

(https://profile.intrav2.42.fr)



Go to

ofile.intrav2.42.fr)

My projects (/)



All projects

(/projects)

jects.intrav2.42.fr)

List projects

(/projects/list)



Git repository

Your projects

arning.intrav2.42.fr)

Left

vosphere/revamps/intrav2.42.fr:intra/2015/activities/malloc/kleneyle



Get Next Line

(/projects/get_next_line)

rum.intrav2.42.fr)

Nous vous demandons, pour le bon déroulement de cette notation :

FdF (/projects/fdf)



- De rester courtois, poli, respectueux, constructif, lors de cet échange. Le lien de confiance entre la communauté 42 et vous en dépend.

meta.intrav2.42.fr)

- De bien mettre en évidence auprès de la personne notée (ou du groupe) les dysfonctionnements éventuels.

Exam C

- D'accepter le fait qu'il y ait parfois des différences d'interprétation sur les demandes du sujet ou l'étendue des fonctionnalités. Restez ouvert d'esprit face à la vision de l'autre (a-t-il raison ou tort ?), et notez le plus honnêtement possible.

Rushes

(/projects/rushes)

Bonne soutenance à tous !

LibftASM

(/projects/libftasm)

Guidelines

Savoir Relier

Vous DEVEZ effectuer tous les tests demandés.

(/projects/savoir-relier)

Attention: ce projet étant complexe, le résultat et son interprétation sont subjectifs. Il faut garder en tête le but de ce projet:


"Ce projet consiste à implémenter un mécanisme d'allocation dynamique de la mémoire."

Ratings

Define the type of error (if there is an error), which ended the correction.

Ok Empty work Incomplete work No author file Invalid compilation Norminette Cheat Crash

Attachments

 Intra - Elearning - How to use mmap
(<https://elearning.intrav2.42.fr/notions/67/subnotions>)

 Sujet (/uploads/document/document/63/malloc.pdf)

Sections

Préliminaires

Tests préliminaires

Vérifiez d'abord les éléments suivants :

- Il y a bien un rendu (dans le dépôt git)
- Fichier auteur valide
- Le Makefile est présent et a bien les règles demandées
- Pas de faute de norme, la Norminette faisant foi
- Pas de triche (fonctions non autorisées...)
- 2 globales sont autorisées : une pour gérer les allocations, et une pour gérer le thread-safe

Si un élément n'est pas conforme au sujet, la notation s'arrête là. Vous êtes encouragés à continuer de débattre du projet, mais le barème n'est pas appliqué.

 Yes

No

Compilation de la librairie

Nous allons commencer par vérifier que la compilation de la librairie génère bien les fichiers demandés dans le sujet, en modifiant HOSTTYPE:

```
$> export HOSTTYPE=Testing
$> make re
...
ln -s libft_malloc_Testing.so libft_malloc.so
$> ls -l libft_malloc.so
libft_malloc.so -> libft_malloc_Testing.so
$>
```

Le Makefile utilise bien HOSTTYPE pour définir le nom de la librairie (libft_malloc_\${HOSTTYPE}.so) et crée bien un lien symbolique libft_malloc.so pointant vers libft_malloc_\${HOSTTYPE}.so ?

Si ce n'est pas le cas, la soutenance s'arrête.

✓ Yes

No

Export des fonctions

Vérifiez avec nm que la librairie exporte bien les fonctions malloc, free, realloc et show_alloc_mem.

```
$> nm libft_malloc.so
0000000000000000 T _free
0000000000000000 T _malloc
0000000000000000 T _realloc
0000000000000000 T _show_alloc_mem
U _mmap
U _munmap
U _getpagesize
U _write
U dyld_stub_binder
$>
```

Les fonctions exportées par la librairie sont marquées d'un T, celles utilisées d'un U (les adresses ont été remplacées par des 0, elles changent d'une librairie à l'autre, tout comme l'ordre des lignes).

Si les fonctions ne sont pas exportées, la soutenance s'arrête.

