Travail pratique #2

IFT-2035

June 9, 2022

1 Survol

Ce TP a pour but de vous familiariser avec le langage C, les pointeurs, la gestion de mémoire explicite. Comme pour le TP précédent, les étapes sont les suivantes:

1. Parfaire sa connaissance de C.
2. Lire et comprendre cette donnée.
3. Compléter le code fourni.
4. Écrire un rapport. Il doit décrire **votre** expérience pendant les points précédents: problèmes rencontrés, surprises, choix que vous avez dû faire, options que vous avez sciemment rejetées, etc... Le rapport ne doit pas excéder 5 pages plus 2 d'annexe (voir plus loin).

Ce travail est à faire en groupes de 2 étudiants. Le rapport (au format LaTeX) et le code sont à remettre par remise électronique avant la date indiquée. Aucun retard ne sera accepté. Indiquez clairement votre nom au début de chaque fichier.

Si un étudiant préfère travailler seul, libre à lui, mais l'évaluation de son travail n'en tiendra pas compte. Des groupes de 3 ou plus sont **exclus**.

2 Une bibliothèque de gestion mémoire

Vous devez écrire une bibliothèque de gestion mémoire pour des chaînes de caractères qui fournisse les fonctions suivantes:

```
typedef struct String String;

String *str_alloc   (size_t size);
size_t  str_size    (String *str);
char   *str_data    (String *str);
void    str_free    (String *str);

String *str_concat  (String *s1, String *s2);

void    str_compact  (void);
size_t  str_livesize (void);
size_t  str_freesize (void);
size_t  str_usedsize (void);
```

Libre à vous de définir `struct String` comme bon vous semble. La fonction *str_alloc* crée une nouvelle chaîne de taille *size*. La fonction *str_size* renvoie la taille d'une chaîne et la fonction *str_data* renvoie le tableau de bytes de la chaîne. Finalement *str_free* permet de libérer l'espace occupé par une chaîne et *str_compact* permet de compacter l'espace mémoire occupé par toutes les chaînes de manière à réduire la fragmentation (c'est à dire réduire le nombre de "trous" libres entre les espaces encore utilisés, pour les remplacer par un seul (ou un petit nombre) de plus gros trou(s)).

Pour que la compaction soit possible, chaque `String` sera composé de 2 objets en mémoire: l'objet de type `struct String` qui a une taille fixe indépendente de la longueur de la chaîne et qui ne peut pas être déplacé par *str_compact*, et un autre objet qui contient les bytes de la chaîne de caractères (et cet objet-là peut être déplacé par *str_compact*).

La compaction présume que le reste du programme (le *mutateur*), n'utilise pas de *str_data* pendant la compaction et de plus il doit faire attention à ne plus utiliser les *str_data* obtenus avant la compaction vu qu'ils peuvent ne plus être valides.

Vous devez écrire le code dans un fichier `stralloc.c`. Cette bibliothèque n'a pas le droit d'utiliser *malloc* pour obtenir de la mémoire. À la place, il faudra utiliser des fonctionnalités de plus bas niveau, plus précisément *mmap* et *munmap*. Ces fonctions peuvent varier un peu entre différents systèmes d'exploitation, donc assurez-vous que votre code fonctionne dans un système GNU/Linux raisonnablement récent.

mmap ne peut allouer que des multiples de pages (i.e. des multiples de 4KB par exemple). Il faudra donc souvent les partager entre plusieurs chaînes de caractères. De plus *mmap* est relativement coûteux, donc il est préférable de l'utiliser moins souvent en allouant plus de mémoire à la fois.

Les fonctions *str_livesize*, *str_freesize*, et *str_usedsize* permettent de mesurer quel espace est utilisé pour quoi.

3 Ce que vous devez faire

- Écrire des tests dans `tests.c`.
- Écrire le code dans `stralloc.c`.
- Analyser le comportement de votre code (maximum 2 pages en annexe du rapport). Il s'agit ici d'une analyse principalement théorique: mémoire effectivement consommée pour stocker une chaîne de taille N; nombre d'opérations nécessaires pour allouer une nouvelle chaîne, pour libérer une chaîne, pour compacter l'espace; impact de la fragmentation, ...
- Écrire le reste du rapport.

Le code fourni dans `stralloc.h` ne peut pas être changé, à moins d'une très bonne raison.
Remarquez que ce TP peut se décomposer en 3 parties:

1. Implanter seulement *str_alloc*, *str_size* et *str_data*; utiliser pour *str_free* et *str_compact* une implantation triviale qui ne fait rien.
2. Modifier le code pour implanter un vrai *str_free*.
3. Modifier le code pour implanter un vrai *str_compact*.

4 Remise

Pour la remise, vous devez remettre trois fichiers (`stralloc.c`, `tests.c`, et `rapport.tex`) par la page Moodle (aussi nommé StudiUM) du cours. Assurez-vous que le rapport compile correctement sur `ens.iro` (auquel vous pouvez vous connecter par SSH).

5 Détails

- La note sera divisée comme suit: 25% pour le rapport, 15% pour les tests, 60% pour le code C.
- Un bon point de départ pour vous aider à décider comment gérer vos pages de mémoire est à `http://www.memorymanagement.org/articles/alloc.html`.
- Tout usage de matériel (code ou texte) emprunté à quelqu'un d'autre (ou trouvé sur le web) doit être dûment mentionné, sans quoi cela sera considéré comme du plagiat.
- Chaque ligne de code doit faire moins de 80 caractères. Tout dépassement sera considéré comme une erreur.
- Votre code doit compiler avec `gcc -Wall` sans générer plus d'avertissements que le code fourni.
- Vérifiez la page web du cours, pour d'éventuels errata, et d'autres indications supplémentaires.
- La note est basée d'une part sur des tests automatiques, d'autre part sur la lecture du code, ainsi que sur le rapport. Le critère le plus important, est que votre code doit se comporter de manière correcte. Ensuite, vient la qualité du code: plus c'est simple, mieux c'est. S'il y a beaucoup de commentaires, c'est généralement un symptôme que le code n'est pas clair; mais bien sûr, sans commentaires le code (même simple) est souvent incompréhensible.