НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

КИЇВСЬКИЙ ПОЛІТЕХНИЧНИЙ ІНСТИТУТ

ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

Кафедра обчислювальної техніки

К У Р С О В А Р О Б О Т А

*з дисципліни:* «Паралельні та розподілені обчислення»

*на тему:* «Розробка програмного забезпечення для

паралельних комп’ютерних систем»

Виконала: Косейкіна Г.С.

Група: ІО- 91

Перевірив: \_\_\_\_\_\_\_\_\_\_\_\_\_ Корочкін О.В.

Захищено з оцінкою\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_2012 р.

Київ - 2012 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

КИЇВСЬКИЙ ПОЛІТЕХНИЧНИЙ ІНСТИТУТ

ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

Кафедра обчислювальної техніки

***ЗАВДАННЯ НА КУРСОВУ РОБОТУ***

*з дисципліни:* «Паралельні та розподілені обчислення»

*на тему:* «Розробка програмного забезпечення для

паралельних комп’ютерних систем»

Дата отримання 28.03.2012р.

Термін виконання 10.05.2012р.

Керівник роботи доц.. Корочкін О.В.

ЗАВДАННЯ

1.Реалізація семафорів у різних мовах та бібліотеках

2.Розробка програмного забезпечення для масштабованих паралельних комп’ютерних систем. Мови та бібліотеки паралельного програмування Win32 та Ада. Засоби організації взаємодії процесів: семафори, мютекси, критичні секції, події, механізм посилки повідомлень. Математична задача: MА = α\*MB+β\*MO\*MX\*max(MK).

3. Тестування програмного забезпечення в багатоядерної комп’ютерної системі

ПЕРЕЛІК ГРАФИЧНОЇ ДОКУМЕНТАЦІЇ

1. Схема алгоритму основної програми для ПКС з ЗП

2. Схеми алгоритмів процесів для ПКС з ЗП

3. Схема алгоритму основної програми для ПКС ЛП

4. Схеми алгоритмів процесів для ПКС ЛП

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Жуков І., Корочікн О. Паралельні та розподілені обчислення. – Київ,

Корнійчук, 2005. – 226 с.

2. Корочкин А.В. Ада 95: Введение в программирование. - Киев; Свит, .

1998. - 260 с.

ПІДПИС КЕРІВНИКА РОБОТИ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Технічне завдання на курсову роботу**

1 Область застосування

1. Розроблюване програмне забезпечення може бути застосовано в галузі паралельних математичних обчислень для масштабованих комп'ютерних систем з загальної та локальної пам'яттю.

2 Підстави для розробки

Підставою для розробки служить ТЗ на курсову роботу.

3 Мета курсової роботи

Метою даної роботи є закріплення отриманих знань з дисципліни "Паралельні та розподілені обчислення ", а також отримання навичок і досвіду в розробці програмного забезпечення для комп'ютерних систем з різною структурною організацією.

4 Призначення роботи

Розробка програмного забезпечення (ПЗ) для ПКС з загальною та локальною пам'яттю. Проведення досліджень ефективності ПЗ.

5 Вихідні дані та виконання роботи

5.1 Завдання для розділу 1: Реалізація семафорів у різних мовах та бібліотеках.

5.2 Завдання для розділів 2 і 3: Математична задача: MА = =α\*MB+β\*MO\*MX\*max(MK). Розглядаються матриці розмірності N.

5.3 Структура паралельної комп'ютерної системи з загальною пам'яттю (рис. 1).

5. 4 Структура паралельної комп'ютерної системи з локальною пам'яттю (рис 2).

Загальна пам’ять

…

P

1

MX, MK, β

MB, MO, α, MA

Рис. 1. Структура ПКС з ЗП

1

1a

1b

…

0

Pa

Pb

P

MB, MO, α, MX, MK, β, MA

Рис.2. Структура ПКС ЛП

5.5 Мови та бібліотеки програмування та засоби організації взаємодії процесів: мова C++ і Win32 для розділу 2, Ада і механізм рандеву - для розділу 3.

У КР необхідно:

Розробити алгоритм розв'язання заданого математичної задачі і досліджувати паралельний властивості завдання в рамках концепції необмеженого паралелізму для N і Р, де N - розмірність матриць, Р - кількість процесорів.