✓ Yes

No

Tests de fonctionnalité

Commencez par vous faire un script de lancement qui ne modifie les variables d'environnement que le temps de lancer un programme de test. Il devra s'appeler run.sh, et être exécutable: \$> cat run.sh #!/bin/sh export DYLD_LIBRARY_PATH=. export DYLD_INSERT_LIBRARIES="libft_malloc.so" export DYLD_FORCE_FLAT_NAMESPACE=1 \$@

Test de malloc

TEST DE MALLOC

Nous allons faire un premier programme de test qui ne fait pas de malloc, afin d'avoir une base de comparaison:

```
$> cat test0.c
#include

int main()
{
    int i;
    char *addr;

    i = 0;
    while (i < 1024)
    {
        i++;
    }
    return (0);
}
$> gcc -o test0 test0.c
$> /usr/bin/time -l ./test0
0.00 real 0.00 user 0.00 sys
491520 maximum resident set size
0 average shared memory size
0 average unshared data size
0 average unshared stack size
139 page reclaims
0 page faults
0 swaps
0 block input operations
0 block output operations
0 messages sent
0 messages received
0 signals received
0 voluntary context switches
1 involuntary context switches
$>
```

Nous allons ensuite ajouter un malloc, et écrire dans chaque allocation pour s'assurer que la page mémoire est bien allouée en mémoire physique par le MMU.

Le système n'alloue vraiment la mémoire d'une page que lorsqu'on écrit dedans, donc même en faisant un mmap plus grand que la demande de malloc, cela ne modifie pas "page reclaims".

```
$> cat test1.c
#include
```

```
int main()
{
```

```
int i;
char *addr;

i = 0;
while (i < 1024)
{
    addr = (char*)malloc(1024);
    addr[0] = 42;
    i++;
}
return (0);
}
$> gcc -o test1 test1.c
$> /usr/bin/time -l ./test1
0.00 real 0.00 user 0.00 sys
1544192 maximum resident set size
0 average shared memory size
0 average unshared data size
0 average unshared stack size
396 page reclaims
0 page faults
0 swaps
0 block input operations
0 block output operations
0 messages sent
0 messages received
0 signals received
0 voluntary context switches
1 involuntary context switches
$>
```

Notre programme test1 a demandé 1024 fois 1024 octets, donc 1Mbyte.

On peut le constater en faisant la différence avec le programme test0 :

- soit entre les lignes "maximum resident set size", on obtient un peu plus d'1Mbyte
- soit entre les lignes page reclaims qu'on multipliera par la valeur de getpagesize(3)

Testons maintenant les deux programmes avec notre librairie:

```
$> ./run.sh /usr/bin/time -l ./test0
0.01 real 0.00 user 0.00 sys
708608 maximum resident set size
0 average shared memory size
0 average unshared data size
0 average unshared stack size
214 page reclaims
0 page faults
0 swaps
0 block input operations
1 block output operations
```

```
0 messages sent
0 messages received
0 signals received
0 voluntary context switches
1 involuntary context switches
$>./run.sh /usr/bin/time -l ./test1
0.00 real 0.00 user 0.00 sys
4902912 maximum resident set size
0 average shared memory size
0 average unshared data size
0 average unshared stack size
1238 page reclaims
0 page faults
0 swaps
0 block input operations
0 block output operations
0 messages sent
0 messages received
0 signals received
0 voluntary context switches
2 involuntary context switches
$>
```

On constate dans cet exemple que ce malloc a utilisé 1024 pages soit 4MBytes pour stocker 1Mbyte.

Comptez le nombre de page utilisées et ajustez la note comme suit:

- moins de 255 pages, la mémoire réservée est insuffisante: 0
- 1023 pages et plus, le malloc fonctionne mais consomme une page minimum à chaque allocation: 1
- entre 513 pages et 1022 pages, le malloc fonctionne mais l'overhead est trop important: 2
- entre 313 pages et 512 pages, le malloc fonctionne mais l'overhead est très important: 3
- entre 273 pages et 312 pages, le malloc fonctionne mais l'overhead est important: 4
- entre 255 et 272 pages, le malloc fonctionne et l'overhead est raisonnable: 5

Rate it from 0 (failed) through 5 (excellent)



Les zones pré-allouées

Vérifiez dans le code source que les zones pré-allouées en fonctions des différentes tailles de malloc permettent d'y stocker au moins 100 fois la taille max pour ce type de zone. Vérifiez également que la taille des zones est un multiple de `getpagesize()`.

Si un de ces points fait défaut, laissez Non.

