Projet : le système de fichier MiniFS

Généralités

Il s'agit d'implémenter un système de fichiers à la Unix par simulation au-dessus du système de fichiers d'Unix.

Le système de fichier MiniFS

Un système de fichiers MiniFS est constitué par un objet nommé disque MiniFS. Le disque est un ensemble d'objets nommés noeuds destinés à contenir des données. Il existe deux types de noeuds : les répertoires et les fichiers. Un répertoire est un objet pouvant contenir des noms d'autres noeuds (répertoires ou fichiers). Les répertoires servent donc à nommer les objets. Ceci permet de construire sur un disque un arbre d'objets.
La simulation utilisera comme support du disque MiniFS un fichier du système hôte.

Le disque MiniFS

Le disque (il s'agira d'un fichier Unix pour la simulation) est constitué de blocs de taille fixe (unité de lecture/écriture). Ces blocs étant répartis en trois classes : la zone de description, la zone des noeuds et la zone de données, chacune étant de taille fixe et connue. La zone des noeuds contient la liste des noeuds utilisés ou libres ainsi que les attributs associés aux noeuds ; cette zone pouvant etre vue comme un tableau de structure noeud (voir plus bas). La zone de données contient les blocs de données associés aux noeuds voire certaines informations liées aux données (localisation).

```
#define TAILLE_BLOC 128 /* Unité: octets */
#define TAILLE_ZONE_DESCRIPTION 4 /* Unité: blocs */
#define TAILLE_ZONE_NOEUDS 12 /* Unité: blocs */
#define TAILLE_ZONE_DONNEES 240 /* Unite: blocs */
```

La zone des noeuds

Constituée de blocs contigus, elle contient la table des noeuds (donc morcelée par blocs).

```
#define NUMERO_BLOC_DEBUT_ZONE_NOEUDS 2
#define NOEUDS_PAR_BLOCS              (TAILLE_BLOC/sizeof(struct noeud))
#define NOEUD2BLOC(numero_noeud)      (numero_noeud/NOEUDS_PAR_BLOC+NUMERO_BLOC_DEBU
#define NOEUDINBLOC(numero_noeud)     (numero_noeud%NOEUDS_PAR_BLOC)
```

La structure d'un noeud

Un noeud est donc constitué d'attributs et de données. Sa représentation dans la zone des noeuds est de taille fixe et respecte la structure suivante :

```
struct noeud {
  short etat;
  short type;
  short taille;
  short liens;
  short blocs[12];
};
struct stat {
  short etat;
  short type;
  short taille;
  short liens;
};
typedef struct stat *stat_p;
#define NOEUD_LIBRE    0
#define NOEUD_UTILISE  1
```

```
#define EST_LIBRE(pn)  ((pn)->etat==NOEUD_LIBRE)
#define FIC            0
#define REP            1
#define IS_REP(pn)     ((pn)->type==REP)
...
```

Le champ `etat` (considéré comme un booléen) permet de déterminer si le noeud est libre ou utilisé (accessible à travers une référence).
Le champ `type` peut prendre l'une des deux valeurs (au moment de son allocation) : `REP` (pour un noeud répertoire) ou `FIC` (pour un noeud fichier).
Le champ `taille` contient la taille exprimée en octets de l'objet.
Le champ `liens` contient le nombre de noms associés au noeud.
Le champ `blocs` contient les numéros des premiers blocs de données associées au noeud. Le douzième entier pointe sur un bloc de données contenant la suite de la liste des blocs du noeud (le dernier entier de ce bloc pointant lui-meme sur un autre bloc de données contenant la suite de la liste ad lib.)

Les répertoires

Les répertoires sont des noeuds dont les données sont à interpréter de façon très particulière : il s'agit d'une liste de couples nom, numéro de noeud permettant d'associer à un noeud particulier (donc non libre) un nom que l'on pourra utiliser comme élément d'une référence. Une entrée de répertoire (couple) possède la structure suivante :

```
struct entree_repertoire {
  char  nom[30]; /* nom du lien associé au noeud */
  short numero;  /* numero du noeud associé au nom */
};
typedef struct entree_repertoire *entrep_p;
```

Le champ numéro doit contenir un numero de noeud valide lorsque l'entrée est occupée. Si l'entrée a été effacée, numéro doit être à zéro.
Le champ nom contient une chaine de caractères à la C (terminée par ASCII 0), mais jamais de longueur nulle ! De plus les noms ne doivent pas contenir le caractère dit de séparation utilisé pour composer les noms en références.

La zone des données

Constituée de blocs, on peut en distinguer quatre types : les blocs de données des fichiers (sans structure particulière), les blocs de données associés aux répertoires, les blocs de données utilisés pour constituer la liste des blocs d'un noeud et les blocs de données libres.

