МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

Сучасні операційні системи

Лабораторна робота №1. Аллокатор пам’яті загального призначення

*Виконав студент гр. ІС-72*

*Гороховський І.О.*

Київ

НТУУ «КПІ»

2020

**Мета роботи:** розробити аллокатор загального призначення.

**Завдання:**

Розробити аллокатор загального призначення, беручи до уваги наступні умови:

* Області пам'яті можна виділяти будь-яким доступним способом.
* Функції mem\_alloc (), mem\_realloc () і mem\_free () повинні відповідати наведеним прототипам.
* Адреси пам'яті, які повертаються функціями mem\_alloc() і mem\_realloc(), повинні бути вирівняні на границю 4 байт.
* Оптимізувати час пошуку вільного блоку пам'яті і час звільнення зайнятого блоку.
* Написати функцію, яка повинна виводити на консоль стан областей пам'яті.

**Опис розробленого алгоритму:**

malloc: Виділяє заданий число байт і повертає покажчик на них. Якщо пам'яті недостатньо, повертає покажчик на NULL (null pointer);

free: Приймає на вхід покажчик на область в пам'яті, виділеної за допомогою malloc і повертає її для подальшого використання в програмі або операційній системі.

brk: brk () - це дуже простий системний виклик. System break - це крайня межа розміченій для процесу пам'яті. brk () просто переміщує цю межу вперед / назад, щоб збільшити або зменшити обсяг виділеної пам'яті. В програмі ми будемо використовувати конструкцію brk(0) для того щоб отримати поточний показчик на останню вільну пам’ять.

void \* malloc (long numbytes): Виділяє в пам'яті numbytes байт і повертає покажчик на перший з них;

void free (void \* firstbyte): firstbyte - покажчик отриманий за допомогою malloc () і пам'ять за яким необхідно звільнити.

1. Якщо наш аллокатор не був ініціалізований, то ініціалізуємо його.

2. Додати sizeof (struct mem\_control\_block) до розміру запитуваної пам'яті;

3. Почнемо з managed\_memory\_start.

4. Чи є він покажчиком на last\_valid address?

5. Якщо так:

   A. Значить не знайдено жодного підходящого ділянки пам'яті

     повідомте ОС що вам необхідно більше пам'яті і повертайтеся сюди.

6. Якщо немає:

   A. Поточний блок не зайнятий? (Mem\_control\_block-> is\_available == 1)?

   B. Якщо так:

      I) є-ли він досить великим (mem\_control\_block-> is\_available> = запитуваної пам'яті)?

      II) Якщо так:

           a. Помічаємо пам'ять як недоступну (mem\_control\_block-> is\_available = 0)

           b. Переміщаємо покажчик за mem\_control\_block і повертаємо його

      III) В іншому випадку:

           a. Переміщаємося на "size" байт вперед

           b. Повертаємося до кроку 4

   C. Якщо немає:

      I) Рухаємося на "size" байт вперед

      II) Повернення на крок 4

**Розроблене програмне забезпечення:**

|  |
| --- |
| #include <unistd.h> // sbrk |
|  | #include "stdio.h" // printf |
|  |  |
|  |  |
|  | int has\_initialized = 0; |
|  | void \*managed\_memory\_start; |
|  | void \*last\_valid\_address; |
|  |  |
|  | void malloc\_init() |
|  | { |
|  | last\_valid\_address = sbrk(0); |
|  |  |
|  | managed\_memory\_start = last\_valid\_address; |
|  |  |
|  | has\_initialized = 1; |
|  | } |
|  |  |
|  | struct mem\_control\_block { |
|  | int is\_available; |
|  | int size; |
|  | }; |
|  |  |
|  | void free(void \*firstbyte) { |
|  | struct mem\_control\_block \*mcb; |
|  |  |
|  | mcb = firstbyte - sizeof(struct mem\_control\_block); |
|  |  |
|  | mcb->is\_available = 1; |
|  |  |
|  | return; |
|  | } |
|  |  |
|  | void \*malloc(long numbytes) { |
|  |  |
|  | void \*current\_location; |
|  |  |
|  | struct mem\_control\_block \*current\_location\_mcb; |
|  |  |
|  | void \*memory\_location; |
|  |  |
|  | if(! has\_initialized) { |
|  | malloc\_init(); |
|  | } |
|  |  |
|  | numbytes = numbytes + sizeof(struct mem\_control\_block); |
|  |  |
|  | memory\_location = 0; |
|  |  |
|  | current\_location = managed\_memory\_start; |
|  |  |
|  | while(current\_location != last\_valid\_address) |
|  | { |
|  | current\_location\_mcb = |
|  | (struct mem\_control\_block \*)current\_location; |
|  |  |
|  | if(current\_location\_mcb->is\_available) |
|  | { |
|  | if(current\_location\_mcb->size >= numbytes) |
|  | { |
|  |  |
|  | current\_location\_mcb->is\_available = 0; |
|  |  |
|  | memory\_location = current\_location; |
|  |  |
|  | break; |
|  | } |
|  | } |
|  |  |
|  | current\_location = current\_location + |
|  | current\_location\_mcb->size; |
|  | } |
|  |  |
|  |  |
|  | if(! memory\_location) |
|  | { |
|  | sbrk(numbytes); |
|  |  |
|  | memory\_location = last\_valid\_address; |
|  |  |
|  | last\_valid\_address = last\_valid\_address + numbytes; |
|  |  |
|  | current\_location\_mcb = memory\_location; |
|  | current\_location\_mcb->is\_available = 0; |
|  | current\_location\_mcb->size = numbytes; |
|  | } |
|  |  |
|  |  |
|  | memory\_location = memory\_location + sizeof(struct mem\_control\_block); |
|  |  |
|  | return memory\_location; |
|  | } |
|  |  |
|  |  |
|  | void \*remalloc(void \*adr, long numbytes) { |
|  | if (! has\_initialized){ |
|  | malloc\_init(); |
|  | } |
|  |  |
|  | if (!adr){ |
|  | return malloc(numbytes); |
|  | } |
|  |  |
|  | void \*current\_location = adr; |
|  |  |
|  | struct mem\_control\_block \*current\_location\_mcb; |
|  |  |
|  | current\_location\_mcb = (struct mem\_control\_block \*)current\_location; |
|  |  |
|  |  |
|  | if (current\_location\_mcb->size >= numbytes){ |
|  | current\_location\_mcb->size = numbytes; |
|  | current\_location = current\_location + |
|  | current\_location\_mcb->size; |
|  | } |
|  |  |
|  | // if (current\_location\_mcb->size < numbytes) |
|  | // { |
|  | // int \*new\_tmp = malloc(numbytes); |
|  | // for (int i=0; i < current\_location\_mcb->size; i++){ |
|  | // new\_tmp[i] = ((int \*)current\_location)[0]; |
|  | // } |
|  | // free(current\_location); |
|  | // for (int i=0; i < current\_location\_mcb->size; i++){ |
|  | // printf("+%d", new\_tmp[i]); |
|  | // } |
|  | // new\_tmp = (void\*)new\_tmp; |
|  | // |
|  | // return new\_tmp; |
|  | // } |
|  |  |
|  |  |
|  | return current\_location; |
|  |  |
|  | } |
|  |  |
|  |  |
|  | int main() { |
|  | int number = 10; |
|  | printf("Allocating memory for a digit:\n"); |
|  | int \*a = malloc(8); |
|  | a = &number; |
|  | printf("A value = %d\n", \*a); |
|  | printf("A index in memory = %d\n", a); |
|  |  |
|  | printf("Allocating memory for an array size of 4:\n"); |
|  | int \*arr = malloc(4 \* sizeof(int)); |
|  | if(arr) { |
|  | for(int n=0; n<4; n++) // Заполнение массива |
|  | arr[n] = n\*n; |
|  | printf("\n\nArray:\n"); |
|  | for(int n=0; n<4; n++) // Вывод его содержимого |
|  | printf("--arr[%d] is %d\n",n,arr[n]); |
|  | } |
|  |  |
|  | printf("\n\nReallocating memory for an array size of 4 to size of 3:\n"); |
|  | arr = remalloc(arr, 3 \* sizeof(int)); |
|  | if (arr) { |
|  | printf("\n\nArray:\n"); |
|  | for(int n=0; n<3; n++) // Вывод его содержимого |
|  | printf("--arr[%d] is %d\n",n,arr[n]); |
|  | } |
|  |  |
|  | printf("\n\nReallocating memory for an array size of 4 to size of 10:"); |
|  | arr = remalloc(arr, 10 \* sizeof(int)); |
|  | if (arr) { |
|  | printf("\n\nArray:\n"); |
|  | for(int n=0; n<10; n++) // Вывод его содержимого |
|  | printf("--arr[%d] is %d\n",n,arr[n]); |
|  | } |
|  | free(arr); |
|  | return 0; |
|  | } |

**Результати роботи:**

