НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

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

(повна назва кафедри, циклової комісії)

**КУРСОВА РОБОТА**

з дисципліни «Паралельні та розподілені обчислення»

(назва дисципліни)

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

Студента (ки) 3 курсу ІО-24 групи

напряму підготовки 050102 «Комп’ютерна інженерія»

Вакуленко М.В.

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- 2015 рік

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 6.050102 «Комп’ютерна інженерія»

*(шифр і назва)*

***З А В Д А Н Н Я***

НА КУРСОВУ РОБОТУ СТУДЕНТУ

Вакуленку Максиму Володимировичу

*(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 18 травня 2014 р.

3. Вхідні дані до роботи

- аналіз засобів роботи з процесами у мові Java

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

- огляд засобів роботи з процесами в мові Ада

- математична задача A = (B + C \* MX) \* (MO \* MK) + α \* E

- структури ПКС ОП та ПКС ЛП

- мови і бібліотеки програмування: Ада

- засоби організації взаємодії процесів: захищений модуль

5. Перелік графічного матеріалу

- структурна схема ПКС ОП

- структурна схема ПКС ЛП

- схеми алгоритмів процесів і головної програми для ПРГ1

- схеми алгоритмів процесів і головної програми для ПРГ2.

7. Дата видачі завдання ХХХХХХХ

***КАЛЕНДАРНИЙ ПЛАН***

|  |  |  |
| --- | --- | --- |
| №  з/п | Назва етапів виконання КР | Строк виконання етапів КР |
| 1 | Виконання Розділу 1 | 23.02.2015 |
| 2 | Виконання Розділу 2 | 23.03.2015 |
| 3 | Виконання Розділу 3 | 23.04.2015 |
| 4 | Тестування програм ПРГ1 та ПРГ2 | 10.05.2015 |
| 7 | Оформлення КР | 17.05.2015 |
| 8 | Захист КР | 18.05.2015 |

**Студент \_\_\_\_\_\_\_\_\_** Вакуленко М.В.

( підпис ) (прізвище та ініціали)

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

( підпис ) (прізвище та ініціали)

**ПОЯСНЮВАЛЬНА  
ЗАПИСКА**

**ЗМІСТ**

[ВСТУП 3](#_Toc417597925)

[РОЗДІЛ 1. АНАЛІЗ ЗАСОБІВ РОБОТИ З ПРОЦЕСАМИ У МОВІ JAVA 3](#_Toc417597926)

[1.1. Поняття процесу 3](#_Toc417597927)

[1.2. Процеси в мові JAVA 3](#_Toc417597928)

[1.2.1 Об’єкти потоку 3](#_Toc417597929)

[1.2.2 Створення потоку 3](#_Toc417597930)

[1.2.3 Керування потоком 3](#_Toc417597931)

[1.2.4 Модель пам’яті 3](#_Toc417597932)

[1.2.5 Синхронізація 3](#_Toc417597933)

[1.2.6 Блокування та synchonized-блоки 3](#_Toc417597934)

[1.2.7 Volatile поля 3](#_Toc417597935)

[1.2.8 Фінальні поля 3](#_Toc417597936)

[1.2.9 Історія 3](#_Toc417597937)

[1.3. Висновки до розділу 1 3](#_Toc417597938)

[РОЗДІЛ 2. РОЗРОБКА ПРОГРАМИ ПРГ1 ДЛЯ ПКС СП 3](#_Toc417597939)

[2.1 Рішення початкової математичної задачі 3](#_Toc417597940)

[2.2. Тестування ПЗ 3](#_Toc417597941)

[2.3. Висновки до розділу 2 3](#_Toc417597942)

[РОЗДІЛ 3. РОЗРОБКА ПРОГРАМИ ПРГ2 ДЛЯ ПКС ЛП 3](#_Toc417597943)

[3.1 Рішення початкової математичної задачі 3](#_Toc417597944)

[3.2. Тестування ПЗ 3](#_Toc417597945)

[3.3. Висновки до розділу 3 3](#_Toc417597946)

[ДОДАТКИ 3](#_Toc417597947)

СМОТРИТЕ ЗМІСТ В МЕТОДИЧКЕ!!!!!

# ВСТУП

Для розв’язання багатьох задач (прогноз погоди, задачі гідро- і газодинаміки, квантової хімії, астрономії, спектроскопії, біології, ядерної фізики) необхідна висока продуктивність та висока швидкість передачі інформації по каналах зв’язку, великі об’єми оперативної і постійної пам’яті, які не можуть забезпечити типові обчислювальні засоби. Одним з шляхів забезпечення таких вимог є організація паралельних та розподілених обчислень і відповідних технічних засобів їх реалізації.

Причому, ефективність паралельної обробки залежить як від продуктивності комп’ютерів, так і від розмірів і структури пам’яті, пропускної здатності каналів зв’язку, використаних мов програмування, компіляторів, операційних систем, чисельних методів та інших математичних досліджень. Такий широкий обсяг параметрів вимагає проведення досліджень на різних рівнях: на рівні розпаралелення алгоритмів, створення спеціальних мов програмування, компіляторів, багатопроцесорних систем, неоднорідних систем, кластерів і систем, що розподілені на великих територіях.

По каждому разделу……

В даній курсовій роботі розроблюються алгоритми обчислення заданого математичного виразу в паралельній комп’ютерній системі з спільною пам’яттю та локальною пам’яттю.

Порівнюється прискорення виконання обчислень та ефективність двох паралельних алгоритмів.

# РОЗДІЛ 1. АНАЛІЗ ЗАСОБІВ РОБОТИ З ПРОЦЕСАМИ У МОВІ JAVA

## Поняття процесу

Процес — абстрактне поняття, що включає опис певних дій, пов’язаних з виконанням програми в комп’ютерній системі [1]. При запуску прикладної програми в середовищі операційної системи, створюється спеціальний об’єкт – процес, –призначений для підтримки її виконання. Може здатися, що програма й процес – поняття схожі, вони фундаментально відрізняються.

Програма – це статичний набір команд, а процес – контейнер для ресурсів, які використовуються при виконанні екземпляра програми [2].

Системна реалізація та функції процесу в різних операційних системах дещо відмінні. Для кожного процесу у ОС створюється так званий «віртуальний адресний простір», до якого процес має прямий доступ. Цей простір належить процесу, містить тільки його дані і знаходиться в повному його розпорядженні. Операційна система ж відповідає за те, як віртуальний простір процесу проектується на фізичну пам'ять.

На найвищому рівні абстракції процес у ОС Windows містить:

1. закритий віртуальний адресний простір
2. програму, що виконується
3. список відкритих дескрипторів (handles) різних системних ресурсів
4. контекст захисту (security context), який називають маркером доступу (access token)
5. унікальний ідентифікатор процесу
6. мінімум один потік.

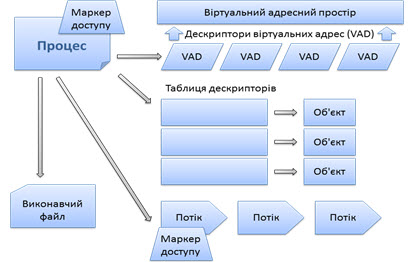


Рис. 1.1 Процес та його ресурси

За допомогою системи віртуальної пам'яті (virtual memory) отримується логічний образ пам'яті, який не збігається з її фізичною структурою. При зверненні процесу за віртуальною адресою, система віртуальної пам'яті транслює цю адресу в фізичну адресу. Вона також перешкоджає безпосередньому доступу процесу до віртуальної пам'яті, зайнятої іншими процесами або ОС. Для виконання коду ОС або доступу до пам'яті ОС потік повинен виконуватися в режимі ядра (kernel mode). Більшість процесів - це процеси користувальницького режиму (user mode).

Потік режиму користувача здійснює отримання доступу до ОС, викликаючи системний сервіс. Під час виклику сервісу відбувається перемикання з режиму користувача в режим ядра. В ОС перевіряються аргументи, передані сервісу, після чого виконується сервіс. Перед поверненням управління програмі, ОС перемикає потік назад в режим користувача.

Крім закритого адресного простору, з кожним процесом пов'язаний набір системних ресурсів. Маркер доступу (Access Token) приєднує до процесу ОС. Це об'єкт системи, який містить інформацію про права зареєстрованого в системі користувача, якого представляє даний процес.

Нижче маркера доступу розташований набір структур даних, створений диспетчером віртуальної пам'яті для відстеження віртуальних адрес, використовуваних процесом.

Під час запуску процесу проходять такі стадії:

1. користувач з допомогою вказує програму, яку потрібно виконати
2. операційна система створює адресний простір для процесу і структури, які описують новий процес
3. заповнюються структури, які описують новий процес
4. з файлу, який містить виконавчий файл, в адресний простір процесу копіюються код і дані
5. встановлюється стан процесу «готовий до виконання»
6. новий процес додається до черги процесів, які очікують на процесор
7. керування повертається оболонці користувача

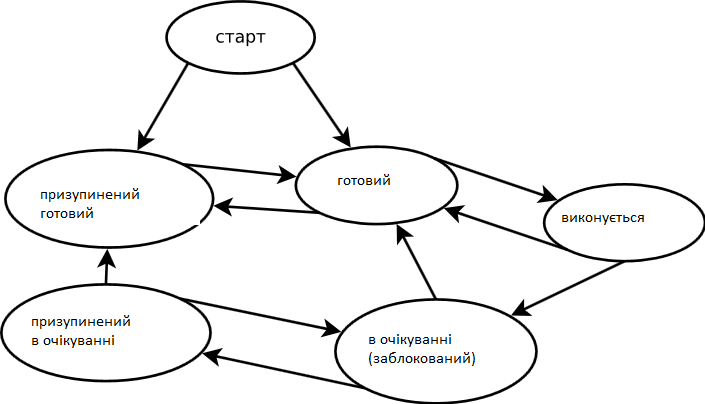


Рис. 1.2 Стани процесу

Кожному процесу мають бути виділені певні системні ресурси, зокрема:

1. процесор
2. пам'ять
3. доступ до пристроїв вводу-виводу

Процес має певний пріоритет, який впливає на кількість процесорного часу, який виділятиметься його [потокам](https://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D1%82%D1%96%D0%BA_(Windows)). У Windows кожен процес має один із таких класів пріоритету:

1. IDLE\_PRIORITY\_CLASS,
2. BELOW\_NORMAL\_PRIORITY\_CLASS,
3. NORMAL\_PRIORITY\_CLASS,
4. ABOVE\_NORMAL\_PRIORITY\_CLASS,
5. HIGH\_PRIORITY\_CLASS,
6. REALTIME\_PRIORITY\_CLASS.

За замовчуванням клас пріоритету процесу набуває значення NORMAL\_PRIORITY\_CLASS. При створенні процесу функцією CreateProcess можна задавати клас пріоритету. Для зміни класу пріоритету існуючого процесу використовується функція API SetPriorityClass, для визначення пріоритету існуючого процесу – функція GetPriorityClass.

Процеси, якими керує система, можуть використовувати найнижчий IDLE\_PRIORITY\_CLASS, що запобігатиме конфлікту таких процесів із процесами вищого пріоритету. Якщо виконується потік процесу з класом HIGH\_PRIORITY\_CLASS, інші потоки в системі не отримуватимуть процесорного часу. Якщо кілька потоків одночасно мають такий пріоритет, вони втрачають свою ефективність.

