НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_Кафедра обчислювальної техніки\_\_\_\_\_\_\_\_\_\_\_\_

(повна назва кафедри, циклової комісії)

**КУРСОВИЙ ПРОЕКТ**

з дисципліни «Паралельне програмування»

(назва дисципліни)

на тему: «Розробка програмного забезпечення для паралельних комп’ютерних систем»

Студента 3 курсу \_\_ІП-42\_\_ групи

напряму підготовки 050103 «Програмна інженерія»

\_ Кафтанатія Богдана Сергійовича \_\_\_\_\_\_\_\_\_

(прізвище та ініціали)

Керівник доцент Корочкін О.В.

Національна оцінка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_\_\_\_\_

Оцінка: ECTS \_\_\_\_\_\_\_\_\_\_\_

Члени комісії \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали

Київ- 2017 р.

Національний технічний університет України

“Київський політехнічний інститут”

Факультет (інститут) інформатики та обчислювальної техніки

( повна назва )

Кафедра обчислювальної техніки

( повна назва )

Освітньо-кваліфікаційний рівень бакалавр

Напрям підготовки 6.050103 «Програмна інженерія»

# (шифр і назва)

## ***З А В Д А Н Н Я***

### НА КУРСОВИЙ ПРОЕКТ СТУДЕНТУ

\_\_\_\_\_\_\_\_\_\_Кафтанатію Богдану Сергійовичу\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(прізвище, ім’я, по батькові)

1. Тема роботи «Розробка програмного забезпечення для паралельних

комп’ютерних систем»

керівник роботи Корочкін Олександр Володимирович к.т.н.**,** доцент

( прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

2. Строк подання студентом роботи 11 травня 2017 р.

3. Вхідні дані до роботи

* Математична задача МА = МВ \* МС + max(Z) \* MO \* MK
* Структури ПКС ОП та ПКС ЛП
* Мови і бібліотеки програмування: Java

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

* порівняння реалізації механізму моніторів секції в мовах і бібліотеках паралельного програмування
* розробка і тестування програми ПРГ1 для ПКС ОП
* розробка і тестування програми ПРГ2 для ПКС ЛП

5. Перелік графічного матеріалу

* Структурна схема ПКС СП
* Структурна схема ПКС ЛП
* Структура класу TaskControl ПРГ1
* Структурна схема взаємодії задач ПГР2

7. Дата видачі завдання \_\_\_\_\_­\_2.03.2017\_\_\_\_\_

#### **КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |
| --- | --- | --- |
| №  з/п | Назва етапів виконання КР | Строк виконання етапів КР |
| 1 | Виконання розділу 1 | 13.03.2017 |
| 2 | Виконання розділу 2 | 03.04.2016 |
| 3 | Виконання розділу 3 | 24.04.2016 |
| 4 | Оформлення КР | 8.05.2016 |
| 5 | Перевірка КР викладачем | 11.05.2016 |
| 6 | Захист КР | 18.05.2016 |

**Студент \_\_\_\_\_\_\_\_\_ \_\_ Кафтанатій Б.С.\_\_\_\_**

( підпис ) (прізвище та ініціали)

**Керівник роботи \_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_Корочкін О.В.\_\_\_\_\_\_**

( підпис ) (прізвище та ініціали)

ЗМІСТ

ВСТУП……………………………………………………………………………

РОЗДІЛ 1. ПОРІВНЯННЯ РЕАЛІЗАЦІЇ МЕХАНІЗМУ МОНІТОРІВ СЕКЦІЇ

В МОВАХ І БІБЛІОТЕКАХ ПАРАЛЕЛЬНОГО ПРОГРАМУВАННЯ ……...

* 1. Загальні поняття……………………………………………………
  2. Монітори в мові Ада……………………………………………….
  3. Монітори в мові Java……………………………………………….
  4. Монітори в мовах C# та C++………………………………………
  5. Висновки до розділу 1……………………………………………..

РОЗДІЛ 2. РОЗРОБКА ПРОГРАМИ ПРГ1 ДЛЯ ПКС ОП……………………

* 1. Огляд паралельно обчислювальної системи…………………………..
  2. Розробка паралельного математичного алгоритму……………………
  3. Аналіз задачі з точки зору КНП………………………………………..
  4. Розробка алгоритмів процесів………………………………………….
  5. Розробка схеми взаємодії процесів…………………………………….
  6. Розробка програми ПРГ1……………………………………………….
  7. Тестування програми ПРГ1 …………………………………………….
  8. Висновки до розділу 2 …………………………………………………..

РОЗДІЛ 3. РОЗРОБКА ПРОГРАМИ ПРГ2 ДЛЯ ПКС ЛП ……………………

* 1. Розробка паралельного математичного алгоритму………………….
  2. Розробка алгоритмів процесів…………………………………………
  3. Розробка схеми взаємодії процесів…………………………………..
  4. Розробка програми ПРГ2………………………………………………
  5. Тестування програми ПРГ2……………………………………………
  6. Висновки до розділу 3…………………………………………………

ОСНОВНІ РЕЗУЛЬТАТИ І ВИСНОВКИ ДО РОБОТИ……………………..

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ………………………………………

ДОДАТКИ…………………………………………………….............................

**ВСТУП**

Курсова робота по дисципліні «Паралельні і розподілені обчислення» складається з трьох розділів.

В першому розділі «Порівняння реалізації механізму моніторів в мовах і бібліотеках паралельного програмування» описаний механізм моніторів в паралельних програмах, а також його застосування в різних мовах та бібліотеках програмування.

Другий та третій розділи присвячені розробці програми для обчислення математичної задачі в паралельній комп’ютерній системі зі спільною та локальною пам’яттю відповідно. Програмне забезпечення для комп’ютерної системи зі спільною пам’яттю розроблено на мові Java. Проведено тестування отриманих програмних продуктів і зроблено висновки по їх ефективності.

Лістинги та алгоритми розроблених програм наведено у додатках.

**РОЗДІЛ 1. ПОРІВНЯННЯ РЕАЛІЗАЦІЇ МЕХАНІЗМУ МОНІТОРІВ СЕКЦІЇ В МОВАХ І БІБЛІОТЕКАХ ПАРАЛЕЛЬНОГО ПРОГРАМУВАННЯ**

Листів

Лист

1

Літера

6.050102

ІП-42

**Розробка програмного забезпечення для паралельних комп’ютерних систем**

НТУУ КПІ 17 4211 - 000 ПЗ

Зм.