Виконати розробку програми вирішення заданого математичного виразу для заданих структур ПКС на заданих язиках (бібліотеках) програмування. Описати структуру взаємодії завдань.

Виконати дослідження ефективності ПЗ завдання в залежності від значень Р і N. При цьому підрахувати часи виконання математичної задачі на різній кількості процесорів. Визначити значення коефіцієнтів прискорення (Кп) та ефективності (Ке).

6 Вимоги до складу і параметрів технічних засобів

Тестування ПЗ виконується з використання багатоядерної ПКС

7 Вимоги до інформаційної та програмної сумісності

Програма повинна працювати під управлінням ОС Windows.

8 Спеціальні вимоги

Тестування виконується на 4-х ядерної системі

9 Етапи і стадії розробки

- Узгодження технічного завдання

- Виконання курсової роботи.

- Захист курсової роботи.

Зміст

[ВСТУП 7](#_Toc324387382)

[РОЗДІЛ 1 Реалізація семафорів 8](#_Toc324387383)

[1.1 Реалізація семафорів у мові Ада 8](#_Toc324387384)

[1.2 Реалізація семафорів у мові Java 11](#_Toc324387385)

[1.3 Реалізація семафорів в мові C# 17](#_Toc324387386)

[1.4 Реалізація семафорів у бібліотеці Win32 22](#_Toc324387387)

[Висновки до розділу 1: 26](#_Toc324387388)

[РОЗДІЛ 2 Розробка програмного забезпечення для ПКС з ЗП 28](#_Toc324387389)

[2.1 Розробка програми 28](#_Toc324387390)

[2.2 Проведення досліджень ефективності розробленої програми в реальній 4-х ядерній системі 31](#_Toc324387391)

[Висновки до розділу 2: 33](#_Toc324387392)

[РОЗДІЛ 3 Розробка програмного забезпечення для ПКС з ЛП 35](#_Toc324387393)

[3.1 Розробка програми 35](#_Toc324387394)

[3.2 Проведення досліджень ефективності розробленої програми в реальній 4-х ядерній системі 38](#_Toc324387395)

[Висновки до розділу 3: 39](#_Toc324387396)

[ВИСНОВКИ 41](#_Toc324387397)

[СКОРОЧЕННЯ 42](#_Toc324387398)

[ЛІТЕРАТУРА 43](#_Toc324387399)

[Додаток А. Лістинг програми для ПКС з ЗП. 44](#_Toc324387400)

[Додаток Б. Лістинг програми для ПКС з ЛП 50](#_Toc324387401)

[Додаток В. Алгоритм головної програми для ПКС з ЗП 59](#_Toc324387402)

[Додаток Г. Алгоритм процесів для ПКС з ЗП 61](#_Toc324387403)

[Додаток Ґ. Алгоритм головної програми для ПКС з ЛП 63](#_Toc324387404)

[Додаток Д. Алгоритм процесів для ПКС з ЛП 65](#_Toc324387405)

# ВСТУП

В даній курсовій роботі в першому розділі оглянуто різноманітні реалізації семафорів – спеціальних механізмів для вирішення задачі взаємного виключення та задачі синхронізації. Описані реалізації в мовах Ада, Java, C# та у бібліотеці Win32. Порівняно їх недоліки та переваги.

В другому розділі розроблено програму для обчислення математичної задачі у паралельній комп’ютерній системі з загальною пам’яттю. Описані етапи розробки програми та наведено сирцевий код, написаний за допомогою засобів бібліотеки Win32. Проведено тестування отриманого програмного продукту та зроблені відповідні висновки.

В третьому розділі розроблено програму для обчислення математичної задачі у паралельній комп’ютерній системі з локальною пам’яттю. Описані етапи розробки програми та наведено сирцевий код, написаний за допомогою механізму рандеву мови Ада. Проведено тестування отриманого програмного продукту та зроблені відповідні висновки.

Після цих розділів зроблені загальні висновки по курсовій роботі.

Наведено список використаних скорочень та список використаної літератури.

Наведені алгоритми основних програм та окремих потоків, лістинги програм.

