TD 2 : Les bibliothèques, pourquoi et comment les utiliser

ED6 - Licence 3 - Université Paris Diderot

Séances du 22 et 29 février 2012

L'objectif de ces travaux dirigés sera de vous familiariser avec la notion de bibliothèque C, en utilisant la bibliothèque C standard de GNU, la bibliothèque readline et éventuellement ncurses, enfin le dernier exercice vous indique les étapes pour créer une bibliothèque C.

Exercice 1 – Quelques fonctions de la Glibc

La bibliothèque standard C

La bibliothèque standard C est une composante essentielle d'un système d'exploitation de type Unix. C'est cette bibliothèque qui est censée fournir tous les appels système, i.e. toutes les fonctions nécessaires pour interagir avec le système d'exploitation (manipulation de fichiers, envoi de signaux entre processus, etc.). La Glibc, bibliothèque standard C de GNU/Linux présente ainsi plus d'un millier de fonctions.

1. Lire la documentation de `strtol(3)` (avec `man 3 strtol` dans un terminal, ou dans Emacs avec `M-x man` suivi par `strtol(3)`). Écrire dans `statistiques.c` une fonction `long* eval_string(char* buf)` qui prend en argument une chaîne de caractères pour laquelle on supposera qu'elle contient une liste d'entiers écrits en base 10 et séparés par des espaces. Cette fonction renverra un tableau contenant dans sa première case le nombre d'entiers lus et dans les suivantes chacun de ces entiers. Vous ne pouvez utiliser que les fonctions `strtol` et `malloc`.
2. Définir une constante `MAX_LINE_LENGTH` égale à 200. Lire la documentation de `fgets(3)`. Écrire une fonction main qui lit une ligne de l'entrée standard (on supposera qu'elle ne contient que des entiers séparés par des espaces) et affiche la moyenne des entiers présents sur celle-ci. Si la ligne a plus de `MAX_LINE_LENGTH` caractères, le programme doit s'arrêter et renvoyer un code d'erreur -1.
3. Préproccessez votre fichier `statistiques.c` avec `gcc -E statistiques.c -o statistiques.i`. Où se trouve le fichier `stdio.h` ? Quel est le code de retour de la commande `grep MAX_LINE_LENGTH statistiques.i` et de `grep MAX_LINE_LENGTH statistiques.c` ?
4. Poursuivez la compilation de votre programme avec `gcc -Wall statistiques.i -o statistiques`. Testez votre programme.

Pour que les programmes puissent exécuter des fonctions de la bibliothèque standard, il existe une version compilée de toutes les fonctions de la bibliothèque standard (habituellement dans le fichier `/lib/libc.so.6`). Lors de la création d'un exécutable, `gcc` effectue automatiquement une liaison dynamique avec cette version compilée de la bibliothèque C standard.

5. Modifier le `main` pour qu'il lise indéfiniment des lignes, en ayant le même comportement. Compilez, testez.
6. Si l'on veut que le programme lise une seule ligne ou qu'au contraire, il en lise indéfiniment, il faut modifier le fichier source, puis le recompiler. Nous allons utiliser les flags d'exécution pour pouvoir faire ce choix non plus à la compilation, mais à l'exécution.

Ainsi, on veut que `./statistiques` lise indéfiniment des lignes, mais `./statistiques -1` ne lira qu'une seule ligne. Si une autre option est lue, le programme doit afficher un message d'erreur explicitant l'utilisation de ce programme. Lire la documentation de `getopt(3)`, puis implémentez ce comportement. A-t-on besoin d'écrire `#include <getopt.h>` ? Compilez, testez.

7. Modifier votre programme pour qu'il affiche la moyenne et/ou l'écart type pour chaque ligne selon que le programme ait un flag `-m` et/ou `-e`. On rappelle que l'écart type d'un ensemble de n éléments $(x_1, \ldots, x_n)$ dont la moyenne est $\bar{x}$ est donnée par la formule $\sqrt{\sum_{i=1}^{n} \frac{(\bar{x} - x_i)^2}{n}}$

   Libérez-vous la mémoire que vous allouez ?

   

   Bien que la Glibc comporte de nombreuses fonctions mathématiques (`sqrt`, `cosl`, `exp`, etc.), elles n'ont pas été compilées dans `/lib/libc.so.6`, mais dans `/lib/libm.so.6` (avec un `m` comme dans "maths"). Comment demander à `gcc` de lier votre exécutable vis-à-vis de ce fichier ?

   Compilez, testez.

8. Compiler statiquement vis-à-vis de la bibliothèque math [1]. Quelle est la taille de l'exécutable obtenu ? En quoi les sorties de `ldd` et `nm` diffèrent-elles sur les exécutables statiques et dynamiques ?

