Факултет техничких наука
Универзитет у Новом Саду

Напредни алокатор меморије

Индустријски комуникациони протоколи
Предметни пројекат

Аутори:
PR 83/2020 Бојана Михајловић
PR 77/2020 Никола Куравица

16. јануар 2024.

Садржај

1. Увод

AHM (Advanced Heap Manager) има циљ да реши проблем heap contention-a. То је проблем који настаје у вишенитном програмирању када више нити покушава да приступи истом ресурсу који је заштићен критичном секцијом. Тада само једна нит може да ради, а све остале су блокиране.

AHM користи конфигурабилан број хипова и захтеве за алокацијом извршава на хипу који има најмање алоцираних бајта.

Хип на ком ће се заузети меморија је онај који има најмање алоцираних бајта.

Такође, AHM води рачуна о томе да се приликом деалокације меморија враћа у хип из ког је алоцирана.

AHM нуди следеће функције:

- *void* ahm_malloc(int bytes);*
- *void ahm_free(void* pointer);*

2. Дизајн

2.1. AHM

AHM је структура која ради над HeapManager структуром, односно, користи њене функције. Такође, садржи хеш табелу у којој чува податак која адреса (показивач) је алоциран на ком хипу.

Хеш табела у себи садржи хеш функцију која кључ (показивач у меморији) претвара у хеш вредност, ствара нови чвор, и смешта га у одговарајући бакет (bucket).

Хеш мапа у себи такође има дефинисану минималну величину табеле, тренутан број елемената, тренутну величину табеле (број бакета), као и функције потребне за алокацију и деалокацију бакета и елемената у табели.

Сама по себи хеш табела није thread-safe, па се убацује у обмотач (wrapper) структуру речник. Речник се састоји из хеш табеле, свог приватног хипа и критичне секције. Управо та критична секција омогућава речнику да буде thread-safe.

Приликом покретања, AHM иницијализује једну инстанцу хип менаџера и речника. Приликом гашења, AHM уништава речник, хип менаџер, као и све структуре унутар њих.

AHM нуди две функције за рад са меморијом: ahm_malloc() и ahm_free().

ahm_malloc() прво од хип менаџера добија меморију. Затим показивач који добије као повратну вредност смешта у речник, заједно са хеднлером хипа у који је меморија смештена.

ahm_free() прво из речника избацује чвор чији је кључ показивач на меморију која се ослобађа. Том приликом добија хендлер за хип у ком је дата меморија алоцирана и позива функцију из хип менаџера за деалокаицју меморије.

2.2. Heap Manager

HeapManager је структура која у себи садржи низ хипова (односно њихових хендлера), низ са бројем алоцираних бајта за сваки хип, максималну величину хипа, број тренутно коришћених хипова, максималан број хипова и тренутно коришћени хип (његов индекс у низу).

Да би био thread-safe, осим овога у себи садржи и критичну секцију.

Приликом његове иницијализације, хип менаџер прави низ са maximal_heap_count елемената и иницијализује сва места за хендлере на NULL, а све елементе у низу алоцираних бајта иницијализује на -1.

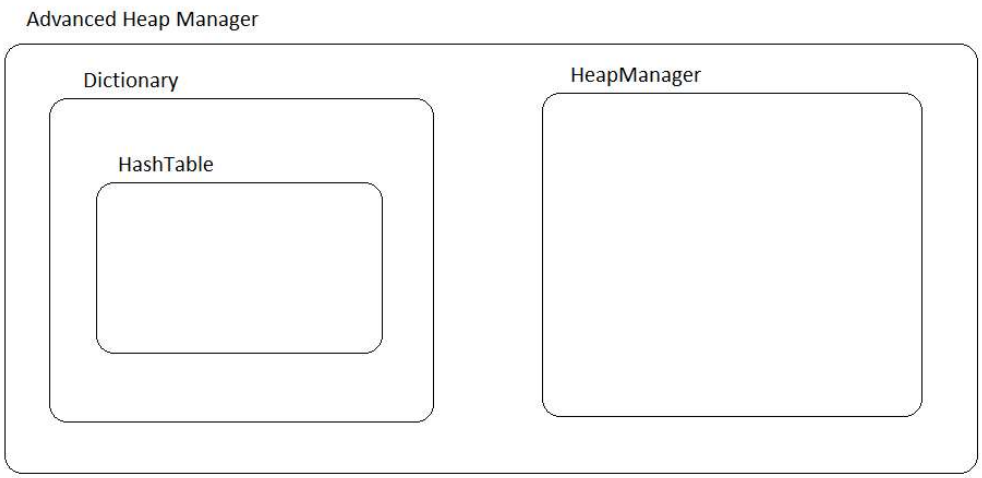
Приликом деиницијализације, низ хипова и броја алоцираних бајта се ослобађа, и сви хипови се уништавају.

