TD/TP : Programmation système avec le système de fichiers Linux
Accès bas niveau aux fichiers

IUT Informatique

2014/2015

1 Introduction

Chaque programme en exécution (un processus) possède 3 fichiers qui lui sont associés. Ces fichiers sont identifiés par des descripteurs. Un descripteur de fichier est un entier qui permet d'accéder au fichier.

Au démarrage d'un processus, voici les 3 descripteurs qui lui sont automatiquement associés :

- 0 : entrée standard
- 1 : sortie standard
- 2 : sortie d'erreurs

On peut avoir d'autres descripteurs en ouvrant des fichiers avec la fonction open.

1.1 La fonction write

La signature de la fonction write est :

```
#include <unistd.h>
size_t write(int fildes, const void *buf, size_t nbytes);
```

Cette fonction va écrire nbytes à partir de l'adresse *buf* dans le fichier identifié par le descripteur *filedes*. Cette fonction retourne le nombre de bytes (octets) effectivement écrits. Si 0 est retourné alors aucun octet n'a été écrit ; un retour de -1 indique une erreur et la description de l'erreur se trouvera dans la variable globale *errno*.

1.2 la fonction write : premier exercice

Objectif : savoir compiler un programme C

```
#include <unistd.h>
#include <stdlib.h>

int main()
{
if ((write(1, "Bonjour\n", 8)) != 8)
write(2, "Erreur sur le descripteur 1\n",28);
exit(0);
}
```

Compilez et testez ce programme !

1.3 read

```
#include <unistd.h>
size_t read(int fildes, void *buf, size_t nbytes);
```

La fonction *read* va lire *nbytes* octets du fichier indiqué par le descripteur *fildes* et copier ces octets à l'adresse indiquée par *buf*.

1.4 Exercice

```
#include <unistd.h>
#include <stdlib.h>
int main()
{
char buffer[128];
int nread;
nread = read(0, buffer, 128);
if (nread == -1)
write(2, "Erreur\n", 7);
if ((write(1,buffer,nread)) != nread)
write(2, "Erreur\n",7);
exit(0);
}
```

- Veuillez saisir et compiler ce programme
- Testez ce programme avec en utilisant les commandes suivantes :
  - `echo "ceci est un test" | ./mon_programme`
  - `./mon_programme <fichier_test.txt`

2 La fonction open

```
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
int open(const char *path, int oflags);
int open(const char *path, int oflags, mode_t mode);
```

La fonction open rend un service assez simple mais très utile : elle transforme un chemin d'un fichier en un descripteur manipulable par les fonctions C.

Si cette transformation est réussie, le descripteur du fichier est retourné. Le descripteur de fichier retourné est unique et cette valeur ne peut pas être partagée donc avec d'autre processus en cours d'exécution. Si deux processus ouvrent le même fichier, ils auront deux valeurs distinctes de descripteurs.

Le chemin du fichier est spécifié dans la variable *path*. *oflags* indique les actions à entreprendre à l'ouverture du fichier. On spécifie ces actions avec une combinaison d'actions obligatoires et d'autres optionnelles.

Les actions obligatoires concernent le type d'ouverture :

- O_RDONLY : ouvre le fichier en lecture seulement

– O_WRONLY : ouvre le fichier en écriture seulement
– O_RDWR : ouvre le fichier en écriture et lecture

Les actions optionnelles sont :
– O_APPEND : les données écrites sont placées à la fin du fichier
– O_TRUNC : la longueur du fichier est mise à zéro
– O_CREAT : création du fichier si c'est nécessaire. Les permissions initiales sont indiquées par la variable *mode*.
– O_EXCL : utilisé avec O_CREAT pour assurer que l'appel de la fonction va effectivement créer le fichier.

Pour faire une combinaison des commandes (obligatoires et optionnelles) on utilise |. Par exemple : O_CREAT|O_WRONLY|O_TRUNC, ouvre un fichier en écriture, avec création si c'est nécessaire, et met son contenu à zéro.

2.1 Création de fichier avec des permissions initiales

On peut également spécifier les droits associés à un fichier lors de sa création. On utilise pour cela un certain nombre d'indicateurs de droits qu'on peut combiner avec l'opérateur |

Les indicateurs de droits sont :
– S_IRUSR : permission de lecture pour le propriétaire
– S_IWUSR : permission d'écriture pour le propriétaire
– S_IXUSR : permission d'exécution pour le propriétaire
– S_IRGRP : permission de lecture pour le groupe
– S_IWGRP : permission d'écriture pour le groupe
– S_IXGRP : permission d'exécution pour le groupe
– S_IROTH : permission de lecture pour les autres
– S_IWOTH : permission d'écriture pour les autres
– S_IXOTH : permission d'exécution pour les autres

Par exemple :

```
open ("monfichier", O_CREAT, S_IRUSR|S_IXOTH);
```

Demande la création du fichier si c'est nécessaire des droits de lecture pour le propriétaire et exécution pour les autres.

3 La fonction close

```
#include <unistd.h>
int close(int fildes);
```

Cette fonction permet de mettre fin à l'association entre le descripteur et le fichier. La valeur du descripteur devient alors libre et utilisable pour un autre fichier.

4 La fonction ioctl

La fonction *ioctl* permet de contrôler le comportement des services et des équipements en indiquant des commandes et les paramètres de ces commandes.

```
#include <unistd.h>
int ioctl(int fildes, int cmd, ...);
```

Pour avoir la liste exacte des commandes et les paramètres des commandes il faut se reporter à la documentation de chaque service ou équipement.

5 Réalisation d'un programme de copie

5.1 Copie caractère par caractère

Réalisez un programme copie_caractere.c qui copie un fichier vers un autre caractère par caractère. Le fichier source sera appelé 'ficher.in' et la copie 'fichier.out'. le fichier 'fichier.out' devra avoir les droits de lecture et d'exécution pour l'utilisateur.

5.2 Copie par blocs

Réalisez un programme copie_bloc.c qui copie un fichier vers un autre bloc par bloc. Le bloc sera représenté par un tableau de 1024 caractères (char bloc[1024])