Programmation Système Avancée

TP3 – Système de fichiers

Objectif

Le but de ce TP est de continuer le développement de votre kernel en implémentant un système de fichiers très rudimentaire. A la fin de ce travail pratique vous aurez atteints trois objectifs :

1. Conception d'un système de fichiers simple.
2. Conception d'outils permettant de manipuler une image d'un système de fichiers.
3. Implémentation du support du système de fichiers au sein du kernel.
    - API d'accès aux fichier via le concept de descripteurs de fichiers.

A la fin de ce TP, votre kernel sera donc capable de :

- lire un fichier - entièrement ou partiellement ;
- déterminer si un fichier existe ;
- obtenir des informations sur un fichier ;
- itérer sur les fichiers d'un système de fichiers.

Pour parvenir à ces différents objectif, ce TP se décompose en deux parties.

Développement des outils côté système « hôte »

La première partie est dédiée au développement des outils permettant de construire l'image du système de fichiers qui sera utilisée par votre kernel. L'image du système de fichiers ainsi créée sera passée à QEMU qui l'interprétera comme un disque physique. Pour manipuler cette image, il vous faudra implémenter les outils nécessaires : création d'une image vide, insertion de fichiers au sein de l'image, etc. Ces outils seront des programmes s'exécutant sur le système hôte, en l'occurence GNU Linux.

Développement de l'accès au système de fichiers depuis le kernel

La deuxième partie du TP est dédiée à l'implémentation des fonctions systèmes permettant au kernel de manipuler le système de fichiers : ouverture, lecture, itération sur les fichiers, etc. La lecture de fichiers s'inspire de la sémantique POSIX pour l'accès aux fichiers (open, read, close, etc.).

Structure du système de fichiers

L'étude des systèmes de fichiers est un sujet très vaste pouvant aisément occuper un semestre entier.

Le but de ce projet est de concevoir et implémenter un système de fichiers extrêmement simple mais suffisamment abouti et flexible pour être réellement utilisé dans votre système d'exploitation. Pour des raisons de temps, il n'est pas nécessaire que votre système de fichiers soit hiérarchique. Le système de fichiers à concevoir doit toutefois satisfaire aux propriétés suivantes :

- Le système de fichiers doit commencer par le superblock. Celui-ci doit au moins contenir :
    - la signature du système de fichiers ;
    - le numéro de version du système de fichiers ;
    - le label (ou nom) du système de fichiers ;
    - la taille des blocs de données (multiple de 512).

- Chaque fichier est composé de méta-données (informations liées au fichier) et de blocs de données (contenu du fichier).
- Chaque fichier doit au moins posséder les méta-données suivantes : nom (30 caractères minimum) et taille.
- Un fichier doit pouvoir être facilement créé.
- Un fichier doit pouvoir facilement grandir (i.e. possibilité d'ajouter des blocs de données).
- Un fichier doit pouvoir facilement être effacé.
- Dans le cas de petits fichiers, éviter une fragmentation interne excessive (typiquement, maximum 4KB par petit fichier).
- Le système de fichiers doit au moins pouvoir gérer des fichiers jusqu'à 256 KB.

Vous êtes complètement libres quand à la structure de votre système de fichiers. Ceci dit, il est impératif que la structure choisie permette de répondre aux contraintes énoncées ci-dessus.

L'unité de base pour le stockage du contenu des fichiers est le bloc. La taille d'un bloc est typiquement définie par l'utilisateur et est toujours un multiple d'un nombre de secteurs. Il est considéré ici que la taille d'un secteur est 512 bytes. Même si ce n'est pas toujours le cas (aujourd'hui beaucoup de disques volumineux possèdent des secteurs de 4KB), c'est ce qui sera considéré dans le cadre de ce projet.

Partie 1 : Outils pour la gestion du système de fichiers

Pour l'instant, votre kernel n'a aucun concept de système de fichiers. Il n'est d'ailleurs pas encore capable de lire un secteur sur le disque (chose qui sera réalisée dans la deuxième partie).

Le but de cette première partie est de développer les outils nécessaires à la manipulation de l'image du système de fichiers. En particulier, vous devrez développer une série d'outils permettant de :

1. Créer l'image d'un système de fichiers vide.
2. Afficher les informations d'une image d'un système de fichiers.
3. Insérer un fichier du système hôte dans l'image d'un système de fichiers (cette dernière doit évidemment d'abord avoir être créée).
4. Lister les fichiers présents dans une image de système de fichiers.
5. Effacer un fichier d'une image d'un système de fichiers.