Хип менаџер AHM-у нуди операције HeapOperacije_dobavi_memoriju и HeapOperacije_oslobodi_memoriju. Оне се позивају у склопу ahm_malloc() и ahm_free().

Ове HeapOperacije користе функције из HeapFunkcije. То су HeapFunkcije_alociraj() и HeapFunkcije_dealociraj.

Heap Funkcije раде директно са системом, позивају HeapAlloc() и HeapFree() из heapapi.h

2.3. Дијаграм компоненти



3. Структуре података

Централни део АНМ у ком се складиште сви подаци потребни за алокаицју и деалокацију је речник. Речник је изабран као структура која има време O(1) за читање и писање. То значајно убрзава процес у односу на неке друге структуре, попут листе која има време O(n), где је n дужина листе.

Због могуће колизије, речник има бакете. Речник има могућност промене своје величине и реорганизације ако увиди да превише често долази до колизије.

Речник има своју критичну секцију која омогућава да буде thread-safe, са обзиром да се АНМ библиотека користи у вишенитним апликацијама.

У речник се смештају елементи који садрже кључ - показивач на алоцирану меморију, и вредност - хендлер хипа из ког је меморија алоцирана.

Овако једноставно памтимо ком хипу враћамо коју меморију приликом деалокације.

4. Резултати тестирања

4.1. Опис тестова

У решењу су имплементирана два теста. Један имплементира вишенитну апликацију која треба да алоцира и деалоцира 4 гб меморијем користећи функције из АНМ, а затим и уграђене функције. Други имплементира клијент и сервер у ком сервер за алокаицју меморије користи функције АНМ, док клијент алоцира исту ту количину меморије са уграђеним функцијама.

4.2. Тест апликација

Тест са апликацијом приликом покретања захтева аргумент који говори колико хипова ће се користити у АНМ (максималан број хипова). Постоји више тестова који можемо да бирамо у зависности са колико нити желимо да радимо.

Тест се састоји од алокације и деалокације 20000 елемената од 200кб/број_нити, тј. у тесту са 20 нити ће се алоцирати 20000 елемената од 10кб, и то ће паралелно урадити 20 нити. У току теста се пре почетка алокације покрене тајмер који се зауставља на крају. Бројач броји колико је успешних алокација извршено. Ако је тај број на крају теста једнак 400000 (у случају теста са 20 нити), значи да су све алокације и деалокације успешно извршене.

Исти принцип се примењује на тест са уграђеним функцијама за рад са меморијом.

