Université Paul Sabatier **UE : ED4INFDM**
L2 **Systèmes d'exploitation 2**

MINI PROJET

Simulation d'un système de gestion de fichiers

mars-avril 2013

Vous allez réaliser un mini projet en `C`. Ce projet sera à rendre le **7 avril 2013 à minuit** et la recette aura lieu lors de la séance de TD-machine qui suivra (semaine du 8 au 12 avril).
Vous devrez fournir les programmes sources `C` et un mini rapport (5 pages maxi) décrivant les fonctions et programmes que vous avez écrits ou modifiés, ainsi que des exemples d'utilisation. Vous rendrez ces documents sous forme d'une archive (tar) contenant les fichiers sources `C`, le `Makefile` et le rapport au format pdf.

Sujet

Vous avez vu en cours le principe de fonctionnement du système de gestion de fichiers *(SGF)* sous UNIX et notamment l'allocation des blocs du périphérique de masse (disque dur) pour les données des fichiers.
Nous vous proposons d'écrire un ensemble de programmes permettant de simuler le fonctionnement d'un *SGF* simplifié. Vous allez donc écrire des fonctions de base du *SGF* ainsi que des programmes permettant de réaliser des commandes d'accès à ce *SGF*.

Précision : chaque programme devra vérifier la validité des paramètres qui lui sont passés et chaque fonction et programme devra gérer les erreurs éventuelles et afficher des messages clairs pour l'utilisateur.

sources `C`

Pour vous aider, nous mettons à votre disposition un ensemble de programmes écrits en `C`, réalisant quelques commandes d'accès au *SGF*, ainsi que les fichiers `mon_SGF.c` et `mon_SGF.h` qui contiennent les structures de données, les constantes et quelques fonctions de base du *SGF*.

périphérique de mémoire de masse simulé

Normalement, un *SGF* enregistre et lit les données par bloc sur un périphérique de mémoire de masse (un disque dur par exemple). Pour la simulation, nous utilisons un fichier dans lequel notre *SGF* vient lire et écrire. Ce fichier (nommé `.device_name`) a une taille fixe et il est créé par une commande de formatage fournie et dont le source `C` se trouve dans le fichier `mon_formater.c`
Nous vous fournissons gracieusement un exemple de ce fichier `.device_name` pour que vous puissiez tester notamment les fonctions de lecture sans avoir écrit les fonctions d'écriture.

nommage des fichiers

Tous les fichiers que nous utilisons et les commandes associées ont pour préfixe `mon_` pour les distinguer des commandes UNIX : par exemple `mon_formater` , `mon_ls` , `mon_mkdir` , `mon_rmdir` ...

compilation avec la commande `make`

Afin de simplifier la compilation, nous vous fournissons gracieusement un fichier `Makefile` qui contient les descriptions de compilation des différents fichiers et commandes à créer. Pour l'utiliser, par exemple pour compiler le programme `mon_formater`, il suffit de taper "`make mon_formater`" et la commande `make` compilera d'abord `mon_SGF.c` si c'est nécessaire, puis `mon_formater.c` pour générer un exécutable `mon_formater`. Si vous voulez compiler tous les programmes décrits dans le fichier `Makefile`, tapez "`make all`". Si vous voulez effacer tous les exécutables, tapez "`make clean`". Si vous ajoutez des fichiers à compiler, prenez exemple sur ceux qui sont déjà dans le `Makefile`.

1 La bibliothèque de fonctions de base du *SGF*

1.1 En-tête

Le fichier `mon_SGF.h` contient les structures de données et les macro-constantes à utiliser dans tous les programmes que vous réaliserez. Vous y trouverez notamment deux structures de base du *SGF* :

`inoeud_t` : qui décrit le type de structure de données d'un *inoeud* (*inode*).

`superbloc_t` : qui décrit le type de structure de données du *superbloc*.

Vous constaterez que le *inoeud* est très simple. Il dispose d'un tableau de liens directs vers les blocs de données du fichier mais pas de liens indirects comme le *SGF* d'UNIX vu en cours.

Ensuite, la structure décrite par le type `SGF_t`, permet de regrouper le *superbloc*, la table des *inoeuds* ainsi que le numéro du fichier dans lequel nous simulons le *SGF*. Cette structure est utilisée comme paramètre dans la plupart des fonctions du *SGF*.

Enfin, la structure décrite par le type `dir_element_t` correspond à un élément de la liste des éléments d'un répertoire. Cette liste est réalisée par un tableau dynamique d'éléments `dir_element_t` et dont le dernier élément est un *inoeud* de type `NO_INOEUD` ce qui permet de marquer la fin de liste.

Vous devez utiliser ces définitions pour vos programmes. Vous ne pouvez pas les modifier, sous aucun prétexte. Vous pouvez, si et seulement si c'est nécessaire, y ajouter des définitions.

