Master M1 Informatique, UJF Grenoble
TD-TP Bibliothèques d'Entrées-Sorties

Vincent Danjean, Nicolas Fournel, Florent Bouchez

année 2013-2014

Résumé

Cette fiche ne sera traitée que cette semaine. Vous traiterez en dehors des séances prévues les exercices et questions non résolues. Les objectifs de cette séance de TD/TP sont les suivants :

- ▷ développer une bibliothèque d'entrées-sorties de niveau utilisateur
- ▷ fournir un mécanisme de gestion de tampon pour obtenir un résultat similaire aux fonctions de `stdio.h`
- ▷ comprendre la fabrication et l'utilisation d'une bibliothèque dynamique

Les exercices portant la mention "Travail personnel" figurant en fin defeuille de TD sont optionnels : le temps imparti aux une séances de TD ne permettra sans doute pas de les traiter durant celles-ci et les évaluations de l'UE ne porteront pas sur les notions qu'ils permettent d'aborder. Il sont donnés à titre de travail personnel. Si vous ne parvenez pas à les traiter seul n'hésitez pas à demander un peu d'aide à vos enseignants.

1 Introduction

La principale motivation pour construire bibliothèque d'entrées-sorties est de minimiser les appels au noyau du système, en particulier à `read` et `write`. Un tampon est associé à chaque fichier qui aura été ouvert par la bibliothèque. Les opérations de lecture et d'écriture devront essayer au maximum de prendre ou de déposer les données dans le tampon sans avoir à invoquer le système.

2 Une interface de programmation

Cette interface est simplifiée et ne comprend que les 4 fonctions suivantes :

```
#include <stdes.h>

FICHIER *ouvrir(char *nom, char mode);
int fermer(FICHIER *f);
int lire(void *p, unsigned int taille, unsigned int nbelem, FICHIER *f);
int ecrire(void *p, unsigned int taille, unsigned int nbelem, FICHIER *f);
```

La fonction `ouvrir` ouvre un fichier dont le `nom` est passé comme premier paramètre. Le second paramètre indique le `mode` d'ouverture du fichier. Il peut être ouvert en lecture (`mode == 'L'`) ou en écriture (`mode == 'E'`). Aucun autre mode n'est accepté. La fonction `ouvrir` renvoie un pointeur sur un objet de type `FICHIER`. La fonction `ouvrir` renvoie `NULL` si le fichier ne peut pas être ouvert.

La fonction `lire` lit `nbelem` éléments de données tenant sur `taille` octets, depuis le fichier pointé par `f` et les stocke à l'emplacement mémoire pointé par `p`. La fonction `lire` retourne le nombre d'octets lus. Le fichier doit avoir prélablement été ouvert en mode `'L'`.

La fonction `ecrire` écrit `nbelem` éléments de données tenant sur `taille` octets stockés à l'emplacement mémoire pointé par `p` dans le fichier pointé par `f`. La fonction `ecrire` retourne le nombre d'octets écrits. Le fichier doit avoir été ouvert en mode `'E'`.

La fonction `fermer` ferme le fichier pointé par `f`. La structure de donnée allouée à l'ouverture pourra être soit libérée soit réutilisée pour un autre fichier.

Les prototypes de ces 4 fonctions sont définis dans le fichier `stdes.h`. Ce fichier contient également la déclaration du type `FICHIER`.

Question .1. *Définir le contenu du type* `FICHIER`.

Question .2. *Définir la structure de données dans laquelle on retrouve l'ensemble des fichiers qui auront été ouverts par la bibliothèque.*

Question .3. *Définir la fonction* `ouvrir`.

Question .4. *Définir la fonction* `fermer`.

Question .5. *Définir la fonction* `lire`.

Question .6. *Définir la fonction* `ecrire`.

Voici un petit exemple d'utilisation de la bibliothèque. Cet exemple ouvre deux fichiers dont les noms sont passés en paramètre. Ce programme copie le contenu du premier fichier dans le second.

```
#include <unistd.h>
#include "stdes.h"

int main (int argc, char **argv)
{
  FICHIER *f1;
  FICHIER *f2;
  char c;

  if (argc != 3)
      exit (-1);

  f1 = ouvrir (argv[1], 'L');
  if (f1 == NULL)
      exit (-1);

  f2 = ouvrir (argv[2], 'E');
  if (f2 == NULL)
      exit (-1);

  while (lire (&c, 1, 1, f1) == 1)
    {
      ecrire (&c, 1, 1, f2);
    }
  fermer (f1);
  fermer (f2);
}
```

3 Entrées-Sorties formatées

Nous souhaitons maintenant compléter cette interface avec des fonctions d'entrées-sorties formatées (du type `fprintf` ou `fscanf`). Une des particularités de ces fonctions est qu'elles ont un nombre variable de paramètres. Vous utiliserez pour cela les fonctions `va_start`, `va_end` et `va_arg` dont les prototypes se trouvent dans `stdarg.h`. Notre interface va être étendue avec deux nouvelles fonctions ;

```
int fecriref (FICHIER *fp, char *format, ...);

int fliref (FICHIER *fp, char *format, ...);
```

Le contenu des formats est similaire à ceux que vous utilisez avec les fonctions classiques de `<stdio.h>`. Trois types de données vont pouvoir être manipulés.

