ДОСЛІДЖЕННЯ ЗАСОБІВ ПОТОКОВОГО ПРОГРАМУВАННЯ, ВІЛЬНИХ ВІД ПЕРЕШКОД БЛОКУВАНЬ І ОЧІКУВАНЬ

OBSTRUCTION-, LOCK-, AND WAIT-FREE PROGRAMMING TOOLS

Виконав: Андрійченко Даниїл, студент ІПЗ – 4

Керівник: Бублик Володимир Васильович

АКТУАЛЬНІСТЬ ТЕМИ

Переваги структур даних, вільних від блокувань та очікувань:

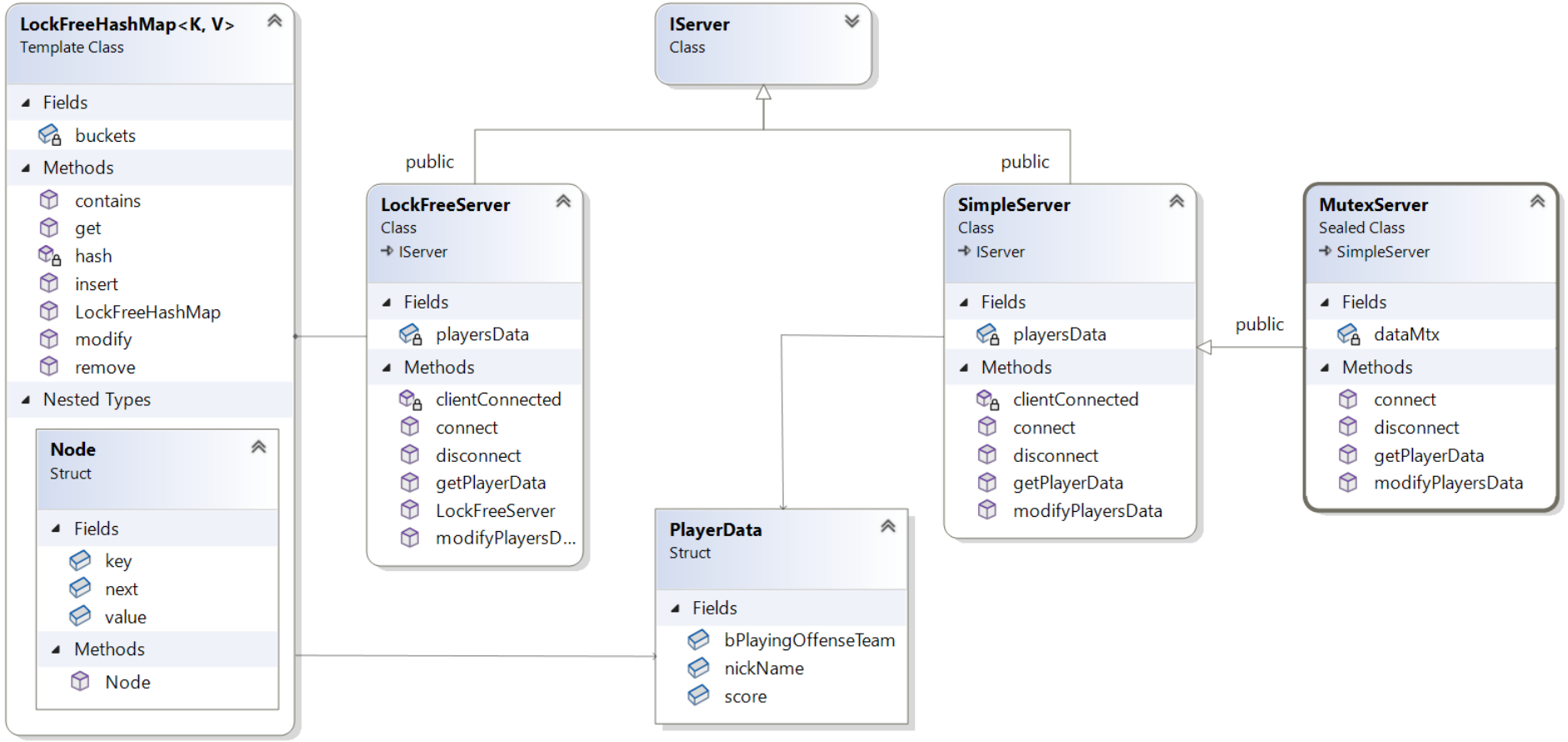
- Використання паралельності без блокування
- Краще піддаються масштабуванню
- Відсутність взаємних блокувань
- Підвищена відмовостійкість системи
- Низька латентність операцій

