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 групи ІО-24

напряму підготовки 050102

«Комп’ютерна інженерія»

\_\_\_\_\_\_Кондратюка В.Ю\_\_\_\_\_\_\_\_\_\_\_\_

(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- 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 КУРСОВИЙ ПРОЕКТ СТУДЕНТУ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Кондратюк Віталій Юрійович\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(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 2015 р.

3. Вихідні дані до роботи

- порівняння реалізації механізмів семафорів в мовах і бібліотеках паралельного програмування

- математична задача

- структури ПКС ОП та ПКС ЛП

- мови і бібліотеки програмування:

- засоби організації взаємодії процесів:

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

- порівняння реалізації механізму семафорів 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 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 2.02. 2015

***КАЛЕНДАРНИЙ ПЛАН***

|  |  |  |
| --- | --- | --- |
| №  з/п | Назва етапів виконання КП | Строк виконання етапів КП |
| 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 \_\_\_\_\_\_\_\_\_ \_\_**Кондратюк В,Ю.**\_\_\_\_\_\_**

( підпис ) (прізвище та ініціали)

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

( підпис ) (прізвище та ініціали)

**ЗМІСТ**

[РОЗДІЛ 1. ПОРІВНЯННЯ РЕАЛІЗАЦІЇ МЕХАНІЗМУ СЕМАФОРІВ В МОВАХ І БІБЛІОТЕКАХ ПАРАЛЕЛЬНОГО ПРОГРАМУВАННЯ 3](#_Toc414829480)

[1.1 Семафори в мові Ada 3](#_Toc414829481)

[1.2 Семафори в мові Java 3](#_Toc414829482)

[1.3 Семафори в бібліотеці Win32 3](#_Toc414829483)

[1.4 Семафори в мові C# 3](#_Toc414829484)

[1.5 Висновки до розілу 1 3](#_Toc414829485)

[РОЗДІЛ 2. РОЗРОБКА ПРОГРАМИ ПРГ1 ДЛЯ ПКС ЗП 3](#_Toc414829486)

[2.1 Розробка паралельного математичного алгоритму 3](#_Toc414829487)

[2.2 Розробка алгоритмів процесів 3](#_Toc414829488)

[2.3 Розробка схеми взаємодії процесів 3](#_Toc414829489)

[2.4 Розробка програми ПРГ1 3](#_Toc414829490)

[1.5 Висновки до розілу 2 3](#_Toc414829485)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 3](#_Toc414829491)

[ДОДАТКИ 3](#_Toc414829492)

[3.1 Додаток А. Лістинг до ПРГ1 3](#_Toc414829493)

# РОЗДІЛ 1. ПОРІВНЯННЯ РЕАЛІЗАЦІЇ МЕХАНІЗМУ СЕМАФОРІВ В МОВАХ І БІБЛІОТЕКАХ ПАРАЛЕЛЬНОГО ПРОГРАМУВАННЯ

Семафор — це універсальний [механізм](http://uk.wikipedia.org/wiki/%D0%9C%D0%B5%D1%85%D0%B0%D0%BD%D1%96%D0%B7%D0%BC_(%D0%B7%D0%BD%D0%B0%D1%87%D0%B5%D0%BD%D0%BD%D1%8F)) для організації взаємодії [процесів](http://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81_(%D1%96%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) (в термінології операційних систем сімейства Windows — [потоків](http://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D1%82%D1%96%D0%BA_(Windows))) [ 1]. Розв'язує задачі взаємного виключення та синхронізації [потоків](http://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D1%82%D1%96%D0%BA_(Windows)). Він є одним з найстаріших засобів розподілення доступу [процесів](http://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81_(%D1%96%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)), що працюють паралельно, до критичних ресурсів. Семафори використовуються для контролю доступу до спільного ресурсу, або для синхронізації [процесів](http://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81_(%D1%96%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) ([потоків](http://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D1%82%D1%96%D0%BA_(Windows))). Визначення семафору зроблено нідерландським вченим [Едсгером Дейкстрою](http://uk.wikipedia.org/wiki/%D0%95%D0%B4%D1%81%D0%B3%D0%B5%D1%80_%D0%94%D0%B5%D0%B9%D0%BA%D1%81%D1%82%D1%80%D0%B0), деякий час використовувався термін Семафор Дейкстри. Семафор — це об'єкт [ядра](http://uk.wikipedia.org/wiki/%D0%AF%D0%B4%D1%80%D0%BE_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D1%96%D0%B9%D0%BD%D0%BE%D1%97_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B8) [ОС](http://uk.wikipedia.org/wiki/%D0%9E%D0%A1), який можна розглядати як лічильник, що містить ціле число в діапазоні від 0 до заданого при його створенні максимального значення. При досягненні семафором значення 0 він переходить у несигнальний стан, при будь-яких інших значеннях лічильника – його стан сигнальний. ГДЕ ССЫЛКИ???

Традиційне позначення семафора : S. Операції, які можна виконати над семафором:

1. Ініціалізація - встановлення початкового значення семафору.
2. Операція P(S): Вона перевіряє стан семафору. Якщо семафор не рівний нулю, то виконується операція S:=S-1. Інакше, процес блокується, поки S=0.
3. Операція V(S): Ця операція збільшує значення семафору на 1. Тобто виконується операція S:=S+1.

В залежності від значень, які може приймати семафор він поділяється на:

* Двійковий : здатний приймати значення 0 та 1.
* Трійковий : здатний приймати значення 0, 1 та 2.
* і т. д.

З допомогою семафорів можна розв'язати задачі синхронізації потоків та використання спільного ресурсу багатьма потоками.

## Семафори в мові Ada

Вбудована підтримка багатозадачності є унікальною і широко відомою особливістю мови програмування Ада, яка суттєво відрізняє його від більшості сучасних мов програмування. Слід особливо підкреслити, що підтримка багатозадачності забезпечується не за допомогою будь-яких розширень або зовнішніх бібліотек, а за допомогою строго стандартизованих засобів, які вбудовані безпосередньо в мову програмування. Забезпечення підтримки багатозадачності на рівні мови програмування має важливе значення.

Завдання синхронізації двох процесів полягає в тому, що в од­ному процесі, наприклад Б, у визначеній точці (точці події) вико­нується подія (обчислення даних, уведення або виведення даних і т. ін.), а другий процес А у визначеній точці (точці очікування події) блокується доти, доки ця подія не відбудеться і він зможе продов­жити своє виконання. Точка події і точка очікування - це точки синхронізації Якщо процес А вийшов на точку синхронізації, коли подія вже відбулася, то він не блокується і продовжує виконува­тись.

Існує декілька схем синхронізації процесів :

* один процес очікує на подію в одному процесі (а);
* декілька процесів очікують на подію в одному процесі (b);
* один процес очікує на події в кількох процесах (с).

Для вирішення завдання синхронізації, яке іноді називають синхронізацією за подіями (eventsynchronization), можна викорис­товувати різні механізми синхронізації процесів, таки як семафо­ри, події, монітори.Для вирішення завдання синхронізації в мові Ада95 можна за­стосувати механізм семафорів, а також захищені модулі.

Застосування семафорів. Розглянемо приклад, у якому задача А чекає на подію, яку мають містити задачі В, наприклад, введення даних (змінної х). Для синхронізації процесів з уведення використовуємо семафор СігналПроПодію з початковим значенням false, яке встановлюється автоматично під час створювання сема­фора. Очікування події в процесі А реалізовано за допомогою опе­рації Suspend\_\_Until\_True (СігналПроПодію), а посилання сигналу про подію в задачі В реалізовано за допомогою операції SetTrue(СігналПроПодію) .

procedureСинхронізація is

X: integer; -- глобальна змінна

СігналПроПодію: Suspension\_Object; -- семафор

— задача, що чекає на подію

taskA;

task body A is

begin

точка очікування події

Suspend\_Until\_True(СігналПроПодію);

end A;

-- задача, де пройде подія

task В;

task body В is

begin

get(X); -- введення даних(подія, наяку чекає А)

Set\_True(СігналПроПодію); -- сигнал задачі А

end В; begin

null ;

end Синхронізація;

Механізм семафорів запропонував математик Е.Дейкстра. У класичній інтерпретації механізм семафорів - це спеціальний захищений тип Semaphore та дві неподільні операції над змінною цього типу: P(S) і V(S).Неподільність операції означає, що її не можна переривати, поки не завершиться її виконання.