Клас найвищого пріоритету використовується для потоків, які повинні відповідати на найважливіші події. Якщо прикладна програма виконує завдання, яке потребує цього класу пріоритету, слід використовувати функцію SetPriorityClass для тимчасового підвищення класу пріоритету і відразу після виконання завдання відновлювати звичайний пріоритет.

Інший спосіб для вирішення подібного завдання – створити процес високого пріоритету, всі потоки якого будуть блокованими більшість часу і тимчасово розблоковуватимуться при виникненні певної події. Клас пріоритету REALTIME\_PRIORITY\_CLASS припиняє виконання системних потоків, які керують вводом від миші, клавіатури та фоновим записом на диск, що може призвести до втрати даних або нефункціональності всієї системи. Цей клас може використовуватися для програм, які повинні працювати безпосередньо з апаратним забезпеченням або які виконують швидкі, обмежені в часі завдання.

Процес містить таку інформацію (список не повний):

1. командний рядок запуску виконуваного процесом [застосунку](https://uk.wikipedia.org/wiki/%D0%97%D0%B0%D1%81%D1%82%D0%BE%D1%81%D1%83%D0%BD%D0%BE%D0%BA);
2. інформація про відведений процесу адресний простір, включно зі стеком;
3. посилання на поточний робочий каталог і кореневий каталог процесу (останній служить для обмеження доступу процесу до файлової структури);
4. таблиця відкритих процесом файлів;
5. так зване оточення процесу, тобто перелік заданих для даного процесу змінних з їх поточними значеннями;
6. атрибути, що визначають права і привілеї процесу
7. таблиця обробників сигналів;
8. вказівка на батьківський процес;
9. призначена для користувача маска або маска доступу — вказівка на те, які права треба видалити при створенні нового файлу або каталогу із стандартного набору прав, присвоєних файлу (каталогу).
10. перелік [потоків](https://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D1%82%D1%96%D0%BA_(Windows)) процесу (якщо [операційна система](https://uk.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D1%96%D0%B9%D0%BD%D0%B0_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0) підтримує [багатопотоковість](https://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D1%82%D1%96%D0%BA_(Windows)))

Під час запуску процесу проходять такі стадії:

1. користувач з допомогою вказує програму, яку потрібно виконати
2. [операційна система](https://uk.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D1%96%D0%B9%D0%BD%D0%B0_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0) створює адресний простір для процесу і структури, які описують новий процес
3. заповнюються структури, які описують новий процес
4. з файлу, який містить виконавчий файл, в адресний простір процесу копіюються код і дані
5. встановлюється стан процесу «готовий до виконання»
6. новий процес додається до черги процесів, які очікують на процесор
7. керування повертається оболонці користувача

Найважливішою складовою частиною процесу є потік. Процес як системний об'єкт призначений для "обслуговування" потоків. Сам по собі процес не виконує ніяких дій, для цього призначений потік. Крім закритого адресного простору й одного або кількох потоків, у кожного процесу є ідентифікатор захисту і список відкритих дескрипторів об'єктів (файли, розділи загальної пам'яті), або синхронізуючих об'єктів – м’ютексів, подій і семафорів. Кожен процес має контекст захисту, який зберігається в об'єкті – маркері доступу. Маркер доступу містить ідентифікацію захисту й визначає повноваження даного процесу. Дескриптори віртуальних адрес(virtual address descriptors, VAD) – це структури даних, які використовуються диспетчером пам'яті для обліку віртуальних адрес, задіяних процесом.

Базовою функцією API для створення процесу є CreateProcess. Основні етапи створення процесу функцією CreateProcess такі:

1. Відкривається файл образу (ЕХЕ), що буде виконуватися в процесі.
2. Створюється об'єкт "процес" виконавчої системи.
3. Створюється первинний [потік](https://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D1%82%D1%96%D0%BA_(Windows)) (стек, контекст і об'єкт "потік" виконавчої системи).
4. Підсистема Windows повідомляється про створення нового процесу й [потоку](https://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D1%82%D1%96%D0%BA_(Windows)).
5. Починається виконання первинного [потоку](https://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D1%82%D1%96%D0%BA_(Windows)) (якщо не зазначено прапорець CREATE\_SUSPENDED- створити "завислим").
6. У контексті нового процесу й потоку ініціалізується адресний простір (наприклад, завантажуються необхідні DLL) і починається виконання програми.

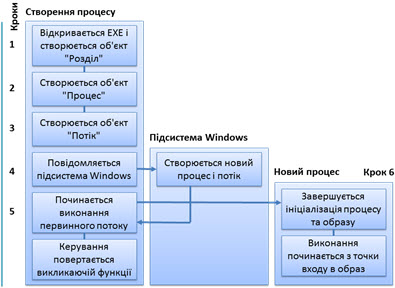


Рис. 1.3 Створення процесу в ОС Windows

Після створення процесу з ним співставляється унікальний глобальний системний ідентифікатор- ProcessID.

## Процеси в мові JAVA

Мова програмування Java і JVM (Java Virtual Machine) розроблені з підтримкою паралельних обчислень, всі обчислення виконуються в контексті потоку. Кілька потоків можуть спільно використовувати об’єкти та ресурси; кожен потік виконує свої інструкції (код), але потенційно може отримати доступ до будь-якого об’єкта в програмі. В обов’язки програміста входить координація (або «синхронізація») потоків під час операцій читання і запису поділюваних об’єктів. Синхронізація потоків потрібна для гарантування того, що одночасно до об’єкта можливе було звернення тільки одного потоку, щоб запобігти звернення потоків до неповністю оновлених об’єктів в той час, як з ними працює інший потік. У мові Java є вбудовані конструкції підтримки синхронізації потоків.

Більшість реалізацій віртуальної машини Java використовують єдиний процес для виконання програми і в мові програмування Java поняття паралельні обчислення найчастіше пов’язують з потоками.

### Об’єкти потоку

Потоки поділяють між собою ресурси процесора, зокрема пам'ять і відкриті файли. Такий підхід веде до ефективної, але потенційно проблематичною, комунікації. Кожна програма має хоча б один потік, що виконується. Потік, з якого починається виконання програми, називається головним або основним. Головний потік здатний створювати додаткові потоки у вигляді об'єктів Runnable або Callable. (Інтерфейс Callable схожий на Runnable тим, що обидва вони розроблені для класів, екземпляри яких будуть виконуватися в окремому потоці. Runnable, однак, не повертає результату і не може викинути проверяемое виняток.)

Кожен потік може бути запланований для виконання на окремому ядрі ЦП, використовувати квантування часу на одноядерному процесорі або використовувати квантування часу на кількох процесорах. В останніх двох випадках система буде періодично перемикатися між потоками, по черзі даючи виконуватися то одному, то іншому потоку. Така схема називається псевдо-паралелізмом. Немає універсального рішення, яке сказало б, як саме потоки Java будуть перетворені в нативні потоки ОС. Це залежить від конкретної реалізації JVM.

У мові Java потік представляється у вигляді об'єкта-нащадка класу Thread. Цей клас інкапсулює стандартні механізми роботи з потоком. Потоками можна управляти або безпосередньо, або за допомогою АБСТРАКТ механізмів, таких як Executor і колекції з пакету java.util.concurrent.

### Створення потоку

Створити новий потік можна двома способами:

1. Реалізацією інтерфейсу Runnable

public class HelloRunnable implements Runnable {

     public void run () {

         System.out.println ("потік 1");

     }

    public static void main (String [] args) {

         (new Thread (new HelloRunnable ())). start ();

     }

}

1. Спадкуванням від класу Thread

public class HelloThread extends Thread {

     public void run () {

         System.out.println ("потік 1");

     }

     public static void main (String [] args) {

         (new HelloThread ()). start ();}}

### Керування потоком

Thread.sleep () - статичний метод класу Thread, який призупиняє виконання потоку, в якому він був викликаний. Під час виконання методу sleep () система перестає виділяти процесорний час для потоку, розподіляючи його між іншими потоками. Метод sleep () може виконуватися або задану кількість часу (мілісекунди або наносекунди), або до тих пір, поки він не буде зупинений перериванням (в цьому випадку він згенерує виняток InterruptedException).

Thread.sleep (1500); // Чекає півтори секунди

Thread.sleep (2000, 100); // Чекає 2 секунди і 100 наносекунд

Переривання - вказівка ​​потоку, що він повинен припинити поточну роботу. Потік може послати переривання викликом методу interrupt () об'єкта Thread, якщо потрібно перервати асоційований з ним потік. Механізм переривання реалізований з використанням внутрішнього методу interrupt status (прапор переривання) класу Thread. Виклик Thread.interrupt () піднімає цей прапор. Будь-який метод, що завершується викиданням InterruptedException, скидає прапор переривання. Перевірити ж, чи встановлений цей прапор, можна двома способами. Перший спосіб - викликати метод bool isInterrupted () 'об'єкта потоку, другий - викликати статичний метод bool Thread.interrupted ()'. Перший метод повертає стан прапора переривання і залишає цей прапор недоторканим. Другий метод повертає стан прапора і скидає його. Зауважте, що Thread.interrupted () - статичний метод класу Thread, і його виклик повертає значення прапора переривання того потоку, з якого він був викликаний.

У мові Java передбачений механізм, що дозволяє одному потоку чекати завершення виконання іншого. Для цього використовується метод join (). Наприклад, щоб головний потік почекав завершення побічної потоку myThready, необхідно виконати інструкцію myThready.join () в головному потоці. Як тільки потік myThready завершиться, метод join () поверне управління, і головний потік зможе продовжити виконання.

Метод join () має перевантажену версію, яка отримує як параметр час очікування. У цьому випадку join () повертає управління або коли завершиться очікуваний потік, або коли закінчиться час очікування. Подібно до методу Thread.sleep () метод join може чекати протягом мілісекунд і наносекунд - аргументи ті ж.

За допомогою завдання часу очікування потоку можна, наприклад, виконувати оновлення анімованої картинки поки головний (або будь-який інший) потік чекає завершення побічного потоку, що виконує ресурсомісткі операції:

Thinker brain = new Thinker (); // Thinker - нащадок класу Thread.

brain.start (); // Почати "обмірковування".

do

{  mThinkIndicator.refresh (); // MThinkIndicator - анімована картинка.

    try {

        brain.join (250); // Почекати закінчення думки чверть секунди.

    } Catch (InterruptedException e) {}}

while (brain.isAlive ()); // Поки brain думає ...

// Brain закінчив думати

Статичний метод Thread.yield () змушує процесор переключитися на обробку інших потоків системи. Метод може бути корисним, наприклад, під час очікування настання події, в разі необхідності перевірки її настання якомога частіше. У цьому випадку можна помістити перевірку події і метод Thread.yield () в цикл:

// Очікування надходження повідомлення

while (! msgQueue.hasMessages ()) // Поки в черзі немає повідомлень

{

     Thread.yield (); // Передати управління іншим потокам

}

### Модель пам’яті

Модель пам'яті Java описує взаємодію потоків через пам'ять в мові програмування Java [3]. Найчастіше, на сучасних комп'ютерах код заради швидкості виконується не в тому порядку, в якому написаний. Перестановка виконується компілятором, процесором і підсистемою пам'яті. Мова програмування Java не гарантує атомарність операцій і послідовну консистентність при читанні або запису полів поділюваних об'єктів. Дане рішення "розв'язує руки" компілятору і дозволяє проводити оптимізацію (таку як розподіл регістрів, видалення загальних підвиразів і усунення зайвих операцій читання), засновані на перестановці операцій доступу до пам'яті [4].