МЕТА РОБОТИ

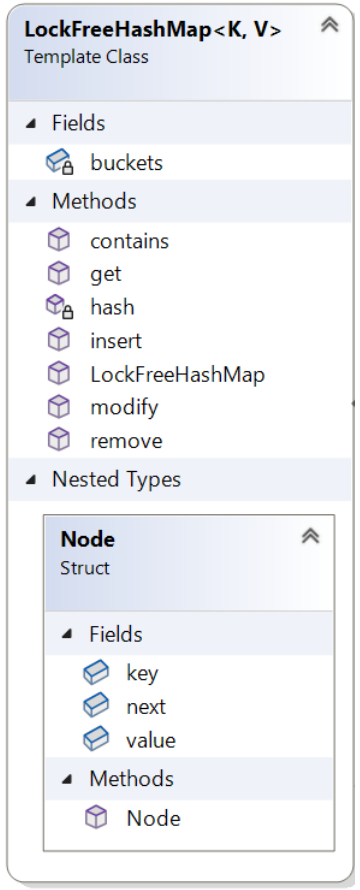
Мета дослідження: Порівняльний аналіз відомих методів синхронізації доступу до структур даних за допомогою двох типів мьютексів та створеними в роботі засобами на основі атомарних операцій.

Мета розробки: розробка та реалізація структур даних (хеш таблиць), вільних від блокувань; створення прототипу високопродуктивної програмної системи обробки неблокованих хеш таблиць.

ДІАГРАМА КЛАСІВ



КЛАС LOCK-FREE HASHMAP



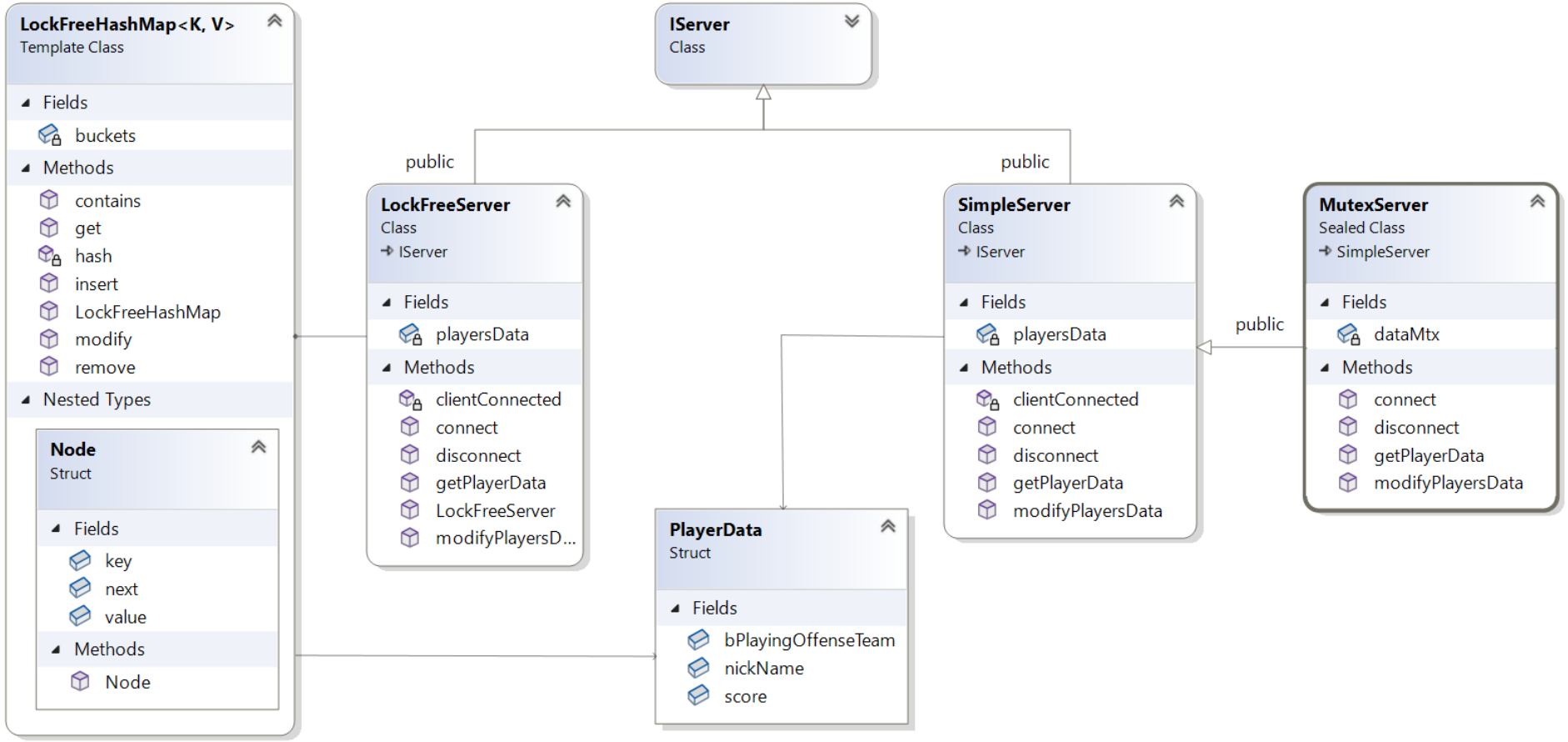
ХЕШ ТАБЛИЦЯ, ВІЛЬНА ВІД БЛОКУВАНЬ

```
template <typename K, typename V>
class LockFreeHashMap
{
public:
    explicit LockFreeHashMap(size_t numBuckets): buckets(numBuckets) {}
    bool insert(const K& key, const V& value);
    bool remove(const K& key);
    bool contains(const K& key) const;
    std::optional<V> get(const K& key) const;
    bool modify(const K& key, std::function<void(V&)>modificationFunction);
private:
    struct Node;
    std::vector<std::atomic<Node*>> buckets;
    size_t hash(const K& key) const;
};
```