У мові Ада механізм семафорів подано у вигляді пакета Synchronous\_Task\_Control в додатку Annex D: Real-Time Systems. Пакет реалізує механізм семафорів таким чином. Семафорний тип забезпечується приватним типом Suspension\_object, операції P(S) і V(S) реалізовані за допо­могою процедур Suspend\_Until\_True () і SetJTrue (). Викори­стовується бінарний логічний семафор, тобто семафорні змінні типу Suspension\_Object набувають значень false і true. Крім указаних процедур, в пакеті реалізовані допоміжні процедури SetFalseOдля встановлення значення семафора в false і Currentstate () для зчитування поточного значення семафора. Специфікація пакета:

package Ada.Synchronous\_Task\_Control is

type Suspension\_Object limited private ;

rocedure Set\_True(S : in out 3uspension\_Object);

procedure Set\_False(S : in out Suspension\_Object);

function Current\_State(S : Suspension\_Object) return Boolean ;

procedure Suspend\_Until\_True(3:in out Suspension\_Object); private

end Ada. Synchronous\_Task\_Control;

## Семафори в мові Java

Синхронізація в Java гарантує, що ніякі два потоки не зможуть виконати синхронізований метод одночасно або паралельно. В Java існує два механізма синхронізації:на рівні класу, та на рівні об’єкта .Мова Java не підтримує семафорів, але вона дозволяє створювати класи, методи яких будуть мати основну властивість процедур семафора – виконуватися в режимі взаємного виключення. Такі методи в Java повинні мати модифікатор synchronized. В Java механізм семафорів реалізований за допомогою моніторів. В Java в кожного екземпляра об'єкта є монітор. Реалізований він native-кодом і контролюється віртуальною машиною. Використовується він так: будь-який нестатичний synchronized-метод при своєму виклику насамперед намагається захопити монітор того об'єкта, у якого він викликаний (на який він може послатися як на this). Якщо це вдалося - метод виконується. Якщо ні – потік блокується і чекає, поки монітор буде відпущений.

Приклад монітору в мові Java.

public class Monitor {

public synchronized void someMethod1(){

//code

}

public synchronized void someMethod2(){

//code

}

}

Якщо два нестатичних методи оголошені як synchronized, то в кожен момент часу з різних потоків в одному об'єкті може бути викликаний тільки один з них. Потік, який викликає метод першим, захопить монітор, і другий потоку доведеться чекати. Є три важливі моменти:

* Це вірно лише для різних потоків. Один і той же потік може викликати синхронізований метод, всередині нього - інший синхронізований метод на тому ж екземплярі. Оскільки цей потік володіє монітором, проблем другий виклик не створить.
* Це вірно лише для викликів методів одного екземпляра. У різних примірників різні монітори, тому одночасний виклик нестатичних методів проблем не створить
* У разі статичних методів має значення тільки одне - чи різні потоки, які викликають синхронізовані методи, чи ні. Про екземпляри тут мови не йде, його роль виконує об'єкт класу.

Для ефективної передачі управління від одного потока іншому в мові Java є методи wait(), notify() і notifyAll() класу Object.

Методи notify() та notifyAll() "пробуджують" потік, що очікує методом wait() (якщо такий є), і переводять його в стан очікування звільнення монітора. Різниця між notify() і notifyAll() в тому, що notify() пробуджує тільки один потік, що очікує методом wait()(який саме - визначити не можна), а notifyAll() - всі такі потоки.[8]

Приклад використання методів wait(), notifyAll() в мові Java.  
class AutoResetEvent {  
 private boolean isSet;  
 public AutoResetEvent(boolean initialState) {  
 isSet=initialState;  
 }  
 public synchronized void await() throws InterruptedException {  
 while(!isSet)  
 wait();  
 isSet=false;  
 }  
 public synchronized void set() {  
 isSet=true;  
 notifyAll();  
 }  
}  
// тестовий потік, який при натисненні ENTER встановлює подію  
class TestThread implements Runnable {  
 AutoResetEvent ev;  
 TestThread(AutoResetEvent ev) {  
 this.ev=ev;  
 }  
 public void run() {  
 System.out.println("press ENTER to set event");  
 try {  
 while(System.in.read()!='\n');  
 ev.set();  
 }  
 catch (Exception e) {  
 }  
 }  
}  
// демонстрація роботы класа AutoResetEvent  
public class jfirts {  
 public static void main(String[] args) {    
 AutoResetEvent ev=new AutoResetEvent(false);  
 Thread t1=new Thread(new TestThread(ev));  
 t1.start();  
 try {  
 ev.await();  
 System.out.println("event is set");  
 }  
 catch(Exception e){  
 }  
 }  
}

У цьому прикладі створюється об'єкт-подія AutoResetEvent і створюється потік, якому передається цей обєкт-подія. Після чого викликається метод AutoResetEvent.await (), який чекає поки другий потік після натискання ENTER не встановив подію в стан «set», після чого метод wait() повертає управління, перед цим встановивши стан об'єкта-події в «reset».

## Семафори в бібліотеці Win32

У загальному випадку, синхронізація - це узгодження дій різних потоків. Середовище Win32 API містить ряд об'єктів, які застосовують для синхронізації потоків - м'ютекси, критичні секції, семафори і події. З семафорами, як і раніше з критичними секціями, ми будемо працювати за допомогою спеціальних функцій з арсеналу Win32. Для початку познайомимося з функцією, яка відповідає за створення семафорів. CreateSemaphore (nil, 1, 1, nil) - це створення семафора без власника і з яким може працювати тільки один потік. Розглянемо докладніше цю функцію.

HANDLE CreateSemaphore (LPSECURITY\_ATTRIBUTES

lpSemaphoreAttributes, // атрибут доступу

LONG lInitialCount, // ініціалізувати початковий стан лічильника

LONG lMaximumCount, // максимальна кількість звернень

LPCTSTR lpName // ім'я об'єкта);

Перший параметр - це атрибути безпеки, які передаються функції для можливості використання семафора іншими процесами. Другий і третій параметри - це початкове і максимальне значення лічильника, вбудованого в кожен семафор. Він рахує кількість потоків, що знаходяться в даний момент часу в семафорі. Само собою, максимальна кількість повинна бути не менше, ніж початкова, а початкова повинна бути не менше нуля. Останній параметр, який передається функції CreateSemaphore при її виклику - це ім'я семафора, яке не повинно містити зворотних слешів (\). Ще одна функція, яка знадобиться нам при роботі з семафорами, - це WaitForSingleObject. Виглядає вона в довідці по Delphi наступним чином: DWORD WaitForSingleObject (HANDLE hHandle, DWORD dwMilliseconds).

DWORD WaitForSingleObject(

Функція завершується в тому випадку, якщо при перевірці стану сигналізуючого об'єкта

відбувається одна з подій:

  · Стан об'єкта змінився на "Включений"

  · Час очікування, визначеного при виконанні функції, минув.

параметри:

  HANDLE hHandle, // Handle сигналізуючого об'єкта

  DWORD dwMilliseconds // Час очікування в мілісекундах);

Як бачите, параметрів у неї небагато - всього два. Ця функція переводить потік в стан очікування звільнення якого-небудь зайнятого ресурсу, з якого він може вийти в двох випадках: або звільнення даного ресурсу іншим потоком, або ж закінчення часу, виділеного на дане очікування. Відповідно, передаються при виклику цієї функції параметри як раз і визначають ці два значення: перший параметр - це дескриптор того об'єкта, звільнення якого буде очікувати наш потік, а другий - це кількість мілісекунд, яке повинне закінчитися до того, як потік завершить очікування. Якщо ми хочемо, щоб потік просто перевірив стан об'єкта, то можемо поставити в якості часу 0, а якщо потрібно, щоб очікування продовжилося до переможного кінця, який настане взагалі невідомо коли, то потрібно використовувати константу INFINITE. Нарешті, третьою функцією, необхідної тим, хто працює з семафорами, є функція ReleaseSemaphore, яка використовується для роботи з лічильником семафора. Вона виглядає наступним чином: BOOL ReleaseSemaphore (HANDLE hSemaphore, LONG lReleaseCount, LPLONG lpPreviousCount). Перший параметр - це дескриптор того самого семафора, з лічильником якого ми з вами будемо мати справу. Другий параметр - це число, на яке ми зменшуємо значення внутрішнього лічильника семафора за допомогою цієї функції. Ну а в третьому параметрі програма збереже попереднє значення лічильника. Третій параметр функції ReleaseSemaphore - це покажчик на цілу змінну. Потрібен він тому, що семафор не належить якомусь одному потоку.