# РОЗДІЛ 1 РЕАЛІЗАЦІЯ СЕМАФОРІВ

Семафор – це універсальний механізм для організації взаємодії процесів[8]. Семафор вирішує задачу взаємного виключення та задачу синхронізації. Зазвичай семафор реалізовано спеціальним захищеним типом. Захищеність означає, що заборонено усі операції над семафорами, окрім трьох – створення семафору, P(S) та V(S), де S – семафор. P(S) – реалізація входу в критичну ділянку у задачі взаємного виключення або очікування сигналу в задачі синхронізації[1]. В цій операції перевіряється значення семафору, якщо S=0, то процес блокується, інакше виконує дію S=S-1 та входить у критичну ділянку або продовжує виконання. V(S)- реалізація виходу з критичної ділянки у задачі взаємного виключення або відправлення сигналу в задачі синхронізації. Ця операція виконує дію S=S+1, що є виходом з критичної ділянки або відправкою сигналу[11]. Семафори реалізовано у мовах Ада, Java, C#, у бібліотеці Win32[1].

# 1.1 Реалізація семафорів у мові Ада

У мові Ада для семафорів створено спеціальний тип Suspension\_object, який є лімітованим захищеним типом. Для використання цього типу необхідно підключити пакет Ada.Synchronous\_Task\_Control[7]. У цьому пакеті визначені такі операції:

- procedure Suspend\_until\_true (S: in out Suspension\_object), що є реалізацією операції P(S);

- procedure Set\_true (S: in out Suspension\_object), що є реалізацією операції V(S);

- procedure Set\_false (S: in out Suspension\_object), що є додатковою операцією та встановлює семафору значення false;

- function Current\_state (S: Suspension\_object) return Boolean, що є додатковою операцією та перевіряє значення семафору[6].

Семафор у мові Ада може приймати два значення – true та false. При створенні семафору йому за замовчанням присвоюється значення false[12].

Приклади використання семафорів у мові Ада.

--Задача взаємного виключення

with Ada.Synchronous\_Task\_Control;

use Ada.Synchronous\_Task\_Control;

with Ada.TEXT\_IO; use Ada.TEXT\_IO;

procedure Cursova1 is

buf : integer; --загальний ресурс

--семафор

S : Suspension\_Object; --false

--задачі

procedure Start is

task T1;

task body T1 is

begin

put ("T1 started");

--якісь дії задачі, які вона виконує паралельно

Suspend\_Until\_True(S);

buf := buf+100; --крітична ділянка

Set\_True(S);

--інші паралельні дії

put ("T1 finished");

end T1;

task T2;

task body T2 is

begin

put ("T2 started);

--якісь дії задачі, які вона виконує паралельно

Suspend\_Until\_True(S);

buf := buf\*78-126; --крітична ділянка

Set\_True(S);

--інші паралельні дії

put ("T2 finished");

end T2;

begin

null;

end Start;

begin --main

Set\_True(S); --оскільки спочатку ресурс вільний

Start;

end Cursova1;

--Задача синхронізації

with Ada.Synchronous\_Task\_Control;

use Ada.Synchronous\_Task\_Control;

with Ada.TEXT\_IO; use Ada.TEXT\_IO;

procedure Cursova2 is

--семафор

S : Suspension\_Object; --false

--задачі

procedure Start is

task T1;

task body T1 is

begin

put ("T1 started");

--якісь дії задачі, які вона виконує паралельно

Set\_True(S); --сигнал від задачі Т1 до Т2

--інші паралельні дії

put ("T1 finished");

end T1;

task T2;

task body T2 is

begin

put ("T2 started");

--якісь дії задачі, які вона виконує паралельно

Suspend\_Until\_True(S); --очікування сигналу з Т1

--інші паралельні дії

put ("T2 finished");

end T2;

begin

null;

end Start;

begin --main

Start;

end Cursova2;

# 1.2 Реалізація семафорів у мові Java

В мові Java семафори реалізовано класом Semaphore пакету java.util.concurent.\*. Цей клас має два конструктори – Semaphore (int X), де X – початкове значення семафору та Semaphore (Boolean X), де X – поведінка P(S) та V(S), якщо X=true, то по черзі, якщо X=false, то випадково[2]. В класі Semaphore визначено такі операції для роботи з семафорами:

* acquire() – метод, що реалізує операцію P(S), підчас якої виконується перевірка значення дозволів семафору, якщо значення семафору більше або дорівнює одиниці, то від значення семафору віднімається одиниця і потік продовжує своє виконання, якщо значення менше, то потік блокується до моменту коли умова виконується. Метод acquire() має модифікацію в якій задається число, на скільки зменшити семафор[5]. Також в Java реалізований схожий метод acquireUninterruptibly[5], цей метод отримує дозвіл від семафору, блокуючи потік, доки дозвіл не буде доступний. Якщо є доступний дозвіл, то отримує його и миттєво повертається, зменшуючи кількість доступних дозволів на одиницю. Якщо немає доступних дозволів, то даний потік стає недіючим для розкладу потоків і очікує, доки інший потік не викличе метод release() для цього семафору і даний потік стає наступним, кому буде надано дозвіл. Якщо даний потік переривається, очікуючи на дозвіл, то він продовжить очікувати, але час, за який потік отримає дозвіл, буде змінено порівняно з часом, за який він отримав би дозвіл без переривання. Коли потік повертається з цього методу, його статус переривання буде встановлено. Тобто цей метод схожий до методу acquire(), але переривання не розблокує потік, а лише змінить час його очікування на дозвіл від семафору. У даного методу є модифікація, за якої задається кількість дозволів.
* availablePermits() – метод, що повертає кількість наявних в цьому семафорі дозволів.
* release() – метод, що реалізує операцію V(S), підчас якої виконується збільшення кількості дозволів (повернення) в семафор на один. Якщо всі потоки намагаються отримати дозвіл, то вибирається потік і йому надається дозвіл. Цей потік включається в список планування потоків. Даний метод може приймати, в якості параметру, на скільки дозволів збільшити семафор.
* drainPermits() – метод, що отримує та повертає всі дозволи, котрі зразу стають доступними.
* getQueuedThreads() - повертає колекцію потоків, які можуть очікувати дозвіл. Оскільки фактичний набір потоків може змінитися під час виконання методу, результат може бути не достовірним. Елементи колекції не мають порядку. Цей метод призначений для поліпшення побудови дочірніх класів, котрі мають більш широкі можливості контролю.
* getQueueLength() - повертає оцінку числа потоків, які очікують дозволу. Значення тільки оцінка, тому що число потоків може мінятися динамічно під час виконання методу. Цей метод призначений для використання в сфері моніторингу стану системи, а не для синхронізації.
* hasQueuedThreads() - надсилає запит, чи чекають якісь потоки методу acquire().
* isFair() - повертає істину, якщо справедливість семафору встановлена в істину.
* reducePermits() - зменшує кількість дозволів. Цей метод може бути корисним в підкласах, котрі використовують семафори для моніторингу ресурсів. Основна відмінність від методу acquire() в тому, що він не блокує потік доки дозвіл не стане доступним. Даний метод може приймати, в якості параметру, на скільки зменшити кількість дозволів, якщо без параметрів то на 1.
* tryAcquire() - отримує вказану кількість дозволів від цього семафора, тільки якщо всі вони доступні під час виклику методу і відразу ж повертається зі значенням доступних дозволів скороченим на задану величину. Якщо дозволів немає, то цей метод повертає негайно зі значенням помилки і число доступних дозволів залишається незмінним. Даний метод може приймати в якості параметрів: кількості дозволів, час очікування виконання методу і кількість дозволів та час очікування виконання методу.

Також в семафорі є методи wait(), notify(), notifyAll() які працюють так само як в потоках, наявність цих методів пояснюється наслідуванням від спільного предка[1].

В мові Java на відміну від мови Ада лічильники семафорів є змінними не булевського типу, а чисельного типу integer. Тобто семафори можуть бути не тільки двійковими, а й багатозначними.

Приклад використання семафорів мови Java.