### Синхронізація

Комунікація потоків здійснюється за допомогою розділення доступу до полів і об'єктів, на які посилаються поля. Дана форма комунікації є гранично ефективною, але робить можливим виникнення помилок двох різновидів: втручання в потік (thread interference) і помилки консистентності пам'яті (memory consistency errors). Для запобігання їх виникненню існує механізм синхронізації.

Перевпорядкуванння (зміна порядку проходження, reordering) проявляється в некоректно синхронізованих багатопоточних програмах, де один потік може спостерігати ефекти, спричинені іншими потоками, і такі програми в змозі виявити, що оновлені значення змінних стають видимими для інших потоків в порядку, відмінному від зазначеного у вихідному коді.

Для синхронізації потоків в Java використовуються монітори, які є високорівневим механізмом, що дозволяє одноразово тільки одному потоку виконувати блок коду, захищений монітором. Синхронізація має кілька аспектів. Взаємне виключення (mutual exclusion) - тільки один потік може володіти монітором, таким чином синхронізація на моніторі означає, що як тільки один потік входить в synchronized-блок, захищений монітором, ніякий інший потік не може увійти в блок, захищений цим монітором, поки перший потік не вийде з synchronized-блоку.

Але синхронізація - це більше ніж просто взаємне виключення. Синхронізація гарантує, що дані, записані в пам'ять до або всередині синхронізованого блоку, стають видимими для інших потоків, які синхронізуються на тому ж моніторі. Після того як ми виходимо з синхронізованого блоку, ми звільняємо (release) монітор, що має ефект скидання (flush) кеша в оперативну пам'ять, так що записи, зроблені нашим потоком, можуть бути видимими для інших потоків. Перш ніж з’явиться змога увійти в синхронізований блок, захоплюється (acquire) монітор, що має ефект оголошення недійсними даних локального процесорного кешу (invalidating the local processor cache), отже змінні будуть завантажені з основної пам'яті. Тоді буде можливим побачити всі записи, зроблені видимими попереднім звільненням (release) монітора. (JSR 133)

### Блокування та synchonized-блоки

Ефект взаємного виключення і синхронізації потоків досягається входженням в synchronized-блок або метод, неявно отримуючий блокування, або отримуючий блокування явним чином (таким як ReentrantLock з пакету java.util.concurrent.locks). Обидва підходи надають однаковий вплив на поведінку пам'яті. Якщо всі спроби доступу до деякого полю захищені однаковим блокуванням, то операції читання-запису цього поля є атомарними.

### Volatile поля

Застосування до поля ключового слово volatile гарантує:

1. (У всіх версіях Java) Доступи до volatile-змінної впорядковані глобально. Це передбачає, що кожен потік, який звертається до volatile-поля, прочитає його значення перед тим, як продовжити замість того, щоб (можливо) використовувати кешовані значення. (Доступи до volatile змінних не можуть перевпорядковуватись один з одним, але вони можуть бути перевпорядковані з не-volatile змінними. Це зводить нанівець корисність volatile полів як засобу передачі сигналу від одного потоку до іншого.)
2. (В Java 5 і пізніших) Запис у volatile поле однаковий ефект для пам'яті, як і звільнення монітора (monitor release), а читання з volatile поля має однаковий ефект для пам'яті, як і захоплення монітора (monitor acquire). Доступ до volatile встановлює відношення яке є гарантією того, що запис в пам'ять одним конкретним виразом буде видимий іншому конкретному виразу.

Volatile-поля є атомарними. Читання з volatile-поля має той самий ефект, як і отримання блокування: дані в робочій пам'яті оголошуються недійсними, значення volatile-поля заново читається з пам'яті. Запис у volatile-поле має той самий ефект для пам'яті, як і звільнення блокування: volatile-поле негайно записується в пам'ять.

### Фінальні поля

Поле, яке оголошено final, називається фінальним і не може бути змінено після ініціалізації. Фінальні поля об'єкта ініціюються в його конструкторі. Якщо конструктор відповідає певним простим правилам, то коректне значення фінального поля буде видимим для решти потоків без синхронізації. Просте правило: посилання this не повинна покинути конструктор до його завершення.

### Історія ЧЕГО?!

Починаючи з JDK 1.2, в Java включений стандартний набір класів-колекцій Java Collections Framework [5].

Doug Lea, який також брав участь у реалізації Java Collections Framework, розробив пакет concurrency, що включає в себе кілька примітивів синхронізації і велика кількість класів, що відносяться до колекцій [6]. Робота над ним була продовжена як частина JSR 166 під головуванням Doug Lea.

Реліз JDK 5.0 включив багато доповнень і пояснень до моделі паралелізму в Java. Вперше API для роботи з паралелізмом розроблені JSR 166 були включені в JDK. JSR 133 надала підтримку для добре визначених атомарних операцій в багатопотоковому / багатопроцесорному оточенні.

І Java SE 6, і Java SE 7 привносять зміни та доповнення до JSR 166 API.

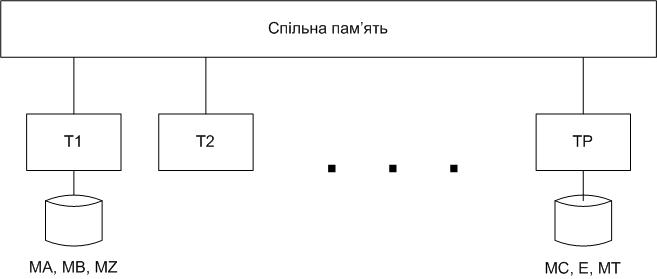
## Висновки до розділу 1

1. Виконаний аналіз засобів організації роботи з процесами у мові Java, який показав, що процеси у даній мові програмування реалізовані у вигляді потоку (thread).
2. Було розглянуто два способи створення потоку у мові Java: за допомогою наслідування класу Thread та з використанням інтерфейсу Runnable И ЧТО? ГДЕ ВЫВОД????
3. Було розглянуто основні методи роботи з потоками у мові Java, зокрема join(), run(), sleep(), start() та інші, що надають широкий спектр інструментів для керування потоками у мові Java.
4. ПРОЦЕССЫ?
5. ВЗАИМОДЕЙСТВИЕ ПРОЦЕССОВ??

# ПЕРЕДЕЛАТЬ!!!!РОЗДІЛ 2. РОЗРОБКА ПРОГРАМИ ПРГ1 ДЛЯ ПКС СП

У цьому розділі буде розглянуто та проаналізовано поставлену математичну задачу, розроблено алгоритм рішення та схему взаємодії потоків для системі з спільною пам’яттю, а на основі вищезазначених досліджень буде створена паралельна програма.

ЗАВДАННЯ????



* 1. **Рішення початкової математичної задачі ?????**

**21. Розробка паралельного мат алгоритма…. И т.д. пл. етапам Смотрите Методичку…….**

Матаматична задача, для якої необхідно створити паралельну програму: MA = MB (MC \* MZ) – max(E) \* MT.

Перш за все розроблюється математичний алгоритм, аналізується наявність спільних ресурсів.

1. zi = max(EH), де і = 1..P
2. m = max (m, zi)
3. MAH = MB(MZ \* MCH) – m \* MTH

Спільні ресурси: MB, MZ, m



Наступним етапом є створення алгоритму для кожного паралельного процесу із зазначенням точок синхронізації та критичних ділянок.

Т1 КД,ТС

1. Читання MB, MZ
2. Сигнал про завершення читання для Т2 - ТР S(2..Р),1
3. Чекати сигналу про завершення читання від Тр WP,1
4. Створити копії MB1 = MB, MZ1 = MZ КД
5. z1 = max(EH)
6. Порівняння m та z1 КД
7. Сигнал задачам Т2 – ТР про завершення порівняння S(2..P),2
8. Чекати сигналу T2 – ТР про завершення порівняння W(2..P-1),1,WP,2
9. Копіювання m1 = m КД
10. Розрахунок MAH = MB1 \* (MZ1 \* MCH) – m1 \* MTH
11. Чекати сигналу Т2 – ТР про завершення розрахунку W(2..P),2, WP,3
12. Вивести МА

Тi (1 < i < P) КУ,ТС

1. Чекати сигналу про завершення читання Т1, ТР WР,1,W1,1
2. Створити копії MBi = MB, MZi = MZ КД
3. Zi = max(EH)
4. Порівняння m та zi КД
5. Сигнал Т1 – Тi-1, Ti+1 - TP про завершення порівняння S(1..i-1),1, S(i+1..P),1
6. Чекати сигналу Т1 – Тi-1, Ti+1 - TP про завершення порівняння W(2..i-1),1, WP,2, W(i+1..P-1),1, W1,2
7. Копіювання mі = m КД
8. Розрахунок MAH = MBі \* (MZі \* MCH) – mі \* MTH
9. Сигнал Т1 про завершення розрахунку S1,2

ТP КУ,ТС

1. Читання MC, MT, E
2. Сигнал про завершення читання для Т1 - ТР-1 S(1..Р-1),1
3. Чекати сигналу про завершення читання від Т1 W1,1
4. Створити копії MBP = MB, MZP = MZ КД
5. ZP = max(EH)
6. Порівняння m та zP КД
7. Сигнал задачам Т1 – ТР-1 про завершення порівняння S(1..P-1),2
8. Чекати сигналу T2 – ТР про завершення порівняння W(2..P-1),1,W1,2
9. Копіювання mP = m КД
10. Розрахунок MAH = MBP \* (MZP \* MCH) – mP \* MTH
11. Сигнал Т1 про завершення розрахунку S1,3

На підставі даного алгоритму будується схема взаємодії процесів, представлена ​​в додатку В «Алгоритми процесів в програмі для ПКС з СП». Як засіб синхронізації обраний механізм монітора, зображенного на рисунку 2.1, в програмі створений клас з даною назвою, який реалізує виконання задач взаємного виключення і синхронізації, представлених вище в алгоритмі.