1.2 Corps

Le fichier `mon_SGF.c` contient le corps des fonctions de base du *SGF*. Il y a des fonctions de :

affichage des erreurs : `affiche_erreur`

lecture - écriture du *SGF* : `ecrire_superbloc`, `lire_superbloc`, `ecrire_table_inoeuds`, `lire_table_inoeuds`, `lire_SGF`, `ouvrir_SGF`, `fermer_SGF`

gestions des blocs : `ecrire_donnees_dans_bloc`, `ecrire_bloc`, `lire_donnees_dans_bloc`, `lire_bloc`

gestions des *inoeuds* et blocs : `inoeud_libre`, `bloc_libre_suivant`, `liberer_blocs_du_inoeud`, `allouer_n_blocs_dans_inoeud`, `ecrire_donnees_dans_inoeud`, `lire_donnees_dans_inoeud`, `creer_inoeud`, `liberer_inoeud`

gestions des répertoires : `lire_liste_rep_dans_inoeud`, `delier_element_du_repertoire`, `lier_element_au_repertoire`, `nom_suivant`, `inoeud_element_dans_repertoire`, `inoeud_designation_element`

Certaines de ces fonctions vous sont gracieusement offertes, d'autres sont à votre charge. Vous êtes invité à lire tous ces codes ainsi que les commentaires qui les accompagnent.

Vous ne pouvez pas modifier le corps (s'il est déjà écrit) ou les prototypes de ces fonctions, sous aucun prétexte. Vous pouvez, si et seulement si c'est nécessaire, ajouter de nouvelles fonctions.

2 Fonctions de base à écrire

Vous devez écrire les fonctions suivantes, en complétant le corps vide qui se trouve dans le fichier `mon_SGF.c` :

`int lire_superbloc (SGF_t *mon_SGF)`
copie dans la structure "`mon_SGF->superbloc`" les données de superbloc lues au début (bloc numéro 0) du fichier ouvert "`mon_SGF->device_num`".
Renvoie 0 si tout va bien ou un code d'erreur sinon.

`int lire_table_inoeuds (SGF_t *mon_SGF)`
copie dans la structure "`mon_SGF->table_inoeuds`" les données de la table des *inoeuds* lues à partir du bloc numéro 1 dans le fichier ouvert "`mon_SGF->device_num`".
Renvoie 0 si tout va bien ou un code d'erreur sinon.

`int lire_SGF (SGF_t *mon_SGF)`
alloue en mémoire, si besoin, les structures "`mon_SGF->superbloc`" et "`mon_SGF->table_inoeuds`", et y copie les données correspondantes lues dans le fichier ouvert "`mon_SGF->device_num`".
Renvoie 0 si tout va bien ou un code d'erreur sinon.

`SGF_t * ouvrir_SGF ()`
alloue en mémoire la structure "`mon_SGF`" et ouvre en lecture- écriture le fichier déjà existant "`DEVICE_NAME`", dont le numéro est copié dans "`mon_SGF->device_num`". Puis lit les données des structures du *SGF*.
Renvoie un pointeur vers une structure *SGF* initialisée et prête à l'emploi pour d'autres opérations sur le *SGF*.

`int inoeud_libre (SGF_t *mon_SGF)`
cherche dans la table des *inoeuds* – la structure "`mon_SGF->table_inoeuds`" – le numéro du premier *inoeud* libre.
Renvoie le numéro du premier *inoeud* libre ou "`NO_INOEUD`" s'il n'y en a pas.

`int bloc_libre_suivant (int num_bloc, SGF_t *mon_SGF)`
lit dans le bloc libre "`num_bloc`", le numéro du bloc libre suivant.
Attention : "`num_bloc`" doit être le numéro d'un bloc libre.
Renvoie le numéro du bloc libre suivant ou "`NULL_BLOC`" s'il n'y en a pas.

`int liberer_blocs_du_inoeud (int inoeud, int premier_bloc, SGF_t *mon_SGF)`
libère les blocs du *inoeud* "`inoeud`" à partir du numéro "`premier_bloc`" jusqu'au dernier utilisé de la table "`liens_directs_blocs`" du "`inoeud`". Les éléments de la table "`liens_directs_blocs`" libérés prennent la valeur "`NULL_BLOC`". Les blocs libérés sont ajoutés en tête de la liste chaînée de blocs libres, en écrivant dans chaque bloc libéré le numéro du bloc libre suivant et en mettant à jour le numéro du premier bloc libre (tête de la liste) dans "`mon_SGF->superbloc->premier_bloc_libre`". Le superbloc et la table des *inoeuds* sont mis à jour sur le *SGF* (écrits dans le fichier du *SGF*).
Renvoie 0 si tout va bien ou un code d'erreur sinon.

`int allouer_n_blocs_dans_inoeud (int nb_blocs_a_allouer, int inoeud, SGF_t *mon_SGF)`
prend dans la liste des blocs libres les "`nb_blocs_a_allouer`" premiers blocs libres et les alloue à l'inoeud "`inoeud`" en mettant le numéro des blocs aux emplacements libres de la table "`liens_directs_blocs`" du *inoeud* "`inoeud`". Met à jour le numéro du premier bloc libre (tête de la liste) dans "`mon_SGF->superbloc->premier_bloc_libre`". Le superbloc et la table des *inoeuds* sont mis à jour sur le *SGF* (écrits dans le fichier du *SGF*).
Renvoie 0 si tout va bien ou un code d'erreur sinon (notamment `EXIT_FS_FULL` s'il n'y a plus de blocs libres ou d'emplacements libres dans la table "`liens_directs_blocs`").

