Travaux Dirigés Programmation Système: Feuille 1

Informatique 2ème année. ENSEIRB 2014/2015

—*Mathieu Faverge* - `mfaverge@enseirb.fr` —

Quelques aspects de la programmation système

▸**Exercice 1.** *Pages de manuel et sections*

La commande `man(1)` permet de localiser les pages de manuel par mots-clés, indépendamment des sections dans lesquelles elles se trouvent, grâce à l'option `-k`. Lorsqu'on cherche précisément une page dont on connaît le nom, on pourra s'aider de `grep(1)` pour sélectionner les pages dont le nom commence exactement par celui que l'on souhaite. Ex. :

```
$ man -k write | grep ^write
```

Le système d'exploitation découpe certaines sections du manuel en sous-sections étiquetées par des lettres. Dans quelle section se trouve la documentation de la fonction de bibliothèque standard `printf()` ? A quoi correspond(ent) l'(es) autre(s) page(s) de manuel que vous avez trouvée(s) sur `printf` ?

▸**Exercice 2.** *Implémentation des appels systèmes*

Un appel système requiert une instruction assembleur spécifique, par exemple `ta` sur Sparc, `int` sur x86, `trap` sur m68k, `syscall` sur x86_64, ...

Une implémentation de la bibliothèque C standard peut être trouvée dans `/usr/lib/libc.a` ou `/usr/lib/x86_64-linux-gnu/libc.a`. Désassemblez cette bibliothèque avec l'utilitaire `objdump(1) -d` et trouvez des fonctions où des appels système ont lieu. Sans forcément connaître ce langage assembleur, trouvez comment le système d'exploitation reconnaît quel appel système est demandé.

▸**Exercice 3.** *Écriture avec `write(2)`*

L'appel système `write(2)` permet d'écrire dans un fichier. Lisez-en la documentation.

Écrivez un programme qui affiche ses arguments sur la sortie standard en utilisant `write(2)`.

Votre programme devra être écrit dans les règles de l'art : tester les erreurs possibles de `write(2)`, afficher le cas échéant un message avec `perror(3)`.

▸**Exercice 4.** *Écriture d'un entier avec `write(2)`*

Ecrivez un programme C `write_number` qui écrit sur sa sortie standard le contenu mémoire d'un entier long (de type `long`) dont la valeur sera 5, ainsi qu'à la suite le nombre d'octets qui ont été écrits par le premier appel à `write(2)`. Vous pourrez utiliser `od -x` pour observer la sortie de votre programme.

▸**Exercice 5.** *Gestion des erreurs*

`perror()` est-elle un appel système ? Pourquoi ?

▸**Exercice 6.** *Lecture avec `read(2)`*

Écrivez un programme C qui lit du texte sur son entrée standard avec l'appel système `read(2)` et écrit sur sa sortie standard avec `write(2)` le texte lu. Le programme se termine lorsque son entrée standard est fermée (utiliser Ctrl-d dans le terminal pour fermer l'entrée standard d'un processus).

Pour la lecture, on utilisera un tableau de caractères dont la taille sera fixée par une constante (`#define ...`).

▸**Exercice 7.** *Entrée standard*

En utilisant `fstat(2)` et `STDIN_FILENO` comme descripteur, affichez le type de l'entrée standard. Pour cela on utilisera les macros `S_IS...` sur le champ `st_mode` de la structure `stat`.

Affichez le résultat dans les cas suivants :

```
./a.out
./a.out < /etc
./a.out < ./a.out
```

▸**Exercice 8.** *Lecture d'un entier avec* `read(2)`
Écrivez un programme C qui lit sur son entrée standard ce qui est produit par le programme `write_number`*, et l'affiche avec* `printf(3)`*.*

Compilez les deux programmes (écrivain et lecteur) avec `gcc(1)`*, chacun en deux versions : l'une sans option particulière, l'autre avec* `-m32`*, qui produira un exécutable 32 bits. Vérifiez avec* `file(1)` *que vos deux exécutables sont différents.*

Testez toutes les combinaisons possibles de vos programmes, et réfléchissez à des solutions aux divers problèmes que vous aurez rencontrés.

▸**Exercice 9.** *Lecture et écriture d'une structure*
Recommencez en modifiant vos programmes pour écrire et lire au lieu du `long` *une structure du type suivant, où* `c` *contiendra 'a' et* `l` *contiendra 5 :*

```
struct nopad {
    char c;
    long l;
};
```

Après l'avoir analysé, proposez une solution à ce nouveau problème.