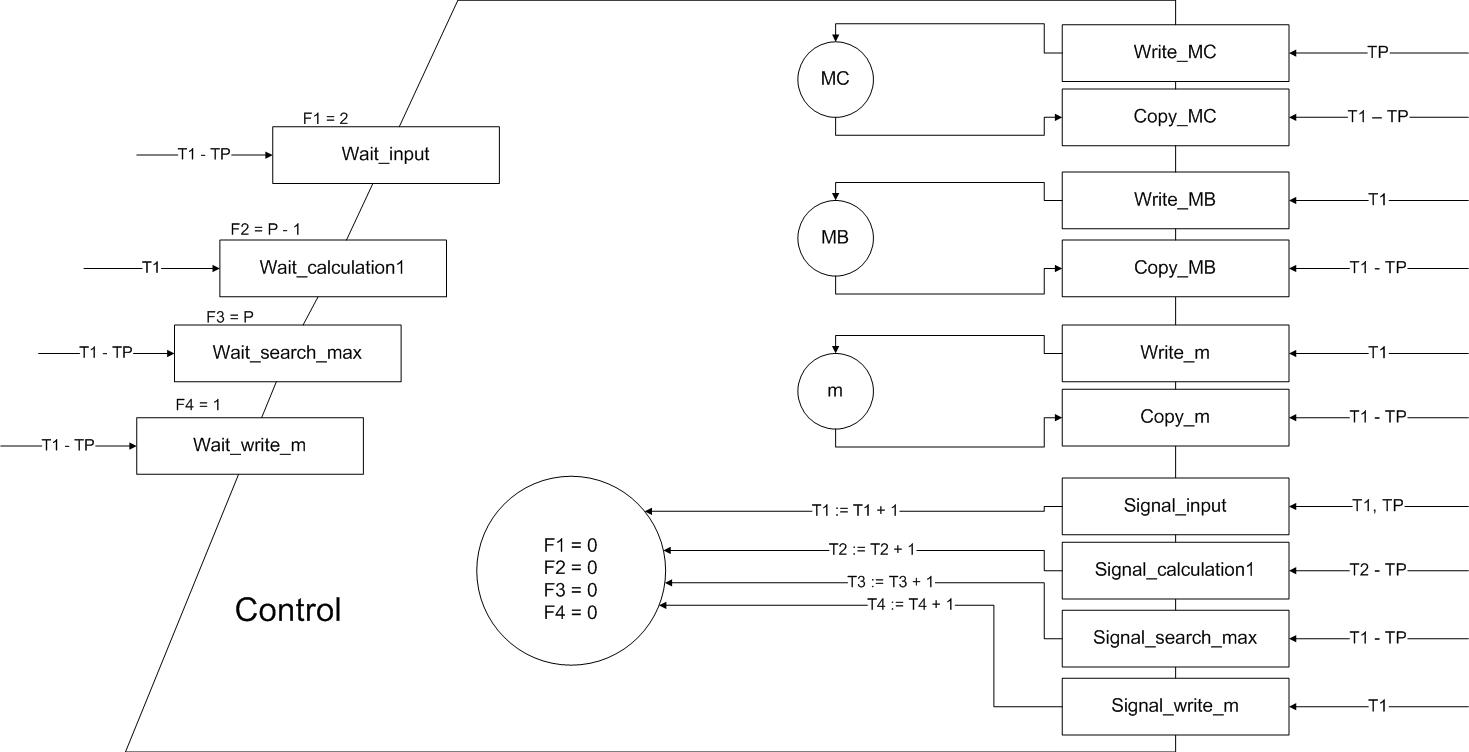


Рис. 2.1. Монітор

Використовуються такі методи:

- Write\_MC - запис в монітор СР ()

- Write\_MB - запис в монітор СР ()

- Write\_m - запис в монітор знайденого максимуму, що є спільним ресурсом

- Copy\_MC - створення локальної копії МС

- Copy\_MB - створення локальної копії МB

- Copy\_m - створення локальної копії m

- Wait\_input - очікування закінчення введення Т1 і ТР (синхронізація по вводу). Прапор F1 спочатку встановлений в 0. При закінченні введення якихось із цих потоків, виконується операція F1++. Якщо F1 = 2, всі потоки повідомляються про це, і отримують можливість доступу до даних і до подальшого виконання програми.

- Wait\_calculation1 - очікування закінчення рахунку (синхронізація з виведення). Прапор F2 спочатку встановлений в 0. При закінченні рахунку якимось із потоків, виконується операція F2++. Якщо F1 = P-1, перший потік отримує повідомлення про це і допускається до подальшого виконання програми.

- Wait\_search\_max - очікування закінчення розрахунку максимального елементу вектора. Прапор F3 спочатку встановлений в 0. При закінченні рахунки яким або з потоків, виконується операція F3++. Якщо F1 = Р, всі потоки отримують повідомлення про цю подію і допускається до подальшого виконання програми.

- Wait\_write\_m – очікування закінчення запису першим потоком максимального елементу вектора до монітору.

- Signal\_input - повідомлення від потоку про завершення їм введення даних. F1++.

- Signal\_calculation1 - повідомлення від потоку про завершення їм розрахунку. F2++.

- Signal\_search\_max - повідомлення від потоку про завершення їм розрахунку максимального елементу. F3++.

- Signal\_write\_m - повідомлення від першого потоку про завершення їм запису максимального елементу до монітора. F4++.

**2.2. Тестування ПЗ**

Для дослідження необхідно визначити час виконання розробленої програми в реальній ПКС. Тому при тестуванні послідовно використовуються 1, 2, 3, 4 процесорів, для яких визначаються час виконання Програми 1. При цьому встановлюється декілька значень розмірності векторів (матриць) N = 900, 1800, 2400. В таблиці 2.1 відображаються значення часу для різних N та Р.

ОПИСАНИЕ КОМПЬЮТЕРА ДЛЯ ТЕСТИРОВАНИЯ….. СКОЛЬКО ЯДЕР?

Таблиця 2.1 - Час виконання програми в ПКС з СП (в секундах)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N//P | Т1 | Т2 | Т3 | Т4 |
| 900 | 41,3 | 20,97 | 14,13 | 10,60 |
| 1800 | 329,3 | 169,1 | 113 | 87,31 |
| 2400 | 778,68 | 391,2 | 265,57 | 212,26 |

Базуючись на отриманих результатах, вираховуємо значення коефіцієнтів прискорення . У таблиці 2.2 приведено результати цих розрахунків.

Таблиця 2.2. Значения Кп для ПКС з СП

|  |  |  |  |
| --- | --- | --- | --- |
| N//P | Т2 | Т3 | Т4 |
| 900 | 1,96 | 2,92 | 3,89 |
| 1800 | 1,94 | 2,91 | 3,77 |
| 2400 | 1,98 | 2,84 | 3,66 |

На основі отриманих коефіцієнтів прискориння будується графік змінення КП в залежності від Р та N. Графік наведено на рис. 2.2. На осі абцис разташована розмірність векторів та матриць, ординатами зазначено коефіцієнт прискорення. Для позначення результатів різних багатопроцесорних систем використовуються різні типи ліній та маркеров.

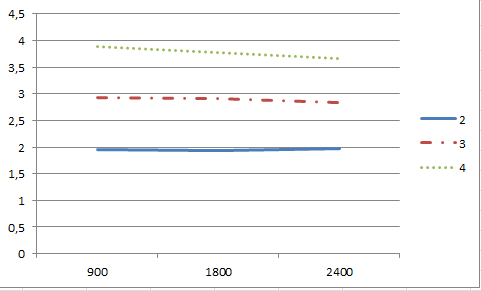


Рисунок 2.2 - Графік залежності коефіцієнта прискорення від розмірності матриць і векторів для ПКС з СП.

Розрахуємо коефіцієнти ефективності та заповнимо їми таблицю 2.3. Для розрахунку використовуються формули  .

Таблиця 2.3 - Значення Ке для програми ПКС з СП

|  |  |  |  |
| --- | --- | --- | --- |
| N//P | Т2 | Т3 | Т4 |
| 900 | 0,98 | 0,97 | 0,97 |
| 1800 | 0,97 | 0,97 | 0,95 |
| 2400 | 0,96 | 0,95 | 0,91 |

На основі отриманих коефіцієнтів ефективності будується графік змінення Ке в залежності від Р та N. Графік наведено на рис. 2.3. На осі абцис разташована розмірність векторів та матриць, ординатами зазначено коефіцієнт ефективності. Для позначення результатів різних багатопроцесорних систем використовуються різні типи ліній та маркеров.

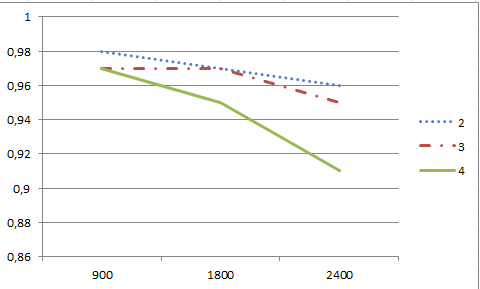


Рисунок 2.3 - Графік залежності коефіцієнта прискорення від розмірності матриць і векторів для ПКС з СП.

**2.3. Висновки до розділу 2**

1. У даному розділі досліджені результати тестування паралельної програми для системи з загальною пам’яттю, написаної на мові Java. Тестування провдилось для 2, 3 та 4 потоків. ЭТО НЕ ВЫВОД!!!!

2. Коефіцієнт прискорення приймає значення у проміжку від 1,94 до 3,89. Найвищі значення цього коефіцієнту приходяться на систему з чотирьма потоками, найменше прискорення отримане для системи з двома потоками. Характер графіків коефіцієнтів прискорення однаковий для систем з 2, 3 та 4 ядрами.

3. Коефіцієнт ефективності приймає значення у проміжку від 0,91 до 0,98. Найвище значення цього коефіцієнту отримане на тесті з параметрами Р = 2, N = 900; найнижче – P = 4, N = 2400. Тобто, при збільшенні кількості ядер та розмірності матриць, коефіцієнт ефективності спадає.

4. 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 вирогідність виникнення досить значущої похибки. ???????

# РОЗДІЛ 3. РОЗРОБКА ПРОГРАМИ ПРГ2 ДЛЯ ПКС ЛП

У цьому розділі буде розглянуто та проаналізовано поставлену математичну задачу, розроблено алгоритм рішення та схему взаємодії задач, а на основі вищезазначених досліджень буде створена паралельна програма для системи з локальною пам’яттю.

ЗАДАНИЕ!!!!

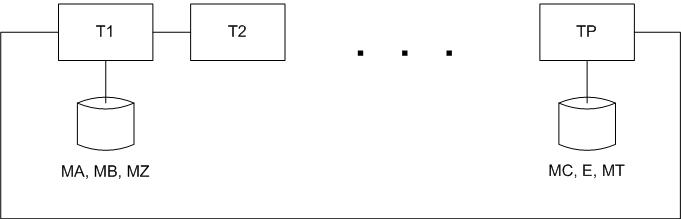


Рис.3.1 ЧХХХХХХХХХХХХХХХХХ

**3.1 Рішення початкової математичної задачі ???????**

Матаматична задача, для якої необхідно створити паралельну програму: MA = MB (MC \* MZ) – max(E) \* MT.

Перш за все розроблюється математичний алгоритм.

1. zi = max(EH), де і = 1..P
2. m = max (m, zi)
3. MAH = MB(MZ \* MCH) – m \* MTH

Наступним етапом є створення алгоритму для кожної задачі. Використовується концепція рандеву. Блок-схеми алгоритмів задач розміщені у додатку Г.

