et les sous-types. Voici donc deux exemples de demandes qui devront être traitées par `ftc` : `ftc . -mime text` ou `ftc . -mime text/html`. Pour cette option, il est encore une fois conseillé de s'appuyer sur une librairie de bon aloi. On attend que la détection des types se fassent selon l'extension du fichier et non selon son encodage.
8. Écrire le code pour accéder au contenu du fichier puis implémenter la recherche de pattern à l'intérieur du fichier (équivalent de la commande grep). Cette option prendra le nom '`-ctc` ' (catch the cat). Si la gestion des regex n'est pas en place, il est également possible de commencer à l'implémenter avec des chaînes de caractères normales. On attend de cette option qu'elle renvoie les fichiers contenant la chaîne de caractère passée en paramètre.
9. Ecrire une option '-dir' qui permettra de faire de la recherche par nom sur les répertoires plutôt que sur les fichiers. Dans le cas où aucun paramètre n'est passé à cette option, elle se contentera de lister les dossiers présents depuis le point de départ.
10. Mettre en place une gestion des messages d'erreurs. En cas de passage de paramètre incorrect ou de problème dans la recherche, on attend que le programme renvoie un message d'erreur explicite sur un canal dédié. Ce méchanisme devra permettre de filtrer les résultats corrects des erreurs.
11. Apportez les touches finales au code, nettoyez le et testez le convenablement.

4 Extensions possibles

Les extensions réalisées seront valorisées. Il n'est pas nécessaire d'avoir complété la première partie pour les implémenter mais je recommande à nouveau de se pencher sur les premières fonctionnalités en priorité ! Les fonctionnalités optionnelles devront être décrites dans le rapport. Comme indiqué plus haut la partie obligatoire du projet sera notée sur 16, pour obtenir une meilleure note il faudra donc implémenter plusieurs des extensions présentées ci-dessous. Ces options n'étant pas obligatoires, les choix de conception réalisés leur de leur implémentation pourront être justifiés dans le rapport.

1. Ajouter de la couleur dans les résultats pour améliorer leur lisibilité. Cette option ne fera pas l'objet de tests automatiques mais devra donc être documentée dans votre rapport. Cet affichage ne devra être activé qu'en présence du flag '-color' pour éviter de mettre en péril vos résultats sur les tests automatiques.
   Quelques exemple de mise en couleur :
   - Ecriture en couleur du nom du fichier trouvé par rapport à son path
   - Choisir la couleur en fonction des droits de l'utilisateur sur le fichier
   - Colorer différement en fonction du type mime
   - Colorer différement les fichiers et dossiers

Commandes	Description
-test	Cette option ne prend pas de paramètre mais indique qu'on est dans le comportement de test. Dans ce contexte, le programme affichera le flag suivant et la valeur qu'il prend en paramètre.
-name	Cherche les fichiers dont le nom correspond à la chaîne de caractère. Si la partie regex n'a pas encore été traitée, il s'agira du nom complet (nom.extension). Si cette option a été atteinte, il pourra s'agir d'une regex.
-size	Cherche les noms correspondant à un critère de taille. L'utilisateur pourra passer des options de types -9c ou +6k pour chercher respectivement les fichiers contenant moins de 9 bytes ou plus de 6 KiB.
-date	Cherche les fichiers dont la date de dernier accès correspond à la période passée en paramètre de l'option.
-mime	Cherche les fichiers selon leur type mime. Cette option doit accepter en paramètre soit le type recherché, soit le type/sous-type.
-ctc	Cherche une chaîne de caractère (ou une regex) à l'intérieur des fichiers et renvoie les fichiers qui la contienne.
-dir	Mets en place la recherche par nom sur les dossiers.
Commandes optionnelles	**Description**
-color	Affiche les resultats en couleur.
-perm	Chercher les fichiers par leur permissions.
-link	Demande à `ftc` de suivre les liens symboliques.
-threads	Lance `ftc` en multithreading. On passera en paramètre de cette option le nombre de threads voulu.
-ou	Indique au programme de traiter les différentes options passées dans la ligne de commande comme un 'OU' plutôt qu'un 'ET'.

Table 3 – Récapitulatif des options

2. Gérer la recherche par date à l'aide de mot clé (par exemple, chercher les fichiers dont les dates de modification correspondent à "now", "today", "yesterday", "this month"). Cette option étendra le comportement de base de l'option '-date'.
3. Permettre la recherche selon le type de permission du fichier (accès en lecture par l'utilisateur, en exécution par le groupe...). Cette option sera accessible à partir du flag '-perm'. Les permissions recherchées seront passées avec le format octal (pour chercher les fichiers pour lesquels l'utilisateur a tous les droits et les autres rien, on fera par exemple : `ftc . -perm 700`).
4. Ajouter une option ('-link') qui demande à `ftc` de suivre les liens symboliques lors du parcours de l'arbre.
5. Ajouter un flag '-ou' qui indiquera à votre programme de faire un 'OU' logique sur les conditions plutôt qu'un 'ET'.
6. Gérer les fichiers encodés en UTF-8 plutôt qu'en ASCII.
7. Pour les plus motivés, lancer `ftc` en multithreads pour améliorer ses performances. Si cette option est intégrée au projet, il sera intéressant d'ajouter au rapport une partie benchmark comparant les performances du programme avec et sans cette option. Il sera aussi intéressant de le comparer aux temps d'exécution de la fonction find.
8. Pour les plus masochistes, essayer de refaire `ftc` pour le rendre opérant dans une distribution Windows.
9. Trouver le code du chat et l'intégrer au rapport !

5 « Rendu » du projet.

Le « rendu » du projet se fera via l'instance Gitlab de TELECOM Nancy. En dehors de la configuration correcte de votre dépôt, il n'y a rien à faire le jour de la fin du projet. L'intégration continue et les tests blancs, vous permettront de vérifier la bonne configuration de votre dépôt.

Votre dépôt de groupe sera créé et configuré par l'école. L'adresse vous sera communiquée une fois les groupes formés. Pour ce faire, vous devrez envoyer un mail par binôme à l'adresse `maiwenn.racouchot@inria.fr` avant le **26/10/2022**. Il est inutile que les deux membres du binôme le fasse, il suffira de mettre l'autre en copie. Si vous vous retrouvez seul, merci de m'envoyer un mail pour le signaler, je pourrais ainsi former des groupes avec les personnes restantes voire former un groupe de trois si cela devait être nécessaire. Si je n'ai pas reçu de mail d'ici le 26, vous serez attribué au hasard à un groupe.

Votre dépôt Git devra contenir, à la racine du dépôt :

— Le code source de votre projet (éventuellement dans un dossier `src`)

— Un ficher `AUTHORS` listant les noms et prénoms des membres du groupe (une personne par ligne)

— Un `Makefile` (**à la racine !**) compilant votre projet en créant un fichier exécutable (**à la racine**) nommé `ftc`

— Un fichier `rapport.pdf` contenant votre rapport au format PDF.

6 Calendrier

Des ≪tests blancs≫ seront effectués à au moins trois reprises, aux alentours du **08/11/22**, du **29/11/22**, et du **10/12/22**. 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.

La version finale de votre projet est à rendre pour le **dimanche 18/12/2022 à 23h59**. Il sera récupéré directement sur vos dépôts git. 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.