Національний технічний університет України «КПІ ім. Ігоря Сікорського»

Факультет Інформатики та Обчислювальної Техніки

Кафедра Автоматизованих Систем Обробки Інформації та Управління

Лабораторна робота №1

з дисципліни «Операційні системи»

Виконав:

студент гр. ІС-71

Большой Олександр

Перевірив:

Дифучин А. Ю.

Київ – 2020

Опис алгоритму

Було створено клас власного алокатора для контролю над пам'яттю *MyCppAllocator*. Блоки пам'яті представлені за допомогою структури *mem\_block*, що включає: покажчик на блок, довжину блоку, покажчики на попередній та наступний блоки. Для контролю вільних/зайнятих блоків використовуються два списки.

**Виведення даних про пам’ять (mem\_dump):**

Знаходимо перший блок пам'яті, та від нього через покажчики на наступний блок, ідемо до кінця контрольованої нами пам'яті. Під час проходження по блокам, виводимо їх величину та статус вільний/зайнятий.

**Запис нових даних (mem\_alloc):**

Вирівнюємо величину блоку до числа, кратного 4 байтам. Проглядаємо список вільних блоків на наявність підходящого по величині.

Якщо ні:

1. виділяємо пам’ять в кучі під блок за допомогою пакету *<Windows.h>*, якщо пам’ять не виділена повертаємо *nullptr*;
2. створюємо блок, встановлюємо покажчик блоку на виділену область пам’яті, робимо даний блок хвостом нашої контрольованої області;
3. повертаємо покажчик на даний блок користувачеві.

Якщо так:

1. якщо величина блока рівна потрібній користувачеві величині, повертаємо покажчик на нього;
2. якщо величина блока менша за потрібну величину, «обрізаємо» його: створюється ще один блок, який вказує на «лишню» область, записується у список вільних. Потрібний, уже урізаний блок повертаємо користувачеві через покажчик.

**Зміна існуючої пам’яті (mem\_realloc):**

Викликаємо метод звільнення пам’яті (mem\_free), а потім метод запису даних (mem\_alloc).

**Звільнення пам’яті (mem\_free):**

Видаляємо блок зі списку зайнятих, додаємо до списку вільних. Якщо справа або зліва від блоку були вільні блоки, об’єднуємо їх разом. Це допоможе уникнути надлишкової фрагментації пам’яті.

Оцінка пошуку блоків

Оскільки вільні блоки зберігаються у окремому списку, максимальна кількість перевірок блоків буде у випадку, якщо підходящий вільний блок останній або взагалі не існує. Це буде O(Pвільн), де Pвільн – кількість елементів у списку вільних блоків. Те ж стосується і пошуку блоку для звільнення – O(Pзайн), де Pзайн – кількість елементів у списку зайнятих блоків.

Оцінка службової пам’яті

Для кожного блоку має зберігатись екземпляр його структури, що включає в себе покажчик на область пам’яті, величину області. Також потрібно зберігати списки покажчиків на блоки, щоб відрізняти вільні та зайняті блоки.

Недоліки та переваги

* Велике використання службової пам’яті
* Якщо змінити розмір блоку методом realloc, і при цьому новий розмір буде більший за попередній + вільний простір справа, то в пам’яті виділиться новий блок і дані старого туди не перенесуться
* Код не є кросплатформеним
* Внутрішня фрагментація існує, доки ми вирівнюємо блоки до розмірів, кратних 4 байтам
* Адреси пам’яті вирівнюються на границю в 4 байти, це призводить до більшої універсальності при зміні уже створених блоків
* Швидкий пошук на наявність вільних блоків
* При звільненні блоку, він об’єднується з вільними блокам справа і зліва (якщо такі наявні). Це допомагає зменшити зовнішню фрагментацію пам’яті
* Метод mem\_dump дозволяє проглянути стан контрольованої пам’яті

Зменшення фрагментації пам’яті

Як було сказано у останньому пункті переваг програми, завдяки об’єднанню вільних блоків зовнішня фрагментація зменшується. Однак проблема внутрішньої фрагментації є наявною, оскільки блоки вирівнюються до розмірів, кратних 4 байтам.

Лістинг програми

Код програми з коментарями доступний за посиланням: <https://github.com/alexboliam/OS_labs/blob/master/MyCppAllocator/Source.cpp>

Приклад роботи аллокатора

На рис. 1 наведено консольний результат виконання програми, усі дії описані там же.

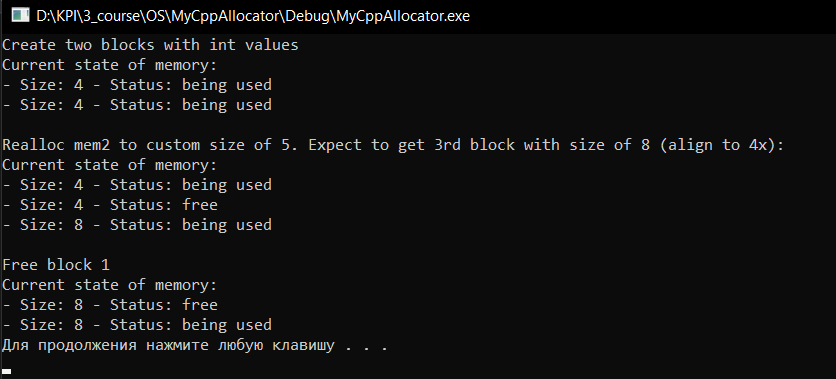


Рис. 1 – Консоль з результатом виконання програми