Exercice 2 – Utilisation de la bibliothèque readline

Readline, pour les interfaces en ligne de commande

Sous les systèmes de type Unix, lorsqu'un programme est exécuté, trois flux sont automatiquement créés : l'entrée standard `stdin` (en lecture seule pour le programme) et les sorties standard `stdout` et d'erreur `stderr` (en écriture seule pour le programme). Lorsque le programme est lancé depuis un terminal (avec un interpréteur de commande ou shell), les entrées clavier sont connectées à l'entrée standard (et bien souvent aussi au display afin que l'utilisateur dispose d'un contrôle visuel sur ses entrées clavier), et les sorties standard et d'erreur sont connectés au display. Comme le clavier est relié directement à l'entrée standard, il n'y a pas d'édition possible de la part de l'utilisateur. GNU/readline permet d'ajouter des fonctions d'édition et d'historique à ces interfaces en ligne de commande.

1. Lire la documentation de la fonction `char* readline (const char *prompt)` dans la page de manuel `readline(3)`. Modifiez `statistiques.c` pour utiliser `readline` au lieu de `fgets` pour lire sur l'entrée standard. Compilez, testez.
2. Quels sont les fichiers supplémentaires nécessaires pour compiler votre programme ? Quels sont les fichiers supplémentaires nécessaires pour exécuter votre programme ?
3. Utilisez une macro préprocesseur `USE_READLINE`, qui fait que votre code n'utilise `readline` que si elle est définie (votre code utilise `fgets` lorsqu'elle n'est pas définie.
4. Consulter la documentation —à l'aide de `info readline` dans un terminal ou dans Emacs en cherchant `readline` dans le buffer créé par `C-h i`—et déduisez de la section 2.1 comment utiliser la fonction `add_history` pour activer l'historique. Compilez et testez cette nouvelle fonctionnalité

1. avec l'option `-static` qui doit précéder les `-l` de chaque bibliothèque que l'on souhaite statiquement lier.

Exercice 3 – Utilisation de la bibliothèque ncurses

Ncurses, pour des interfaces en mode texte

La bibliothèque ncurses est une bibliothèque du projet GNU (donc libre) fournissant une API pour le développement d'interfaces en mode texte.
Une interface en mode texte se démarque d'une interface en ligne de commande (comme celle fournie par readline), car l'interface utilisateur occupe la totalité de l'écran, et n'est donc pas limité au traitement ligne par ligne.
Elle offre notamment la possibilité d'afficher un caractère en tout point de l'écran, ce que cet exercice se propose d'exploiter.

1. Consulter la page de manuel de `ncurses(3ncurses)`. Comme beaucoup de bibliothèques, pour être utilisée elle nécessite un appel à une fonction d'initialisation et doit effectuer un appel à une fonction de nettoyage avant que le programme ne termine.
2. À l'aide de la fonction `mvaddnstr` écrire à une position aléatoire sur l'écran une chaîne de caractères.
   Indice : `refresh(3ncurses)`, `rand(3)`
3. Modifiez votre programme pour qu'il effectue 100 fois l'opération suivante : afficher la chaîne à une position aléatoire, attendre 500ms, effacer l'écran.
   Indice : `erase(3ncurses)`, `usleep(3)`
4. Modifiez votre programme pour qu'il boucle jusqu'à ce que l'utilisateur appuie sur une touche
   Indice : `cbreak(3ncurses)`, `nodelay(3ncurses)`, `noecho(3ncurses)`, `getch(3ncurses)`

Exercice 4 – Une (toute) petite bibliothèque partagée

1. Dans un répertoire `libhello`, écrire dans un fichier `hello.c` une fonction `void hello()` qui affiche une salutation [2] sur la sortie standard. Exportez l'entête de cette fonction dans `hello.h`.
2. Nous allons maintenant créer à partir de ce fichier C une bibliothèque dynamique. Compilez avec la commande `gcc -shared hello.c -o libhello.so` [3]
   Quelles sont les sorties de `ldd` et `nm` sur `libhello.so` ?
3. **Dans un autre répertoire** créez un fichier `test-hello.c` qui appelle la fonction `hello` dans le `main`. Quel(s) fichier(s) `.h` faut-il inclure ? Quel type d'`#include` faut-il utiliser ?
4. Préprocessez votre fichier (avec l'option `-E` de `gcc`), est-il normal que le compilateur ne trouve pas `hello.h`. Utilisez l'option `-I` de `gcc` pour corriger ce problème.
5. Compilez maintenant votre fichier `test-hello.c`, à quelle étape votre compilation échoue-t-elle ?
   Il faut montrer au compilateur où se trouve notre bibliothèque : utilisez les options `-l` et `-L` pour lui indiquer `libhello.so`.
6. Que se passe-t-il lorsqu'on essaie de lancer l'exécutable produit ? Que renvoie `ldd` ?
7. Exécutez votre programme (supposément nommé `test-hello`) avec l'invocation shell suivante :
   `LD_LIBRARY_PATH=/home/moi/ed6/tp2/libhello ./test-hello`

2. En panne d'inspiration ? Essayez-donc `C-h C-h` dans Emacs.
3. L'option `-shared` demande à `gcc` de créer un "shared object", cet objet est écrit dans le fichier `libhello.so` : le `.so` indique qu'il s'agit d'une bibliothèque partagée, et le préfixe `lib` permettra à `gcc` de lier vis-à-vis de cet objet.