TP1 Système de fichiers

nicolas.ollinger@univ-orleans.fr

22 septembre 2017

Contrôle continu : *Sont à rendre selon les modalités détaillées sur la page celene, une archive contenant la bibliothèque de la partie 2 modifiée par vos soin et l'utilitaire* `bcp`*.*

1 Utilitaires POSIX

Exercice 1. À l'aide des fonction POSIX `mkdir`, `rmdir` et `link`, écrivez les programmes suivants qui seront réutilisés dans les TP suivants :

`mkdir directory_name` crée le répertoire passé en argument avec le mode `0777` ;

`rmdir directory_name` supprime le répertoire vide passé en argument ;

`link source_file target_file` crée `target_file`, un lien physique vers `source_file`.

N'oubliez pas de gérer les cas d'erreur à l'aide de `perror`.

Exercice 2. Écrivez une fonction `copy(const char *src, const char *dst, mode_t mode)` qui copie le fichier de chemin d'accès `src` vers le chemin d'accès `dst` avec les droits `mode` en utilisant les fonctions POSIX `open`, `read`, `write` et `close`. Utilisez cette fonction pour écrire le programme `cp source_file target_file` qui sera réutilisé dans un TP suivant.

Exercice 3. Écrivez le programme `mv source_file target_file` qui sera réutilisé dans un TP suivant à l'aide de la fonction POSIX `rename` et, lorsqu'elle ne fonctionne pas, de la fonction `copy` de l'exercice précédent et de la fonction POSIX `unlink`.

2 Entrées/sorties bufferisées

L'archive `bio.tar.gz` (à décompresser avec la commande `tar`) contient un squelette de fonctions de gestion d'entrées/sorties bufferisées. Décompressez cette archive et, à l'aide de la commande `make`, compilez le programme `cat`. Exécutez le programme `./cat` obtenu sur `cat.c` et `bio.c`. Comparez les résultats avec le "vrai" `cat`. Le programme est buggué !

GDB, pour GNU DeBugger, est un programme permettant d'inspecter l'exécution d'autres programmes. Ceci est très utile lorsque vous avez un programme qui ne fonctionne pas correctement pour comprendre d'où vient l'erreur. Nous allons l'utiliser pour corriger le programme `cat`.

Avant toute chose, il convient de recompiler le programme avec l'option `-g` qui ajoute les informations de débuggage au programme. Une fois cela fait, il suffit de charger le programme dans GDB à l'aide de la commande `gdb cat`. Vous trouverez en annexe la liste des commandes principales de GDB ainsi que des exemples d'utilisation.

Exercice 4. À l'aide de GDB, corrigez la bibliothèque `bio.c` fournie. Vérifiez qu'alors le programme `cat` fourni fonctionne correctement.

Exercice 5. Ajoutez à la bibliothèque une fonction d'écriture ainsi qu'une fonction de vidage de tampon, de prototypes respectifs `ssize_t bwrite(void *buf, ssize_t size, BFILE *stream)` et `int bflush(BFILE *stream)`. Modifiez aussi la structure de données `BFILE` et la fonction `bopen` pour tenir compte des différents modes d'accès : lecture seule, écriture seule et lecture/écriture.

Exercice 6. Écrivez l'utilitaire `bcp` qui copie un fichier en utilisant `bopen`, `bread`, `bwrite`, `beof` et `bclose`. On prendra soin de faire de la taille du tampon utilisée un paramètre défini avec `#define`.

Exercice 7 (*facultatif*)**.** À l'aide de `time` (`man time`) et `gnuplot` (`man gnuplot`), comparez les trois implémentation de `cp` que vous possédez en exécutant dans `/tmp` un grand nombre de copies de fichiers de tailles conséquentes, en faisant varier, en paramètre, la taille du tampon utilisé par votre programme. Tracez les trois courbes obtenues.

3 Manipulation de l'environnement

Exercice 8. L'environnement courant est accessible à travers la variable `char **environ;`. Réécrivez la fonction `char *getenv(const char *name);` en accédant directement à `environ`.

Exercice 9 (*facultatif*)**.** Saurez-vous réécrire, en faisant attention à la manipulation de la mémoire, la fonction `int setenv(const char *envname, const char *envval, int overwrite);` ? Il convient d'utiliser `calloc`, `realloc` et des variables statiques.

Références

[POSIX 2008] *The Open Group Base Specifications Issue 7.* IEEE Std 1003.1-2008. 2001–2008. http://www.opengroup.org/onlinepubs/9699919799/

[MAN] *NetBSD Manual Pages.* http://man.netbsd.org/

A GDB

A.1 Quelques commandes

Voici quelques commandes principales de GDB ainsi que des exemples d'utilisation. Ces commandes possèdent une notation courte (d'une ou deux lettres) et une notation longue.

`h [cmd] help` affiche l'aide sur la commande `cmd`

`r [args] run` lance le programme avec les arguments `args`

`c continue` continue l'execution du programme (après un breakpoint par exemple)

`s [count] step` Avance l'exécution du programme de `count` lignes (1 par défaut).

`n [count] next` Même chose que step mais ne prend pas en compte l'appel de sous-fonctions.

`bt backtrace`. Affiche la pile (ainsi que les variables locale si l'argument `full` est donné).

`p [expr] print` Affiche la valeur de l'expression `expr`

`display [expr]` Similaire à `print` mais affiche de nouveau la valeur après chaque `next` ou `step`.

`b [pos] breakpoint` Ajoute un *breakpoint* à la position `pos` (ligne ou fonction).

`wa [var] watch` Surveille la variable var.

`l list` Affiche le code actuellement exécuté.

A.2 Exemple de session

```
gdb ./cat
<snip>

(gdb) b bio.c:bread
Breakpoint 1 at 0x1b20: file bio.c, line 36.

(gdb) run bio.c
Starting program: /home/jlapin/bio/cat bio.c
Reading symbols for shared libraries ++. done

Breakpoint 1, bread (buf=0xbffff4e0, size=100, stream=0x800000) at bio.c:36
36 if ((stream==NULL)||(stream->mode!=BMODE_READ)) {

(gdb) l
31 ssize_t bread(void *buf, ssize_t size, BFILE *stream)
32 {
33 char *ptr;
34 ssize_t more;
35
36 if ((stream==NULL)||(stream->mode!=BMODE_READ)) {
37 errno = EBADF;
38 return 0;
39 }
40 more=size;

(gdb) wa more
Hardware watchpoint 2: more

(gdb) c
```

```
Continuing.
Hardware watchpoint 2: more

Old value = 1927426484
New value = 100
bread (buf=0xbffff4e0, size=100, stream=0x800000) at bio.c:41
41 ptr=buf;


(gdb) bt full
#0  bread (buf=0xbffff4e0, size=100, stream=0x800000) at bio.c:41
ptr = 0x5 <Address 0x5 out of bounds>
more = 100
#1  0x00001e80 in main (argc=2, argv=0xbffff5ac) at cat.c:20
bf = (BFILE *) 0x800000
nb = -1881139340
buf = '\0' <repeats 99 times>
```