Виконав

Керівник

Конс.

Н. контр.

Зав. каф.

№ докум.

Кафтанатій Б.С.

Корочкін О.В.

Корочкін О.В.

Стіренко С.Г

Лит

Підпис

Дата

1

* 1. **Загальні поняття**

Ідея монітору, яку запропонував Б. Хансен і розвинув Ч. Хоар, ґрунтується на об’єднанні змінних, що описують спільний ресурс, і дій, які визначають засобі доступу до спільного ресурсу [2][3]. Монітор – програмний модуль, що містить (ховає) змінні та надає процедури для роботи з ними, причому доступ до змінних можливий тільки через процедури монітора.

Монітор – засіб розподілу ресурсів і взаємодії процесів[1]. Це призначення монітора реалізується за допомогою властивостей, якими наділені процедури монітора. Характерна особливість процедур монітора – взаємне виключення ними одне одного. У будь-який момент часу може виконуватись тільки одна процедура монітора. З спроби виклику іншим процесом процедури, що виконується, або іншої процедури монітора цей процес блокується і розміщується в черзі блокованих процесів доти, поки активний процес не закінчить виконання процедури монітора. Тобто в моніторі не може «знаходитись» більше одного процесу. Така властивість процедур монітора забезпечує взаємне виключення процесів, які працюють з монітором.

У моніторі декларують локальні змінні (спільні змінні), які захищені монітором, і процедури монітора. Значення локальних змінних можуть бути встановлені під час створення монітора. Далі значення цих змінних можуть бути прочитані або зміненні процесами тільки за допомогою процедур, визначених в моніторі.

Загальна структура монітора:

monitor Ім’я монітора;

- - Опис локальних даних

- - Опис процедур для доступу по даних

begin

- - Ініціалізація локальних даних

end Ім’я монітора;

Властивості процедур монітора забезпечують вирішення завдання взаємного виключення за доступу до спільних ресурсів, об’явленими в моніторі. При цьому монітор формує чергу процесів, які викликали процедури монітора і є блокованими через зайнятість монітора (тобто спільного ресурсу).

Приклад монітора:

monitor Склад;

Товар: Data;

procedure На\_Склад (T: in Data);

procedure Зі\_Складу (T: out Data);

begin

Товар:= 0.0ж

end Склад;

* 1. **Монітори в мові Ада**

Концепцію моніторів у стандарті мови Ада реалізовано у вигляді спеціальних програмних модулів – захищених модулів (protected units) [5],[6]. Їх призначення – розширення можливості мови для програмування паралельних процесів, зокрема, для вирішення проблеми доступу до спільних ресурсів і синхронізації процесів. Крім того, захищені модулі забезпечують підтримку різних парадигм систем реального часу, для розроблення яких мову Ада використовують в першу чергу.

Спільні дані і операції над ними (захищені операції) об’єднуються в захищеному модулі, аналогічно тому, як це робиться в інших модулях мови Ада – пакетах. Доступ до спільних ресурсів можливий тільки через захищені операції, які мають властивості, що дозволяють вирішити завдання взаємного виключення під час роботи зі спільними ресурсами[1].

Як і всі модулі в мові, захищені модулі складаються зі специфікації і тіла.

Специфікація захищеного модуля:

PROTECTED [TYPE] Ім’я\_Захищеного\_Модуля [Дискримінант]

- - Опис\_Захищених\_Операцій

[PRIVATE]

- - Опис\_Захищених\_Елементів

END Ім’я\_Захищеного\_Модуля;

Захищенні модулі – це:

* + *Захищені функції,*
  + *Захищені процедури,*
  + *Захищені входи.*

Захищенні функції забезпечують доступ тільки до читання захищених елементів. Але дозволяють робити це одночасно всім процесам автоматичним копіюванням елементів, які зчитуються. Це порушую головну властивість процедур монітора, яка дозволяє знаходитися в моніторі тільки одному процесу, але «порушення» дозоляє скоротити час доступу до захищених елементів і не має будь-яких наслідків, оскільки зміна даних заборонена і не виконується.

Захищені процедури забезпечують ексклюзивний доступ до захищених елементів через читання і запис.

Захищенні входи забезпечують ті самі функції, що й захищені процедури, додатково реалізуючи за допомогою бар’єрів ексклюзивний (умовний) доступ до тіла захищеного входу. Це дозволяє реалізувати за допомогою входів вирішення завдання умовної синхронізації.

Приватна частина специфікації обмежує видимість захищених елементів: операцій і об’єктів, що описані в ній. Спільні дані, доступ до яких контролюється захищеним модулем, описуються в приватній частині його специфікації[1].

Приклад 1.1 Специфікації захищеного модуля

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

--- Ада. Захищений модуль --

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

protected Контроль is

procedure Включення; -- захищені підпрограми

function Перемкнути (X: Float);

end Контроль.

protected type Сенсор is

entry Чекати; -- захищенні входи

entry Сигнал;

procedure Зміна\_Стану(x: in float);

function Замір\_Стану return float;

private

Прп: Boolean := False; -- Прапор

Стан: Float;

end

protected Блок232(Номер: in Positive) is

entry Параметри(X: out integer);

procedure Зміна\_Параметру(Y: in integer);

private

- - захищений елемент

Об’єкт: array(1 .. Номер) of integer;

end Блок232;

Тіло захищеного модуля реалізує захищені операції, які об’явлені і його специфікації, використовуючи для цього локальні ресурси, які можуть бути об’явлені в тілі модуля.

PROTECTED BODY Ім’я\_Захищеного\_Модуля IS

- - Локальні\_Описи

BEGIN

- - Реалізація захищених операцій і захищених елементів

END Ім’я\_Захищеного\_Модуля;

Захищені процедури і функції реалізуються в тілі захищеного модуля, як це робиться в тілі пакета. На відмінну від модулів-задач, реалізація захищеного входу в тілі захищеного модуля не пов’язана з оператором приймання accept, а виконується за допомогою тіла входу, в якому обов’язково використовується бар’єр.