Т1 (i=1)

1. Ввід MB, MZ
2. Отримати E(1..(P/2)\*H), MC(1..N, 1..(P/2)\*H), MT(1..N, 1..(P/2)\*H) від TP
3. Відправити MZ, MB в TP
4. if P>2 then Відправити MB, MZ, E(H+1..(P/2)\*H), MC(H+1..(P/2)\*H), MT(H+1..(P/2)\*H) в T2
5. Рахувати z = max(EH)
6. Відправити z в T(i+1)
7. if (P=2) then Отримати m від TP
8. else if (P>2) Отримати m від T2
9. Рахувати MAh
10. If (P>2) Отримати MA(H+1..p/2\*H) від T2
11. Отримати MA(p/2\*H+1..N) від TP
12. Вивести МА

ТT1(i) (i in 2..P/2)

1. Отримати MB, MZ, E(1..(P/2-i)\*H), MC(1..(P/2-i)\*H), MT(1..(P/2-i)\*H) від T(i-1)
2. Якщо і < (P/2-1), відправити MB, MZ, E(Н+1..(P/2-i)\*H), MC(Н+1..(P/2-i)\*H), MT(Н+1..(P/2-i)\*H) в T(i+1)
3. Рахувати z = max(EH)
4. Передати z T(i+1)
5. Отримати m від T(i+1)
6. Передати m в T(i-1)
7. Рахувати МАН
8. Якщо і < (P/2-1), отримати МA(i\*H+1..P/2\*H) від T(i+1)
9. Передати МA((i-1)\*H+1..P/2\*H) в T(i-1)

ТT2(i) (i in P/2+1..P-1)

1. Отримати MB, MZ, E(p/2\*Н+1..N-H), MC(1..N, p/2\*Н+1..N-H), MT(1..N, p/2\*Н+1..N-H) від T(i+1)
2. Якщо і > P/2+2, передати MB, MZ, E(p/2\*Н+1..Н(і-1)), MC(1..N, p/2\*Н+1..Н(і-1)), MT(1..N, p/2\*Н+1..Н(і-1))в T(i-1)
3. Рахувати z = max(EH)
4. Якщо і > P/2+2, Відправити z в T(i-1).
5. Якщо і = P/2+1, отримати z від T(i+1) та T(i-1) та визначити з них максимальний. Передати m до T(i+1), T(i-1).
6. Якщо і > P/2+2, отримати m від T(i-1), та передати його до T(i+1).
7. Рахувати МAh
8. Якщо і > P/2+2, отримати МAH((P/2)\*H+1..H\*(i-1)) від T(i-1)
9. Відправити МAH((P/2)\*H+1..H\*i)в T(i+1)

ТР (i=P)

1. Ввід MC, MT, E
2. Відправити E(1..(P/2)H), MC(1..N,1..(P/2)H), MT(1..N,1..(P/2)H) в T1
3. Отримати MB, MZ від T1
4. if P>2 then Відправити MB, MZ, E(p/2+1..N-H), MC(1..N, p/2+1..N-H), MT(1..N, p/2+1..N-H)в T(p-1)
5. Рахувати z = max(EH)
6. if (P=2) then Передати z T1 and отримати m від Т1
7. if (P>2) then Передати z T(P-1) and отримати m від Т(P-1)
8. Рахувати MAh
9. If (P>2) Отримати MAH(p/2\*H+1..N-H) від T(p-1)
10. Передати MAH(p/2\*H+1..N) T1

Для реалізації поставленого завдання використовується механізм Ada рандеву. Основна ідея механізму рандеву: у специфікації задачі публікуються різні входи (entry) у задачу, в яких вона готова чекати звернення до неї від інших задач. Далі, в тілі задачі вказуються інструкції прийняття звернень до відповідних входів, зазначених у специфікації цієї задачі. Механізм рандеву є одночасно як механізмом синхронізації, так і механізмом межзадачного обміну даними.

У найпростішому випадку, коли розглядається взаємодія тільки двох задач, задача-клієнт, що бажає звернутися до іншої задачі (задачі-серверу), ініціює звернення до входу задачі-сервера. Після цього задача-сервер відгукується на звернення задачі-клієнта, беручи звернення до цього входу. Таким чином, взаємодія двох задач здійснюється в ситуації, коли задача-клієнт звернулася до входу, а задача-сервер готова прийняти це звернення.

Базуючись на алгоритмі задач та концепції рандеву, створюється схема взаємодії задач, зображена на рис.3.1.

Лістинг програми розміщується в Додатку Б «Лістинг програми ПКС ЛП».

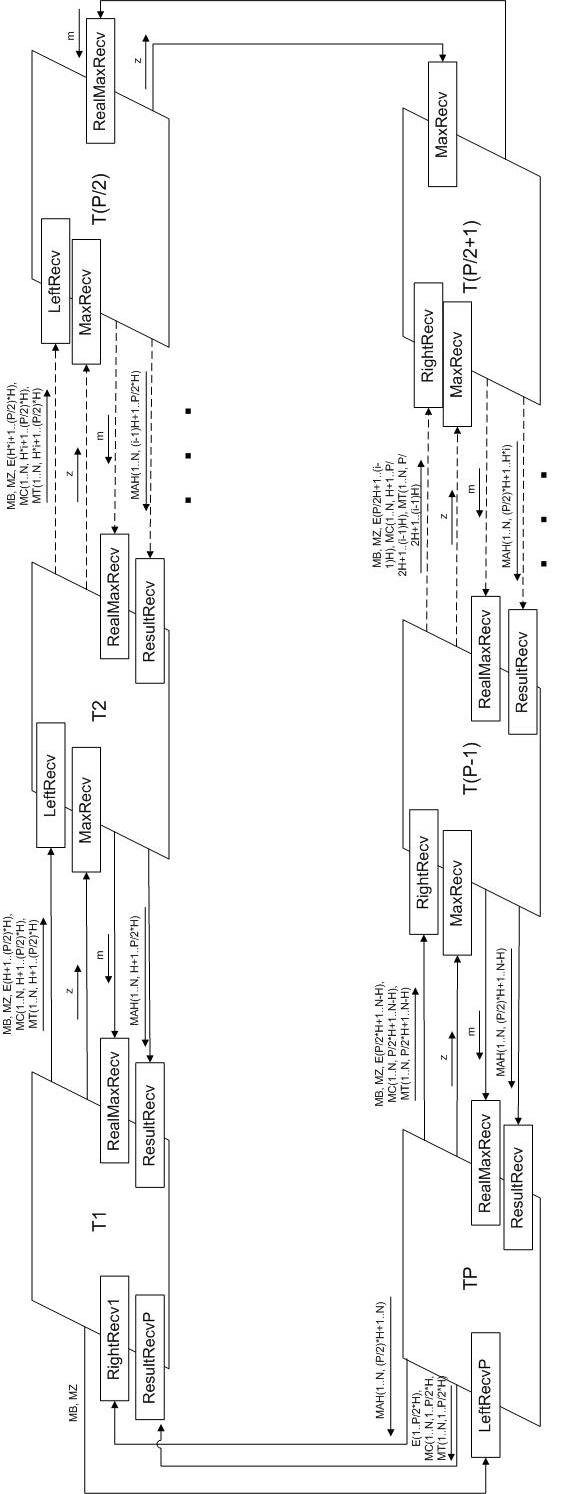


Рис. 3.2. Схема взаємодії задач ПОДПИСЬ ИДЕТ ПОД РИСУНКОМ, А НЕ ЗБОКУ!!!!

**3.2. Тестування ПЗ**

Для дослідження необхідно визначити час виконання розробленої програми в реальній ПКС. Тому при тестуванні послідовно використовуються 1, 2, 4, 6 процесорів, для яких визначаються час виконання Програми 1. При цьому встановлюється декілька значень розмірності векторів (матриць) N = 1000, 2000, 3000. В таблиці 3.1 відображаються значення часу для різних N та Р.

Таблиця 3.1 - Час виконання програми в ПКС з ЛП (в секундах)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N//P | Т1 | Т2 | Т4 | Т6 |
| 900 | 59,4 | 14,625 | 7,569 | 5,616 |
| 1800 | 352,62 | 134,67 | 85,64 | 70,59 |
| 2400 | 897,12 | 432,04 | 228,77 | 131,36 |

Базуючись на отриманих результатах, вираховуємо значення коефіцієнтів прискорення . У таблиці 3.2 приведено результати цих розрахунків.

Таблиця 3.2. Значения Кп для ПКС з ЛП ЧТО ЭТО!!!!!!!

|  |  |  |  |
| --- | --- | --- | --- |
| N//P | Т2 | Т4 | Т6 |
| 900 | 4,06 | 7,84 | 10,57 ??? |
| 1800 | 2,62 | 4,12 | 4,99 |
| 2400 | 2,08 | 3,92 | 6,83 |

На основі отриманих коефіцієнтів прискориння будується графік змінення КП в залежності від Р та N. Графік наведено на рис. 3.2. На осі абцис разташована розмірність векторів та матриць, ординатами зазначено коефіцієнт прискорення. Для позначення результатів різних багатопроцесорних систем використовуються різні типи ліній та маркеров.

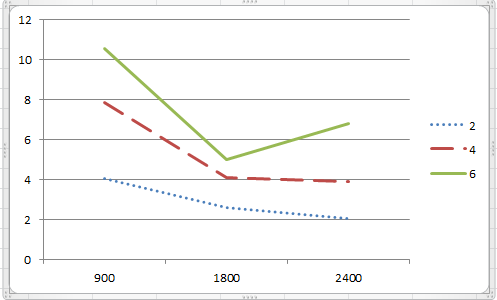


Рис. 3.2 . Графік залежності коефіцієнта прискорення від розмірності матриць і векторів для ПКС з ЛП.

Розрахуємо коефіцієнти ефективності та заповнимо їми таблицю 3.3. Для розрахунку використовуються формули  .

Таблиця 3.3 - Значення Ке для програми ПКС з ЛП

|  |  |  |  |
| --- | --- | --- | --- |
| N//P | Т2 | Т4 | Т6 |
| 1000 | 2,03 | 1,96 | 1,76 |
| 2000 | 1,31 | 1,03 | 0,83 |
| 3000 | 1,04 | 0,98 | 1,14 |

На основі отриманих коефіцієнтів ефективності будується графік змінення Ке в залежності від Р та N. Графік наведено на рис. 3.3. На осі абцис разташована розмірність векторів та матриць, ординатами зазначено коефіцієнт ефективності. Для позначення результатів різних багатопроцесорних систем використовуються різні типи ліній та маркеров.

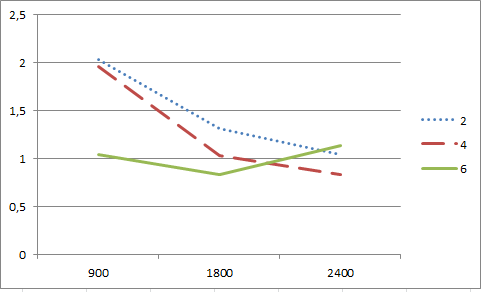


Рисунок 3.3 - Графік залежності коефіцієнта прискорення від розмірності матриць і векторів для ПКС з ЛП.

**3.3. Висновки до розділу 3**

1. У даному розділі досліджені результати тестування паралельної програми для системи з локальною пам’яттю, написаної на мові Ada. Тестування провдилось для 2, 4 та 6 потоків. ЯДЕР СКОЛЬКО???

2. Коефіцієнт прискорення приймає значення у проміжку від 2,08 до 10,57. Найвищі значення цього коефіцієнту приходяться на систему з шістьма потоками, найменше прискорення отримане для системи з двома потоками. Характер графіків коефіцієнтів прискорення однаковий для систем з 2, 4 та 6 ядрами.

3. Коефіцієнт ефективності приймає значення у проміжку від 0,98 до 2,03. Найвище значення цього коефіцієнту отримане на тесті з параметрами Р = 2, N = 900. Тобто, при збільшенні кількості ядер та розмірності матриць, коефіцієнт ефективності спадає.

4. 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 вирогідність виникнення досить значущої похибки.

ОБЩИЕ ВЫВОДЫ ГДЕ??

ЛИТЕРАТУРА?

**ДОДАТКИ**

**Додаток А**

**Додаток А**

**Лістинг програми ПРГ1**

**ШАПКА?**

GNAT GPL 2013 (20130314)

Copyright 1992-2013, Free Software Foundation, Inc.