Видаляються семафори з програми за допомогою функції CloseHandle. Вона має єдиний параметр - дескриптор того семафора, який більше нам з вами не знадобиться в цьому додатку.

## Семафори в мові C#

Рішення проблеми синхронізації в C# полягає в тому, щоб синхронізувати потоки і обмежити доступ до ресурсів на час їх використання яким-небудь потоком. Для цього використовується ключове слово lock. Оператор lock визначає блок коду, всередині якого весь код блокується і стає недоступним для інших потоків до завершення роботи поточного потоку. Приклад:

static object locker = new object();

public static void Count()

    {

        lock (locker)

        {

            x = 1;

            for (int i = 1; i < 9; i++)

            {

                Console.WriteLine("{0}: {1}", Thread.CurrentThread.Name, x);

                x++;

                Thread.Sleep(100);

            }

        }

    }

Для блокування з ключовим словом lock використовується об'єкт-заглушка, в даному випадку це змінна locker. Коли виконання доходить до оператора lock, об'єкт locker блокується, і на час його блокування доступ до блоку коду має тільки один потік. Після закінчення роботи блоку коду, об'єкт locker звільняється і стає доступним для інших потоків.

Механізм семафорів в мові C# реалізований за допомогою класу Semaphore. Клас Semaphore- обмежує число потоків, які можуть одночасно отримувати доступ до ресурсу або множини ресурсів.

Ієрархія наслідування:

[System.Object](https://msdn.microsoft.com/ru-ru/library/system.object(v=vs.110).aspx)   
  [System.MarshalByRefObject](https://msdn.microsoft.com/ru-ru/library/system.marshalbyrefobject(v=vs.110).aspx)  
    [System.Threading.WaitHandle](https://msdn.microsoft.com/ru-ru/library/system.threading.waithandle(v=vs.110).aspx)  
      System.Threading.Semaphore

Синтаксис:

[ComVisibleAttribute(false)]

[HostProtectionAttribute(SecurityAction.LinkDemand, Synchronization = true,

ExternalThreading = true)]

public sealed class Semaphore : WaitHandle

Використовуйте клас Semaphore для управління доступом до множини ресурсів. Потоки входять в семафор, викликаючи метод WaitOne, успадкований від класу WaitHandle, і звільняють семафор викликом методу Release.

Лічильник на семафорі зменшується на одиницю кожного разу, коли потік входить в семафор, і збільшується на одиницю, коли потік звільняє семафор. Коли лічильник дорівнює нулю, наступні запити блокуються, поки інші потоки не звільнять семафор. Коли семафор звільнений всіма потоками, лічильник має максимальне значення, задане при створенні семафора. Гарантований порядок, в якому б блоковані потоки входили в семафор, наприклад FIFO або LIFO, відсутні. Потік може виконувати вхід в семафор кілька разів, викликаючи багаторазово метод WaitOne. Щоб звільнити деякі з цих входів, потік може викликати перегружений метод Release () без параметрів кілька разів, або викликати перегружений метод Release (Int32), який вказує кількість звільняються входів.

Клас Semaphore не забезпечує потокової ідентифікації для викликів методів WaitOne або Release. Забезпечити, щоб потоки не звільняли семафор занадто багато разів - відповідальність програміста. Припустимо, що у семафора максимальне значення лічильника дорівнює двом, і два потоки A і B входять в семафор. Якщо помилка програмування в потоці B змушує його викликати метод Release двічі, обидва виклику закінчуються успішно. Лічильник на семафорі досяг максимального значення, і якщо потік A викличе метод Release, буде видано виключення SemaphoreFullException.

Семафори бувають двох типів: локальні семафори та іменовані системні семафори. При створенні об'єкта Semaphore за допомогою конструктора, що дозволяє передавати параметр з ім'ям семафора, об'єкт зв'язується з маючим дане ім'я семафором операційної системи. Іменовані системні семафори доступні в межах всієї операційної системи і можуть бути використані для синхронізації дій процесів. Можна створити кілька об'єктів Semaphore, що представляють один і той же іменований системний семафор, і використовувати метод OpenExisting для відкриття існуючого іменованого системного семафора. Локальний семафор існує тільки всередині одного процесу. Він може використовуватися будь-яким потоком в процесі, що має посилання на локальний об'єкт Semaphore. Кожен об'єкт Semaphore є окремим локальним семафором.

## Висновки до розділу 1 ШИРИНА????? ВЫРОВНЯТЬ!!!

1.Ада.У мові Ада механізм семафорів подано у вигляді пакета Synchronous\_Task\_Control в додатку Annex D: Real-Time Systems. Пакет реалізує механізм семафорів таким чином. Семафорний тип забезпечується приватним типом Suspension\_object, операції P(S) і V(S) реалізовані за допо­могою процедур Suspend\_Until\_True () і SetJTrue ().

2.Java.В Java механізм семафорів реалізований за допомогою моніторів. В Java в кожного екземпляра об'єкта є монітор. Реалізований він native-кодом і контролюється віртуальною машиною.

3. Win32.Середовище Win32 API містить ряд об'єктів, які застосовують для

синхронізації потоків - м'ютекси, критичні секції, семафори і події.

4. C#.Рішення проблеми синхронізації в C# полягає в тому, щоб синхронізувати потоки і обмежити доступ до ресурсів на час їх використання яким-небудь потоком. Для цього використовується ключове слово lock. Оператор lock визначає блок коду, всередині якого весь код блокується і стає недоступним для інших потоків до завершення роботи поточного потоку.

5. Порівняння. В мові програмування Ада механізм семафорів реалізований за допомогою змінних типу Suspension\_Object які набувають значень false і true, і декількох процедур, які відповідають за зміну цього значення на протилежний. Наприклад механізм семафорів в бібліотеці Win32 відрізняється тим, що там зміна типу HANDLE використовується в якості лічильника семафора і набуває значення не false і true,а значення цілої константи. Абсолютно інший механізм реалізації семафорів у мові Java, там він реалізований за допомогою методів які мають модифікатор synchronized. Використовується він так: будь-який нестатичний synchronized-метод при своєму виклику насамперед намагається захопити монітор того об'єкта, у якого він викликаний (на який він може послатися як на this). Якщо це вдалося - метод виконується. Якщо ні – потік блокується і чекає, поки монітор буде відпущений.

# РОЗДІЛ 2. РОЗРОБКА ПРОГРАМИ ПРГ1 ДЛЯ ПКС ЗП

ВСТУПЛЕНИИ И ЗАДАНИЕ!!!!

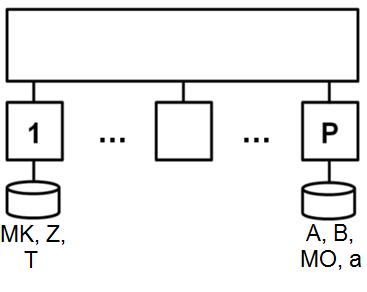


Рис. 2.1. Структура ПКС

Вихідна математична задача: A = B(MO\*MK)+a\*max(X)\*T

## Розробка паралельного математичного алгоритму

Відповідно до концепції необмеженого паралелізму:

* Кожна операція виконується за одиницю часу
* Час передачі даних між процесорами не враховується
* Застосовується будь-яка потрібна (необмежена) кількість процесорів

Вихідна математична задача вміщує в собі окремі дії, які не можуть бути виконані паралельно, тобто задача не є повністю паралельною.

При розробці паралельного алгоритму передбачається, що розмірність задачі (N) більше або дорівнює числу процесорів (P) (N> = P), причому N кратне P, що дозволяє розбити задачу на окремі частини Н, кожен з яких обробляється своїм завданням. Якщо N не кратне P, то останній процес буде обробляти частину даних, більшу ніж H.