Описання тіла захищеного входу:

ENTRY Ім’я\_Захищеного\_Входу WHEN Умова IS

BEGIN

. . . - - Послідовність\_Операторів

END Ім’я\_Захищеного\_Входу;

Конструкція When Умова – це бар’єр, де Умова – логічний вираз, який визначає відкритий або закритий вхід. Перевірка умови в бар’єрі виконується під час виклику захищеного входу. Якщо значення виразу Умова дорівнює true, то вхід відкритий і виконується тіло захищеного входу, інакше вхід є зачинений і виконання процесу, який викликав цей вхід, блокується до того часу, поки значення виразу Умова в бар’єрі не буде змінено в true іншою задачею за допомогою захищеної процедури або іншого захищеного входу.

В прикладі 1.2. наведено реалізацію тіла захищеного модуля Сенсор.

Приклад 1.2 Специфікації захищеного модуля

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

--- Ада. Тіло захищеного модуля ---

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

protected body Сенсор is

- - тіло захищеного входу з бар’єром

entry Чекати when Прп is

begin

Прп:= False;

end Чекати;

- - тіло захищеного входу з бар’єром

entry Сигнал when not Прп is

begin

Прп:= True;

end Сигнал;

procedure Зміна\_Стану(x : in float) is

begin

Стан := x;

end Зміна\_Стану;

function Замір\_Стану return float is

begin

return Стан;

end Замір\_Стану;

end Сенсор;

Структур захищеного модуля Сенсор показано на рис. 1.1. Захищену функцію Замір\_Стану() і процедуру Замір\_Стану() зображено справа, захищені входи Чекати() і Сигнал() – зліва. Захищені елементи (Стан, Прп) зображено всередині захищено модуля в овалі.

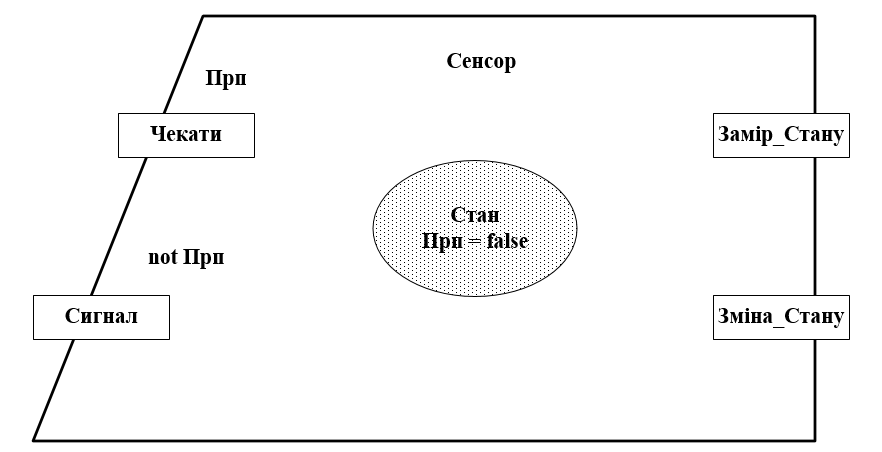


Рис. 1.1. Структура захищеного модуля

Виклик захищеної функції дозволяє процесу зчитувати дані із захищеного модуля. Кілька процесів можуть виконувати таке читання одночасно, викликаючи потрібні функції. Під час виконання читання в тілі захищеної функції заборонено зміну даних. Тіло захищеної функції може містити виклик іншої захищеної функції, але не виклик захищеної процедури.

Виклик захищеної процедури дозволяє процесу як читати, так і змінювати інформацію в захищеному модулі. На відмінну від захищеної функції під час виконання захищеної процедури дозволяється змінювати дані. Якщо кілька процесів виконують виклик захищених процедур, то тільки один з них отримує можливість роботи з викликаною процедурою. У тілі захищеної процедури дозволено виклик як захищеної функції, так і захищеної процедури.

Виклик закритого захищеного входу приводить до блокування процесу до того часу, поки вхід не стане відкритим, тобто умовний вираз в бар’єрі набуде значення True. Це може статися в разі виконання іншим процесом потрібної захищеної операції, пов’язаної зі змінними, використаними в бар’єрі.

Блокований процес розміщується в черзі, яка пов’язана із входом, а також за змінними, що використовуються в бар’єрі. Якщо вхід відкритий, то виконується тіло входу.

Приклад 1.3.

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

--- Ада. Захищений модуль у завданні взаємного виключення ---

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

protected Буфер is

procedure Додати(Вклад: out Positive);

procedure Видалити(Вклад: out Positive);

private

Лічильник: Integer:= 0;

end Буфер;

- - тіло захищеного модуля

protected body Буфер is

procedure Додати(Вклад: out Positive) is

begin

Лічильник := Лічильник + 1;

Вклад := Лічильник;

end Додати;

procedure Видалити(Вклад: out Positive) is

begin

Лічильник := Лічильник - 1;

Вклад := Лічильник;

end Видалити;

end Буфер;

Задачі додають або зменшують значення змінної Лічильник, викликаючи процедури Додати і Видалити захищеного модуля Буфер:

Буфер.Додати(Зарплата); Буфер.Видалити(Плата);

Захищений модуль Буфер гарантує синхронізований доступ задач до захищеної змінної Лічильник. Черги під час роботи із захищеним модулем не створюються, оскільки використовуються тільки захищені процедури, а не захищені входи.

У прикладі 1.4. захищений модуль Вклад виконує роль буфера, куди задачі Клієнт\_А і Клієнт\_В записують і звідки читають дані. Захищений модуль Вклад повинен забезпечувати взаємовиключний доступ задач до спільного ресурсу, яким є змінна Рахунок, а також синхронізацію процесів залежно від стану ресурсу.

Приклад 1.4.

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

--- Ада. Захищений модуль у завданні взаємного виключення та ---

--- синхронізації процесів ---

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

protected Вклад is

entry В\_Банк(M : in Гроші);

entry З\_Банку(M : out Гроші);

private

Рахунок: Гроші;

Прапор: Boolean := False;

end Вклад;

- - - - - - - - -

protected body Вклад is

entry В\_Банк(M : in Гроші) when Прапор = False is

begin

Рахунок := M;

Прапор := True;

end В\_Банк;

entry З\_Банку(M : out Гроші) when Прапор = true is

M := Рахунок;

Прапор := False;

end З\_Банку;

end Вклад;

- - - - - - - - -

task Клієнт\_А;

task body Клієнт\_А is

Дохід : Гроші;

begin

. . . .