 Yes

No

Tests de free

Nous allons rajouter simplement un free à notre programme de test:

```
$> cat test2.c
#include

int main()
{
    int i;
    char *addr;

    i = 0;
    while (i < 1024)
    {
        addr = (char*)malloc(1024);
        addr[0] = 42;
        free(addr);
        i++;
    }
    return (0);
}
$> gcc -o test2 test2.c
```

Nous allons comparer le nombre de "page reclaims" à celui de test0 et test1. S'il y a autant de "page reclaims" ou plus que test1, le free ne marche pas.

```
$> ./run.sh /usr/bin/time -l ./test2
```

Le free fonctionne-t-il ? (moins de "pages reclaims" que test1)

☒ Yes

No

Qualité du free

test2 a maximum 3 "page reclaims" en plus que test0 ?

☒ Yes

No

Test de realloc

```
$> cat test3.c
#include
#include

#define M (1024 * 1024)

void print(char *s)
{
    write(1, s, strlen(s));
}
```

```
write(1, s, strlen(s));
}

int main()
{
char *addr1;
char *addr3;

addr1 = (char*)malloc(16*M);
strcpy(addr1, "Bonjours\n");
print(addr1);
addr3 = (char*)realloc(addr1, 128*M);
addr3[127*M] = 42;
print(addr3);
return (0);
}
$> gcc -o test3 test3.c
$> ./run.sh ./test3
Bonjours
Bonjours
$>
```

Ca fonctionne comme dans l'exemple ?

☒ Yes

☐ No

Test de realloc ++

Dans test3.c, modifiez le corps de la fonction main de la manière suivante:

```
int main()
{
char *addr1;
char *addr2;
char *addr3;

addr1 = (char*)malloc(16*M);
strcpy(addr1, "Bonjours\n");
print(addr1);
addr2 = (char*)malloc(16*M);
addr3 = (char*)realloc(addr1, 128*M);
addr3[127*M] = 42;
print(addr3);
return (0);
}
```

Ca fonctionne toujours ?

☒ Yes

☐ No

Gestions des erreurs

Testez les cas particuliers et d'erreurs.

```
$> cat test4.c
#include
#include
#include

void print(char *s)
{
write(1, s, strlen(s));
}

int main()
{
char *addr;

addr = malloc(16);
free(NULL);
free((void *)addr + 5);
if (realloc((void *)addr + 5, 10) == NULL)
print("Bonjours\n");
}
$> gcc -o test4 test4.c
$> ./run/sh ./test4
Bonjours
```

En cas d'erreur, realloc doit renvoyer NULL. Est-ce que le "Bonjours" est affiché comme dans l'exemple ?

Si le programme réagit de façon malsaine (segfault ou autres), la soutenance s'arrête et vous devez sélectionner Crash en haut du barème.

 Yes

No

Test de show_alloc_mem

```
$> cat test5.c
#include

int main()
{
malloc(1024);
malloc(1024 * 32);
malloc(1024 * 1024);
malloc(1024 * 1024 * 16);
malloc(1024 * 1024 * 128);
show_alloc_mem();
}
```

```
return (0);  
}  
$> gcc -o test5 test5.c -L. -lft_malloc  
$> ./test5
```

L'affichage correspond au sujet et à la répartition TINY/SMALL/LARGE du projet ?

☒ Yes

☐ No

Bonus

Accès concurrentiels

Le projet gère les accès concurrentiels des threads grâce à la bibliothèque pthread et aux mutexes.

Comptez les cas applicables :

- un mutex empêche plusieurs threads d'entrer simultanément dans la fonction malloc
- un mutex empêche plusieurs threads d'entrer simultanément dans la fonction free
- un mutex empêche plusieurs threads d'entrer simultanément dans la fonction realloc
- un mutex empêche plusieurs threads d'entrer simultanément dans la fonction show_alloc_mem

Rate it from 0 (failed) through 5 (excellent)



Autres bonus

S'il y a d'autres bonus, comptez-les ici. Les bonus doivent être 100% fonctionnels et un minimum utiles (à la discrétion du correcteur).

Exemple de bonus:

- Lors d'un free, le projet "défragmente" la mémoire libre en regroupant les blocs libres concomitants en un seul
- Malloc possède des variables d'environnement de debug
- Une fonction permet de faire un dump hexa des zones allouées
- Une fonction permet d'afficher un historique des allocations mémoire effectuées
- ...

Rate it from 0 (failed) through 5 (excellent)



Conclusion

Leave a comment on this correction

*** (required) Comment**

Finish correction