|  |  |  |
| --- | --- | --- |
|  | УНИВЕРЗИТЕТ У НОВОМ САДУ  **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ** | A black and white logo  Description automatically generated |

Зорица Суботић E3 60/2014

**ИНДУСТРИЈСКО КОМУНИКАЦИОНИ ПРОТОКОЛИ У ИНФРАСТРУКТУРНИМ СИСТЕМИМА**

ПРОЈЕКАТ

- Примењено софтверско инжењерство (ОАС) –

Нови Сад, 2024.

**САДРЖАЈ**

1. ОПИС РЕШАВАНОГ ПРОБЛЕМА И ЦИЉЕВИ
2. ОПИС РЕШЕЊА ПРОБЛЕМА
3. СТРУКТУРЕ ПОДАТАКА
4. ТЕСТИРАЊЕ И РЕЗУЛТАТИ ТЕСТОВА

# ПРЕДЛОЗИ ЗА ДАЉА УСАВРШАВАЊА

**ОПИС РЕШАВАНОГ ПРОБЛЕМА И ЦИЉЕВИ**

У модерним вишенитним апликацијама, ефикасно управљање меморијом представља критичан изазов који значајно утиче на перформансе и стабилност система. Традиционални приступи управљању меморијом суочавају се са неколико значајних проблема у вишенитном окружењу. Када више нити истовремено покушава да алоцира или ослободи меморију, може доћи до загушења због потребе за синхронизацијом приступа заједничким ресурсима.

Ово загушење често резултира значајним падом перформанси, посебно у системима са великим бројем нити. Додатни проблем представља фрагментација меморије, која се јавља када се меморија временом дели на мале, неискоришћене делове. Ово не само да смањује ефикасност искоришћења меморије, већ може довести и до ситуација где систем не може да задовољи захтеве за већим континуалним блоковима меморије, иако укупна количина слободне меморије може бити довољна.

Имплементација ***thread-safe*** механизама:

* Осигуравање конзистентности меморијских операција у вишенитном окружењу
* Елиминација ***race*** ***condition***-а и ***deadlock***-ова
* Минимизација времена проведеног у критичним секцијама

Оптимизација перформанси:

* Смањење времена чекања нити на приступ меморијским ресурсима
* Ефикасна расподела меморијских захтева преко више хипова
* Балансирање оптерећења између доступних меморијских ресурса

Управљање меморијским ресурсима:

* Прецизно праћење свих меморијских алокација и деалокација
* Превенција цурења меморије кроз робустан систем праћења
* Минимизација фрагментације меморије
* Оптимизација искоришћења доступне меморије

**ОПИС РЕШЕЊА ПРОБЛЕМА И ДИЗАЈН СИСТЕМА**

Имплементирано решење се заснива на систему управљања меморијом који користи више независних хипова и напредне механизме синхронизације. Централну компоненту система представља ***AdvancedHeapManager*** класа која координира рад свих компоненти и имплементира следеће кључне концепте:

Архитектура система:

* Модуларна структура са јасно дефинисаним компонентама
* Вишеслојни систем праћења меморијских алокација
* Aрхитектура која омогућава лако проширивање функционалности

Механизми синхронизације:

* Употреба критичних секција за заштиту приступа дељеним ресурсима
* Имплементација ***fine-grained*** ***locking*** механизама за минимизацију контенције
* Коришћење **атомичних** операција за ажурирање дељених бројача
* Хијерархијски систем закључавања који спречава ***deadlock*** ситуације

***Round-Robin*** алгоритам:

* Равномерна дистрибуција меморијских захтева преко свих доступних хипова
* Атомично инкрементирање бројача за избор следећег хипа
* Аутоматско прескакање неактивних или преоптерећених хипова

Управљање меморијским блоковима:

* Ефикасно праћење величине и локације сваке алокације
* Повезивање алокација са специфичним хиповима
* Двоструко повезане листе за брзо проналажење и ажурирање информација о алокацијама

A diagram of a server

Description automatically generated

**Слика 1: Дијаграм дизајна система**

**СТРУКТУРЕ ПОДАТАКА**

**Array<T>**

* Генеричка имплементација динамичког низа
* Аутоматско проширивање капацитета по потреби
* Ефикасно управљање меморијом са минималним бројем реалокација
* Оптимизовани методи за додавање и приступ елементима
* Управљање животним веком објеката

template<typename T>

class Array {

private:

T\* data;

size\_t capacity;

size\_t length;

public:

Array(size\_t initialCapacity = 4) : capacity(initialCapacity), length(0) {

data = new T[capacity];

