Système d'exploitation – TD4
Systèmes de fichiers

1 Taille et occupation disque

Les résultats observés dans les questions ci-dessous peuvent varier selon le type de système de fichiers utilisé (ext4, ext3, NFS, tmpfs, ...). Ouvrir `/proc/mounts` et trouver les différents points de montage pour savoir quel système de fichiers vous allez utiliser ensuite.
Pour éviter d'écrire plein de programmes en C, on pourra utiliser l'outil `dd` pour remplir des bouts de fichier. Par exemple, pour copier 1 bloc (`count`) de taille 1 (`bs`) du fichier `/dev/zero` (`if`) vers le fichier `fichier` (`of`) en sautant 1000000 blocs (`seek`), on pourra par exemple utiliser :

```
dd if=/dev/zero of=fichier count=1 bs=1 seek=1000000
```

1.a) Créez un fichier contenant un seul caractère. Observez sa taille avec les commande `du -h` et `stat`. Expliquez les différents chiffres observés.

1.b) Agrandissez votre fichier jusqu'à ce que l'occupation disque augmente et expliquez. Continuez plusieurs fois et essayez de prédire l'occupation disque en fonction de la taille réelle. Agrandissez le fichier jusqu'à des mega voire des gigaoctets. Votre prédiction est-elle bonne ? Pourquoi le système de fichiers se comporte-t-il ainsi ?

1.c) Créez maintenant un fichier de taille nulle et expliquez comment son occupation disque peut être nulle.

1.d) Créez maintenant un fichier ne contenant qu'un octet à la position 10000. Expliquez sa taille et son occupation disque. Et si on remplace 10000 par 10 millions, ou 1000 milliards ?

2 Système de fichiers virtuels avec FUSE

Les fichiers sources sont disponibles à côté du sujet de TD sur la page du cours.

Cette partie nécessite des droits spéciaux, indisponible sur les machines ENSEIRB. Pour vos portables, des images de machine virtuelle pour KVM et VirtualBox ont été préconfigurées [1].

Sinon vous pouvez utiliser vos portables sans machine virtuelle, mais il faudra installer les paquets de développement de FUSE pour avoir `/usr/include/fuse.h` (par exemple le paquet `libfuse-dev` sous Debian/Ubuntu). Et ensuite vérifiez que vous êtes dans le groupe `fuse` ou que vous avez accès à `/dev/fuse`.

FUSE (Filesystem in USErspace) permet d'implémenter un système de fichiers sans avoir à écrire un module noyau. Un pilote existant (module noyau fuse) se charge de transférer les requêtes utilisateur via le VFS vers un processus qui exécute notre code. Les appel-système se traduisent donc en appel à des fonctions que nous allons définir. Cela permet d'implémenter des systèmes fichiers exotiques, par exemple la lecture d'un fichier tar.gz sans le décompresser.

1. Voir la page `goglin/teaching/Systeme/VM.xhtml` à coté de la page du cours.

2.1 Système de fichiers virtuels Hello

2.a) Téléchargez les sources en ligne et compilez-les. Montez le système de fichiers virtuels hello sur le répertoire `mnt` en lançant `./hello mnt`.

Les logs du système de fichiers sont envoyés dans le fichier `hello.log` dans le répertoire courant. Affichez-les au fur et à mesure dans un autre terminal avec `tail -f hello.log`.

Listez le contenu du répertoire du montage, affichez le contenu du fichier et expliquez ce qui se passe ainsi que le contenu des logs.

Pour deboguer, on pourra attacher gdb après montage avec `gdb -p $(pidof hello)`. Enfin, on démontera le système de fichiers plus tard avec `fusermount -u mnt`.

2.b) Ajoutez un second fichier virtuel avec un contenu différent.

2.2 Système de fichiers virtuels grepfs

Nous utilisons maintenant FUSE pour créer un système de fichiers appliquant la commande `grep` de manière transparente. On travaillera sur un seul répertoire dont les sous-répertoires sont ignorés. Si le fichier `toto` existe dans ce répertoire, il apparaîtra tel quel dans le système de fichiers. Il apparaîtra également dans `foo/toto` sous la forme d'un fichier dont le contenu est restreint aux lignes contenant `foo`. Et également dans `foo/bar/toto` avec uniquement les lignes contenant `foo` et `bar`.

2.c) Montez le système de fichiers sur le répertoire `mnt` avec `./grepfs . mnt` et affichez les logs dans un autre terminal avec `tail -f ./grepfs.log`. Expliquez la sortie de `ls -l mnt`, `ls mnt/foo/bar` et `cat mnt/foo/bar/grepfs.c`.

2.d) Complétez l'implémentation de `grep_realpath()` et `grep_getattr()` pour récupérer les vrais attributs des vrais fichiers, et si ça a échoué, regarder le répertoire. On utilisera `LOG()` si nécessaire pour vérifier que `stat()` est bien appelé sur les bons fichiers. Expliquer la sortie de `ls mnt/foo/bar` et `cat mnt/foo/bar/grepfs.c`.

2.e) Dans `grep_read()`, parsez le chemin pour découvrir les tags au début. Par exemple, on extraira `foo` de `mnt/foo/grepfs.c` ou `foo` et `bar` de `mnt/foo/bar/grepfs.c`. Construisez la commande correspondante à lancer avec `popen()`.

2.f) Améliorez `readdir()` pour n'afficher que les fichiers, pas les sous-répertoires.
Ensuite n'affichez que les fichiers dont le contenu matche le chemin. Par exemple `ls mnt/foo/` ne doit lister que les fichiers qui contiennent `foo`.

3 Accès concurrents

Soient les deux processus :

```
r ()                                        /* Processus R */
{
  int             fd;
  char            buf1[512];
  char            buf2[512];

  fd = open ("test", O_RDONLY);
  read (fd, buf1, sizeof (buf1));           /* Lecture R1 */
  read (fd, buf2, sizeof (buf2));           /* Lecture R2 */
  close (fd);
}
```

```
w ()                                    /* Processus W */
{
  int             fd;
  char            buf[512];
  int             i;

  fd = open ("test", O_WRONLY);
  for (i = 0; i < 512; i ++)
    buf[i] = 'a';
  write (fd, buf, sizeof (buf));        /* Écriture W1 */
  for (i = 0; i < 512; i ++)
    buf[i] = 'b';
  write (fd, buf, sizeof (buf));        /* Écriture W2 */
  close (fd);
}
```

Le fichier test contient initialement 1024 caractères 'c'. On lance ces deux processus en parallèle.

3.a) Que peut-il se passer ? Quels peuvent être les états du fichier et des tampons buf des deux processus après chacune des entrées/sorties ?

3.b) Peut-on avoir dans un tampon buf du processus r une suite du genre aaa...aaaccc...ccc ou bbb...bbbccc...ccc ?

3.c) Peut-on avoir un appel système "atomique" ? Est-ce souhaitable ? Est-ce raisonnable ? Comment le système peut-il gérer cela ?

3.d) Comment peut-on modifier ces deux processus pour s'assurer que le processus qui fait la première entrée/sortie fasse sa seconde avant l'autre processus ? En d'autres termes, comment deux processus coopérants peuvent-ils assurer leur exclusion mutuelle d'accés au fichier ?