//Задача взаємного виключення

import java.util.concurrent.\*;

public class Cursova1 {

public static int *buf*;

public static Semaphore *S* = new Semaphore(1);

public static void main(String[] args) {

Thread1 T1 = new Thread1();

Thread2 T2 = new Thread2();

T1.start();

T2.start();

try {

T1.join();

T2.join();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

public class Thread1 extends Thread{

public void run(){

System.*out*.println("T1 started");

//паралельні дії потоку

try {

Cursova1.*S*.acquire();

} catch (InterruptedException e) {

e.printStackTrace();

}

Cursova1.*buf* = Cursova1.*buf*+100; //Крітична ділянка

Cursova1.*S*.release();

//ще якісь паралельні дії

System.*out*.println("T1 finished");

}

}

public class Thread2 extends Thread{

public void run(){

System.*out*.println("T2 started");

//паралельні дії потоку

try {

Cursova1.*S*.acquire();

} catch (InterruptedException e) {

e.printStackTrace();

}

Cursova1.*buf* = Cursova1.*buf*\*74+26; //Крітична ділянка

Cursova1.*S*.release();

//ще якісь паралельні дії

System.*out*.println("T1 finished");

}

}

//Задача синхронизації

import java.util.concurrent.\*;

public class Cursova2 {

public static int *buf*;

public static Semaphore *S* = new Semaphore(0);

public static void main(String[] args) {

Thread1 T1 = new Thread1();

Thread2 T2 = new Thread2();

T1.start();

T2.start();

try {

T1.join();

T2.join();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

public class Thread1 extends Thread{

public void run(){

System.*out*.println("T1 started");

//паралельні дії потоку

Cursova2.*S*.release(); //сигнал

//ще якісь паралельні дії

System.*out*.println("T1 finished");

}

}

public class Thread2 extends Thread{

public void run(){

System.*out*.println("T2 started");

//паралельні дії потоку

try {

Cursova2.*S*.acquire(); //очікування сигналу

} catch (InterruptedException e) {

e.printStackTrace();

}

//ще якісь паралельні дії

System.*out*.println("T1 finished");

}

}

# 1.3 Реалізація семафорів в мові C#

Семафор в C# має декілька варіацій: Semaphore, SemaphoreSlim[3].

Semaphore обмежує кількість потоків, які можуть отримати доступ до ресурсу або пулу ресурсів одночасно. Даний семафор має декілька конструкторів, а саме з значеннями: максимальної кількості одночасних записів і можливо резервування деяких записів; максимальної кількості одночасних записів і можливо резервування деяких записів для виклику потоку, і можливо, із зазначенням найменування об’єкта системного семафора; максимальної кількості одночасних записів і можливо резервування деяких записів для виклику потоку, і можливо, із зазначенням найменування об’єкта системного семафора, і вказати змінну, яка отримує значення, яке вказує, чи є новий семафор системи новостворений; максимальної кількості одночасних записів і можливо резервування деяких записів для виклику потоку, і можливо, із зазначенням найменування об’єкта системного семафора, і вказати змінну, яка отримує значення, яке вказує, чи є новий семафор системи новостворений та визначення контролю безпеки доступу до системи семафора. В класі Semaphore визначені наступні методи:

* WaitOne() - блокує даний потік до отримання сигналу. Цей метод має декілька модифікацій з різними параметрами за яких до поведінки методу додаються такі дії: спостерігання за CancellationToken; використовуючи 32-розрядне число, яке визначає тайм-аут; використовуючи TimeSpan щоб вказати тайм-аут; використовуючи 32-розрядне число, яке визначає очікування, і вказівки, чи потрібно виходити з домену синхронізації до закінчення очікування; використовуючи TimeSpan, який визначає час очікування, і вказівки, чи потрібно виходити з домену синхронізації до закінчення очікування;
* Dispose() - звільняє всі ресурси, використовувані поточним екземпляром класу WaitHandle.
* Release()- виходить з семафора і повертає попереднє значення лічильника. Є модифікація для виходу з семафору задану кількість разів.
* TryOpenExisting() - відкриває вказаний іменований семафор, якщо він вже існує, і повертає значення, яке вказує, чи є операція успішної. Даний метод має модифікацію в якій: метод відкриває вказаний іменований семафор, якщо він вже існує, з бажаною безпекою доступу і повертає значення, яке вказує, чи є операція успішною[5].