H=N/P

Таким чином, отримано алгоритм виконання математичної задачі:

1. mi=max()



1. m=max(mi,m)
2. AH = B( \* MK)+a\*\*TH



Загальний ресурс: m, B, MK, a

## Розробка алгоритмів процесів

При розробці алгоритмів позначимо перший процес – T1, останній процес – TP, процес між першим і останнім під номером і (1<i<P) – Ti. Тоді алгоритми кожного процесу матимуть наступний вигляд

|  |  |  |
| --- | --- | --- |
| **T1** | | **Точки**  **синхронізації** |
| 1 | Введення MK, Z, T |  |
| 2 | Сигнал про введення даних для T2 - TP | S1 |
| 3 | Чекати сигнал про завершення введення даних в TP | W1 |
| 4 | Обрахувати: m1=max() |  |
| 5 | Обрахувати: m=max() | КС |
| 6 | Сигнал про готовність для T2 - TP | S2 |
| 7 | Чекати сигнал про готовність від T2 - TP | W2 |
| 8 | Копіювати: m1=m; B1=B; a1 = a; MK1=MK | КС |
| 9 | Обрахувати:AH = B1( \* MK1)+a1\*\*TH |  |
| 10 | Сигнал про готовність для TP | S2 |

|  |  |  |
| --- | --- | --- |
| **TI** | | **Точки**  **синхронізації** |
| 1 | Чекати сигнал про завершення введення даних в T1, TP | W1 |
| 2 | Обрахувати: mi=max() |  |
| 3 | Обрахувати: m=max() | КС |
| 4 | Сигнал про готовність для T1 - TP | S1 |
| 5 | Чекати сигнал про готовність від T1 - TP | W2 |
| 6 | Копіювати: mi=m; Bi=B; ai = a; MKi=MK | КС |
| 7 | Обрахувати:AH = Bi( \* MKi)+ai\*\*TH |  |
| 8 | Сигнал про готовність для TP | S2 |
|  |  |  |
| **TP** | | **Точки**  **синхронізації** |
| 1 | Введення B, MO, a |  |
| 2 | Сигнал про введення даних для T1 – TP-1 | S1 |
| 3 | Чекати сигнал про завершення введення даних в T1 | W1 |
| 4 | Обрахувати: mp=max() |  |
| 5 | Обрахувати: m=max() | КС |
| 6 | Сигнал про готовність для T1 – TP-1 | S2 |
| 7 | Чекати сигнал про готовність від T1 – TP-1 | W2 |
| 8 | Копіювати: mp=m; Bp=B; ap = a; MKp=MK | КС |
| 9 | Обрахувати:AH = Bp( \* MKp)+ap\*\*TH |  |
| 10 | Чекати сигнал про готовність для T1- TP-1 | W3 |
| 11 | Вивід А |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## 

## Розробка схеми взаємодії процесів

На рисунку 2.3 зображено розроблену схему взаємодії процесів

W1

S1

W1

S1

TP

TI

EnterP

Enter1

W1

Calcm

Calcm

Calcm

S2

S2

S1

W2

W2

T1

W2

Finish

S3

Finish

S2

W2

Рис 2.3 схема взаємодії процесів

m, B, a, MK – загальні ресурси

Копіювання виконується за допомогою моніторів:

**Mon\_a, Mon\_B, Mon\_m, Mon\_MK**

Для сигналів вводу використані семафори:

**EnterT1, EnterTP**

Для сигналу про завершення обчислень m використаний масив подій:

**Calcm**

Для сигналу про завершення обчислень використаний масив подій:

**Finish**

## Розробка програми ПРГ1

На основі розроблених алгоритмів задач та схеми взаємодії задач розроблено паралельну програму для вирішення заданої математичної задачі за допомогою бібліотеки Win32. Програму розроблено на мові програмування C++.

Програма складається з 2 класів:

**Data**– клас вміщує в собі необхідні для обчислень функції

**Program**– головний клас програми, містить в собі основну функцію main(), з якої починається робота програми та функції потоків.

Лістинг до ПРГ1 наведено в додатку А.

## Тестування програми ПРГ1

Для тестування використовувалась паралельна обчислювальна система з наступними апаратними характеристиками:

* процесор: Intel Core I5(4 ядра,4 потоки по 2.5 Ггц);
* оперативна пам'ять: DDR3 1333 МГц, 4000 МБ.

В якості програмного забезпечення виступали:

* операційна система: Microsoft Windows 7 x86 SP1 (Version 6.7.7601);
* середовище розробки і компіляції C++ програми: Microsoft Visual Studio 2013 (Version 10.0.30319.1 RTMRel).

Для вимірювання часу виконання програми використовувався високоточний таймер з бібліотеки Win32.

Тестування проводилось в автоматичному режимі за допомогою bat‑скрипта, який запускав програму із заданою розмірністю матриці на заданій кількості процесорів. Результати тестування записувались у текстовий файл. Код bat‑файлу приведений у додатку .

Для оцінки ефективності програми використовуються коефіцієнти прискорення та ефективності.

Коефіцієнт прискорення  показує скорочення часу виконання паралельної програми в паралельній системі з ** процесорами ** в порівнянні з часом виконання послідовної програми в однопроцесорній системі :



Коефіцієнт ефективності  застосування комп’ютерної системи показує ступінь використання ** процесорів системи:



Результати тестування і проведених досліджень ефективності розробленої програми наведено в таблицях ‑.

Таблиця . – Час виконання програми ПРГ1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N |  |  |  |  |
| 800 | 12448 | 5468 | 4952 | 3968 |
| 1600 | 65490 | 34136 | 29187 | 27500 |
| 2400 | 220068 | 124033 | 120308 | 99166 |

На основі даних із таблиці  виконано розрахунок значень коефіцієнтів прискорення, які наведені в таблиці .

Таблиця . – Коефіцієнти прискорення для програми ПРГ1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Кількість процесорів (P) | | | | |
| N | 1 | 2 | 3 | 4 |
| 800 | 1,00 | 2,00 | 2,51 | 3,13 |
| 1600 | 1,00 | 1,91 | 2,24 | 2,38 |
| 2400 | 1,00 | 1,77 | 1,83 | 2,21 ?????? |

Коефіцієнти ефективності (таблиця ) обчислено за даними таблиці .

Таблиця . – Коефіцієнти ефективності для програми ПРГ1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Кількість процесорів (P) | | | |
| N | 1 | 2 | 3 | 4 |
| 800 | 100,00% | 100,00% | 83,79% | 78,42% |
| 1600 | 100,00% | 95,92% | 74,79% | 72,53% |
| 2400 | 100,00% | 88,71% | 70,97% | 69,47% |

Використовуючи таблиці ‑ побудовано графіки зміни коефіцієнтів прискорення і ефективності в залежності від *N* і .



Рисунок . – Графік зміни коефіцієнту прискорення програми ПРГ1 в залежності від кількості ядер при N=800



Рисунок . – Графік зміни коефіцієнту прискорення програми ПРГ1 в залежності від кількості ядер при N=1600



Рисунок . – Графік зміни коефіцієнту прискорення програми ПРГ1 в залежності від кількості ядер при N=2400



Рис. 2.6. Графік зміни коефіцієнту ефективності програми ПРГ1 в залежності від кількості ядер при N=800



Рисунок . – Графік зміни коефіцієнту ефективності програми ПРГ1 в залежності від кількості ядер при N=1600



Рисунок 2.8 – Графік зміни коефіцієнту ефективності програми ПРГ1 в залежності від кількості ядер при N=2400

# РОЗДІЛ 3. РОЗРОБКА ПРОГРАМИ ПРГ2 ДЛЯ ПКС ЛП

ВСТУПЛЕНИЕ И ЗАДАНИЕ!!!!

Вихідна математична задача: A = B(MO\*MK)+a\*max(X)\*T

Ti1

Ti2

**……**

P

Рис. 3.1. Структура ПКС

## 3.1 Розробка паралельного математичного алгоритму ШРИФТ 14!!!!!!

Відповідно до концепції необмеженого паралелізму:

* Кожна операція виконується за одиницю часу
* Час передачі даних між процесорами не враховується
* Застосовується будь-яка потрібна (необмежена) кількість процесорів

Вихідна математична задача вміщує в собі окремі дії, які не можуть бути виконані паралельно, тобто задача не є повністю паралельною.

При розробці паралельного алгоритму передбачається, що розмірність задачі (N) більше або дорівнює числу процесорів (P) (N> = P), причому N кратне P, що дозволяє розбити задачу на окремі частини Н, кожен з яких обробляється своїм завданням. Якщо N не кратне P, то останній процес буде обробляти частину даних, більшу ніж H.

