Système Unix

TP malloc

Contact b-psu-330@epitech.eu

Abstract:
Ce TP a pour objectif de vous initier à la gestion mémoire et de découvrir le `tas` *et la notion de plage mémoire.*
Ce TP est une première approche à votre projet malloc, qui consistera à reprogrammer les fonctions `malloc`, `realloc` *et* `free`.

Table des matières

.1 Introduction

Ce TP s'articulera autour de 2 grandes étapes.

- Les librairies partagées
- les appels systèmes de gestion mémoire

Passons donc sans plus attendre à l'étape 1 !

.2 Etape 1 : Librairies partagées

Les librairies partagées, ou "shared object" sous Unix, sont des collections de fonctions qui seront dynamiquement chargées par vos programmes lors de leur exécution.
À la différence d'une librairie statique, le contenu des fonctions appelées ne sera alors pas "copié" dans l'exécutable lors du link des différents objets.

A ce titre, votre `malloc` devra se trouver dans une librairie partagée, afin de pouvoir remplacer le malloc système sans avoir à recompiler les programmes voulant l'utiliser.

1. Exercice 1 : Création d'une librairie partagée

   (a) Créez un fichier .c contenant les fonctions :

      i. my_putchar

      ii. my_putstr

      iii. my_strlen

   (b) Lisez le man de `gcc` (1)

   (c) Compilez votre fichier en demandant à gcc de créer une librairie partagée (shared object).

2. Exercice 2 : Utilisation de votre librairie partagée
   Nous allons maintenant créer un programme qui va utiliser les fonctions de notre librairie.

   (a) Créez un fichier .c avec une fonction main qui affiche "Hello World!!!" sur la sortie standard (en utilisant votre `my_putstr`).

   (b) Compilez votre fichier en lui specifiant de linker avec votre librairie partagée (avec le flag -l, comme pour les librairies statiques)

   (c) Utilisez le programme ldd pour afficher les dépendances de votre programme nouvellement compilé.

`Vous devriez alors voir la liste des librairies partagées nécessaires à l'exécution de votre programme, on y trouve entre autre la libC.`

   (d) Exécutez votre programme

Vous devriez avoir un message d'erreur, stipulant que le système ne trouve pas votre librairie partagée. Cela est dû au fait que le système n'est pas paramétré pour aller chercher votre librairie à l'endroit ou elle se trouve.

(e) Definissez la variable d'environnement LD_LIBRARY_PATH en lui spécifiant le chemin ou se trouve votre librairie partagée.

man setenv

Vous pouvez également faire appel à la variable d'environnement LD_PRELOAD pour "précharger" la librairie partagée.

(f) Exécutez votre programme

3. Exercice 3 : Chargement manuel des symboles
   Il existe un autre moyen de charger les symboles (fonctions) de votre librairie, afin de les appeler.
   Il faut pour cela le faire à la main, grace aux fonctions `dlopen`, `dlclose` et `dlsym`.

   (a) Reprenez le programme précédent et remplacez l'appel à my_putstr par :

   i. Chargement de la librairie
   ii. Recuperation du symbole
   iii. Appel du symbole
   iv. Dechargement de la librairie

man dlopen (3)

Quel est l'intérêt de cette méthode selon vous ?

4. Exercice 4 : Hack me !

   (a) Ajoutez à votre librairie partagée une fonction `malloc`, qui se contentera d'afficher "Vive les moutons volants !!!" sur la sortie d'erreur.

   (b) Compilez votre librairie et chargez là avec LD_PRELOAD

   (c) Lancez un "ls" ou un "cat"

.3 Etape 2 : Appels systèmes

Rentrons maintenant dans le vif du sujet en abordant les appels systèmes liés à la gestion de la mémoire.
Les 2 principaux appels sont ceux permettant de manipuler le "break" (sorte d'indicateur de la position "haute" de l'allocation), à savoir `brk` et `sbrk`.
Enfin, nous verrons succintement l'utilisation de `getpagesize` qui pourrait vous être utile si vous souhaitiez optimiser votre malloc.

1. Exercice 1 :

   (a) Lire le man `brk` (2)

   (b) Lire le man `sbrk` (2)

   

   Avez vous bien compris l'utilité de ces fonctions ?

   (c) Lire le man `brk` (2) ... oui encore, ca ne fait pas de mal

   (d) Lire le man `sbrk` (2) ... idem !

2. Exercice 2 : Plages d'adressage
   Ecrivez maintenant un programme qui réalisera les opérations suivantes

   (a) Affichez l'adresse de la fonction main

   (b) Affichez l'adresse d'une fonction d'une librairie partagée (une de celle de votre librairie de l'étape 1 par exemple)

   (c) Affichez l'adresse d'une fonction de la libC

   (d) Affichez l'adresse d'une variable locale à votre main

   (e) Affichez l'adresse d'une chaine constante "toto"

   

   Analysez ces adresses et essayez d'en déduire comment la mémoire pourrait être organisée.

3. Exercice 3 : Allocation "manuelle"
   Attardons nous au fonctionnement de `brk` et `sbrk`

   (a) Utilisez `sbrk` et étudiez la valeur de retour lorsqu'on lui passe en paramètre une valeur positive, négative ou nulle.

   (b) Etendez les "break" et remplissez la zone mémoire nouvellement allouée afin de vérifier que tout fonctionne.

   (c) Tentez de remplir la mémoire au delà du "break" ...

`Vous noterez qu'il ne semble pas y avoir d'appel système pour "libérer" de la mémoire ... qu'elle est donc l'effet de la fonction free ? Comment est-il possible de libérer de la mémoire avec les appels systèmes entrevus ?`

4. Exercice 4 : Pagination
   Ce dernier "exercice" est une ouverture vers une façon d'optimiser les allocations dans votre malloc

   (a) Lisez le man de `getpagesize` (2)

   (b) Reflechissez à comment utiliser la notion de page pour améliorer les performances de votre malloc en limitant le nombre d'appel à des syscalls ?

Ce TP est maintenant fini. Vous devriez donc être en mesure de commencer dès à présent votre projet malloc.

Merci d'avoir suivi ce TP jusqu'au bout et bon courage pour votre projet.