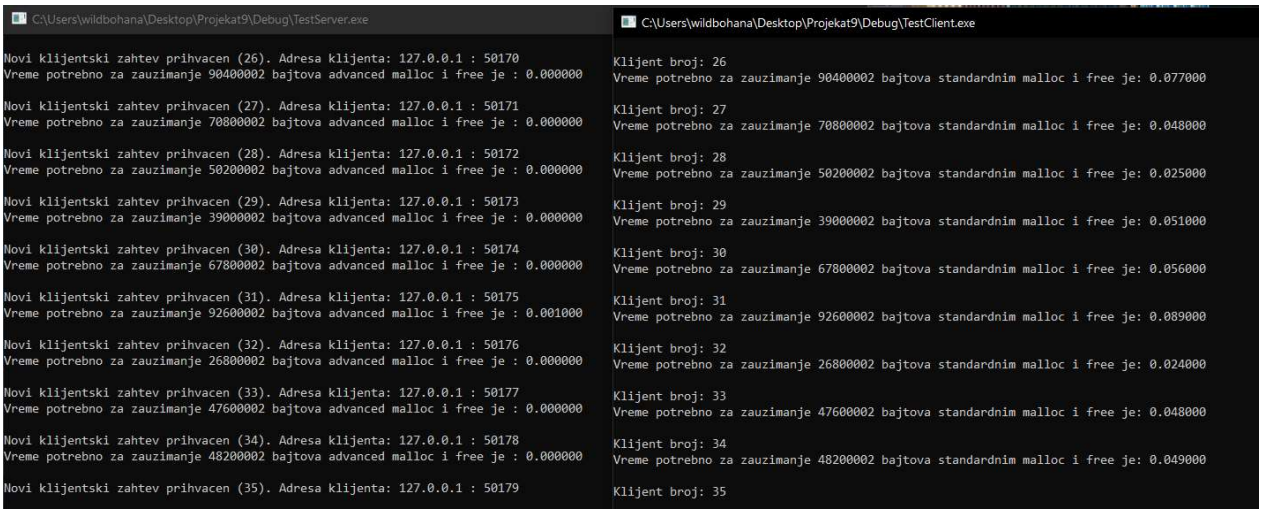
У тесту са 20 нити се могу приметити разлике од 5-10 секунди у користи АНМ. АНМ обави тест у просеку за 2.5 секунде, док уграђене функције обаве тест у просеку за 10 секунди.

```
C:\Users\wildbohana\Desktop\Projekat9\Debug>TestThreads.exe 20

        AHM malloc i free funkcije

Broj niti: 20
Potrebno vreme za zauzimanje i oslobadjanje memorije: 2.859000 sekundi

---------------------------------------------------------------------
NIJE NULL (uspesna alokacija): 400000 elemenata!

        Ugradjene malloc i free funkcije

Broj niti: 20
Potrebno vreme za zauzimanje i oslobadjanje memorije: 11.397000 sekundi

---------------------------------------------------------------------

Pritisnite bilo sta za izlazak iz programa...


C:\Users\wildbohana\Desktop\Projekat9\Debug>
```

4.3. Тест сервер

Други тест има серверску компоненту која успоставља комуникацијом са клијентском компонентом. Клијент шаље поруку насумичне вредности. Та вредност представља колико меморије ће се алоцирати. Клијент ту меморију алоцира употребом уграђених функција за рад са меморијом, док сервер за алокацију и деалокацију користи функције из AHM. За сваку алокацију и деалокацију се мери временска делта и исписује се на конзолу.

И у овом тесту AHM се показује као много ефикаснији од уграђених функција. Серверу је потребно 0.001 секунда за алокацију, док је клијенту потребно око 0.5 секунди.



5. Закључак

AHM који користи HeapAlloc() и HeapFree() функције у склопу ahm_malloc() и ahm_free() се показао много ефикаснијим од уграђених malloc() и free() функција.

Наш HeapManager се може користити само на Windows оперативном систему зато што користи функције из heapapi.h, за разлику од malloc() који се може користити било где, али управо чињеница да користимо Windows API нам нуди одређене оптимизације, и самим тим убрзања.

Такође, malloc() "испод хаубе" користи HeapAlloc(), а ми га користимо директно, тако да и то побољшава ефикасност.

6. Потенцијална унапређења

Постоје разни други алгоритми по којима се може бирати хип на ком ће се алоцирати меморија. Сваки од тих алгоритама је добар за различите типове апликација. Ако знамо за коју намену правимо хип, можемо да прилагодимо алгоритам томе.

За овај наш хип, уз претпоставку да ће се увек алоцирати иста количина меморије (као нпр. у тесту где се увек алоцира 10кб), можда би бољи алгоритам избора хипа био Round Robin.

Следећи корак у развоју хипа би вероватно био раздвајање на SOH (Small Object Heap) и LOH (Large Object Heap), како би се што ефикасније алоцирала меморија за објекте различитих величина. Такође, касније би се могао увести сакупљач отпадака, генерације и остале функционалности које садрже модерни хип менаџери.