H=N/P

Таким чином, отримано алгоритм виконання математичної задачі:

1. mi=max()



1. m=max(mi,m)
2. AH = B\*( \* MK)+a\*\*TH



Загальний ресурс: m, B, MK, a

## 3.2 Розробка алгоритмів процесів

|  |  |
| --- | --- |
|  | Т0 |
| 1) | Введення MO, alpha, T,Z,B, MK |
| 2) | Пересилка MO2H, MK, alpha, T2H, Z,B у задачі Ti, i=(2, 4, ..,2\*P) |
| 3) | Обчислення m=max(ZH) |
| 4) | Отримання mі від задач Ti2 |
| 5) | Обчислення m=max(m, mi) |
| 6) | Пересилка m у задачі Ti2 |
| 7) | Обчислення AH=B(MOH\*MK)+a\*m\*TH |
| 8) | Отримання A2H від задач Ti, i=(2, 4,..,2\*P) |
| 9) | Вивід А |

|  |  |
| --- | --- |
|  | Ti2 |
| 1) | Отримання MO2H, MK, alpha, T2H, Z, B від задачі Т0 |
| 2) | Пересилка MOH, MK, alpha,T2H, Z, B у задачу Ti1 |
| 3) | Обчислення m=max(ZH) |
| 4) | Отримання mі від задачі Ti1 |
| 5) | Обчислення m=max(m, mi) |
| 6) | Пересилка а у задачу T0 |
| 7) | Отримання а від задачі Т0 |
| 8) | Пересилка m у задачу Ті1 |
| 9) | Обчислення AH=B(MOH\*MK)+a\*m\*TH |
| 10) | Отримання AH від задачі Ti1 |
| 11) | Пересилка A2H у задачу Т0 |

|  |  |
| --- | --- |
|  | Ti1 |
| 1) | Отримання MO2H, MK, alpha, T2H, Z, B від задачі Ті2 |
| 2) | Обчислення m=max(ZH) |
| 3) | Посилка z у задачу Ті2 |
| 4) | Отримання z від задачі Ті2 |
| 5) | Обчислення AH=B(MOH\*MK)+a\*m\*TH |
| 6) | Пересилка AH у задачу Ті2 |

**3.3 Розробка схеми взаємодії процесів**

**????????**

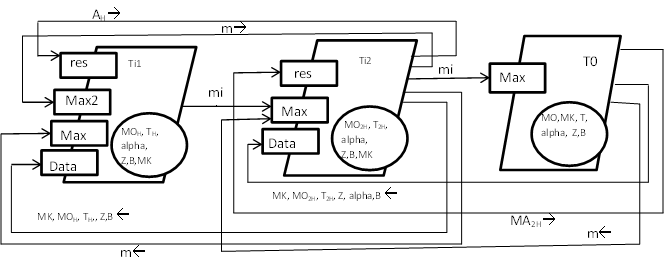


Рис. 3.3 Схема взаємодії процесів

У потоках Тi1 є захищені входи res, Max, Max2 та Data. Вхід res викликається потоками Тi2 в ньому MAH пересилається у потік, що викликав його. Вхід Max викликається потоками Тi2 в ньому потік, що викликав його, отримує а. Вхід Max2 викликається потоками Тi2 в ньому потік, що викликав його, пересилає а. Вхід Data викликається потоками Тi2 в ньому потік, що викликав його, пересилає MO, MBH, MKH, MXH, alpha, beta.

У потоках Тi2 є захищені входи res, Max та Data. Вхід res викликається потоком Т0, в ньому MA2H пересилається у потік Т0. Вхід Max викликається потоком Т0, в ньому Т0 пересилає а. Вхід Data викликається потоком Т0, який пересилає MO, MB2H, MK2H, MX2H, alpha, beta.

У потоці Т0 є захищений вхід Max, який викликається потоками Тi2, які пересилають аі.

**3.4 Розробка програми**

В результаті розробки алгоритмів задач та схеми їх взаємодії розроблено паралельну програму для вирішення заданої математичної задачі за допомогою засобів механізму рандеву в мові Ада[15]. Для пересилки даних використовуються захищені входи. Є три види потоків – Т0, Тi2 та Тi1. В залежності від номеру потоку для нього визначені входи, які пересилають або отримують дані. Потоки Ті2 та Ті1 створюються потоковими типами. Потоки Ті2 створюються у потоці Т0 та створюють в собі потоки Ті1.

Лістинг наведено в додатку Б.

3**.5 Тестування**

**Висновки до розділу**

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

* 1. Semaphore (programing) [Електронний ресурс] .— Режим доступу: http://ukrefs.com.ua/
  2. Семафор (синхронізація) [Електронний ресурс] .— Режим доступу: http://www.ccas.ru/paral/prog/semaphore.html
  3. Синхронізація потоків [Електронний ресурс] .— Режим доступу: <http://www.skipy.ru/technics/synchronization.html#monitors>
  4. Синхронізація потоков [Електронний ресурс] .— Режим доступу: <http://metanit.com/sharp/tutorial/11.4.php>
  5. Семафори [Електронний ресурс] .— Режим доступу:

http://life-prog.ru/view\_linux.php?id=23

* 1. Клас Семафор в c# [Електронний ресурс] .— Режим доступу: http://metanit.com/sharp/tutorial/11.8.php
  2. Семафор (синхронізація) [Електронний ресурс] .— Режим доступу: <http://forum.sources.ru/index.php?showtopic=348502>
  3. Java. Методи классу Object wait(), notify(), notifyAll() [Електронний ресурс] .— Режим доступу: <http://techarxiv.blogspot.com/2007/11/java-object-wait-notify-notifyall.html>
  4. Жуков І.А., Корочкін О.В. Паралельні та розподілені обчислення: Навч. посібник. – К.: Корнійчук, 2005. – 226 с.

# ДОДАТКИ

## Додаток А.

## Лістинг до ПРГ1

**Program**

#include "stdafx.h"

#include <stdio.h>

#include <time.h>

#include <windows.h>

#include <iostream>

#include "Data.cpp"

using namespace std;

int N = 4;

int P = 4;

int H;

int \*A, \*B, \*Z, \*T, \*temp\_v\_1, \*temp\_v\_2, \*temp\_v\_3, \*temp\_v\_4;

int a, m;

int \*\*MO, \*\*MK;

int \*\*temp\_m;

Data \*data;

void T1(void);

clock\_t start\_time;

clock\_t end\_time;

DWORD TI(LPVOID);

DWORD TP(LPVOID);

DWORD \*XWI;

HANDLE \*TasksI;

HANDLE Mut\_B, Mut\_m, Mut\_a, Mut\_MK;

HANDLE EnterT1, EnterTP;

HANDLE \*Calcm;

HANDLE \*Finish;

DWORD TI(LPVOID lpdwThreadParam){

int I = (const int)lpdwThreadParam;

printf("T%d - start\n",I+1);

if(I == P-1)

{

a = 1;

B = data->CreateVector();

MO = data->CreateMatrix();

SetEvent(EnterTP);

}

if(I == 0)

{

Z = data->CreateVector();

//Z[2] = 3;

T = data->CreateVector();

MK = data->CreateMatrix();

SetEvent(EnterT1); //S1

}

//wait enter

if(P>1)

{

if(I==0 || I==P-1)

{

if(I==0)

{

HANDLE endInputEvents[1] = { EnterTP };

WaitForMultipleObjects(1, endInputEvents, true, INFINITE); //W1

}

else

{

HANDLE endInputEvents[1] = { EnterT1 };

WaitForMultipleObjects(1, endInputEvents, true, INFINITE); //W1

}

}

else

{

HANDLE endInputEvents[2] = { EnterT1,EnterTP };

WaitForMultipleObjects(2, endInputEvents, true, INFINITE); //W1

}

}

int val\_s = I\*H;

int val\_e = (I != P - 1) ? (I + 1)\*H : N;

int mi = data->find\_max(Z, val\_s, val\_e);

//m val

WaitForSingleObject(Mut\_m, INFINITE);

m = (m<mi)?mi:m;

ReleaseMutex(Mut\_m);

//wait m

if (P>1)

{

SetEvent(Calcm[I]);

WaitForMultipleObjects(P , Calcm, true, INFINITE); //W2

}

//clone B

WaitForSingleObject(Mut\_B, INFINITE);

int\* BI = data->clone\_vector(B);

ReleaseMutex(Mut\_B);

//clone a

WaitForSingleObject(Mut\_a, INFINITE);

int aI = a;

ReleaseMutex(Mut\_a);

//clone m

WaitForSingleObject(Mut\_m, INFINITE);

int mI = m;

ReleaseMutex(Mut\_m);

//clone MK

WaitForSingleObject(Mut\_MK, INFINITE);

int\*\* MKI = data->clone\_matrix(MK);