- - виклик входу В\_Банк

Вклад.В\_Банк(Дохід);

. . . .

end Клієнт\_А;

- - - - - - - - -

task Клієнт\_В;

task body Клієнт\_В is

Сплата : Гроші;

begin

. . . .

- - виклик входу З\_Банку

Вклад.З\_Банку(Сплата);

. . . .

end Клієнт\_В;

* 1. **Монітори в мові Java**

У кожного об'єкта в Java є свій власний неявний монітор. Коли метод типу synchronized викликається для об'єкта, відбувається звернення до монітора об'єкта щоб визначити, чи виконує в даний момент будь-який інший потік метод типу synchronized для даного об'єкта. Якщо ні, то поточний потік отримує дозвіл увійти в монітор. Вхід в монітор називається також блокуванням (locking) монітора. Якщо при цьому інший потік вже увійшов в монітор, то поточний потік повинен чекати доти, поки інший потік не покине монітор. Таким чином монітор Java вводить черговість в паралельну обробку. Цей спосіб називається також перетворенням в послідовну форму (serialization).

Оголошення методу synchronized не має на увазі, що тільки один потік може одночасно виконувати цей метод, як у випадку критичного ділянки (critical sections) Має на увазі, що в будь-який момент часу тільки один потік може викликати цей метод (або будь-який інший метод типу synchronized) для конкретного об'єкта. Таким чином, монітори Java пов'язані з об'єктами, але не з блоками кода. Два потоки можуть паралельно виконувати один і той же метод типу synchronized за умови, що цей метод викликаний для різних об'єктів.

Монітори не є об'єктами мови Java, у них немає атрибутів або методів. Доступ до моніторів можливий на рівні власного коду JVM.

Синхронізація дозволяє здійснювати блокування потоків, запобігаючи асинхронний доступ до певних методів. Однак часто виникає необхідність узгодження потоків, коли виконання одного потоку може залежати від завершення в іншому потоці запиту на обслуговування або виконання певної операції. При цьому важливо, щоб очікуючий потік або потоки очікували, не використовуючи час ЦП на опитування для постійної перевірки деяких умов. Щоб уникнути втрати часу, пов'язані з опитуванням, Java використовує елегантний механізм взаємодії між потоками через методи wait (), notify (), notifyall (). Усі три методи оголошені в класі Object.

final void wait()

final void notify()

final void notifyall()

wait () - наказує викликаному потоку віддати монітор і перейти в стан очікування, поки який-небудь інший потік не ввійде в той же монітор і не викличе метод notify ().

notify () - активізує один з очікуючих потоків, що викликали метод wait () того ж об'єкта. Запускається потік з найбільшим пріоритетом.

notifyall () - активізує всі очікують потоки, що викликали метод wait () того ж об'єкта.

Метод wait () має додаткову форму, що дозволяє задати період очікування. Усі три методи служать інтерфейсом для взаємодії з монітором об'єкта і їх можна викликати тільки в тому випадку, коли поточний потік володіє правами на монітор об'єкта, тобто всередині методу або блоку типу sysnchronized.

Монітори використовуються як для введення черговості в паралельну обробку, так і для узгодження потоків. Виклик методу wait () для даного об'єкта призупиняє поточний потік і вводить його в чергу очікування за умовою (condition variable wait queue) в моніторі об'єкта. Черга містить список всіх потоків , заблокованих усередині методу wait () для даного об'єкта. Виклик методу notify () переводить єдиний потік в активний режим, повідомляючи про те, що умова змінилося.

Зазвичай wait () поміщається в блок try {} обробки виняткових ситуацій.

synchronized int get() {

while( условие ) {

try {

wait();

} catch() {}

}

. . . . . . . . . . . . . . . .

notify();

. . . . . . . . . . . . . . . .

}

Виклик wait() і notify() для масивів здійснюється в межах синхронізуючого блоку.

Obj[]array = getArray();

synchronized(array){

array.wait();

}

. . . . . . . . . . . . .

synchronized(array){

array.notify();

}

* 1. **Монітори в мовах C# та C++**

Об'єкти Monitor надають можливість синхронізувати доступ до області коду за допомогою отримання та звільнення блокування на заданому об'єкті за допомогою методів Monitor.Enter, Monitor.TryEnter і Monitor.Exit. Методи Monitor.Wait, Monitor.Pulse і Monitor.PulseAll можна використовувати, як тільки на області коду встановлена блокування. Метод Wait знімає блокування, якщо вона встановлена, і очікує оповіщення. При отриманні оповіщення метод Wait повертається і знову отримує блокування. Pulse і PulseAll сигналізують про перехід до наступного потоку в черзі очікування.

Інструкції SyncLock в Visual Basic і lock в C # використовують метод Monitor.Enter для установки блокування і метод Monitor.Exit для її зняття. Перевага використання інструкцій мови полягає в тому, що весь вміст інструкції lock або SyncLock включається в інструкцію Try. Інструкція Try забезпечується блоком Finally, що гарантує зняття блокування.

Monitor блокує об'єкти (зі посилальним типом), а не типи значень. Оскільки методам Enter і Exit можна передати тип значення, він упаковується окремо для кожного виклику. Оскільки при кожному виклику створюється окремий об'єкт, метод Enter ніколи не блокується, і код, імовірно захищається цим методом, насправді не є синхронізувати. Крім того об'єкт, переданий методу Exit, відрізняється від об'єкта, переданого методом Enter, тому Monitor створює виняток SynchronizationLockException з повідомленням "Метод синхронізації об'єкта викликаний з несинхронізованих блоку коду". У наступному прикладі показані ці неполадки.

try

{

int x = 1;

// Виклик Enter () створює загальний об'єкт синхронізації для значення

// з х щораз код виконується, так що Enter ніколи не блокує

Monitor.Enter(x);

try

{

// Код, який повинен бути захищений від монітора.

}

finally

{

// Завжди використовуйте Finally, щоб переконатися, що ви вийти з монітора.

// Виклик Exit () не спрацює !!!

// Об'єкт синхронізації створена для х в Exit () буде відрізнятися

// Чим об'єкт, який використовується в Enter (). SynchronizationLockException

// Буде кинутий.

