CY Cergy-Paris Université

UFR des Sciences et Techniques

Projet Systèmes d'exploitation

Sujet : système d'allocation dynamique de mémoire

« Guide de l'utilisateur »

Rédigé par :

Lydia Ait-Abdelkader
Zitouni Melissa

Avril 2021

Table des matières

Ce présent document a pour but d'aider un programmeur à utiliser le gestionnaire de mémoire dynamique développé. Ce dernier est un mécanisme d'allocation et désallocation de mémoire inspiré du couple malloc() / free() de la librairie standard.
Le programme contient quatre fonctions principales :

- int initMemory(int nBytes)
- void* myalloc(int nBytes)
- int myfree(void* p)
- int freeMemory()

I. Description des fonctions

Nous allons décrire chaque fonction et son utilisation :

1. int initMemory(int nByte)

Cette fonction permet d'initialiser la zone de travail.

- Tout d'abord, l'utilisateur doit saisir la taille de la mémoire qu'il souhaite allouer (nByte).
- Ensuite, on alloue un espace mémoire avec la taille demandée + la taille de méta données en utilisant la fonction prédéfinie 'mmap'.
- Avant de terminer et afin de pouvoir adresser l'espace mémoire à d'autres blocs, on crée une liste chainée et on initialise sa tête avec le premier nœud dont les paramètres sont :
  - 'next' qui représente l'adresse du prochain bloc ,il est initialisé à null,
  - 'capacité' qui représente la taille du bloc actuel, elle est initialisée à la taille de la mémoire totale.
  - 'isfree' qui est un booléen qui teste si le bloc est libre ou pas, il est initialisé à true.
- Enfin, la fonction retourne 0 en cas d'erreur, la taille allouée en cas de succès (nByte).

2. void* myalloc(int nBytes)

Cette fonction permet d'allouer d'une manière dynamique l'espace dans la zone.

- On prend la taille entrée par l'utilisateur.
- Puis on cherche un bloc vide avec un espace suffisant en parcourant la liste chainée, si ce bloc a une taille égale à celle de la mémoire demandée (nBytes), on effectue l'allocation directement et si on ne trouve aucun bloc ayant une taille suffisante, on affiche un message d'erreur
  (fonction find _block).
  Sinon, si on trouve un bloc avec un espace plus grand que nBytes, on le divise en deux, une partie pour la taille mémoire que l'on veut allouer (nBytes) et l'autre va être un bloc vide non alloué (fonction trysplit). Ceci a pour but d'éviter la fragmentation et le gaspillage de l'espace mémoire.

- Enfin, on retourne l'adresse du bloc alloué.

3. **int myfree(void* p)**

Cette fonction permet de libérer un bloc mémoire alloué.

- Il faut que le pointeur 'p' soit supérieur ou égal à l'adresse de début de mémoire (0x04040000) et inférieur ou égal à l'adresse de début+la taille de la mémoire totale. Si ce n'est pas le cas, on retourne -1 car le pointeur n'appartient pas à notre plage d'adressage.
- Afin de pouvoir libérer l'espace alloué, on calcule d'abord l'adresse de début du bloc à libérer sans meta données puis on effectue la desallocation en passant le paramètre 'is_free' à true.
- Ensuite, on teste si le prochain bloc est libre, si c'est le cas on fusionne les deux.

4. **int freeMemory()**

Libérer l'espace mémoire alloué dans initMemory() en utilisant la primitive munmap.
On retourne -1 en cas d'erreur et la taille totale récupérer en cas de sucés

II. Exécution du programme

1. Test simple

- On initialise la mémoire à 2000 bits.
- On alloue l'espace mémoire à quatre blocs, avec une taille de 300, 350, 400 et 200 bits.
- On libère l'espace mémoire alloué aux blocs dont la position est 1, 2 et 3.

NB : le premier bloc possède la position 0.

2. Mode interactif

- On demander à l'utilisateur de saisir la taille de l'espace mémoire qu'il souhaite allouer.
- Ensuite, pour allouer l'espace mémoire à un bloc, il suffit de taper +n où n est la taille à allouer pour ce bloc.
- Puis, pour désallouer l'espace mémoire affecté à un bloc, il suffit de taper –Numb où Numb est le numéro de bloc.
- Enfin, la libération de l'espace mémoire alloué à la première étape s'effectue automatiquement à la fin d'exécution grâce à la fonction FreeMemory.
- Pour quitter, il suffit de taper e.