SemaphoreSlim легка альтернатива семафора, який обмежує кількість потоків, які можуть отримати доступ до ресурсу або пулу ресурсів одночасно. Даний семафор має два конструктори: конструктор ініціалізації нового екземпляра класу SemaphoreSlim, з зазначенням початкового кількість запитів, які можуть бути надані одночасно; та конструктор ініціалізації нового екземпляра класу SemaphoreSlim, з зазначенням початкового та максимального числа запитів, які можуть бути надані одночасно. SemaphoreSlim має методи:

* Release() - виходить з семафору один раз. Даний метод може приймати в якості параметру кількість виходів з семафору.
* Wait()- блокує поточний потік, поки він не може увійти в SemaphoreSlim. Цей метод має декілька модифікацій з різними параметрами за яких до поведінки методу додаються такі дії: спостерігання за CancellationToken; використовуючи 32-розрядне число, яке визначає тайм-аут; використовуючи TimeSpan щоб вказати тайм-аут; використовуючи 32-розрядне число, яке визначає очікування, спостерігаючи за CancellationToken; використовуючи TimeSpan, який визначає час очікування, спостерігаючи CancellationToken;
* WaitAsync() - асинхронне чекання, щоб увійти в SemaphoreSlim. Цей метод має декілька модифікацій з різними параметрами за яких до поведінки методу додаються такі дії: спостерігання за CancellationToken; використовуючи 32-розрядне число, яке визначає тайм-аут; використовуючи TimeSpan щоб вказати тайм-аут; використовуючи 32-розрядне число, яке визначає очікування, спостерігаючи за CancellationToken; використовуючи TimeSpan, який визначає час очікування, спостерігаючи CancellationToken[4].

Обидва семафори в C# можуть викликати виключення SemaphoreFullException - виключення, яке викидається при Semaphore.Release, коли кількість вже на максимумі[13].

Лічильники семафорів в C#, як і в Java, є числами, тому семафори можуть бути багатозначними.

Приклад використання

//Задача взаємного виключення

using System;

using System.Threading;

using System.Text;

namespace ConsoleApplication1

{

class Cursova1

{

static Semaphore S1 = new Semaphore(0, 1);

static int buf;

static void T1() {

Console.WriteLine("T1 started");

//якісь паралельні дії потоку

S1.WaitOne();

buf = buf + 100; //крітична ділянка

S1.Release();

//ще якісь паралельні дії

Console.WriteLine("T1 finished");

}

static void T2()

{

Console.WriteLine("T2 started");

//якісь паралельні дії потоку

S1.WaitOne();

buf = buf\*74 + 26; //крітична ділянка

S1.Release();

//ще якісь паралельні дії

Console.WriteLine("T2 finished");

}

static void Main(string[] args)

{

Thread Thread1 = new Thread(T1);

Thread Thread2 = new Thread(T2);

Thread1.Start();

Thread2.Start();

Thread1.Join();

Thread2.Join();

}

}

}

//Задача синхронизації

using System;

using System.Threading;

using System.Text;

namespace ConsoleApplication2

{

class Cursova2

{

static SemaphoreSlim S1 = new SemaphoreSlim(0, 1);

static int buf;

static void T1() {

Console.WriteLine("T1 started");

//якісь паралельні дії потоку

S1.Release(); //сигнал

//ще якісь паралельні дії

Console.WriteLine("T1 finished");

}

static void T2()

{

Console.WriteLine("T2 started");

//якісь паралельні дії потоку

S1.Wait(); //очікування на сигнал

//ще якісь паралельні дії

Console.WriteLine("T2 finished");

}

static void Main(string[] args)

{

Thread Thread1 = new Thread(T1);

Thread Thread2 = new Thread(T2);

Thread1.Start();

Thread2.Start();

Thread1.Join();

Thread2.Join();

}

}

}

# 1.4 Реалізація семафорів у бібліотеці Win32

В бібліотеці Win32 семафори є змінними спеціального типу HANDLE, за якими слідкує сама система. В цій бібліотеці лічильники семафорів також є багатозначними. Для семафорів визначені такі операції:

* HANDLE CreateSemaphore (LPSECURITY\_ATTRIBUTES lpSemaphoreAttributes, LONG lInitialCount, LONG lMaximumCount, LPCTSTR lpName) – операція, що створює семафор. lpSemaphoreAttributes – параметри захисту, зазвичай встановлено значення NULL. lInitialCount – початкове значення, воно має бути більше за 0 або рівним 0, та меншим за lMaximumCount або рівним йому. Стан семафору сигнальний, коли це число більше за 0 та не сигнальний, коли рівне 0. Значення семафору зменшується на 1 коли функція очікування розблоковує потік, який чекав на цей семафор. Значення семафору збільшується на визначене значення викликанням функції ReleaseSemaphore. lMaximumCount – максимальне значення, має бути більше за 0. lpName – ім’я семафору, яке має бути не більше за MAX\_PATH та може містити в собі будь-які символи, окрім зворотного слешу(\). Порівняння імен регістрозалежне. Якщо lpName співпадає з вже існуючим семафором, то lInitialCount та lMaximumCount ігноруються, тому що вони вже були встановлені при створенні. Якщо lpName є NULL, то семафор створюється без імені. Якщо функція успішна, то вона повертає семафор типу HANDLE. Якщо іменований об’єкт семафору існував до виклику функції, то функція GetLastError повертає ERROR\_ALREADY\_EXISTS. В інших випадках GetLastError повертає нуль. Якщо CreateSemaphore є невдалою, то повертає NULL. Для поширених відомостей про помилку потрібно викликати функцію GetLastError.
* HANDLE WaitForSingleObject (HANDLE hHandle, DWORD dwMilliseconds) – реалізація операції P(S), hHandle – семафор, dwMilliseconds – визначає час тайм-ауту в мілісекундах. Функція повертає значення коли інтервал часу скінчується, навіть якщо стан семафору несигнальний. Якщо dwMilliseconds дорівнює нулю, то функція визначає стан семафору та повертається миттєво. Якщо dwMilliseconds дорівнює INFINITE, тайм-аут функції ніколи не наступає. Якщо виконання функції успішне, то значення, що вона повертає, визначає яка подія змусила її повернутися. Є два такі значення: WAIT\_OBJECT\_0, яке повертається коли стан семафору стає сигнальним та WAIT\_TIMEOUT, яке повертається коли трапляється тайм-аут та стан семафору є несигнальним. WAIT\_FAILED вказує на не успішне виконання функції. Очікування на некоректний семафор змушує функцію повернути WAIT\_FAILED. Для поширених відомостей про помилку потрібно викликати функцію GetLastError.
* BOOL ReleaseSemaphore (HANDLE hSemaphore, LONG lReleaseCount, LPLONG lpPreviousCount) – реалізація операції V(S), hSemaphore – семафор, який повертається функцією CreateSemaphore. lReleaseCount – число, на яке його потрібно збільшити, це число повинне бути більше за нуль. Якщо додавання цього числа до значення, яке вже є, зробить його більше за максимальне значення семафору, то додавання не відбудеться і функція поверне FALSE. lpPreviousCount – попереднє значення семафору. Може бути NULL, якщо попереднє значення не потребується. Якщо функція успішна, то значення, яке вона повертає, є ненульовим. Якщо ні, то повертає нуль. Для поширених відомостей про помилку потрібно викликати функцію GetLastError[3].

Приклад використання:

//Задача взаємного виключення

#include <iostream>

#include <fstream>

#include <Windows.h>

using std::cout;

using std::endl;

HANDLE S1;

int buf;

HANDLE hThread[2];

void Thread1(){

std:: cout << "Task 1 started" << std::endl;

//якісь паралельні дії потоку

WaitForSingleObject(S1, INFINITE);

buf = buf+100; //крітична ділянка

ReleaseSemaphore(S1, 1, NULL);

//ще якісь паралельні дії

std:: cout << "Task 1 finished" << std::endl;

}

void Thread2(){

std:: cout << "Task 2 started" << std::endl;

//якісь паралельні дії потоку

WaitForSingleObject(S1, INFINITE);

buf = buf\*74+26; //крітична ділянка

ReleaseSemaphore(S1, 1, NULL);

//ще якісь паралельні дії

std:: cout << "Task 2 finished" << std::endl;

}

int main (int argc, char\* argv[])

{

int tid[2];

S1 = CreateSemaphore(NULL, 1, 1, NULL);

hThread[1] = CreateThread(NULL, NULL, (LPTHREAD\_START\_ROUTINE) Thread1, &tid[1], 0, 0);

hThread[2] = CreateThread(NULL, NULL, (LPTHREAD\_START\_ROUTINE) Thread2, &tid[1], 0, 0);

return 0;

}

//Задача синхронизації

#include <iostream>

#include <fstream>

#include <Windows.h>

using std::cout;

using std::endl;