}

~Array() {

delete[] data;

}

void add(const T& item) {

if (length >= capacity) {

size\_t newCapacity = capacity \* 2;

T\* newData = new T[newCapacity];

for (size\_t i = 0; i < length; i++) {

newData[i] = data[i];

}

delete[] data;

data = newData;

capacity = newCapacity;

}

data[length++] = item;

}

T& operator[](size\_t index) {

return data[index];

}

size\_t size() const {

return length;

}

};

**MemoryBlock структура**

* Показивач на алоцирану меморију (void\* ptr)
* Величина алокације у бајтовима (size\_t size)
* Индекс хипа који управља блоком (size\_t heapIndex)
* Показивач на следећи блок у листи (MemoryBlock\* next)
* Додатне информације за праћење перформанси

struct MemoryBlock {

void\* ptr;

size\_t size;

size\_t heapIndex;

MemoryBlock\* next;

MemoryBlock(void\* p, size\_t s, size\_t hi) : ptr(p), size(s), heapIndex(hi), next(nullptr) { }

};

**Heap структура**

* Критична секција за синхронизацију приступа
* Показивач на први блок у листи алокација
* Бројач укупног броја активних алокација
* Статистички подаци о коришћењу хипа
* Механизми за праћење фрагментације

struct Heap {

CRITICAL\_SECTION critSection;

MemoryBlock\* allocations;

size\_t totalAllocations;

Heap() : allocations(nullptr), totalAllocations(0) {

InitializeCriticalSection(&critSection);

}

~Heap() {

while (allocations) {

MemoryBlock\* next = allocations->next;

free(allocations);

allocations = next;

}

DeleteCriticalSection(&critSection);

}

};

**ТЕСТИРАЊЕ И РЕЗУЛТАТИ ТЕСТОВА**

Систем је подвргнут тестирању које обухвата различите сценарије употребе и оптерећења:

* Конфигурација тестног окружења
* 8 паралелних нити које симулирају реално оптерећење
* 4 независна хипа за управљање меморијом
* Различите величине алокација (од 1 до 1000 бајтова)
* Насумични временски интервали између операција

**Тестни сценарији**

* Симултане алокације из више нити
* Мешавина малих и великих алокација
* Истовремене алокације и деалокације
* Дуготрајно извршавање са великим бројем операција
* Тестови за проверу цурења меморије
* Стрес тестови са максималним оптерећењем

**Метрике праћења**

* Време извршавања операција
* Дистрибуција оптерећења између хипова
* Број успешних алокација и деалокација
* Праћење фрагментације меморије

**Резултати тестирања**

* Успешна расподела оптерећења преко свих хипова
* Минимална контенција између нити
* Нема детектованих цурења меморије
* Стабилне перформансе под различитим оптерећењима
* Ефикасно искоришћење меморијских ресурса

**Thread 5 freed memory at 000001FC20E206E0**

**Thread Thread 0 freed memory at 000001FC20E222E0**

**Attempting to free invalid pointer**

**Thread 4 freed memory at 000001FC20E21EE0**

**Thread 2 freed memory at 000001FC20E21AE0**

**ttempting to free invalid pointer**

**Attempting to free invalid pointer**

**Thread 3 freed memory at 000001FC20E20AE0**

**------------------------------------**

**| HEAP | Allocations |**

**------------------------------------**

**| 0 | 10 |**

**------------------------------------**

**| 1 | 10 |**

**------------------------------------**

**| 2 | 10 |**

**------------------------------------**

**| 3 | 09 |**

**------------------------------------**

**ПОТЕНЦИЈАЛНА УНАПРЕЂЕЊА**

На основу анализе тренутне имплементације и резултата тестирања, идентификовано је неколико праваца за даље унапређење система:

* Унапређење алгоритма расподеле
* Имплементација адаптивних алгоритама за расподелу оптерећења
* Увођење хеуристика за избор оптималног хипа
* Подршка за приоритизацију критичних алокација
* Динамичко прилагођавање броја активних хипова

Оптимизација перформанси:

* Имплементација ***lockless*** алгоритама где је могуће
* Увођење ***cache-aware*** структура података
* Оптимизација величине и трајања критичних секција
* Побољшање локалности података за боље перформансе

Управљање меморијом:

* Имплементација алгоритама за дефрагментацију
* Подршка за различите стратегије алокације по хипу
* Увођење механизама за компакцију меморије
* Оптимизација праћења меморијских блокова
* Додавање подршке за различите типове меморијских алокација