ReleaseMutex(Mut\_MK);

temp\_m = data->MultMatrix(MKI, MO, temp\_m, val\_s, val\_e);

temp\_v\_1 = data->matrix\_vector\_mul(BI, temp\_m, temp\_v\_1, val\_s, val\_e);

temp\_v\_2 = data->vector\_int\_mul(T, mI, temp\_v\_2, val\_s, val\_e);

temp\_v\_3 = data->vector\_int\_mul(temp\_v\_2, aI, temp\_v\_3, val\_s, val\_e);

A = data->SumVector(temp\_v\_1, temp\_v\_3, A, val\_s, val\_e);

if(P>1)

{

SetEvent(Finish[I]);

if(I==P-1)

{

WaitForMultipleObjects(P-1, Finish, true, INFINITE); //W2

}

}

if(I==P-1)

{

data->OutputVector(A);

end\_time=clock();

int count\_clock = end\_time-start\_time;//ms

cout<<"Count clocks(miliseconds) work:'"<<count\_clock<<"'\n";

}

printf("T%d - finish\n",I+1);

return 0;

}

int \_tmain(int argc, \_TCHAR\* argv[])

{

while(true)

{

cout<<"Write P\n";

cin>>P;

cout<<"Write N\n";

cin>>N;

if(N>=P)

{

break;

}

}

start\_time=clock();

H=N/P;

XWI = new DWORD[P];

TasksI = new HANDLE[P];

Calcm = new HANDLE[P];

Finish = new HANDLE[P - 1];

data = new Data(N);

for (int j = 0; j<P - 1; j++)

{

Calcm[j] = CreateEvent(NULL, 1, 0, NULL);

}

for(int j=0;j<P-1;j++)

{

Finish[j] = CreateEvent(NULL, 1, 0, NULL);

}

A = data->CreateEmptyVector();

temp\_m = data->CreateEmptyMatrix();

temp\_v\_1 = data->CreateEmptyVector();

temp\_v\_2 = data->CreateEmptyVector();

temp\_v\_3 = data->CreateEmptyVector();

temp\_v\_4 = data->CreateEmptyVector();

Mut\_B = CreateMutex(NULL,0, NULL);

Mut\_m = CreateMutex(NULL,0,NULL);

Mut\_a = CreateMutex(NULL,0,NULL);

Mut\_MK = CreateMutex(NULL,0,NULL);

EnterT1 = CreateEvent(NULL, 1, 0, NULL);

EnterTP = CreateEvent(NULL, 1, 0, NULL);

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

{

const int temp\_i = i;

(TasksI[i]) = CreateThread(NULL, 1000000, (LPTHREAD\_START\_ROUTINE)TI, (LPVOID)(temp\_i), 0, &(XWI[i]));

}

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

{

CloseHandle(TasksI[i]);

}

int exit\_prog;

cin>>exit\_prog;

getchar();

return 0;

}

**Data**

#include "stdafx.h"

#include <windows.h>

#include <iostream>

using namespace std;

//#include "Data.cpp"

class Data

{

public:

int N;

int \*\*MA, \*\*MV, \*\*MM, \*\*temp;

int \*B;

Data::Data(int i)

{

N = i;

}

int\*\* CreateMatrix()

{

int \*\*result;

result=new int\*[N];

for(int i=0;i<N;++i)

{

result[i]=new int[N];

}

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

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

result[i][j] = 1;

}

}

return result;

}

int\* CreateEmptyVector()

{

int \*result = new int[N];

for(int i=0;i<N;++i)

{

result[i]=0;

}

return result;

}

int\*\* CreateEmptyMatrix()

{

int \*\*result;

result=new int\*[N];

for(int i=0;i<N;++i)

{

result[i]=new int[N];

}

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

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

result[i][j] = 0;

}

}

return result;

}

int CreateInt()

{

int result =1;

return result;

}

int\* CreateVector()

{

int\* result;

result = new int[N];

for (int j = 0; j < N; j++)

{

result[j] = 1;

}

return result;

}

/////////////////////////////////////////////////////////////////////////////////////////////////

int\*\* MultMatrix(int\*\* a, int\*\* b, int\*\* x, int from, int to)

{

int\*\* res = x;

for (int i = from; i < to; i++)

{

for (int j = 0; j < N; j++)

{

res[i][j] = 0;

}

}

for (int i = from; i < to; i++)

{

for (int j = 0; j < N; j++)

{

for (int k = 0; k < N; k++)

{

res[i][j] += a[i][k] \* b[k][j];

}

}

}

return res;

}

int\*\* MultMatrixInt(int\*\* a, int b, int\*\* x, int from, int to)

{

int\*\* res = x;

for (int i = from; i < to; i++)

{

for (int j = 0; j < N; j++)

{

res[i][j] = 0;}}

for (int i = from; i < to; i++)

{

for (int j = 0; j < N; j++)

{

res[i][j] += a[i][j] \* b;

}

}

return res;

}

void OutputMatrix(int\*\* vector)

{

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

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

printf("%d ",vector[i][j]);

}

printf("\n");

}

printf("\n");

}

void OutputVector(int\* vector)

{

if(N<10)

{

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

printf("%d ",vector[i]);

}

}

printf("\n");

}

int\*\* SumMatrix(int\*\* A, int\*\* B, int\*\* C, int from, int to)

{

int\*\* res = C;

for (int i = from; i < to; i++)

{

for (int j = 0; j < N; j++)

{

res[i][j] = 0;}}

for (int i = from; i < to; i++)

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

{

res[i][j] = A[i][j] + B[i][j];

}

}

return res;

}

int\* SumVector(int\* A, int\* B, int\* res,int from, int to)

{

for (int i = from; i < to; i++)

{

res[i] = A[i] + B[i];

}

return res;

}

int\* matrix\_vector\_mul(int\* x, int\*\* y,int\* c, int from, int to)

{

int\* res = c;

for(int i=from;i<to;i++)

{

res[i]=0;

}

for (int i = from; i < to; i++)

{

for (int j = 0; j < N; j++)

{

res[i] += x[j] \* y[i][j];

}

}

return res;

}

int\* vector\_int\_mul(int\* x, int a,int\* res, int from, int to)

{

for (int i = from; i < to; i++)

{

res[i] = 0;

}

for (int i = from; i < to; i++)

{

res[i] = x[i] \* a;

}

return res;

}

int\* vector\_vector\_mul(int\* x, int\* y,int\* res, int from, int to){

for (int i = from; i < to; i++)

{

res[i] = 0;

}

for (int i = from; i < to; i++)

{

for (int j = 0; j < N; j++)

{

res[i] += x[j] \* y[i];

}

}

return res;

}

int find\_max(int\* x, int from, int to){

int a=x[from];

for(int i=from+1; i<to; i++){

if(x[i]>a){

a=x[i];

}

}

return a;

}

int set\_max(int a, int a1){

if(a1>a) return a1;

else return a;

}

int\*\* clone\_matrix(int \*\*A){

int \*\*res = new int\* [N];

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

res[i] = new int[N];

}

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

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

res[i][j] = A[i][j];

}

}

return res;

}

int\* clone\_vector(int \*A){

int \*res = new int [N];

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

res[i] = A[i];

}

return res;

}

int clone\_int(int a){

int res = a;

return res;

}

};

**Додаток Б.**

**Лістинг до ПРГ2**

**Program**

GNAT GPL 2014 (20140331)

Copyright 1992-2014, Free Software Foundation, Inc.

Compiling: program.adb (source file time stamp: 2015-04-22 18:45:02)

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

2. -- Ada.Randevu

3. -- Task A = B(MO\*MK)+a\*Z

4. -- Rozkydany

5. -- Group IO-24

6. -- 04.04.2015

7. -----------------------------------------------------

8. with Data;

9.

10. with ada.text\_IO; use ada.text\_IO;

11. with ada.integer\_text\_IO; use ada.integer\_text\_IO;

12. with Ada.Synchronous\_Task\_Control; use Ada.Synchronous\_Task\_Control;

13. with Ada.Calendar; use Ada.Calendar;

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

15.

16. procedure Lab1 is

17.

18. N: Integer;

19. P: Integer;

20. H: Integer;

21. C: Integer;

22. last\_H: Integer;

23.

24. procedure Start is

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

26. --25

27. ----------------------------------------------------------------------

28. ----------------------------------------------------------------------

29. task T\_1 is

30. entry RRecv(Bi:in Vector;Ei:in Vector;MUi:in Matrix);

31. entry DRecv(Di:in vector);

32. entry ResRecv(Ai:in Vector);

33. entry start;

34. end T\_1;

35.

