Rapport projet allocateur

yanis kharouni
Zhyuan LIU

March 2020

1 Introduction

Ce projet est fait dans le cadre d'une evaluation sur le modules AISE et consiste en le développement d'un allocateur mémoire dynamique(Malloc/calloc/realloc/free).Il devra pouvoir être lancé en remplacement de l'allocateur système pour n'importe quel type d'application ou de bibliothèque.

2 Implementation

2.1 Structure de donnees

Après avoir consulté plusiers site sur internet, on a voulu faire une structure qui correspond aux codes A5 et B2 dans le cahier des charges. Ça veut dire que on devra être capable de reconsulter les adresses allouées précédament. Puis on a choisi d'adopter une structure de chaîne. Tous les blocs seront connectés l'un après l'autre. Chaque bloc possède un header qui contient les informations nécessaires pour gérer le bloc et la chaîne. La structure du header est comme ci-dessous.

```
union header{
    struct{
        size_t size;
        unsigned is_free;
        union header* next;
    } head;
```

```
ALIGN stub;
};
```

La variable size stock la taille de chaque bloc alloue.
La variable isfree indique si le block est libre ou pas.
Le pointeur next pointe vers le prochain bloc dans la chaine.
ALIGN est prédéfini comme un déterminant de la taille l'union.

2.2 Malloc()

La fonction malloc prend une taille en paramètre et elle cherche un bloc de libre dans la chaîne et de tailles supérieure ou égale à la taille demandée + la taille du header qui vas contenir les informations du bloc.

Quand il trouve un bloc adequat la fonction retourne l'adresse juste après l'adresse pointée par le header c'est la première case mémoire du bloc alloue pour l'utilisateur.

Si on trouve aucun bloc adequat dans la chaîne on alloue un bloc avec mmap() en precisant que la taille est la taille demandee + la taille qu'il faut pour stocker un header.

2.3 Calloc()

La fonction calloc() prend en parametres le nombre d'elements qu'on souhaite allouer ainsi que la taille de chaque element.

Elle calcule la taille totale num * size et appel malloc avec cette meme taille qui retourne un pointeur sur un bloc memoire alloue.

On recupere ce pointeur et on le passe en parametres pour memset() avec 0 en deuxieme parametre et la taille num * size calculee auparavant.

et on retourne ce pointeur a l'utilisateur.

2.4 realloc()

la fonction realloc alloue un bloc memoire de taille superieure a la taille du bloc passe en parametres et conserve les donnees de ce dernier.

on declare une un pointeur de type header et un pointeur vers void.on cast le pointeur passe en parametre sur le type header afin de recuperer un pointeur qu'on stock dans header qui pointe vers les donnees de ce bloc en reculant d'une case memoire.

on appel malloc() avec le nouveau pointeur qui pointe vers void et la nouvelle taille.puis avec memcpy() on copie les donnees de l'ancien bloc vers le nouveau bloc en donnant le nouveau bloc en premier parametre,l'ancien en deuxieme, et la taille de l'ancien bloc.

enfin on libere la memoire de l'ancien bloc avec free();

2.5 free()

free() prend en parametre un pointeur qui pointe vers la premiere case d'une zone memoire qu'onveut redonner au systeme.

on caste ce pointeur en header et on recule d'une case afin d'acceder aux donnees de ce bloc stockees dans son header pour recuperer la taille du bloc.

enfin on appel munmap() avec le pointeur qui pointe sur le header et la taille du bloc.

3 Compilation

La compilation est fait par un cmakelist qui génère ensuite un makefile. Les fichier qui sont liés à la compilation sont situés dqns le fichier cmake-build-debug. Avec les commandes précis dans le README, on pourra compiler et installer la bibliothèque sur le système. Une fois qu'elle soit installée, on peut l'utiliser comme une bibliothèque du système.

Tous les commandes et ses détailles sont expliqués dans le ficher CMakeList.txt. Les commandes à saisir sur le terminal est expliqué dans le Readme.

4 Tests et resultats

4.1 tests

Jusqu'à maintenant, le test de performance est fait par allocation de mémoire de plusieur taille. On calculer le temps d'exécution des allocteur par la fonction clockgettime(). Les paramètres et les résultats sont enregistrés sous test/resultoftest.txt.

4.2 résultats

Test on system function of malloc. Total allocated : 3.000000 mb. Function called : 1024 times. Total time spend : 5 ms.

Test on system function of malloc. Total allocated : 12.000000 mb. Function called : 1024 times. Total time spend : 8 ms.

Test on system function of malloc. Total allocated : 48.000000 mb. Function called : 4096 times. Total time spend : 35 ms.

Test on our own function of malloc. Total allocated : 3.000000 mb. Function called : 1024 times. Total time spend : 26 ms.

Test on our own function of malloc. Total allocated : 12.000000 mb. Function called : 1024 times. Total time spend : 130 ms.

Test on our own function of malloc. Total allocated : 48.000000 mb. Function called : 4096 times. Total time spend : 1931 ms.

Test on our own function of malloc with optimisation flag -O3. Total allocated : 3.000000 mb. Function called : 1024 times. Total time spend : 50 ms.

Test on our own function of malloc with optimisation flag -O3. Total allocated : 12.000000 mb. Function called : 1024 times. Total time spend : 134 ms.

Test on our own function of malloc with optimisation flag -O3. Total allocated : 48.000000 mb. Function called : 4096 times. Total time spend : 1946 ms.

4.3 comparaison

Selon le test, notre allocateur de mémoire implémenté n'est pas autant performant que celui fourni par le système. On pense qu'il peut y avoir plusieurs facteur :

-la fonction allocateur : selon les manuels, l'allocateur du système utilise brk() pour allouer les petites tailles de mémoire, alors que nous n'utilisons que la fonction mmap().

-la structure : la structure de chaine et ce que nous avons construit pout supporter les informations de chaque bloc de mémoire sont encore à optimiser. Plusieur fonctionallités attendues dans le cahier des charges ne sont pas réalisées. Tout ça peuvent influencer la performance de notre allocateur.