`int ecrire_donnees_dans_inoeud (char *data, int data_size, int inoeud, SGF_t *mon_SGF)`
écrit dans les blocs du *inoeud* "`inoeud`", le contenu de "`data`" de "`data_size`" octets. Si le nombre de blocs alloués au *inoeud* "`inoeud`" n'est pas suffisant, alloue plus de blocs, et s'il est trop grand, libère les blocs inutiles. Écrit les données de "`data`" bloc par bloc (écrit "`mon_SGF->superbloc->taille_bloc`" octets de données à chaque fois) jusqu'à l'avant dernier bloc. Écrit les données restantes et pas plus, dans le dernier bloc.
Renvoie 0 si tout va bien ou un code d'erreur sinon.

`int lire_donnees_dans_inoeud (char * data, int data_size, int inoeud, SGF_t *mon_SGF)`
copie dans "`data`" les "`data_size`" octets de données lues dans les blocs du *inoeud* "`inoeud`". Lit les données bloc par bloc (lit "`mon_SGF->superbloc->taille_bloc`" octets de données à chaque fois) jusqu'à l'avant dernier bloc. Lit les données restantes et pas plus, dans le dernier bloc.
Renvoie 0 si tout va bien ou un code d'erreur sinon.

`int creer_inoeud (int type_inoeud, SGF_t *mon_SGF)`
crée avec le premier numéro de *inoeud* libre, un *inoeud* de type "`type_inoeud`" et met à jour la table des *inoeuds*.
Renvoie le numéro du *inoeud* créé si tout va bien ou "`NO_INOEUD`" sinon.

`int liberer_inoeud (int inoeud, SGF_t *mon_SGF)`
libère les éventuels blocs utilisés par le *inoeud* `"inoeud"`, met le type du *inoeud* à `"INOEUD_LIBRE"` et sa taille à zéro. Puis met à jour la table des *inoeuds*.
Renvoie 0 si tout va bien ou un code d'erreur sinon.

3 Commandes d'accès au *SGF*

Vous disposez des sources des programmes des commandes suivantes :

`mon_formater` : formate le périphérique de mémoire de masse, simulé dans un fichier, pour y installer le système de fichiers de *mon_SGF*.

`mon_affiche_SGF` : affiche le *SGF* du périphérique de mémoire de masse, simulé dans un fichier.

`mon_ls` : affiche le contenu du répertoire passé en paramètre. La désignation doit obligatoirement être un chemin absolu.

`mon_mkdir` : créé un nouveau répertoire dont la désignation absolue est passée en paramètre.

`mon_rmdir` : efface le répertoire dont la désignation absolue est passée en paramètre.

3.1 `mon_ls`

Vous allez ajouter le paramètre `"-l"` en option à la commande `mon_ls`. Cette option permet d'afficher la taille en octets en plus du nom et du numéro d'*inoeud* de chaque élément du répertoire passé en paramètre.

3.2 `mon_du`

Programmez la commande `mon_du` (`du` pour *disk usage*), qui affiche la somme des tailles des éléments du répertoire passé en paramètre. Si l'élément est un sous-répertoire, on utilise la taille calculée récursivement par `mon_du` sur ce sous-répertoire.

4 Conseils

Nous vous conseillons d'écrire en premier les fonctions : `lire_superbloc` , `lire_table_inoeuds` , `lire_SGF` , `ouvrir_SGF` et `bloc_libre_suivant`, et de les tester en compilant et en exécutant la commande `mon_affiche_SGF` avec le fichier simulant le périphérique de mémoire de masse fourni (`.device_name`).
Vous pourrez tester au fur et à mesure en commentant les parties du code de `mon_affiche_SGF` pour lesquelles les fonctions de `mon_SGF.c` n'ont pas encore été implémentées.

Ensuite, voici dans l'ordre les fonctions à réaliser et les commandes qu'elles permettent de compiler :

fonction `lire_donnees_dans_inoeud` : commande `mon_ls`.

fonctions `allouer_n_blocs_dans_inoeud` , `creer_inoeud` , `ecrire _donnees_dans_inoeud`, `inoeud_libre` et `liberer_blocs_du_inoeud` : commandes `mon_formater` et `mon_mkdir`.

fonction `liberer_inoeud` : commande `mon_rmdir`.

Faite une copie des fichiers fournis pour travailler avec, notamment du fichier `.device_name` qui sera écrasé à chaque fois que vous utiliserez `mon_formater`.

Reportez-vous au `wiki` de la licence pour des informations complémentaires et l'archive contenant les programmes sources et les fichiers fournis.

Important : si vous trouvez un *bug* dans les programmes fournis, vous aurez des points en plus à condition de le signaler assez tôt : 1 point par semaine avant le rendu.