Monitor.Exit(x);

}

}

catch (SynchronizationLockException SyncEx)

{

Console.WriteLine("A SynchronizationLockException occurred. Message:");

Console.WriteLine(SyncEx.Message);

}

Незважаючи на можливість упаковки змінної типу значення перед викликом методів Enter і Exit, як показано в наступному прикладі, і передачі одного і того ж упакованого об'єкта обом методам, цей спосіб не надає переваг. Зміна змінної ніяк не відбивається на її упакованої копії, та копію неможливо змінити.

int x = 1;

object o = x;

Monitor.Enter(o);

try

{

// Code that needs to be protected by the monitor.

}

finally

{

// Always use Finally to ensure that you exit the Monitor.

Monitor.Exit(o);

}

Необхідно згадати про відмінності у використанні об'єктів Monitor і WaitHandle. Об'єкти Monitor є повністю керованими і переносяться. Крім того, вони більш ефективні відносно вимог до ресурсів операційної системи. Об'єкти WaitHandle представляють об'єкти очікування операційної системи і використовуються при синхронізації керованого і некерованого коду, вони також надають деякі додаткові можливості операційної системи, наприклад можливість очікування одночасно великої кількості об'єктів.

* 1. **Висновки**

Монітори більш гнучкі, ніж семафори. У формі моніторів порівняно легко можна реалізувати різні синхронізуючі примітиви, зокрема семафори і поштові скриньки. Крім того, монітори дозволяють декільком процесам спільно використовувати програму, що представляє собою критичний ділянку.

Реалізація моніторів вимагає розробки спеціальних мов програмування і компіляторів для них. Монітори зустрічаються в таких мовах як паралельний Евклід, паралельний Паскаль, Java, Ada, C # і т.д. Емуляція моніторів за допомогою системних викликів для звичайних широко використовуваних мов програмування не так проста, як емуляція семафорів.

**РОЗДІЛ 2. РОЗРОБКА ПРОГРАМИ ПРГ1 ДЛЯ ПКС ОП**

**2.1. Огляд паралельної обчислюваної системи**

Листів

Лист

**1**

Літера

**6.050102**

**ІП-42**

**Розробка програмного забезпечення для паралельних комп’ютерних систем**

**НТУУ КПІ 17 4211 - 000 ПЗ**

Зм.

Виконав

Керівник

Конс.

Н. контр.

Зав. каф.

№ докум.

Кафтанатій Б.С.

Корочкін О.В.

Корочкін О.В.

Стіренко С.Г

Лит

Підпис

Дата

**1**

Дана паралельна комп’ютерна система, що складається з Р процесорів, двох пристроїв вводу/виводу та спільної пам’яті. Для даної комп’ютерної системи розробити програмне забезпечення для обчислення виразу:

МА = MB \* MC + max(Z) \* MO \* MK

Структурна схема ПКС СП наведена в додатку В. Вхідні та вихідні дані знаходяться на пристроях вводу/виводу так, як показано на схемі.

Мова для розробки програмного забезпечення – Java.

**2.2. Розробка паралельного математичного алгоритму**

1. a*i* = max(ZH), *i* =

a = max(a, a*i*), *i* =

2. MA*H* = MB \* MC*H* + a \* MO \* MK*H*  , де:

Н = N / P, P – кількість процесів,

*MAH* – *Н* рядків матриці *MA*,

*MСH* – *Н* рядків матриці *MС*,

*MKH* – *Н* рядків матриці *MK*.

Спільні ресурси: *a, MO, MB*.

**2.3. Аналіз задачі з точки зору концепції необмеженого паралелізму (КНП)**

Для оцінки необхідного часу обчислень використаємо теорему Мунро-Петерсена, яка для комп’ютерної системи з необмеженим числом процесорів формулюється наступним чином: якщо виконується обчислення скалярної величини, яке потребує *m* бінарних операцій, то необхідний час обчислень tp:

*tp* ≥ [log2(m + 1)].

Для обчислення а = max(Z) необхідно виконати *N* бінарних операцій порівняння. Тому час виконання буде:

*tp1* ≥ [log2(*N* + 1)].

Для обчислення одного елементу добутку матриць MB \* MC необхідно виконати *N* множень та *N – 1* операцій додавання. Час виконання:

*tp2* ≥ [log2(2*N*)] = 1 + [log2*N*].

Для обчислення одного елементу добутку матриць a \* MO\*MK необхідно виконати *N + 1* множень та *N – 1* операцій додавання. Час виконання:

*tp3* ≥ [log2(2*N + 1*)].

Так як другий та третій етап незалежні, то вони можуть виконуватись паралельно. Тому сумарний час їх виконання буде рівний максимальному з двох:

*tp2,3* ≥ max(*tp2*, *tp3*) = *tp3* = [log2(2*N + 1*)].

Для обчислення одного елементу суми матриць MB \* MC + а \* MO \* MK необхідно виконати *N2* операцій додавання. Час виконання:

*tp4* ≥ [log2(*N2* + 1)].

Сумарний час виконання всіх етапів обчислень буде виражатись наступною формулою:

*tp* ≥ *tp1* + *tp2,3* + *tp4* = [log2(*N* + 1)] + [log2(2*N + 1*)] + [log2(*N2* + 1)].

**2.4. Розробка алгоритмів процесів**

Так як розроблюване програмне забезпечення має бути масштабованим, тобто має працювати на системі з будь-якою кількістю процесорів, то зручним варіантом реалізації є написання єдиного алгоритму для всіх задач.

|  |  |
| --- | --- |
| **Крок алгоритму** | **ТС, КД** |
| 1. Якщо *tid = 0*, ввести МВ та МО |  |
| 2. Якщо *tid = 0*, сигнал задачам *1…Р-1* про завершення *вводу 1*. | S*i,1* i = |
| 3. Якщо *tid = Р-1*, вести МС, Z, MK. |  |
| 4. Якщо *tid = Р-1*, сигнал задачам *0…Р-2* про завершення *вводу 2*. | S*i,2* i = |
| 5. Якщо *tid != 0*, чекати сигналу про завершення *вводу 1* від задачі *0*. | W*0,1* |
| 6. Якщо *tid != Р-1*, чекати сигналу про завершення *вводу 2* від задачі *Р-1* | W*P-1,2* |
| 7. Обчислення a*i* = max(Z*H*). |  |
| 8. Обчислення a = max(a, a*i*). | КД |
| 9. Сигнал всім задачам про завершення пошуку максимального. | S*i,3* i = |
| 10. Чекати сигналів про завершення пошуку максимального від всіх задач | W*i3* i = |
| 11. Копія МО*i* = MO, MB*i* = MB, a*i* = a. | КД |
| 12. Обчислення MA*H* = MB*i* \* MC*H* + a \* MO*i* \* MK*H* . |  |
| 13. Якщо *tid != Р-1*, сигнал задачі *Р-1* про завершення обчислень. | S*P-1,4* |
| 14. Якщо *tid = Р-1*, чекати сигналів про завершення обчислення від задач *1…Р-1* | W*P-1,4* |
| 15. Якщо *tid = Р-1*, вивести *МА*. |  |