36. Task T\_P is

37. entry LRecv(alphai:in integer;MTi:in matrix;MKi:in matrix;MOi:in matrix);

38. entry DRecv(Di:in vector);

39. entry start;

40. end T\_P;

41.

42. task Ti is

43. entry RRecv(Bi:in Vector;Ei:in Vector;MUi:in Matrix);

44. entry LRecv(alphai:in integer;MTi:in matrix;MKi:in matrix;MOi:in matrix);

45. entry DN1Recv(Di:in Vector);

46. entry DNPRecv(Di:in Vector);

47. entry ResRecv(Ai:in Vector);

48. entry start;

49. end Ti;

50. --46

51.

52. type T\_Ptr is access Ti;

53. type T1\_Ptr is access T\_1;

54. type TP\_Ptr is access T\_P;

55. type T\_Ptr\_Array is array (1..P) of T\_Ptr;

56. tasks: T\_Ptr\_Array;

57. T1: T1\_Ptr;

58. TP: TP\_Ptr;

59.

60. ----------------------------------------------------------------

61.

62. task body T\_1 is

63. a\_h:Vector(1..H);

64. a:Vector(1..n);

65. b:Vector(1..n);

66.

67. D\_part:vector(1..h);

68. D:vector(1..n);

69.

70. MK:matrix(1..n,1..n);

71. MK\_part:Matrix(1..n,1..h);

72. MK\_send:matrix(1..n,1..n-h);

73.

74. MO:matrix(1..n,1..n);

75. MO\_part:Matrix(1..n,1..h);

76. MO\_send:matrix(1..n,1..n-h);

77.

78. MT:matrix(1..n,1..n);

79. MU\_part:matrix(1..n,1..n);

80. alpha1:integer;

81. temp1:Vector(1..h);

82. temp2:matrix(1..n,1..h);

83. temp3:vector(1..h);

84. E:vector(1..n);

85. begin

86. accept start;

87. Put\_line("T1 start");

88. MK:=fillMatrix(1);

89. MO:=FillMatrix(1);

90. MT:=fillMatrix(1);

91. alpha1:=1;

92.

93. MK\_part:=GetMatrixPart(MK,1,h);

94. MK\_send:=GetMatrixPart(MK,h+1,n-h);

95. MO\_part:=getMatrixPart(MO,1,h);

96. MO\_send:=getMatrixPart(MO,h+1,n-h);

97.

98. --83 --Send to T2

99. tasks(2).LRecv(alpha1,MT,MK\_send,MO\_send);

100.

101.

102. accept RRecv(Bi:in Vector,Ei:in Vector,MUi:in matrix) do

|

>>> "," should be ";"

103. b:=Bi;

104. E:=ei;

105. MU\_part:=GetMatrixPart(MUi,1,h);

106. end;

107.

108. D\_part:=vectorMultiplicationOnMatrix(b,MK\_part);

109. accept DN1Recv(Di:in vector)do

110. d:=concatVector(D\_part,di);

111. end;

112.

113. tasks(2).DRecv(d);

114.

115.

116. A\_h:=addVector(vectorMultiplicationOnMatrix(d,MO\_part), scalarMultiplicationOnVector(alpha1,vectorMultiplicationOnMatri(e,MatrixMultiplication(MT,MU\_part))));

117.

118.

119. --A\_h:=addVector(vectorMultiplicationOnMatrix(B,MatrixMultiplication(MO,MK\_part)),

120. --scalarMultiplicationOnVector(alpha1,Z\_part));

121. --Vector\_output(A\_h);

122.

123. accept ResRecv(ai:in Vector) do

124. a:=concatVector(a\_h,ai);

125. end;

126.

127. Vector\_output(a);

128.

129. Put\_Line("T1 finish");

130. end T\_1;

131. --111

132. task body Ti is

133. A\_h:vector(1..h);

134. A\_send:Vector(1..n-(h\*(id-1)));

135.

136. b:vector(1..n);

137. E:vector(1..n);

138. D\_part:vector(1..h);

139.

140. D\_send:vector(1..n-h\*(id-1));

141. D:vector(1..n);

142.

143.

144. MK\_part:Matrix(1..n,1..h);

145. MK\_send:matrix(1..n,1..n-h\*id);

146.

147. MO\_part:Matrix(1..n,1..h);

148. MO\_send:matrix(1..n,1..n-h\*id);

149.

150. MT:matrix(1..n,1..n);

151.

152. MU\_part:matrix(1..n,1..h);

153. MU\_send:matrix(1..n,1..h\*(id-1));

154. alpha:integer;

155.

156. begin

157. accept Start;

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

159.

160. --if number < center

161. if(id<c)then

162. --get data from prev

163. accept LRecv(alphai:in integer;MTi:in matrix;MKi:in matrix;MOi:in matrix)do

164. alpha:=alphai;

165. MT:=MTi;

166. MK\_part:=getMatrixPart(mKi,1,h);

167. MK\_send:=GetMatrixPart(MKi,h+1,n-h\*id);

168. MO\_part:=GetMatrixPart(MOi,1,h);

169. MO\_send:=GetMatrixPart(MOi,h+1,n-h\*id);

170. end;

171. --send data to next

172. if(id=(p-1))then

173. tP.LRecv(alpha,MT,MK\_send,MO\_send);

174. else

175. tasks(id+1).LRecv(alpha,MT,MK\_send,MO\_send);

176. end if;

177.

178. --get data from next

179. accept RRecv(bi:in vector;ei:in vector,MUi:in matrix)do

|

>>> "," should be ";"

180. b:=Bi;

181. e:=ei;

182. MU\_part:=getMatrixPart(MUi,h\*(id-1)+1,h);

183. MU\_send:=getMatrixPart(MUi,1,h\*(id-1));

184. end;

185. --send data from next to prev

186. if(id=2)then

187. T1.RRecv(b,e,MU\_send);

188. else

189. tasks(id-1).RRecv(b,e,MU\_send);

190. end if;

191.

192. else

193. --if number is center

194. if(id=c)then

195. --get data from next

196. accept RRecv(bi:in vector;Ei:in vector;MUi:in matrix)do

197. B:=Bi;

198. e:=ei;

199. MU\_part:=getMatrixPart(MUi,h\*(id-1)+1,h);

200. MU\_send:=GetMatrixPart(MUi,1,H\*(id-1));

201. end;

202. --get data from prev

203. accept LRecv(alphai:in integer;MTi:in matrix;MKi:in matrix;MOi:in matrix)do

204. alpha:=alphai;

205. MT:=MTi;

206. MK\_part:=getMatrixPart(MKi,1,h);

207. MK\_send:=getMatrixPart(MKi,h+1,n-h\*id);

208. MO\_part:=getMatrixPart(MOi,1,h);

209. MO\_send:=GetMatrixPart(MOi,h+1,n-h\*id);

210. end;

211.

212. --send data from prev to next

213. if(id=P-1)then

214. TP.LRecv(alpha,MT,MK\_send,MO\_send);

215. else

216. tasks(id+1).LRecv(alpha,MT,MK\_send,MO\_send);

217. end if;

218. --send data from next to prev

219. if(id=2)then

220. T1.RRecv(b,e,MU\_send);

221. else

222. tasks(id-1).RRecv(b,e,MU\_send);

223.

224.

225. --if number > center

226. else

|

>>> only one "else" part allowed

227. --get data from next

228. accept RRecv(bi:in vector;ei:in vector;MUi:in matrix)do

229. b:=bi;

230. e:=ei;

231. MU\_part:=getMatrixPart(mUi,h\*(id-1)+1,h);

232. MU\_send:=getMatrixPart(MUi,1,h\*(id-1));

233. end;

234. --send to prev

235. if(id=2)then

236. T1.RRecv(b,e,MU\_send);

237. else

238. tasks(id-1).RRecv(b,e,MU\_send);

239. end if;

240.

241. --get data from prev

242. accept LRecv(alphai:in integer;MTi:in matrix;MKi:in matrix; MOi:in matrix)do

243. alpha:=alphai;

244. MT:=MTi;

245. MK\_part:=getMatrixPart(MKi,1,h);

246. MK\_send:=GetMatrixPart(MKi,h+1,n-h\*id);

247. MO\_part:=GetMatrixPart(MOi,1,h);

248. MO\_send:=getMatrixPart(MOi,h+1,n-h\*id);

249. end;

250. --send to next

251. if(id=p-1)then

252. TP.LRecv(alpha,MT,MK\_send,MO\_send);

253. else

254. tasks(id+1).LRecv(alpha,MT,MK\_send,MO\_send);

255. end if;

256.

