POLYTECH NICE-SOPHIA

SI 3 - 2013-2014

Erick Gallesio

Système

TP à rendre

Implémentation de primitives de haut niveau sur les fichiers

Les exercices de cette feuille consistent à réécrire certaines des primitives de haut niveau sur les fichiers en termes des primitives de bas niveau vues dans la feuille précédente.

Le but du jeu consiste ici à retrouver une écriture des fonctions standard d'entrées/sorties de C ANSI par reverse-engineering *sur le fichier* `<stdio.h>`

Vous ***devez travailler en binôme****.*

La date limite du rendu est le ***1er juin 2014 23h00.***

Entrées/Sortie *bufferisées*

Dans cette feuille nous utiliserons une version du fichier `<stdio.h>` provenant d'une très vieille version du Unix de Sun Microsystems (devenu Oracle depuis). La version que vous utiliserez se trouve à l'URL http://www.polytech.unice.fr/~eg/Posix/TP-stdio. Nous préférons utiliser ici cette version de `stdio.h`, à celle de Gnu, car elle est est beaucoup plus simple à lire et tient dans un fichier unique.
Si l'on regarde le fichier `stdio.h`, on voit que ce fichier définit principalement la structure `_iob`. Cette structure est déclarée de la façon suivante sur Sun:

```
extern struct _iobuf {
        int     _cnt;
        unsigned char *_ptr;
        unsigned char *_base;
        int     _bufsiz;
        short   _flag;
        char    _file;          /* should be short */
} _IOB[];
```

De plus, on a les déclarations suivantes:

```
#define FILE    struct _iobuf
#define stdin   (&_IOB[0])
#define stdout  (&_IOB[1])
#define stderr  (&_IOB[2])
```

Par conséquent, on voit que les fichiers standard C sont en fait des références aux trois premières entrées du tableau global `_IOB`.
Pour comprendre un peu mieux ce qu'il se passe nous allons utiliser ici la fonction `tracer` pour voir les valeur des différents champs d'un `FILE*`. Le code de cette fonction est donné ci-dessous:

```
void tracer(FILE *f)
{
  char buffer[500];

  snprintf(buffer, 500, "Trace FILE %p\n", f);
  write(2, buffer, strlen(buffer));
  snprintf(buffer, 500,
           "\tcnt: %d, ptr: %p, base: %p, bufsiz: %d, flag:%x, file:%d\n\n",
           f->_cnt, f->_ptr, f->_base, f->_bufsiz, f->_flag, f->_file);
  write(2, buffer, strlen(buffer));
}
```

Quelques exemples d'utilisation de cette fonction sont donnés ci-dessous. Ils devraient vous aider à comprendre comment sont gérés les descripteurs de fichiers:

1. affichage de l'état initial des fichiers `stdin`, `stdout` et `stderr`

```
  tracer(stdin);
  tracer(stdout);
  tracer(stderr);
==>
  Trace FILE 0x804b060
          cnt: 0, ptr: (nil), base: (nil), bufsiz: 0, flag:1, file:0
  Trace FILE 0x804b074
          cnt: 0, ptr: (nil), base: (nil), bufsiz: 0, flag:82, file:1
  Trace FILE 0x804b088
          cnt: 0, ptr: (nil), base: (nil), bufsiz: 0, flag:6, file:2
```

2. lecture de deux caractères (mais 3 sont entrés)

```
  tracer(stdin);
  getchar();
  tracer(stdin);
  getchar();
  tracer(stdin);
==>
  Trace FILE 0x804b060
          cnt: 0, ptr: (nil), base: (nil), bufsiz: 0, flag:1, file:0
  abc
  Trace FILE 0x804b060
          cnt: 3, ptr: 0x820e009, base: 0x820e008, bufsiz: 1024, flag:9, file:0
  Trace FILE 0x804b060
          cnt: 2, ptr: 0x820e00a, base: 0x820e008, bufsiz: 1024, flag:9, file:0
```

3. écriture sur la sortie standard (qui est bufferisée par ligne)

```
  tracer(stdout);
  putc('X', stdout);
  tracer(stdout);
  putc('Y', stdout);
  tracer(stdout);
  putc('\n', stdout);
  tracer(stdout);
==>
  Trace FILE 0x804b074
```

```
        cnt: 0, ptr: (nil), base: (nil), bufsiz: 0, flag:82, file:1
Trace FILE 0x804b074
        cnt: -1, ptr: 0x9904411, base: 0x9904410, bufsiz: 1024, flag:8a, file:1
Trace FILE 0x804b074
        cnt: -2, ptr: 0x9904412, base: 0x9904410, bufsiz: 1024, flag:8a, file:1
XY
Trace FILE 0x804b074
        cnt: 0, ptr: 0x9904410, base: 0x9904410, bufsiz: 1024, flag:8a, file:1
```

Questions:

1. En regardant les définitions des différentes macros définies dans ce fichier, essayez de déterminer l'utilisation des différents champs de la structure_`iobuf`.
2. Ecrivez ensuite les primitives _`filbuf` et _`flsbuf` dont vous "devinerez" la spécification en fonction de leur utilisation dans le fichier `<stdio.h>` qui vous a été fourni. La primitive _`filbuf` est beaucoup plus simple à écrire; écrivez cette fonction en premier.
3. Testez vos fonctions sur le programme `cat` (uniquement stdin → stdout).
4. Ecrivez enfin les primitives `fopen`, `setbuf`, `setvbuf`, `fflush` et `fclose`.
5. Tester votre code de `<stdio.h>` avec une version simplifiée de `cp` qui recopie caractère par caractère le fichier source vers le fichier destination.