Compiling: main.adb (source file time stamp: 2015-03-23 13:54:34)

1. with Ada.Text\_IO, Ada.Integer\_Text\_IO, Ada.Calendar;

2. use Ada.Text\_IO, Ada.Integer\_Text\_IO, Ada.Calendar;

3.

4. procedure Main is

5. N: Integer := 4;

6. P: Integer := 4;

7. H: Integer := N / P;

8. StartTime, FinishTime: Time;

9. DiffTime: Duration;

10.

11. type Vector is array(1..N) of Integer;

12. type Matrix is array(1..N) of Vector;

13.

14. MX, MO, MK: Matrix;

15. A, B, C, E, T: Vector;

16. Alf: Integer;

17.

18. procedure MatrixInput(M: out Matrix) is

19. begin

20. for i in 1..N loop

21. for j in 1..N loop

22. M(i)(j) := 1;

23. end loop;

24. end loop;

25. end MatrixInput;

26.

27. procedure VectorInput(V: out Vector) is

28. begin

29. for i in 1..N loop

30. V(i) := 1;

31. end loop;

32. end VectorInput;

33.

34. procedure VectorOutput(V: in Vector) is

35. begin

36. for i in 1..N loop

37. put(V(i));

38. put(" ");

39. end loop;

40. Put\_Line("");

41. end VectorOutput;

42.

43. protected ResourceMonitor is

44. procedure WriteMO(M: in Matrix);

45. procedure WriteB(V: in Vector);

46. procedure WriteC(V: in Vector);

47. procedure WriteT(V: in Vector);

48. procedure WriteAlf(A: in Integer);

49.

50. function CopyAlf return Integer;

51. function CopyB return Vector;

52. function CopyC return Vector;

53. function CopyT return Vector;

54. function CopyMO return Matrix;

55. private

56. Alf: Integer;

57. B: Vector;

58. c: Vector;

59. T: Vector;

60. MO: Matrix;

61. end ResourceMonitor;

62.

63. protected SynchronizeMonitor is

64. procedure SignalInput;

65. procedure SignalCalculate1;

66. procedure SignalCalculate2;

67. entry WaitInput;

68. entry WaitCalculate1;

69. entry WaitCalculate2;

70. private

71. flagInput: Integer := 0;

72. flagCalculate1: Integer := 0;

73. flagCalculate2: Integer := 0;

74. end SynchronizeMonitor;

75.

76. protected body ResourceMonitor is

77. procedure WriteMO(M: in Matrix) is

78. begin

79. MO := M;

80. end WriteMO;

81.

82. procedure WriteB(V: in Vector) is

83. begin

84. B := V;

85. end WriteB;

86.

87. procedure WriteC(V: in Vector) is

88. begin

89. C := V;

90. end WriteC;

91.

92. procedure WriteT(V: in Vector) is

93. begin

94. T := V;

95. end WriteT;

96.

97. procedure WriteAlf(a: in Integer) is

98. begin

99. Alf := a;

100. end WriteAlf;

101.

102. function CopyB return Vector is

103. begin

104. return B;

105. end CopyB;

106.

107. function CopyC return Vector is

108. begin

109. return C;

110. end CopyC;

111.

112. function CopyT return Vector is

113. begin

114. return T;

115. end CopyT;

116.

117. function CopyMO return Matrix is

118. begin

119. return MO;

120. end CopyMO;

121.

122. function CopyAlf return Integer is

123. begin

124. return Alf;

125. end CopyAlf;

126. end ResourceMonitor;

127.

128. protected body SynchronizeMonitor is

129. procedure SignalInput is

130. begin

131. flagInput := flagInput + 1;

132. end SignalInput;

133.

134. procedure SignalCalculate1 is

135. begin

136. flagCalculate1 := flagCalculate1 + 1;

137. end SignalCalculate1;

138.

139. procedure SignalCalculate2 is

140. begin

141. flagCalculate2 := flagCalculate2 + 1;

142. end SignalCalculate2;

143.

144. entry WaitInput

145. when flagInput = 2 is

146. begin

147. null;

148. end WaitInput;

149.

150. entry WaitCalculate1

151. when flagCalculate1 = P is

152. begin

153. null;

154. end WaitCalculate1;

155.

156. entry WaitCalculate2

157. when flagCalculate2 = P is

158. begin

159. null;

160. end WaitCalculate2;

161.

162. end SynchronizeMonitor;

163.

164. task type CalculateTask(taskNumber: Integer);

165.

166. task body CalculateTask is

167. temp2, temp3, first, last: Integer;

168. Temp1:Vector;

169. Ti, Ci, Bi: Vector;

170. alfi: Integer;

171. MOi: Matrix;

172. begin

173. Put\_Line("Task " & Integer'Image(taskNumber) & " started...");

174.

175. if ( taskNumber = 1 ) then

176. -- task1---------------------------------

177. MatrixInput(MX);

178. MatrixInput(MK);

179. VectorInput(B);

180. Alf:=1;

181. ResourceMonitor.WriteB(B);

182. ResourceMonitor.WriteAlf(Alf);

183. SynchronizeMonitor.SignalInput;

184. end if;

185.

186. if ( taskNumber = P ) then

187. -- task P------------------------

188. VectorInput(C);

189. VectorInput(E);

190. MatrixInput(MO);

191. ResourceMonitor.WriteMO(MO);

192. ResourceMonitor.WriteC(C);

193. SynchronizeMonitor.SignalInput;

194. end if;

195. --task 2..p-1-----------------------------

196. SynchronizeMonitor.WaitInput;

197.

198. first := (taskNumber - 1) \* H + 1;

199. last := taskNumber \* H;

200.

201. Bi := ResourceMonitor.CopyB;

202. Ci := ResourceMonitor.CopyC;

203. -- calc1------------------------------------

204. for i in first..last loop

205. Temp1(i):=0;

206. for j in 1..N loop

207. Temp1(I):=Temp1(i)+C(j)\*MX(i)(j);

208. end loop;

209. T(i):=Temp1(i)+Bi(i);

210. end loop;

211.

212. SynchronizeMonitor.SignalCalculate1;

213. SynchronizeMonitor.WaitCalculate1;

214. ------------calc2---------------------------------------

215. Ti := ResourceMonitor.CopyT;

216. MOi := ResourceMonitor.CopyMO;

217. Alfi := ResourceMonitor.CopyAlf;

218.

219. for i in first..last loop

220. temp2:=0;

221. for j in 1..N loop

222. Temp3 := 0;

223. for k in 1..N loop

224. Temp3 := Temp3 + MK(I)(k)\*MOi(k)(j);

225. end loop;

226. Temp2:= Temp2 + Temp3\*T(j);

227. end loop;

228. A(i):= Alfi\*E(i)+Temp2;

229. end loop;

230. -------------------------------------------------------------------------------------------

231. if ( taskNumber = 1 ) then

232. SynchronizeMonitor.SignalCalculate2;

233. SynchronizeMonitor.WaitCalculate2;

234.

235. if ( N <= 8 ) then

236. VectorOutput(A);

237. end if;

238.

239. FinishTime := Clock;

240. DiffTime := FinishTime - StartTime;

241.

242. Put("Time = ");

243. Put(Integer(DiffTime), 5);

244. Put\_Line("");

245. else

246. SynchronizeMonitor.SignalCalculate2;

247. end if;

248.

249. Put\_Line("Task " & Integer'Image(taskNumber) & " finished");

250. end CalculateTask;

251.

252. ---------------------------------------------------------------

253. type CalculateTaskPointer is access CalculateTask;

254. type TasksArray is array(1..P) of CalculateTaskPointer;

255. tArray: TasksArray;

256.

257.

258. begin

259. StartTime := Clock;

260.

261. for i in 1..P loop

262. tArray(i) := new CalculateTask(i);

263. end loop;

264. end Main;

264 lines: No errors

**Додаток Б**

**ДОДАТОК Б.**

**Лістинг ПРГ2**

GNAT GPL 2013 (20130314)

Copyright 1992-2013, Free Software Foundation, Inc.

Compiling: D:\course\course2.adb (source file time stamp: 2015-04-15 22:01:34)

1. -------------------------------------------------------

2. -- Coursework

3. -- MA = MB\*(MC\*MZ)-max(E)\*MT

4. -- Kucenko Lera, IO-11

5. -- 13/05/2014

6. -------------------------------------------------------

7.

8. With Ada.Integer\_Text\_IO;

9. With Ada.Synchronous\_Task\_Control; use Ada.Synchronous\_Task\_Control;

10. With Data;

11. with Ada.Text\_IO; use Ada.Text\_IO;

12. with Ada.Calendar; use Ada.Calendar;

13. with Ada.Float\_Text\_IO; use Ada.Float\_Text\_IO;

14. with Ada.Integer\_Text\_IO; use Ada.Integer\_Text\_IO;

15.

16. PROCEDURE CourseWork IS

17.

18. N: Integer := 6;

19. P: Integer := 3;

20. H: Integer := N/P;

21.

22. procedure Start is

23.

24. startTime, endTime: Time;

25.

26. package Data\_Base is new Data(N); use Data\_Base;

27.

28. task type T1 (id: Integer) is

29. entry RightRecv1(MCi: in Matrix; Ei: in Vector; MTi: in Matrix);

30. entry LeftRecv(MB: out Matrix; MZ: out Matrix; E\_Send:out Vector; MC\_Send:out Matrix; MT\_Send: out Matrix);

31. entry LeftRecvP(MB: out Matrix; MZ: out Matrix);

32. entry MaxRecv(m: in Integer);

33. entry RealMaxRecv(m: in Integer);

34. entry ResultRecv(MV: in Matrix);

35. entry ResultRecvP(MV: in Matrix);

36. entry Start;

37. end T1;

38.

39. task type TP (id: Integer) is

40. entry RightRecv(MB: out Matrix; MZ: out Matrix; E\_Send:out Vector; MC\_Send:out Matrix; MT\_Send: out Matrix);

41. entry RealMaxRecv(m: in Integer);

42. entry MaxRecv(m: out Integer);

43. entry RightRecv1(MC\_Send: out Matrix; E\_Send: out Vector; MT\_Send: out Matrix);

44. entry LeftRecvP(MBi: in Matrix; MZi: in Matrix);

45. entry ResultRecv(MV: in Matrix);

46. entry ResultRecvP(MV: out Matrix);

47. entry Start;

48. end TP;

49.

50. task type TT1 (id: Integer) is

51. entry LeftRecv(MBi: in Matrix; MZi: in Matrix; Ei:in Vector; MCi:in Matrix; MTi: in Matrix);

52. entry MaxRecv(m: in Integer);

53. entry RealMaxRecv(m: in Integer);

54. entry ResultRecv(MV: in Matrix);

55. entry Start;

56. end TT1;

57.

58. task type TT2 (id: Integer) is

59. entry RightRecv(MBi: in Matrix; MZi: in Matrix; Ei:in Vector; MCi:in Matrix; MTi: in Matrix);

60. entry MaxRecv(m: in Integer);

61. entry RealMaxRecv(m: in Integer);

62. entry ResultRecv(MV: in Matrix);

63. entry Start;

64. end TT2;

65.

66. -- tasks array and ptr in task

67. type TT1\_Ptr is access TT1;

68. type TT2\_Ptr is access TT2;

69.

70. type T1\_Ptr is access T1;

71. type TP\_Ptr is access TP;

72.

73. --!!!!!!!!!!!!!!!!!!!!!