Réalisation

On devra entres autres :
- Déterminer les structures, valeurs symboliques, macros-définitions manquantes, corriger les existantes.
- Écrire une bibliothèque permettant de lire/écrire des blocs sur le disque (interdiction formelle d'y lire ou écrire octet par octet). **Bonus** (facultatif) : ajouter un système de caches pour les lectures/écritures sur le disques logique.
- Écrire une commande

  `makeminifs <fichier>`

  permettant de créer un disque MiniFS.
- Écrire une bibliothèque contenant au moins les fonctions suivantes :
  - `desc_t open_minifs(char *reference,int mode)`
    permettant d'ouvrir le noeud associé à la référence. Les modes possibles sont un parmi `M_LEC`, `M_ECR`, `M_EL` combiné avec `M_CRE`, `M_ZER` (troncature en cas d'écriture). Seul le mode `M_LEC` est autorisé sur un répertoire.
  - `int read_minifs(desc_t descripteur,void *tampon,int taille)`
    permettant d'effectuer une lecture.
  - `int write_minifs(desc_t descripteur, void *tampon,int taille)`
    permettant d'effectuer une écriture.
  - `pos_t lseek_minifs(desc_t descripteur,pos_t position,int mode)`
    permettant de déplacer le pointeur courant associé au descripteur. La position étant mesurée par rapport au début

du fichier, si l'on utilise `DEBUT`, par rapport à la position courante si l'on utilise `COURANT` et par rapport à la fin du fichier si l'on utilise `FIN` comme troisième argument.
- `int close_minifs(desc_t descripteur)`
  permettant de fermer un fichier précedemment ouvert par `open_minifs`.
- `entrep_p readdir_minifs(desc_t descripteur)`
  permettant de lire l'entrée courante du répertoire ouvert par `open_minifs`.
- et autres :

```
int link_minifs(char *src,char *dst)
int unlink_minifs(char *src)
int rmdir_minifs(char *reference)
int mkdir_minifs(char *reference)
int stat_minifs(char *reference,stat_p buf)
```

Les différentes fonctions devront etre conformes à la spécification suivante :
- en cas de succès la valeur renvoyée doit etre positive ou nulle (pour un entier ou type compatible), ou un pointeur différent de `NULL` (pour un pointeur),
- en cas d'échec la valeur renvoyée doit etre égale à `-1` (pour un entier) ou `NULL` (pour un pointeur). De plus la fonction `int cause()` doit renvoyer une valeur représentant la cause de l'erreur :

```
#define ENTREE__REPERTOIRE_INCONNUE 1
#define DISQUE_INCONNU              2
#define DESCRIPTEUR_INCONNU         3
#define MAUVAISE_POSITION           4
...
```

- Écrire un programme

  `shell_minifs <fichier>`

  permettant de manipuler un disque MiniFS. Il s'agira d'une "sorte" de shell (très simple) permettant d'effectuer quelques opérations comme :
  - Création et destruction d'un répertoire : : `mkdir <rep>`, `rmdir <rep>`,
  - création et destruction d'un fichier `touch <fic>`, `rm <fic>`
  - Écriture dans un fichier : `fecho <fic> str`. Cette opération simulera l'usuelle commande shell `echo str > fic`.
  - Affichage du contenu d'un fichier : `cat <fic>`, `ls <rep>`.
  - Copie d'un fichier : `cp <fichier> <rep>|<fichier>`
  - Création d'un lien dur : `ln <src> <dst>`
  - Affichage et deplacement du repertoire courant : `pwd`, `cd <rep>`
  - et autant que l'on jugera nécessaire qui permettront de manipuler le système de fichiers...

Modalités de travail.

Ce projet devra être réalisé en langage C en utilisant les appels systèmes POSIX étudiées en cours (essentiellemet `open, read, write, close`. Il devra impérativement fonctionner sur les machines du CMI. Le travail pourra être effectué par groupe de trois étudiants au plus. Inclure les noms des étudiants dans le fichier `README` et dans chaque fichier source.

Modalités de soumission.

Préparer un répertoire contenant le code source, la documentation (fichier `README` en format texte), un fichier `Makefile`, et rien d'autre. Ce répertoire sera comprimé au format `.tgz` et rendu avant la soutenance. Les dates de remise des rapports et de soutenance seront précisées ultérieurement.

Modalités d'évaluation

Dans l'évaluation de votre projet importance particulière sera donnée aux critères suivants :

Documentation. Votre projet contiendra un fichier `README` contenant :
- La description du projet et des prorammes adressé à un utilisateur non-expert.
- Une description concise du projet (adressés aux évaluateurs) décrivant les choix d'implémentation, les structures de données utilisés, les résultats accomplis par rapport aux taches décrites ci-dessus.

Qualité du Code C. Le code source sera bien commenté, les noms des variables, des fonctions, des structures, et des types seront bien choisis. On donnera préférence à des fonction de petite taille. On utilisera de macro-constantes aux lieux des constantes explicites.

Organisation du programme en unités logiques. Chaque unité logique sera contenue dans un fichier séparé `nom.c` (avec `nom` est bien choisi) ayant un fichier en tête `nom.h` par défaut. La liste des unités logiques (donc la liste du code source) apparaîtra dans le fichier `README`, la compilation sera géré par le programme `make` à l'aide d'un fichier `Makefile`.

Fonctionnement de votre code. Le code source sera compilé au moment de l'évaluation, possiblement sur des machines différentes des machines au CMI. Écrivez donc votre code en suivant le standard POSIX.

Votre projet peut être évalué en votre absence. Ajouté des explications éventuelles dans le fichier `README`. Votre présence est par contre nécessaire au moment de la soutenace.

Recommandations

Il est recommandé de procéder avec beaucoup d'attention... Demandez-vous à quoi sert la zone de description d'un disque, ce que l'on y trouvera, où est placé le noeud racine d'un disque, etc. N'hésitez pas à rajouter ce qui semble manquer (à condition d'y réflechir à deux fois).
De plus, il est rappellé que les enseignants sont normalement disposés à répondre à vos questions, profitez-en.
S'il est conseillé d'échanger des informations entre différents groupes de travail, il n'est pas conseillé de copier les uns sur les autres. Le sujet est suffisamment libre d'interprétation pour que chaque groupe trouve une solution originale à chaque problème.