Ces outils doivent s'exécuter sur le système hôte, soit GNU Linux. Les commandes à implémenter sont définies ci-dessous. A noter que les arguments spécifiés pour chaque commande sont les arguments minimum à passer. Vous êtes libres (et parfois même obligés) d'en ajouter.

- **`fs_create label bs fs.img`**
  - Créé l'image d'un système de fichiers vide dans `fs.img`.
  - `label` spécifie le label du système de fichiers.
  - `bs` spécifie la taille d'un block en byte ; la taille doit être un multiple de la taille d'un secteur (512 bytes).
  - La taille du système de fichiers à créer doit être spécifiable d'une manière ou d'une autre ; à vous de décider comment.
- **`fs_add file fs.img`**
  - Ajoute le fichier `file` à l'image `fs.img`.
- **`fs_del file fs.img`**
  - Supprime le fichier `file` de l'image `fs.img`.

- **`fs_list fs.img`**
  - Liste les fichiers présents dans l'image `fs.img` ; sont listés au minimum le nom et la taille de chaque fichier.
- **`fs_info fs.img`**
  - Affiche les informations sur le système de fichiers ; au minimum le type de système de fichiers, la version, le label, et l'espace utilisé/disponible (représentation libre).

Le code source de ces utilitaires doit figurer dans le répertoire `tools` de l'arborescence du projet. Tous comme pour le répertoire `kernel`, un makefile permettra de les compiler.

N'oubliez pas de développer du code modulaire et réutilisable. Ainsi, il est fort probable qu'une grande partie des utilitaires utilisera des fonctions et structures de données communes.

Pensez à afficher une aide sur la syntaxe de vos outils si ceux-ci sont appelés sans arguments. Assurez-vous aussi que vos outils sont robustes et ne crashent pas en cas de mauvais arguments ou mauvaise manipulation.

Rappelez-vous que les utilitaires *hexdump* ou *hexedit* sont vos amis car ils permettent de facilement inspecter le contenu de fichiers binaires. Ceux-ci vous permettent d'accéder facilement à n'importe quel secteur de l'image de votre système de fichiers (par exemple, pour sauter à un offset spécifique du fichier courant avec *hexedit* : pressez entrée, puis indiquez un offset en bytes et pressez entrée à nouveau).

Partie 2 : Implémentation du système de fichiers dans le kernel

Accès au système de fichiers avec QEMU

Pour accéder au système de fichiers depuis votre kernel, il convient tout d'abord de l'indiquer à QEMU. Lors des TP1 et TP2, vous avez créé une image ISO de votre système d'exploitation. Celle-ci est au format ISO-9660 qui est spécifiquement prévu pour être gravé sur CD-ROM ou DVD-ROM (lecture seule). Quand vous passez l'image ISO à QEMU avec l'option `-cdrom image.iso`, celui-ci émule un CD-ROM ayant le contenu de l'image ISO passée en argument.

Le système de fichiers développé résidera lui sur un disque dur. Pour ceci, il est nécessaire d'indiquer à QEMU que la machine à émuler contient un disque dur connecté sur le premier contrôleur de disque. Ceci se fait avec l'option `-hda fs.img` où `fs.img` est l'image du disque et `hda` représente le disque dur du premier contrôleur.

Accéder au premier disque dur depuis le kernel

Comment peut-on accéder aux secteurs du premier disque dur depuis le kernel ? Le seul moyen de le faire est d'écrire un drivers permettant de gérer le protocole ATA. Afin de vous faciliter la tâche, le code C permettant de lire et écrire des secteurs sur le disque dur connecté au premier contrôleur ATA vous est fourni sur Cyberlearn (archive `ide.tar.gz`). N'utilisant pas le DMA, ce code est peu efficace mais ce n'est pas un problème dans le cadre du TP. A noter que ce code utilise les fonctions d'accès aux ports d'entrée/sortie (PIO) implémentées lors du TP1.

Support du système de fichiers dans le kernel

Le kernel ne possèdant aucun concept lié au système de fichiers, il est nécessaire d'en implémenter le support. Ceci est à réaliser selon une API clairement définie et indépendante de la structure sur disque du système de fichiers. Les fonctions à implémenter dans le kernel sont définies ci-dessous :

- **`int file_stat(char *filename, stat_t *stat);`**

Renvoie dans `stat` les méta-informations liées au fichier passé en argument ; la structure `stat_t` doit contenir au minimum le champ `size` qui est la taille du fichier. Retourne 0 en cas de succès et -1 en cas d'échec.