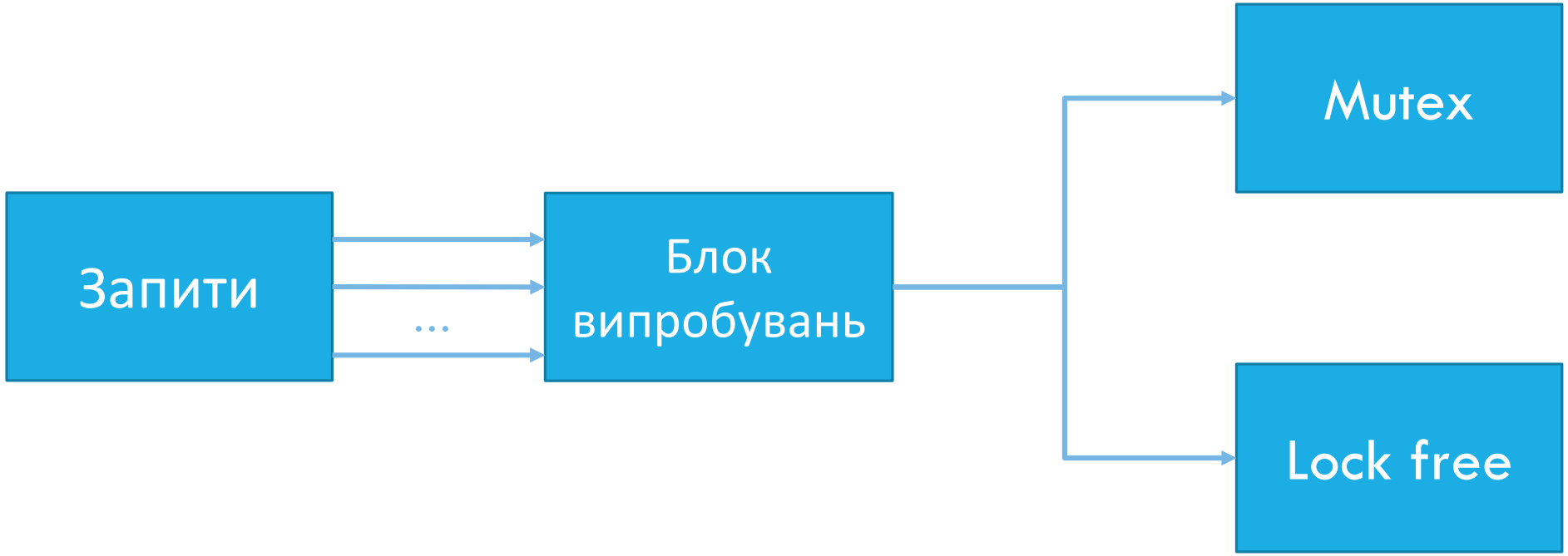
ПРИКЛАД РЕАЛІЗАЦІЇ МЕТОДУ МОДИФІКАЦІЇ

```
bool LockFreeHashMap<K, V>::modify(const K& key, std::function<void(V&)> modificationFunction){
    size_t index   = hash(key);
    Node*  current = buckets[index].load(std::memory_order_acquire);
    while (current)
    {
        if (current->key == key)
        {
            std::shared_ptr<V> oldValue, newValue;
            do
            {
                oldValue = current->value.load(std::memory_order_acquire);
                newValue = std::make_shared<V>(*oldValue);
                modificationFunction(*newValue);
            }
            while (!current->value.compare_exchange_weak(oldValue, newValue, std::memory_order_acq_rel));
            return true;
        }
        current = current->next.load(std::memory_order_acquire);
    }
    return false;
}
```