74. type T\_Ptr\_Array1 is array (2..P/2) of TT1\_Ptr;

75. tasks1: T\_Ptr\_Array1;

76. type T\_Ptr\_Array2 is array (P/2+1..P-1) of TT2\_Ptr;

77. tasks2: T\_Ptr\_Array2;

78. T\_1: T1\_Ptr;

79. T\_P: TP\_Ptr;

80.

81. --\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* T1 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*--

82. task body T1 is

83. MB, MZ, MA:Matrix(1..N, 1..N);

84.

85. -- variables to count

86. E\_H:Vector(1..H);

87. MT\_H, MC\_H:Matrix(1..N,1..H);

88. MAi:Integer;

89. MA\_H: Matrix(1..N, 1..H);

90.

91. MA\_1: Matrix(1..N, 1..(P/2)\*H);

92.

93. E\_Send: Vector(1..(P/2-1)\*H);

94. MT\_Send, MC\_Send:Matrix(1..N,1..(P/2-1)\*H);

95. begin

96.

97. accept Start;

98. Put\_Line("T" & Integer'Image(id) & " start");

99.

100. -- Input data

101. MB:=fillMatrix(1);

102. MZ:=fillMatrix(1);

103.

104. --data from Tp

105. accept RightRecv1(MCi: in Matrix; Ei: in Vector; MTi: in Matrix) do

106. E\_H := getVectorPart(Ei,1,H);

107. MC\_H := getMatrixPart(MCi,1,H);

108. MT\_H := getMatrixPart(MTi,1,H);

109.

110. E\_Send := getVectorPart(Ei,H+1,(P/2-1)\*H);

111. MC\_Send := getMatrixPart(MCi,H+1,(P/2-1)\*H);

112. MT\_Send := getMatrixPart(MTi,H+1,(P/2-1)\*H);

113. end RightRecv1;

114.

115. T\_P.LeftRecvP(MB, MZ);

116. -- Send data to next task

117.

118. if (P > 2) then

119. tasks1(2).LeftRecv(MB, MZ, E\_Send, MC\_Send, MT\_Send);

120. end if;

121.

122. -- count max(EH)

123. MAi := getMaxVectorElement(E\_H);

124.

125. if (P = 2) then

126. accept MaxRecv (m : in Integer) do

127. MAi := Max(m, MAi);

128. end MaxRecv;

129.

130. T\_P.RealMaxRecv(MAi);

131. else

132.

133. Tasks1(2).MaxRecv(MAi);

134.

135. -- recv real max element

136. accept RealMaxRecv (m : in Integer) do

137. MAi := m;

138. end RealMaxRecv;

139. end if;

140.

141. -- MA\_H = MB\*(MC\_H\*MZ)-MAi\*MT\_H

142. MA\_H := SubMatrix(matrixMultiplication(MB, matrixMultiplication(MZ, MC\_H)),scalarMultiplicationOnMatrix(MAi, MT\_H));

143.

144.

145. if (P > 2) then

146. accept ResultRecv (MV : in Matrix) do

147. MA\_1 := concatMatrix(MA\_H, MV);

148. end ResultRecv;

149. accept ResultRecvP (MV : in Matrix) do

150. MA := concatMatrix(MA\_1, MV);

151. end ResultRecvP;

152. else

153. accept ResultRecvP (MV : in Matrix) do

154. MA := concatMatrix(MA\_H, MV);

155. end ResultRecvP;

156. end if;

157.

158. -- OUTPUT RESULT

159. if (N<11) then

160. MatrixOutput(MA);

161. end if;

162.

163. -- output time

164. endTime := clock;

165. Put("Time (s): ");

166. Ada.Float\_Text\_IO.Put(Float(endTime - startTime), Fore => 10, Aft => 10, exp =>0);

167. Put\_Line(" ");

168. Put\_Line("T" & Integer'Image(id) & " end");

169. end T1;

170.

171. --\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* TT1 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*--

172. task body TT1 is

173.

174. -- variables to count

175. E\_H:Vector(1..H);

176. MT\_H, MC\_H:Matrix(1..N,1..H);

177. MB, MZ: Matrix(1..N, 1..N);

178. MAi:Integer;

179. MA\_H: Matrix(1..N, 1..H);

180. temp:Integer;

181. MA\_A:Matrix(1..N, 1..(P/2-id+1)\*H);

182.

183. E\_Send: Vector(1..(P/2-id)\*H);

184. MT\_Send, MC\_Send:Matrix(1..N, 1..(P/2-id)\*H);

185.

186. begin

187. accept Start;

188. Put\_Line("T" & Integer'Image(id) & " start");

189.

190.

191. --data from Tp

192. accept LeftRecv(MBi: in Matrix; MZi: in Matrix; Ei: in Vector; MCi: in Matrix; MTi: in Matrix) do

193. MB := MBi;

194. MZ := MZi;

195. E\_H := getVectorPart(Ei,1,H);

196. MC\_H := getMatrixPart(MCi,1,H);

197. MT\_H := getMatrixPart(MTi,1,H);

198.

199. if(id < P/2) then

200. E\_Send := getVectorPart(Ei,H+1,(P/2-id)\*H);

201. MC\_Send := getMatrixPart(MCi,H+1,(P/2-id)\*H);

202. MT\_Send := getMatrixPart(MTi,H+1,(P/2-id)\*H);

203. end if;

204. end LeftRecv;

205.

206. if(id < P/2) then

207. Tasks1(id+1).LeftRecv(MB, MZ, E\_Send, MC\_send, MT\_Send);

208.

209. end if;

210.

211. -- count max(EH)

212. MAi := getMaxVectorElement(E\_H);

213.

214. accept MaxRecv (M: in Integer) do

215. temp:=M;

216. end MaxRecv;

217.

218. MAi:=Max(MAi,Temp);

219.

220.

221.

222. if (id = P/2) then

223. Tasks2(P/2+1).MaxRecv(MAi);

224. else

225. Tasks1(id+1).MaxRecv(MAi);

226. end if;

227.

228. -- recv real max element

229. accept RealMaxRecv (m : in Integer) do

230. MAi := m;

231. end RealMaxRecv;

232.

233. if(id-1 > 1) then

234. Tasks1(id-1).RealMaxRecv(MAi);

235. else

236. T\_1.RealMaxRecv(MAi);

237. end if;

238.

239. -- MA\_H = MB\*(MC\_H\*MZ)-MAi\*MT\_H

240. MA\_H := SubMatrix(matrixMultiplication(MB, matrixMultiplication(MZ, MC\_H)),scalarMultiplicationOnMatrix(MAi, MT\_H));

241.

242. if(id = P/2) then

243. if(id = 2) then

244. null;

245. else

246. Tasks1(id-1).ResultRecv(MA\_H);

247. end if;

248. else

249. if (Id = 2) and (id < P/2) then

250. accept ResultRecv (MV : in Matrix) do

251. MA\_A := concatMatrix(MA\_H, MV);

252. end ResultRecv;

253. T\_1.ResultRecv(MA\_A);

254. else

255. Put("azaza");

256. accept ResultRecv (MV : in Matrix) do

257. MA\_A:= concatMatrix(MA\_H, MV);

258. end ResultRecv;

259.

260. Tasks1(Id-1).ResultRecv(MA\_A);

261. end if;

262. end if;

263.

264. end TT1;

265.

266.

267. --\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* TT2 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*--

268. task body TT2 is

269.

270. -- variables to count

271. E\_H:Vector(1..H);

272. MT\_H, MC\_H:Matrix(1..N,1..H);

273. MB, MZ: Matrix(1..N, 1..N);

274. MAi:Integer;

275. MA\_H: Matrix(1..N, 1..H);

276.

277. E\_Send: Vector(1..H\*(id-P/2-1));

278. MT\_Send, MC\_Send:Matrix(1..N, 1..H\*(id-P/2-1));

279.

280. MA\_A: Matrix(1..N, 1..(id-p/2)\*H);

281.

282. begin

283. accept Start;

284. Put\_Line("T" & Integer'Image(id) & " start");

285.

286. --data from Tp

287. accept RightRecv(MBi: in Matrix; MZi: in Matrix; Ei: in Vector; MCi: in Matrix; MTi: in Matrix) do

288. MB := MBi;

289. MZ := MZi;

290. E\_H := getVectorPart(Ei,H\*(id-P/2-1)+1,H);

291. MC\_H := getMatrixPart(MCi,H\*(id-P/2-1)+1,H);

292. MT\_H := getMatrixPart(MTi,H\*(id-P/2-1)+1,H);

293.

294. if(id > P/2+1) then

295. E\_Send := getVectorPart(Ei,1,H\*(id-P/2-1));

296. MC\_Send := getMatrixPart(MCi,1,H\*(id-P/2-1));

297. MT\_Send := getMatrixPart(MTi,1,H\*(id-P/2-1));

298. end if;

299. end RightRecv;

300.

301. if(id > P/2 + 1) then

302. Tasks2(id-1).RightRecv(MB, MZ, E\_Send, MC\_send, MT\_Send);

303. end if;

304.

305. -- count max(EH)

306. MAi := getMaxVectorElement(E\_H);

307.

308. accept MaxRecv (M: in Integer) do

309. MAi:=Max(MAi,M);

310. end MaxRecv;

311.

312. if (id = p/2 +1) then

313. accept MaxRecv (M: in Integer) do

314. MAi:=Max(MAi,M);

315. end MaxRecv;

316.

317. Tasks1(P/2).RealMaxRecv(MAi);

318. Tasks2(Id+1).RealMaxRecv(MAi);

319. else

320. Tasks2(id-1).MaxRecv(MAi);

321. end if;

322.

323. if(id > P/2 + 1) then

324. -- recv real max element

325. accept RealMaxRecv (m : in Integer) do

326. MAi := m;

327. end RealMaxRecv;

328.

329. if (id = P-1) then

330. T\_P.RealMaxRecv(MAi);

331. else

332. Tasks2(Id+1).RealMaxRecv(MAi);

333. end if;

334. end if;

335.

336. -- MA\_H = MB\*(MC\_H\*MZ)-MAi\*MT\_H

337. MA\_H := SubMatrix(matrixMultiplication(MB, matrixMultiplication(MZ, MC\_H)),scalarMultiplicationOnMatrix(MAi, MT\_H));

338.

339. if(id = P/2+1) then

340. if(Id = P-1) then

341. null;

342. else

343. Tasks2(id+1).ResultRecv(MA\_H);

344. end if;

345. else

346. if (id = P-1) and (id > P/2+1) then

347. accept ResultRecv (MV : in Matrix) do

348. MA\_A:= concatMatrix(MV, MA\_H);

349. end ResultRecv;

350.

351. T\_P.ResultRecv(MA\_A);

352.

353. else

354. accept ResultRecv (MV : in Matrix) do

355. MA\_A:= concatMatrix(MV, MA\_H);

356. end ResultRecv;

357.

358. Tasks2(id+1).ResultRecv(MA\_A);

359. end if;

360. end if;

361.

362. end TT2;

363.

364.

365. --\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* TP \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*--

366. task body TP is

367. -- variables to count

368. E\_H: Vector(1..H);

369. MC\_H, MT\_H :Matrix(1..N, 1..H);

370. MAi:Integer;

371. E: Vector(1..N);

372. MT, MC, MB, MZ: Matrix(1..N, 1..N);

373. MA\_H:Matrix(1..N, 1..H);

374. E\_Send: Vector(1..(P/2)\*H-H);

375. MT\_Send, MC\_Send:Matrix(1..N,1..(P/2)\*H-H);

376.