257.

258. end if;

259. end if;

260. --count Dn

261. D\_part:=vectorMultiplicationOnMatrix(b,MK\_part);

262. accept DN1Recv(Di:in vector)do

263. D\_send:=concatVector(D\_part,di);

264. end;

265. --send Dn

266. if(id=2)then

267. T1.DN1Recv(D\_send);

268. else

269. tasks(id-1).DN1Recv(D\_send);

270. end if;

271.

272. accept DRecv(Di:in vector)do

273. D:=di;

274. end;

275.

276. A\_h:=addVector(vectorMultiplicationOnMatrix(d,MO\_part), scalarMultiplicationOnVector(alpha1,vectorMultiplicationOnMatri(e,MatrixMultiplication(MT,MU\_part))));

277. accept resRecv(ai:in vector)do

278. A\_send:=concatVector(A\_h,Ai);

279. end;

280.

281. --send result

282. if(id=2)then

283. T1.ResRecv(A\_send);

284. else

285. tasks(id-1).ResRecv(A\_send);

286. end if;

287.

288.

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

290. end ti;

|

>>> missing "end if;" for "if" at line 161

291. ---------------------------------------------------------

292. task body T\_P is

293. A\_h:vector(1..h);

294.

295. b:vector(1..n);

296. E:vector(1..n);

297. D\_part:vector(1..h);

298. D:vector(1..n);

299. MK\_part:Matrix(1..n,1..h);

300. MO\_part:Matrix(1..n,1..h);

301. MT:matrix(1..n,1..n);

302.

303. MU:matrix(1..n,1..n);

304. MU\_part:matrix(1..n,1..h);

305. MU\_send:matrix(1..n,1..n-lastH);

306. alpha:integer;

307. begin

308. accept Start;

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

310. MU:=fillMatrix(1);

311. e:=fillVector(1);

312. B:=fillVector(1);

313.

314. MU\_part:=getMatrixPart(MU,h\*(id-1)+1,lastH);

315. MU\_send:=getMatrixPart(MU,1,n-lastH);

316.

317. --send data

318. tasks(p-1).RRecv(b,e,MU\_send);

319. --get data

320. accept LRecv(alphai:in integer;MTi:in matrix;MKi:in matrix;MOi:in matrix)do

321. alpha:=alphai;

322. MT:=MTi;

323. MK\_part:=getMatrixPart(MKi,1,lastH);

324. MO\_part:=GetMatrixPart(MOi,1,lastH);

325. end;

326.

327. D\_part:=vectorMultiplicationOnMatrix(b,MK\_part);

328. tasks(p-1).DN1Recv(D\_part);

329.

330. accept DRecv(Di:in vector)do

331. d:=di;

332. end;

333.

334. A\_h:=addVector(vectorMultiplicationOnMatrix(d,MO\_part), scalarMultiplicationOnVector(alpha1,vectorMultiplicationOnMatri(e,MatrixMultiplication(MT,MU\_part))));

335.

336. tasks(p-1).ResRecv(A\_h);

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

338. end T\_P;

339.

340.

341. begin

342. --create tasks

343. for i in 2..(p-1) loop

344. tasks(i):=new T(id=>i);

345. end loop;

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

347. TP:=new T\_P(p);

348. --start tasks

349. t1.start;

350. TP.start;

351. for i in 2..(p-1)loop

352. tasks(i).start;

353. end loop;

354.

355. end Start;

356.

357. begin

358.

359. Put\_Line("Input N:");

360. Ada.Integer\_Text\_IO.Get(N);

361. Put\_Line("Input P:");

362. Ada.Integer\_Text\_IO.Get(P);

363. if (P>2) then

364. H := N/P;

365. C :=P/2+1;

366. last\_H := N-(P-1)\*H;

367. Start;

368. else

369. Put\_Line("Error! P must be more than 2.");

370. end if;

371.

372. end Lab1;

372 lines: 4 errors

**Data**

GNAT GPL 2014 (20140331)

Copyright 1992-2014, Free Software Foundation, Inc.

Compiling: data.ads (source file time stamp: 2015-04-22 18:45:02)

1. generic

2. N: Positive := 3;

3.

4. package Data is

5.

6. type vector is array(Positive range <>) of integer;

7. type matrix is array(Positive range <>, Positive range <>) of integer;

8. procedure vector\_output(x: vector);

9.

10.

11. function fillMatrix(number: in Integer) return Matrix;

12. function fillVector(number: in Integer) return Vector;

13. function getMatrixPart(MM: in Matrix; start: in Integer;size: in Integer) return Matrix;

14. function getVectorPart(V: in Vector; start: in Integer; size: in Integer) return Vector;

15.

16. function matrixMultiplication(m1 : in Matrix; m2 : in Matrix) return Matrix;

17. function vectorMultiplicationOnMatrix(v: in Vector; m: in Matrix) return Vector;

18. function scalarMultiplicationOnVector(s: in Integer; v: in Vector) return Vector;

19.

20. function addVector(v1: in Vector; v2: in Vector) return Vector;

21.

22. function concatVector(V1: Vector; V2: Vector) return Vector;

23.

24.

25. end Data;

26.

27. with ada.text\_IO; use ada.text\_iO;

|

>>> end of file expected, file can have only one compilation unit

28. WITH Ada.integer\_Text\_IO;

29. use Ada.Integer\_Text\_IO;

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

31.

32.

33. package body Data is

34.

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

36. size: in Integer) return Vector is

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

38. begin

39. for i in 1..size loop

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

41. end loop;

42. return R;

43. end;

44.

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

46. size: in Integer) return Matrix is

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

48. begin

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

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

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

52. end loop;

53. end loop;

54. return MR;

55. end;

56.

57. ------------------------------------------------------

58. -- fill matrix by one number

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

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

61. begin

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

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

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

65. end loop;

66. end loop;

67. return M;

68. end fillMatrix;

69.

70. -- fill vector by one number

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

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

73. begin

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

75. V(i):=number;

76. end loop;

77. return V;

78. end fillVector;

79.

80.

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

82. in Matrix ) return Matrix is

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

84. begin

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

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

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

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

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

90. end loop;

91. end loop;

92. end loop;

93. return MR;

94. end matrixMultiplication;

95.

96.

97. function vectorMultiplicationOnMatrix(v: in Vector;

98. m: in Matrix) return Vector is

99. R : Vector(1..m'Length(2));

100. begin

101. for i in 1..m'Length(2) loop

102. R(i) := 0 ;

103. for k in 1..N loop

104. R(i) := R(i) + v(k)\*m(k, i);

105. end loop;

106. end loop;

107. return R;

108. end vectorMultiplicationOnMatrix;

109.

110. -- multiplication scalar on matrix

111. function scalarMultiplicationOnVector(s: in Integer;

112. v: in Vector) return Vector is

113. R: Vector(1..v'Length);

114. begin

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

116. R(i) := v(i)\*s;

117. end loop;

118. return R;

119. end scalarMultiplicationOnVector;

120.

121. function addVector(v1: in Vector; v2: in Vector) return Vector is

122. R: Vector(1..v1'Length(1));

123. begin

124. for i in 1..v1'Length(1) loop

125. R(i):=v1(i)+v2(i);

126. end loop;

127. return R;

128. end addVector;

129.

130. function concatVector(V1: Vector; V2: Vector) return Vector is

131. R: Vector(1..V1'Length(1)+V2'Length(1));

132. begin

133.

134. for i in 1..V1'Length(1) loop

135. R(i):=V1(i);

136. end loop;

137.

138. for i in 1..V2'Length(1) loop

139. R(I+V1'Length(1)):=V2(i);

140. end loop;

141. return R;

142. end concatVector;

143.

144. ---------------------------------------------------------

145. procedure vector\_output(x: in vector) is

146. begin

147. if(x'length<=8) then

148. for i in 1..x'length loop

149. Ada.integer\_Text\_IO.put(x(i));

150. end loop;

151. else

152. new\_line;

153. Ada.integer\_Text\_IO.put(x(1));

154. end if;

155. new\_line(2);

156.

157. end vector\_output;

158.

159.

160.

161.

162.

163.

164. end Data;

164 lines: 1 error