**2.5. Розробка схеми взаємодії процесів**

Під час виконання даного етапу була розроблена структура класу-монітора TaskControl. Він використовується для синхронізації паралельних потоків. На даному етапі визначався набір захищених елементів, що будуть знаходитись у моніторі, а також множина захищених операцій. Набір захищених елементів визначається множиною спільних ресурсів (див. паралельний математичний алгоритм) та множиною змінних, що використовуються в якості умов. Семантика захищених операцій обиралася виходячи з завдання мiнiмiзацiї кількості захищених операцій.

Клас TaskControl містить поля *а, MО, MВ* для зберігання відповідних спільних ресурсів, а також поля inputCount, calcCount, maxCount для організації умов виконання методів монітору. Структура класу TaskControl наведена в додатку А. В класі містяться наступні методи:

– waitInput — для очікування введення даних в потоках T(0) та T(P-1);

– waitCalc — для очікування закінчення обчислень;

– waitMax — для очікування закінчення пошуку максимального;

– inputMО — для введення MО;

– inputMВ — для введення MВ;

– copyMB — для копіювання спільного ресурсу MB;

– copyMO — для копіювання спільного ресурсу MO;

– copyAlpha — для копіювання спільного ресурсу a;

– computeMaxAlpha — для виконання операцій над спільним ресурсом a;

– signalInputDone — для сигналу про завершення вводу даних;

– signalCalcDone — для сигналу про завершення обчислень;

– signalMaxDone — для сигналу про завершення пошуку максимального;

**2.6. Розробка програми ПРГ1**

Програма для системи з спільною пам’яттю написана на мові Java. Основні класи програми:

– Main — основний клас. Містить головний метод, що запускається JVM при старті програми. Головний метод формує ідентифікатори потоків, запускає потоки та вимірює час їх виконання. В основному класі знаходиться константа P, змінюючи яку можна виконати налаштування програми під конкретну комп’ютерну систему;

– TaskWorker — задачний тип, реалізує інтерфейс Runnable;

– TaskControl — клас-монітор, який вирішує задачі синхронізації та взаємного виключення, а також зберігає спільні ресурси;

Повний лістинг програми наведено у додатку Д.

**2.7. Тестування програми ПРГ1**

Для тестування використовувалась паралельна обчислювальна система з наступним апаратним забезпеченням:

- процесор: Intel Core i3-3110М (2,40~ГГц, 4 ядра, 8~Мб кешу третього рівня);

- оперативна пам'ять: DDR3 2 \* 4096~Мб 1600~МГц;

В якості програмного забезпечення використовувались:

- операційна система: Windows 10;

- компілятор та віртуальна машина Java: Sun Java 1.8.0 121, 64-бітна версія.

Таблиця 2.1. Час виконання програми з спільною пам’яттю(значення в мілісекундах)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | T1 | T2 | T3 | T4 |
| 1000 | 34840 | 19698 | 16552 | 14125 |
| 1500 | 150114 | 82097 | 70501 | 54857 |
| 2000 | 399660 | 246161 | 192348 | 169779 |
| 2500 | 894746 | 533993 | 454630 | 395999 |

Підрахунок коефіцієнту прискорення (КП) виконується за формулою

КП = Т1 / ТР

Таблиця 2.2. Значення КП для програми з спільною пам’яттю

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | Р | | | |
| 1 | 2 | 3 | 4 |
| 1000 | 1 | 1,768707 | 2,104882 | 2,466549 |
| 1500 | 1 | 1,828496 | 2,129246 | 2,73646 |
| 2000 | 1 | 1,623572 | 2,077796 | 2,354001 |
| 2500 | 1 | 1,675576 | 1,968075 | 2,259465 |

Підрахунок коефіцієнту ефективності (КЕ) відбувається за формулою

КЕ = КП / Р

Таблиця 2.3. Значення КЕ для програми з спільною пам’яттю

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | Р | | | |
| 1 | 2 | 3 | 4 |
| 1000 | 1 | 0,884354 | 0,701627 | 0,616637 |
| 1500 | 1 | 0,914248 | 0,709749 | 0,684115 |
| 2000 | 1 | 0,811786 | 0,692599 | 0,5885 |
| 2500 | 1 | 0,837788 | 0,656025 | 0,564866 |

Рисунок 2.1. Графік залежності КП від кількості процесорів при N = 1000

Рисунок 2.2. Графік залежності КП від кількості процесорів при N = 1500

Рисунок 2.3. Графік залежності КП від кількості процесорів при N = 2000

Рисунок 2.4. Графік залежності КП від кількості процесорів при N = 2500

Рисунок 2.5. Графік залежності КT від кількості процесорів при N = 1000

Рисунок 2.6. Графік залежності КT від кількості процесорів при N = 1500

Рисунок 2.7. Графік залежності КT від кількості процесорів при N = 2000

Рисунок 2.8. Графік залежності КT від кількості процесорів при N = 2500

**2.8. Висновки до розділу 2**

– У даному розділі досліджені результати тестування паралельної програми для системи з загальною пам’яттю, написаної на мові Java. Тестування проводилось для 2, 3 та 4 потоків. Для розрахунку коефіцієнтів прискорення та ефективності була протестована окремо створена послідовна програма.

– Коефіцієнт прискорення приймає значення у проміжку від 1.62 до 2.73. Найвищі значення цього коефіцієнту приходяться на систему з чотирма потоками, найменше прискорення отримане для системи з двома потоками. Характер графіків коефіцієнтів прискорення однаковий для систем з 2, 3 та 4 ядрами.

– При збільшенні кількості ядер P > 2 коефіцієнт прискорення зменшується. Це в основному пов’язано з тим, що пропускна здатність пам’яті ділиться між всіма ядрами та під час запитів до пам’яті ядра простоюють. Крім того, деякий час витрачається на синхронізацію.