377. E\_Send1: Vector(1..(P/2)\*H);

378. MT\_Send1, MC\_Send1: Matrix(1..N, 1..(P/2)\*H);

379.

380. begin

381. accept Start;

382. Put\_Line("T" & Integer'Image(id) & " start");

383. -- Input data

384.

385. MT:=fillMatrix(1);

386. MC:=fillMatrix(1);

387. E:=fillVector(1);

388.

389. -- create data to count

390. MT\_H := GetMatrixPart(MT, n-H+1, H);

391. MC\_H := GetMatrixPart(MC, n-H+1, H);

392. E\_H := getVectorPart(E, N-H+1, H);

393.

394.

395. MT\_Send1 := getMatrixPart(MT, 1, (P/2)\*H);

396. MC\_Send1 := getMatrixPart(MC, 1, (P/2)\*H);

397. E\_Send1 := getVectorPart(E, 1, (P/2)\*H);

398.

399. T\_1.RightRecv1(MC\_Send1, E\_Send1, MT\_Send1);

400.

401. accept LeftRecvP(MBi: in Matrix; MZi: in Matrix) do

402. MB:=MBi;

403. MZ:=MZi;

404. end;

405.

406. if (P > 2) then

407. MT\_Send := getMatrixPart(MT, (P/2)\*H+1, (P/2-1)\*H);

408. MC\_Send := getMatrixPart(MC, (P/2)\*H+1, (P/2-1)\*H);

409. E\_Send := getVectorPart(E, (P/2)\*H+1, (P/2-1)\*H);

410.

411. tasks2(id-1).RightRecv(MB, MZ, E\_Send, MC\_Send, MT\_Send);

412. end if;

413.

414. -- count max(EH)

415. MAi := getMaxVectorElement(E\_H);

416.

417. if ( P > 2) then

418. -- send max to prev

419. tasks2(P-1).MaxRecv(MAi);

420. else

421. T\_1.MaxRecv(MAi);

422. end if;

423.

424. -- recv real max element

425. accept RealMaxRecv (m : in Integer) do

426. MAi := m;

427. end RealMaxRecv;

428.

429. -- MA\_H = MB\*(MC\_H\*MZ)-MAi\*MT\_H

430. MA\_H := SubMatrix(matrixMultiplication(MB, matrixMultiplication(MZ, MC\_H)),scalarMultiplicationOnMatrix(MAi, MT\_H));

431.

432. -- send result to prev

433. if( P > 2) then

434. accept ResultRecv(MV: in Matrix) do

435. T\_1.ResultRecvP(ConcatMatrix(MV, MA\_H));

436. end ResultRecv;

437. else

438. T\_1.ResultRecvP(MA\_H);

439. end if;

440.

441. Put\_Line("T" & Integer'Image(id) & " end");

442. end TP;

443.

444. begin

445. -- Create tasks

446. for i in 2..(P/2) loop

447. tasks1(i) := new TT1(id => i);

448. end loop;

449.

450. for i in P/2+1..P-1 loop

451. tasks2(i) :=new TT2(id => i);

452. end loop;

453.

454. T\_1 := new T1(1);

455. T\_P := new TP(P);

456.

457. -- Start tasks

458. startTime := clock;

459.

460. T\_1.Start;

461. T\_P.Start;

462.

463. for i in 2..P/2 loop

464. tasks1(i).Start;

465. end loop;

466.

467. for i in P/2+1..p-1 loop

468. tasks2(i).Start;

469. end loop;

470.

471. end Start;

472.

473. BEGIN

474. Start;

475. END CourseWork;

475 lines: No errors

GNAT GPL 2013 (20130314)

Copyright 1992-2013, Free Software Foundation, Inc.

Compiling: D:\course\data.adb (source file time stamp: 2015-04-15 22:01:26)

1. with Ada.Integer\_Text\_IO;

2. with Ada.Text\_IO; use Ada.Text\_IO;

3.

4. package body Data is

5.

6. -- output vector in console

7. procedure vectorOutput(v: in Vector) is

8. begin

9. for i in 1..v'Length loop

10. Ada.Integer\_Text\_IO.Put(v(i));

11. Ada.Text\_IO.Put(" ");

12. end loop;

13. Ada.Text\_IO.Put\_Line(" ");

14. end vectorOutput;

15.

16. -- output matrix in console

17. procedure matrixOutput(m: in Matrix) is

18. begin

19. for i in 1..m'Length(1) loop

20. for j in 1..m'Length(2) loop

21. Ada.Integer\_Text\_IO.Put(m(i,j));

22. Ada.Text\_IO.Put(" ");

23. end loop;

24. Ada.Text\_IO.Put\_Line("");

25. end loop;

26. end matrixOutput;

27.

28. -- fill matrix by one number

29. function fillMatrix(number: in Integer) return Matrix is

30. M : Matrix(1..N,1..N);

31. begin

32. for i in 1..M'Length(1) loop

33. for j in 1..M'Length(2) loop

34. M(i,j):=number;

35. end loop;

36. end loop;

37. return M;

38. end fillMatrix;

39.

40. -- fill vector by one number

41. function fillVector(number: in Integer) return Vector is

42. V : Vector(1..N);

43. begin

44. for i in 1..V'Length loop

45. V(i):=number;

46. end loop;

47. return V;

48. end fillVector;

49.

50. -- get part from matrix

51. function getMatrixPart(MM: in Matrix; start: in Integer;

52. size: in Integer) return Matrix is

53. MR: Matrix(1..N,1..size);

54. begin

55. for i in 1..MM'Length(1) loop

56. for j in start..(start+size-1) loop

57. MR(i,j-start+1):=MM(i,j);

58. end loop;

59. end loop;

60. return MR;

61. end;

62.

63. -- get part of vector

64. function getVectorPart(V: in Vector; start: in Integer;

65. size: in Integer) return Vector is

66. R: Vector(1..size);

67. begin

68. for i in start..(start+size-1) loop

69. R(i-start+1):=V(i);

70. end loop;

71. return R;

72. end;

73.

74. -- find max element in vector

75. function getMaxVectorElement(v: in Vector) return Integer is

76. max : Integer;

77. begin

78. if (V'Length > 0 ) then

79. max := v(1);

80. for i in 1..v'Length loop

81. if v(i)>max then

82. max := v(i);

83. end if;

84. end loop;

85. else

86. max:= -1\*(2 \*\* 31 - 1);

87. end if;

88. return max;

89. end getMaxVectorElement;

90.

91. -- multiplication matrix on matrix

92. function matrixMultiplication(m1 : in Matrix; m2 :

93. in Matrix ) return Matrix is

94. MR :Matrix(1..m1'Length(1),1..m2'Length(2));

95. begin

96. for i in 1..m1'Length(1) loop

97. for j in 1..m2'Length(2) loop

98. MR(i, j) := 0 ;

99. for k in 1..m1'Length loop

100. MR(i, j) := MR(i, j) + m1(i, k)\*m2(k, j);

101. end loop;

102. end loop;

103. end loop;

104. return MR;

105. end matrixMultiplication;

106.

107. -- multiplication scalar on matrix

108. function scalarMultiplicationOnMatrix(s: in Integer;

109. M: in Matrix) return Matrix is

110. MR :Matrix(1..m'Length(1),1..m'Length(2));

111. begin

112. for i in 1..M'Length(1) loop

113. for j in 1..M'Length(2) loop

114. MR(I,J) := M(i,J) \* s;

115. end loop;

116. end loop;

117. return MR;

118. end scalarMultiplicationOnMatrix;

119.

120. -- sub two matr

121. function SubMatrix(M1: in Matrix; M2: in Matrix) return Matrix is

122. MR: Matrix(1..N, 1..M1'Length(2));

123. begin

124. for i in 1..N loop

125. for j in 1..M1'Length(2) loop

126. MR(i,J) := M1(i,J) - M2(i,J);

127. end loop;

128. end loop;

129.

130. return MR;

131. end SubMatrix;

132.

133. -- concatenation two matrix in one

134. function concatMatrix(M1: Matrix; M2: Matrix) return Matrix is

135. MR: Matrix(1..N, 1..M1'Length(2) + M2'Length(2));

136. begin

137. for i in 1..N loop

138. for j in 1..M1'Length(2) loop

139. MR(i,J) := M1(i,J);

140. end loop;

141. end loop;

142.

143. for i in 1..N loop

144. for j in M1'Length(2)+1..M1'Length(2)+M2'Length(2) loop

145. MR(i,J) := M2(I,J-M1'Length(2));

146. end loop;

147. end loop;

148. return MR;

149. end concatMatrix;

150.

151. -- count max of two numbers

152. function max(a1:in Integer; a2: in Integer) return Integer is

153. max: Integer;

154. begin

155. if(a1>a2) then

156. max:=a1;

157. else

158. max:= a2;

159. end if;

160. return max;

161. end max;

162.

163. end Data;

Compiling: D:\course\Data.ads (source file time stamp: 2015-04-15 22:01:30)

1. generic

2. N: Positive := 3; -- size of vector and matrix

3.

4. package Data is

5. type Vector is array (Positive range <>) of Integer; -- vector type

6. type Matrix is array (Positive range <>, Positive range <>) of Integer; -- matrix type

7.

8. procedure vectorOutput(v: in Vector); -- output vector in console

9. procedure matrixOutput(m: in Matrix); -- output matrix in console

10. function fillMatrix(number: in Integer) return Matrix; -- fill matrix by one number

11. function fillVector(number: in Integer) return Vector; -- fill vector by one number

12. function getMatrixPart(MM: in Matrix; start: in Integer; -- get part from matrix

13. size: in Integer) return Matrix;

14. function getVectorPart(V: in Vector; start: in Integer; -- get part of vector

15. size: in Integer) return Vector;

16. function getMaxVectorElement(v: in Vector) return Integer; -- find max element in vector

17. function matrixMultiplication(m1 : in Matrix; m2 : in Matrix) -- multiplication matrix on matrix

18. return Matrix;

19.

20. function scalarMultiplicationOnMatrix(s: in Integer; M: in Matrix) -- multiplication scalar on matrix

21. return Matrix;

22. function SubMatrix(M1: in Matrix; M2: in Matrix) return Matrix; -- sub two matrix

23. function concatMatrix(M1: Matrix; M2: Matrix) return Matrix; -- concatenation two vectors in one

24. function max(a1:in Integer; a2: in Integer) return Integer; -- count max of two numbers

25. end Data;

163 lines: No errors

**Додаток В**



*Изм.*

*Лист*

*№ докум.*

*Подпись*

*Дата*

*Лист*

1

*Схема алгоритму процесів для ПКС із СП*

*Розроб.*

*Бута С.О.*

*Провір.*

*Корочкін О.В.*

*Реценз.*

*Н. Контр.*

*Утверд.*

*Додаток В*

*Алгоритми процесів*

*Лит.*

*Листов*

9

*НТУУ “КПІ” ФІОТ*



*Изм.*

*Лист*

*№ докум.*

*Подпись*

*Дата*

*Лист*

2

*Схема алгоритму процесів для ПКС із СП*

*Розроб.*

*Бута С.О.*

*Провір.*

*Корочкін О.В.*

*Реценз.*

*Н. Контр.*

*Утверд.*

*Додаток В*

*Алгоритми процесів*

*Лит.*

*Листов*

9

*НТУУ “КПІ” ФІОТ*