ДІАГРАМА КЛАСІВ



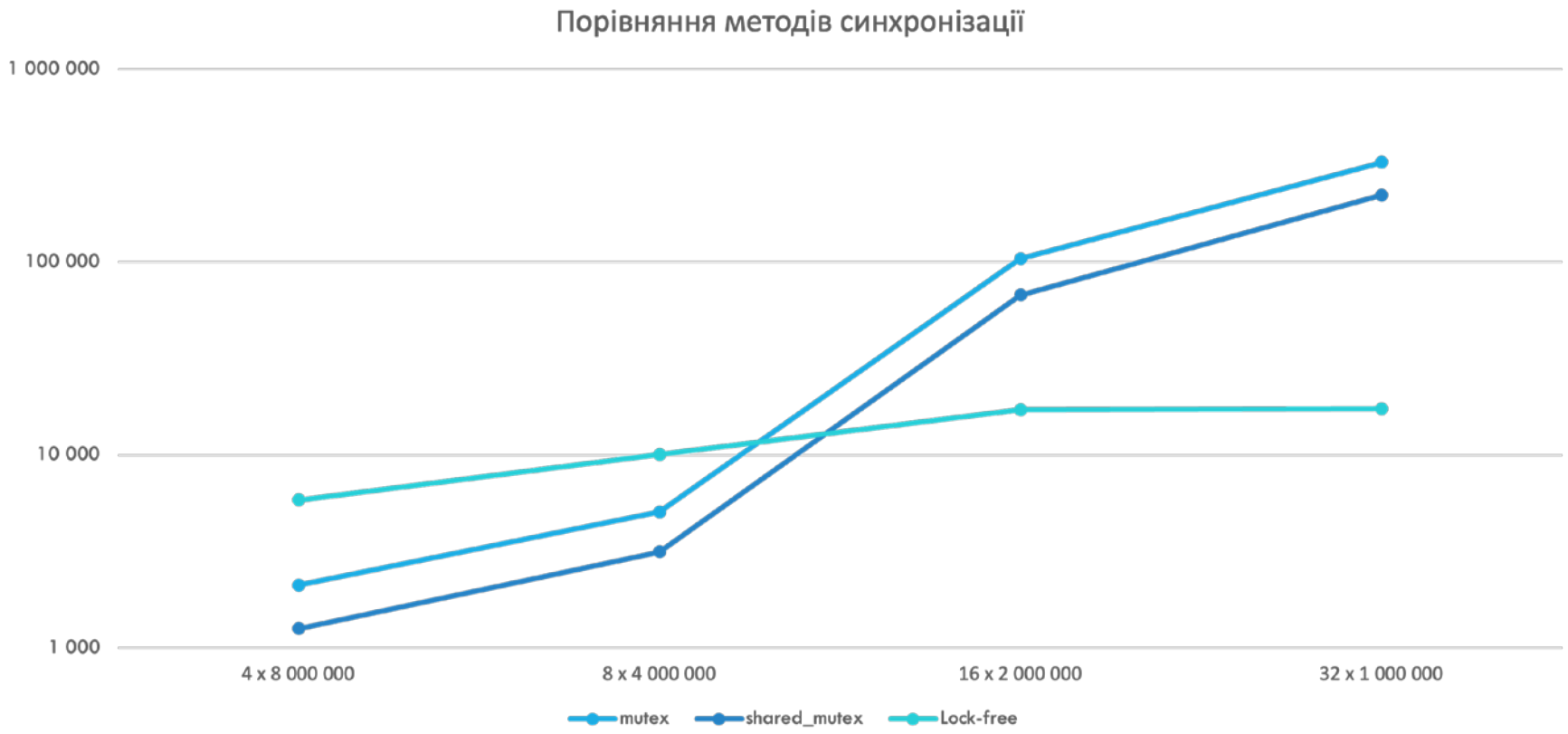
СЦЕНАРІЙ ВИПРОБУВАННЯ НЕБЛОКУВАЛЬНОЇ ХЕШ ТАБЛИЦІ



ПОРІВНЯЛЬНИЙ АНАЛІЗ БЛОКУВАЛЬНИХ ТА ВІЛЬНИХ ВІД БЛОКУВАНЬ МЕТОДІВ СИНХРОНІЗАЦІЇ

К-сть потоків та операцій/час, *ms*	mutex	shared_mutex	Lock-free sequential consistency	Lock-free acquire/release semantics
4 x 8 000 000	2 112	1 257	5 852	5 623
8 x 4 000 000	5 061	3 148	10 063	9 804
16 x 2 000 000	104 162	67 524	17 180	17 035
32 x 1 000 000	329 353	222 324	17 400	17 231

ПОРІВНЯЛЬНИЙ АНАЛІЗ БЛОКУВАЛЬНИХ ТА ВІЛЬНИХ ВІД БЛОКУВАНЬ МЕТОДІВ СИНХРОНІЗАЦІЇ



ПОРІВНЯЛЬНИЙ АНАЛІЗ БЛОКУВАЛЬНИХ ТА ВІЛЬНИХ ВІД БЛОКУВАНЬ МЕТОДІВ СИНХРОНІЗАЦІЇ

К-сть потоків та операцій/час, *ms*	mutex	shared_mutex	Lock-free sequential consistency	Lock-free acquire/release semantics
4 x 8 000 000	2 112	1 257	5 852	5 623
8 x 4 000 000	5 061	3 148	10 063	9 804
16 x 2 000 000	104 162	67 524	17 180	17 035
32 x 1 000 000	329 353	222 324	17 400	17 231

ПРИКЛАДИ ВИКОРИСТАННЯ НЕБЛОКУВАЛЬНИХ СТРУКТУР ДАНИХ

- Операційні системи та системне програмування
- Багатокористувацькі ігри
- Фінансові системи та транзакційні системи
- Бази даних
- Телекомунікації та мережі
- Системи реального часу
- Високопаралельні системи
- Багатоядерні та багатопроцесорні системи
- Імітації та моделювання
- Паралельні алгоритми

ПРАКТИЧНА ЗНАЧИМІСТЬ РЕЗУЛЬТАТІВ РОБОТИ

- Розроблена в ході роботи хеш таблиця, вільна від блокувань, призначена для створення високопродуктивних багатопотокових програмних засобів
- Галузева та освітня цінність полягатиме у впровадженні результатів роботи в курси інженерії програмного забезпечення

ВИСНОВКИ ТА РЕКОМЕНДАЦІЇ ДЛЯ ПОДАЛЬШИХ ДОСЛІДЖЕНЬ

Компроміс між приростом продуктивності неблокувальних структур даних і часом, витраченим на їх розробку.

Такі структури особливо цінні у сценаріях високопродуктивних систем реального часу.

У подальших роботах можна дослідити програмування без очікувань шляхом створення демонстраційного прототипу багатокористувацької програмної системи, включаючи планування виконання потоків на рівні ОС.

ДЯКУЮ ЗА УВАГУ!