– Результати дослідження мають похибку, зумовлену тим, що процесори виділяються операційною системою не на монопольне використання, тобто, процесорний час може бути в будь-який час передано сторонній програмі. Чим менший час виконання програми, тим більша вірогідність виникнення досить значущої похибки.

**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ**

1. Жуков И. А., Корочкін О. В. Паралельні та розподілені обчислення – К.: «Корнійчук», 2014. – 102-111 с.
2. Дейтел Д. введення в операційні системи. – М.: Мир, 1989. – 360 с.
3. Солов’єв Г.Н., Нікітін В.Д. Операційні системи ЕОМ – М.: Вищ. школа., 1989 – 255с.
4. Шильд.Г. Java 8 издание. Полное руководство. Москва-Санкт-Петербург-Киев. 2012.-280 с.
5. Корочкін О.В. Ада95: Введення в програмування – К.: Світ, 1999 – 260 с.
6. Корочкін О., Жужель М. Авдієв А., Корочкін Д. Захищений модуль як універсальний засіб синхронізації процесів – Вісн. НТУУ«КПІ», Інформатика, управління та обчислювальна техніка, 2001, К.: - №34, С. 137 - 145

**ДОДАТКИ**

**Додаток А**

**Структура класу TaskControl ПРГ1**



**Додаток Б**

**Структурна схема взаємодії задач ПГР2**

**Додаток В**

**Структурна схема ПКС СП**



**Додаток Г**

**Структурна схема ПК ЛП**

**Додаток Д**

**Лістинг програми ПРГ1**

Main.java

package com.kaftanatiy.CourseWork2017Program1;

import com.kaftanatiy.CourseWork2017Program1.primitives.MyMatrix;

import com.kaftanatiy.CourseWork2017Program1.primitives.MyVector;

import com.kaftanatiy.CourseWork2017Program1.threads.TaskControl;

import com.kaftanatiy.CourseWork2017Program1.threads.TaskWorker;

public class Main {

private static int N = 2500;

private static int P = 4;

private static int H = N / P;

public static final int maxValue = 5;

public static long startTime = 0;

public static long endTime = 0;

private static MyMatrix MA = new MyMatrix(N);

private static MyMatrix MC = new MyMatrix(N);

private static MyMatrix MK = new MyMatrix(N);

private static MyVector Z = new MyVector(N);

public static void main(String[] args) {

if(P > N) {

P = N;

H = 1;

}

Thread[] tasks = new Thread[P];

TaskControl monitor = new TaskControl();

for (int i = 0; i < P; i++) {

tasks[i] = new Thread(new TaskWorker(i, monitor));

}

startTime = System.currentTimeMillis();

for (int i = 0; i < P; i++) {

tasks[i].start();

}

}

public static int getN() {

return N;

}

public static int getP() {

return P;

}

public static int getH() {

return H;

}

public static MyMatrix getMA() {

return MA;

}

public static MyMatrix getMC() {

return MC;

}

public static MyMatrix getMK() {

return MK;

}

public static MyVector getZ() {

return Z;

}

public static long getTime() {

return endTime - startTime;

}

}

TaskControl.java

package com.kaftanatiy.CourseWork2017Program1.threads;

import com.kaftanatiy.CourseWork2017Program1.Main;

import com.kaftanatiy.CourseWork2017Program1.primitives.MyMatrix;

/\*\*

\* Monitor class to synchronize threads

\* @author Bogdan Kaftanatiy

\*/

public class TaskControl {

private int inputCount = 0;

private int maxCount = 0;

private int calcCount = 0;

private int alpha = Integer.MIN\_VALUE;

private MyMatrix MO = new MyMatrix(Main.getN());

private MyMatrix MB = new MyMatrix(Main.getN());

public synchronized void waitInput() {

while (inputCount < 2) {

try {

wait();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

public synchronized void waitCalc() {

while (calcCount < Main.getP() - 1) {

try {

wait();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

public synchronized void waitMax() {

while (maxCount < Main.getP()) {

try {

wait();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

public synchronized void inputMO() {

MO.generate(Main.maxValue);

}

public synchronized void inputMB() {

MB.generate(Main.maxValue);

}

public synchronized void computeMaxAlpha(int alpha\_i) {

if(alpha\_i > alpha)

alpha = alpha\_i;

}

public synchronized MyMatrix copyMO() {

return new MyMatrix(MO);

}

public synchronized MyMatrix copyMB() {

return new MyMatrix(MB);

}

public synchronized int copyAlpha() {

return alpha;

}

public synchronized void signalInputDone() {

inputCount++;

if(inputCount == 2)

notifyAll();

}

public synchronized void signalCalcDone() {

calcCount++;

if (calcCount == Main.getP() - 1)

notifyAll();

}

public synchronized void signalMaxDone() {

maxCount++;

if (maxCount == Main.getP())

notifyAll();

}

}

TaskWorker.java

package com.kaftanatiy.CourseWork2017Program1.threads;

import com.kaftanatiy.CourseWork2017Program1.Main;

import com.kaftanatiy.CourseWork2017Program1.primitives.MyMatrix;

import java.util.concurrent.ForkJoinPool;

/\*\*

\* Class to execute the task

\* @author Bogdan Kaftanatiy

\*/

public class TaskWorker implements Runnable {

private int tid;

private TaskControl monitor;

private MyMatrix MOcopy;

private MyMatrix MBcopy;

private int alpha\_copy;

public TaskWorker(int tid, TaskControl monitor) {

this.monitor = monitor;

this.tid = tid;

}

@Override

public void run() {

if (tid == 0) {

monitor.inputMB();

monitor.inputMO();

monitor.signalInputDone();

}

if (tid == Main.getP() - 1) {

Main.getMC().generate(Main.maxValue);

Main.getMK().generate(Main.maxValue);

Main.getZ().generate(Main.maxValue);

monitor.signalInputDone();

}

monitor.waitInput();

int startIndex = tid \* Main.getH();

int endIndex = (tid != Main.getP() - 1) ? (tid + 1) \* Main.getH() : Main.getN();