Exemple : l'utilisateur choisit d'allouer un espace mémoire de 20000 bits, puis décide d'allouer au premier bloc 600 bits et au second 300 bits puis libère l'espace alloué à ce dernier.

```
lydia@l:~/Users\Asus\lydia\L3\os\miniProjet$ alloc -i
Veuillez initialiser la zone mémoire:
20000

option 1: pour l'allocation d'un bloc avec une taile n tapez: +n
option 2: pour la desallocation d'un bloc avec un numero Numb tapez: -NumB
option 3: pour terminer tapez : e

+600
Etat de la memoire apres l'allocation de 600 bits :
adress: 0x4040000
size: 600
is_free: 0

adress: 0x4040269
size: 19366
is_free: 1

================

option 1: pour l'allocation d'un bloc avec une taile n tapez: +n
option 2: pour la desallocation d'un bloc avec un numero Numb tapez: -NumB
option 3: pour terminer tapez : e

+300
Etat de la memoire apres l'allocation de 300 bits :
adress: 0x4040000
size: 600
is_free: 0

adress: 0x4040269
size: 300
is_free: 0

adress: 0x40403a6
size: 19049
is_free: 1

================

option 1: pour l'allocation d'un bloc avec une taile n tapez: +n
option 2: pour la desallocation d'un bloc avec un numero Numb tapez: -NumB
option 3: pour terminer tapez : e

-1
Etat de la memoire apres la liberation du bloc num 1 :
adress: 0x4040000
size: 600
is_free: 0

adress: 0x4040269
size: 19349
is_free: 1

================

option 1: pour l'allocation d'un bloc avec une taile n tapez: +n
option 2: pour la desallocation d'un bloc avec un numero Numb tapez: -NumB
option 3: pour terminer tapez : e

e
la memoire est libre
```

- On voit qu'à l'issu de l'allocation de 600 bits, il reste 19366 bits libres (20000-600-34= 19366). 34 représente les bits de méta données du bloc.
- Ensuite, lors de l'allocation de 300 bits à un autre bloc, on voit que la mémoire contient les 600bits déjà alloués, ce qui fait qu'il reste 19049 bits (20000-600-300-51-=19049)
- Enfin, lors de la désallocation de l'espace alloué au bloc dont la position est 1 (300 bits), la mémoire contient 600 bits et il reste 19349 bits libres.

3. Mode ligne de commande

Syntaxe à respecter : -l taille_espace_memoire_a_allouer +taille_bloc -position_bloc

- Exemple : **-l 2000 +200 +400 -1**

```
lydia@l:~/Users\Asus\lydia\L3\os\miniProjet$ alloc -l 2000 +200 +400 -1
Etat de memoire apres l'initialisation:
adress: 0x4040000
size: 2000
is_free: 1

================
Etat de la memoire apres l'allocation de 200 bits :
adress: 0x4040000
size: 200
is_free: 0

adress: 0x40400d9
size: 1783
is_free: 1

================
Etat de la memoire apres l'allocation de 400 bits :
adress: 0x4040000
size: 200
is_free: 0

adress: 0x40400d9
size: 400
is_free: 0

adress: 0x404027a
size: 1366
is_free: 1

================
Etat de la memoire apres la liberation du bloc num 1 :
adress: 0x4040000
size: 200
is_free: 0

adress: 0x40400d9
size: 1766
is_free: 1

================
la memoire est libre
```

- On initialise de la mémoire à 2000 bits puis on alloue 200 bits au premier bloc et 400 bits au second. Ensuite on désalloue l'espace mémoire alloué au second bloc dont la position est 1. Enfin, on voit que la mémoire occupée est de 200 bits et il reste 1766 bits

4. Mode batch

Il suffit de taper -f test.txt.

Remarque : ‘test.txt’ contient : **20000 +300 +250+450 -2 -1**

- 20000 : initialiser la mémoire à 20000 bites.
- +300 : allouer 300 bits au bloc numéro 0.
- +250 : allouer 300 bits au bloc numéro 1.
- +450 : allouer 450 bits au bloc numéro 2.
- -2 : désallouer l’espace mémoire affecté au bloc numéro 2 (celui auquel nous avons alloué 450 bits)
- -1 : désallouer l’espace mémoire affecté au bloc numéro 1 (celui auquel nous avons alloué 250 bits)

III. Makefile

- L’exécution du programme avec les différents modes est très facile grâce au makefile, en effet il suffit juste taper le mot make, puis le mot alloc suivi du mode souhaité avec ses propres paramètres.
- Makefile permet également de créer la doc html.

IV. Doxygen

Une documentation a été générée sous forme de pages html afin de faciliter la compréhension du programme.