ENSEA
Ecole Nationale Supérieure de l'Electronique et de ses Applications

3e année – IS – Systèmes et Réseaux

TP 2 – Allocateur de mémoire sécurisé

C. BARÈS

Objectifs

Le but de ce TP est de développer un allocateur de mémoire se substituant aux fonctions malloc, calloc, free… Les allocateurs standards de la glibc ne sont pas très performants, mais ils fonctionnent aussi bien avec 1 octet que 1 Go.

Il existe toute une série d'allocateurs de substitution, très efficaces pour certains cas d'utilisation de la mémoire[1].

Vous allez créer un allocateur qui permettra de vérifier, de manière basique, les corruptions mémoires liées aux débordements de buffer en écriture.

Principe

On développera des fonctions qui se substitueront aux fonctions d'allocation/désallocation mémoire de la glibc. Dans un premier temps, vous nommerez vos fonctions `malloc_3is` et `free_3is`. Quand vous serez sûrs de vos fonctions, vous pourrez les renommer en malloc et free, elle remplacerons alors les versions de la glibc[2].

Vous allez créer une liste chaînée reliant les blocs mémoire vides (disponibles). Lors d'une demande d'allocation mémoire via l'appel à `malloc_3is` , les cas suivants se présentent :

- Un bloc de taille suffisante est disponible dans la liste chaînée des blocs disponibles : le bloc mémoire (ou au moins une partie de ce bloc) est retourné à la fonction appelante. La taille du bloc disponible est alors diminué, ou bien le bloc est alors supprimé de la liste des bloc disponible.
- Aucun bloc de taille suffisante n'est disponible : le tas est agrandi en utilisant l'appel à la fonction sbrk (note : `sbrk(0)` renvoie l'adresse actuelle du haut du tas).

Lors de la désallocation (libération) via l'appel à `free_3is` d'une zone précédemment allouée, le bloc libre est replacé dans la liste chaînée. Cette liste est classée par ordre croissant des adresses des blocs de mémoire disponibles. Si le bloc mémoire libéré est contigu avec un bloc mémoire déjà libre, les deux blocs sont alors fusionnés (évitant ainsi une trop grande fragmentation de la mémoire).

L'allocateur mémoire développé ici va de plus permettre de contrôler les débordements mémoires. Pour cela, les blocs mémoire transmis à l'utilisateur (via `malloc_3is`) ou récupérés (via `free_3is`) vont être contrôlés pour constater l'absence de débordement mémoire

[1] Voir https ://en.wikipedia.org/wiki/C_dynamic_memory_allocation

[2] Il existe aussi des « hook » dans la glic : `malloc_hook` et `free_hook` qui permettent de détourner les appels originaux, mais ils sont *deprecated* (problème en multiprocesseur).

(et signaler l'erreur dans le cas contraire). Chaque bloc mémoire transmis sera encadré par un `magic_number` (`0x012345678ABDEFL`). En cas de débordement mémoire en écriture, ces nombres seront modifiés.

Une simple vérification de la valeur de ces nombres permet de signaler l'erreur à l'utilisateur. Une amélioration simple de la vitesse d'allocation est d'initialiser les nouveaux blocs disponibles avec de très grandes tailles initiales…

Structures nécessaires

Chaque bloc mémoire contenu dans la liste chaînée des blocs disponibles sera défini d'après la figure 1.

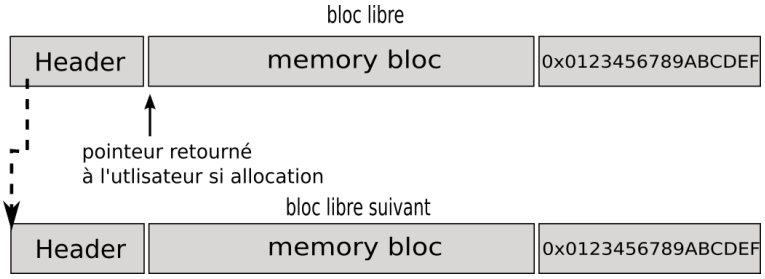

Figure 1 : Chaînage des blocs mémoires **disponibles**.

La structure HEADER présente avant le « memory bloc » est la suivante :

```
typedef struct HEADER_TAG {
    struct HEADER_TAG * ptr_next; /* pointe sur le prochain bloc libre */
    size_t bloc_size; /* taille du memory bloc en octets*/
    long magic_number; /* 0x0123456789ABCDEFL */
} HEADER;
```

Lorsqu'un bloc est fourni à l'utilisateur, il est supprimé de la liste chaînée et `ptr_next` est mis à `NULL`.

Lorsqu'un bloc est récupéré, ses `magic_numbers` sont vérifiés et le bloc est réintégré dans la liste chaînée : `ptr_next` est modifié et `bloc_size` est actualisé s'il y a fusion de blocs mémoires contigus.

Implémentation de l'allocateur

Vous testerez votre allocateur de mémoire à l'aide d'un programme simple effectuant des allocations/libérations de mémoire via l'appel à `malloc_3is` et `free_3is`.

Dans l'ordre :

1. Allocations multiples
2. Libération d'un bloc
3. Vérification des débordements de mémoire en écriture grâce aux nombres magiques
4. Réutilsation d'un bloc libérer (si la taille est compatible)
5. Découpage des gros blocs si nécessaires
6. Fusion des blocs libres adjacents
7. Préallocation de mémoire