MyRecursiveMaxTask myRecursiveMaxTask = new MyRecursiveMaxTask(Main.getZ().getPart(startIndex, endIndex));

int alpha\_i = new ForkJoinPool(Main.getH()).invoke(myRecursiveMaxTask);

monitor.computeMaxAlpha(alpha\_i);

monitor.signalMaxDone();

monitor.waitMax();

MOcopy = monitor.copyMO();

MBcopy = monitor.copyMB();

alpha\_copy = monitor.copyAlpha();

for (int i = 0; i < Main.getN(); i++){

for (int j = startIndex; j < endIndex; j++){

int M1 = 0;

int M2 = 0;

for (int k = 0; k < Main.getN(); k++){

M1 += MBcopy.get(i, k) \* Main.getMC().get(k, j);

M2 += MOcopy.get(i, k) \* Main.getMK().get(k, j);

}

int value = M1 + alpha\_copy \* M2;

Main.getMA().set(i, j, value);

}

}

if (tid == Main.getP() - 1)

monitor.waitCalc();

else

monitor.signalCalcDone();

if (tid == Main.getP() - 1) {

Main.endTime = System.currentTimeMillis();

System.out.println("All threads ended calculations. Result time(ms): " + Main.getTime());

if(Main.getMA().getDimension() < 10)

System.out.println("Result MA:\n" + Main.getMA());

else

System.out.println("Result was calculated. Matrix is too large");

}

}

}

MyRecursiveMaxTask.java

package com.kaftanatiy.CourseWork2017Program1.threads;

import com.kaftanatiy.CourseWork2017Program1.primitives.MyVector;

import java.util.ArrayList;

import java.util.List;

import java.util.concurrent.RecursiveTask;

/\*\*

\* Recursive task for fork join

\* @author Bogdan Kaftanatiy

\*/

public class MyRecursiveMaxTask extends RecursiveTask<Integer> {

private MyVector vector;

public MyRecursiveMaxTask(MyVector vector) {

this.vector = vector;

}

@Override

protected Integer compute() {

if(vector.getDimension() > 10) {

List<MyRecursiveMaxTask> subtasks = createSubtasks();

for(MyRecursiveMaxTask subtask : subtasks) {

subtask.fork();

}

int result = Integer.MIN\_VALUE;

for(MyRecursiveMaxTask subtask : subtasks) {

int temp = subtask.join();

if(temp > result)

result = temp;

}

return result;

}

else

return vector.getMaxElement(0, vector.getDimension());

}

public List<MyRecursiveMaxTask> createSubtasks() {

List<MyRecursiveMaxTask> subtasks = new ArrayList<>();

MyRecursiveMaxTask subtask1 = new MyRecursiveMaxTask(this.vector.getPart(0, this.vector.getDimension()/2));

MyRecursiveMaxTask subtask2 = new MyRecursiveMaxTask(this.vector.getPart(this.vector.getDimension()/2 + 1, this.vector.getDimension()));

subtasks.add(subtask1);

subtasks.add(subtask2);

return subtasks;

}

}

MyVector.java

package com.kaftanatiy.CourseWork2017Program1.primitives;

import java.util.Random;

/\*\*

\* Model of Vector with size N

\* @author Bogdan Kaftanatiy

\*/

public class MyVector {

private int dimension;

private int[] array;

public MyVector(int dimension) {

this.dimension = dimension;

array = new int[dimension];

}

public MyVector(int[] array) {

this.array = array;

this.dimension = array.length;

}

public void generate(int maxValue) {

Random generator = new Random();

for (int i = 0; i < dimension; i++) {

array[i] = generator.nextInt(maxValue);

}

}

public int getMaxElement(int startIndex, int endIndex) {

if (startIndex < 0 || endIndex > dimension || startIndex >= endIndex)

throw new IllegalStateException();

int result = Integer.MIN\_VALUE;

for (int i = startIndex ; i < endIndex; i++) {

if(array[i] > result)

result = array[i];

}

return result;

}

public MyVector getPart(int startIndex, int endIndex) {

if(endIndex <= startIndex || endIndex > dimension || startIndex < 0)

throw new IllegalStateException();

int length = endIndex - startIndex;

int[] resultArray = new int[length];

System.arraycopy(this.array, startIndex, resultArray, 0, length);

return new MyVector(resultArray);

}

public int getDimension() {

return dimension;

}

@Override

public String toString() {

String result = "";

for (int i = 0; i < dimension; i++) {

result += array[i] + "\t";

}

return result;

}

}

MyMatrix.java

package com.kaftanatiy.CourseWork2017Program1.primitives;

import java.util.Random;

/\*\*

\* Model of matrix with size N\*N

\* @author Bogdan Kaftanatiy

\* \*/

public class MyMatrix {

private int dimension;

private int[][] array;

public MyMatrix(int dimension) {

this.dimension = dimension;

array = new int[dimension][dimension];

}

public MyMatrix(MyMatrix other) {

this.dimension = other.dimension;

array = new int [this.dimension][this.dimension];

for (int i = 0; i < this.dimension; i++) {

for(int j = 0; j < this.dimension; j++) {

this.array[i][j] = other.array[i][j];

}

}

}

public void generate(int maxValue) {

Random generator = new Random();

for (int i = 0; i < dimension; i++){

for (int j = 0; j < dimension; j++){

array[i][j] = generator.nextInt(maxValue);

}

}

}

public int get(int i, int j) {

return array[i][j];

}

public void set(int i, int j, int value) {

array[i][j] = value;

}

public int getDimension() {

return dimension;

}

public MyMatrix multipleMatrix(MyMatrix other) {

if(this.dimension != other.dimension)

throw new IllegalArgumentException();

MyMatrix result = new MyMatrix(dimension);

for (int i = 0; i < dimension; i++){

for (int j = 0; j < dimension; j++){

result.array[i][j] = 0;

for (int k = 0; k < dimension; k++){

result.array[i][j] += this.array[i][k] \* other.array[k][j];

}

}

}

return result;

}

public MyMatrix multipleNumber(int number) {

MyMatrix result = new MyMatrix(this);

for (int i = 0; i < dimension; i++) {

for (int j = 0; j < dimension; j++){

result.array[i][j] \*= number;

}

}

return result;

}

@Override

public String toString() {

String result = "";

for (int i = 0; i < dimension; i++){

for (int j = 0; j < dimension; j++){

result += array[i][j] +"\t";

}

result += "\n";

}

return result;

}

}

**Додаток Є**

**Лістинг програми ПРГ2**