- ▷ caractère : %c
- ▷ chaîne : %s
- ▷ entier : %d

Voici un petit exemple illustrant l'utilisation de la fonction `fecriref`.

```
#include <unistd.h>
#include "stdes.h"

int main (int argc, char **argv)
{ FICHIER *f1, *f2;
  if (argc != 2) exit (-1);

  f1 = ouvrir (argv[2], 'E');
  if (f1 == NULL) exit (-1);

  fecriref (f1, "  %c  %s 12\n", 'a', "bonjour");
  fecriref (f1, " %d \n", -1257);

  fermer (f1);
}
```

L'exécution du programme génère un fichier dont le contenu est le suivant :

```
bash$ test_format resultat
bash$ cat resultat
  a  bonjour 12
 -1257
bash$
```

Pour implémenter ces deux nouvelles fonctions, vous pourrez vous servir des fonctions `lire` et `ecrire` définies dans la partie précédente.

Question .7. *Donner l'implémentation de la fonction* `fecriref`.

Question .8. *Donner l'implémentation de la fonction* `fliref`.

4 Manipulation de bibliothèques

La dernière partie de cette fiche est dédiée à la génération d'une bibliothèque statique et d'une bibliothèque dynamique. Une bibliothèque statique est reliée au programme exécutable pendant la phase d'édition de lien, c'est à dire avant l'exécution du programme. Une bibliothèque statique est habituellement stockée dans un fichier ayant comme extension `.a`. L'avantage d'une bibliothèque est qu'il n'est plus nécessaire de spécifier la liste des modules objets. Dans le cas de ce TP, la bibliothèque n'est constituée que d'un seul fichier objet mais une bibliothèque peut être constituée à partir d'un nombre arbitraire de fichiers objets. L'inconvénient d'une bibliothèque statique est que le programme exécutable contient la bibliothèque et donc que la taille du programme final peut être importante. Voici la séquence de commande utilisée pour générer une bibliothèque statique :

```
bash$ gcc -c stdes.c
bash$ ar q libstdes.a stdes.o
bash$ gcc -c test_format.c
bash$ gcc -o test_format test_format.o -L. -lstdes
```

La commande `ar` permet de manipuler des bibliothèques statiques. (`man ar` pour obtenir plus d'informations). L'option `-L` permet d'indiquer à `gcc` dans quel répertoire(s) chercher les bibliothèques. L'option `-l` permet de spécifier un nom de bibliothèque statique à intégrer au programme durant l'édition de liens. On ne spécifie ni le préfixe `lib`, ni l'extension `.a`.

Question .9. *Construire le fichier* `Makefile` *générant le programme exécutable avec une bibliothèque statique.*

Une bibliothèque partagée est reliée au programme au moment de son exécution. Le code de la bibliothèque n'est donc pas inclus dans le fichier du programme exécutable. Une des avantages est que si la bibliothèque venait à être modifiée (correction de bug par exemple), la nouvelle version serait prise en compte sans qu'il n'y ait d'édition de lien à refaire sur les programmes utilisant cette bibliothèque partagée. Cette caractéristique peut aussi être un inconvénient si la signature des fonctions de la bibliothèque est changée : dans ce cas il faut corriger et recompiler les programme qui l'utilisent afin qu'ils fonctionnent. En outre une bibliothèque partagée doit être installée dans le système pour qu'un programme qui l'utilise puisse s'en servir.

La commande `ldd` affiche les bibliothèques partagées nécessaires pour l'exécution d'un programme. La variable d'environnement `LD_LIBRARY_PATH` permet de spécifier une liste de répertoires dans lesquels sont recherchées les bibliothèques partagées. Sur l'exemple suivant, nous avons ajouté le répertoire de travail comme répertoire où devront être recherchées les bibliothèques partagées.

```
bash$ gcc -c stdes.c
bash$ gcc -shared -o libstdes.so.1 stdes.o
bash$ gcc -c test_format
bash$ gcc -o test_format test_format.o libstdes.so.1
bash$ test_format resultat
test_format: error while loading shared libraries:
libstdes.so.1: cannot open shared object file: No such file or directory
bash$ ldd test_format
linux-gate.so.1 =>  (0xffffe000)
libstdes.so.1 => not found
libc.so.6 => /lib/i686/cmov/libc.so.6 (0xb7e63000)
/lib/ld-linux.so.2 (0xb7fc6000)
bash$ export LD_LIBRARY_PATH=.
bash$ ldd test_format
linux-gate.so.1 =>  (0xffffe000)
libstdes.so.1 => ./libstdes.so.1 (0xb7faf000)
libc.so.6 => /lib/i686/cmov/libc.so.6 (0xb7e4f000)
/lib/ld-linux.so.2 (0xb7fcd000)
bash$ test_format resultat
bash$ cat resultat
  a  bonjour 12
 -1257
bash$
```

Question .10. *Construire le fichier* `Makefile` *générant le programme exécutable avec une bibliothèque dynamique.*