- **`bool file_exists(char *filename);`**
  Renvoie `true` si le fichier passé en argument existe.
- **`int file_open(char *filename);`**
  Ouvre un fichier et renvoie un descripteur de fichier pour y accéder ou -1 en cas d'échec.
- **`int file_read(int fd, void *buf, uint count);`**
  Essaie de lire `count` bytes depuis le fichier référencé par `fd` et les place dans le buffer `buf`. Renvoie le nombre de bytes lus, ou 0 en cas de fin de fichier, ou -1 en cas d'erreur.
- **`int file_seek(int fd, uint offset);`**
  Positionne la position pointeur du fichier ouvert (référencé par le descripteur `fd`) à `offset` par rapport au début du fichier. Renvoie la nouvelle position ou -1 en cas d'échec.
- **`void file_close(int fd);`**
  Ferme le fichier référencé par le descripteur `fd`.
- **`file_iterator_t file_iterator();`**
  Créé un itérateur permettant d'itérer sur les fichiers du système de fichiers.
- **`bool file_has_next(file_iterator_t *it);`**
  Renvoie `true` si il y a encore un fichier sur lequel itérer.
- **`void file_next(char *filename, file_iterator_t *it);`**
  Copie dans filename le nom du prochain fichier pointé par l'itérateur.

Typiquement, voici comment itérer sur les fichiers du système de fichiers :

```
char filename[MAX_FILENAME_LENGTH];
stat_t st;
file_iterator_t it = file_iterator();
while (file_has_next(&it)) {
     file_next(filename, &it);
     file_stat(filename, &st);
     printf("%s (%d bytes)\n", filename, st.size);
}
```

En ce qui concerne les fonctions manipulant des descripteurs de fichiers, le kernel maintiendra une table des descripteurs de fichiers ouverts. Une tâche (ou le kernel) aura donc la possibilité d'ouvrir plusieurs fichiers (voir le même, plusieurs fois en lecture) « en même temps » et obtenir un descripteur de fichier par fichier ouvert. Ainsi, à tout descripteur de fichier sera associé une position dans le flux de bytes de chaque fichier ouvert.

Splash screen

Afin de rendre votre kernel plus attractif, modifiez votre kernel afin qu'il affiche un « *splash screen* » à la manière de GNU Linux au moment du démarrage. Vous pouvez par exemple afficher un logo en mode texte[1] ou alors un message en ASCII Art[2] affichant un message de bienvenue à l'utilisateur. Ce logo ou *splash screen* sera chargé depuis un fichier se trouvant dans le système de

1 http://www.glassgiant.com/ascii/
2 http://patorjk.com/software/taag

fichiers. Le logo pourra ainsi être changé sans avoir à toucher au kernel.

Make

Pensez à optimiser les dépendances de votre projet afin que lorsque l'utilisateur tape « `make run` », rien ne soit recréé ou recompilé inutilement. L'image du système de fichiers doit être générée à la volée (incluant le splash screen, fichiers divers, etc.) avant que l'OS démarre. Si un fichier la composant est modifié, alors l'image du système de fichiers doit être regénérée.

Conseils

Commencez d'abord par réfléchir à la conception de votre système de fichiers : quelle politique d'allocation utiliser ? Comment les information seront-elles stockées sur le disque ? Etc.

Ensuite, développez les outils pour la gestion de l'image de votre système de fichiers. Vérifiez que vos outils fonctionnent correctement avec *hexedit*. Une fois validé, commencez le support du système de fichiers au sein du kernel.

Attention à la sérialisation des structures : par défaut celles-si peuvent être « paddées » par le compilateur pour des raisons de performances. Pour s'assurer que le compilateur n'ajoute aucun padding, utilisez l'attribut `__attribute__((packed))`.

Travail à rendre

Vous me rendrez, sur Cyberlearn, l'arborescence complète de votre projet (code source complet depuis le TP1) selon les formalités de rendu décrites dans le document « Consignes pour les travaux pratiques ».

L'arborescence de votre projet doit respecter les consignes décrites dans « Consignes pour les travaux pratiques » et le code respecter celles décrites dans « Consignes générales pour l'écriture du code » et « Consignes pour l'écriture du code C ».

Dans votre archive, vous me rendrez aussi un rapport décrivant clairement et de manière détaillée :

- La structure du système de fichiers, en particulier le format sur disque.
- Des exemples illustrant comment sont stockés des fichiers de tailles différentes (petits, grands, etc.).
- Il faut qu'une personne externe au projet puisse facilement comprendre comment sont stockés les fichiers (meta-données et contenu).
- Les avantages et inconvénients du système de fichiers choisi.