HANDLE S1;

int buf;

HANDLE hThread[2];

void Thread1(){

std:: cout << "Task 1 started" << std::endl;

//якісь паралельні дії потоку

ReleaseSemaphore(S1, 1, NULL); //сигнал

//ще якісь паралельні дії

std:: cout << "Task 1 finished" << std::endl;

}

void Thread2(){

std:: cout << "Task 2 started" << std::endl;

//якісь паралельні дії потоку

WaitForSingleObject(S1, INFINITE); //очікування сигналу

//ще якісь паралельні дії

std:: cout << "Task 2 finished" << std::endl;

}

int main (int argc, char\* argv[])

{

int tid[2];

S1 = CreateSemaphore(NULL, 0, 1, NULL);

hThread[1] = CreateThread(NULL, NULL, (LPTHREAD\_START\_ROUTINE) Thread1, &tid[1], 0, 0);

hThread[2] = CreateThread(NULL, NULL, (LPTHREAD\_START\_ROUTINE) Thread2, &tid[1], 0, 0);

return 0;

}

# Висновки до розділу 1:

1. Розглянуто реалізацію семафорів у різних мовах. Показано, що у будь-яких мовах семафори мають операції створення або конструктори, реалізацію операції P(S) та реалізацію операції V(S), що є основними операціями для використання семафорів.
2. Розглянуто сутність семафорів у різних мовах. Було виявлено, що в мові Ада семафори можуть бути тільки двійковими, а в мовах Java, C# та бібліотеці Win32 вони можуть бути багатозначними, що спрощує задачу синхронізації, оскільки не потрібно створювати багато двійкових семафорів для однотипних сигналів.
3. Виявлено, що в реалізації P(S) у мовах Ада та C# та у бібліотеці Win32 може виконуватись тільки одна дія після виконання умови S==1: S=S-1, а у мові Java може виконуватись також S=S-k, що використовується для багатозначних семафорів та зменшує код порівняно з іншими реалізаціями семафору.
4. Виявлено, що в реалізація V(S) у мові Ада значення семафору може збільшитися лише на 1, а у мовах Java, C# та у бібліотеці Win32 на будь-яке число в межах максимального значення семафору.
5. В результаті вивчення документації по реалізаціям семафорів в різних мовах визначено, що в мові Java можна обрати як будуть розблоковуватися процеси – по черзі або випадково. В інших мовах такої можливості немає.

# РОЗДІЛ 2 Розробка програмного забезпечення для ПКС з ЗП

# 2.1 Розробка програми

**Аналіз вихідної задачі на внутрішній паралелізм з використанням концепції необмеженого паралелізму**

Концепція необмеженого паралелізму складається з трьох принципів[14]:

1. Всі операції виконуються за одиницю часу (t=1);
2. Час передачі даних не враховується (t=0);
3. Кількість процесорів необмежена (P=∞).

Для аналізу задачі на внутрішній паралелізм необхідно визначити кількість операцій при необмеженій кількості процесорів та при рівності часу їх виконання без врахування часу передачі даних.

MА = α\*MB+β\*MO\*MX\*max(MK)

1

max

\*

\*

2

\*

3

\*

4

+

Рис.2.1 Ярусно-паралельна форма виразу

З ярусно-паралельної форми виразу (рис.2.1) видно, що кількість операцій за концепцією необмеженого паралелізму дорівнює n = 4. Оскільки 4≠1, то задача не повністю паралельна.

**Розробка паралельного математичного алгоритму вихідної математичної задачі з виділенням спільних ресурсів**

При розробці паралельного алгоритму передбачається, що розмірність задачі (N) більше або дорівнює числу процесорів (P) (N> = P), причому N кратно P, що дозволяє розбити задачу на окремі шматочки Н, кожен з яких обробляється своїм завданням.

H=N/P

Таким чином, отримано алгоритм виконання математичної задачі:

1. ai=max(MKH), i=(1..P);
2. a=max(a, ai), I=(1..P);
3. MAH=α\*MBH+β\*(MO\*MXH)\*a

Загальні ресурси: α, МО, β, а.

**Розробка алгоритмів кожного паралельного процесу з визначенням точок синхронізації та критичних ділянок**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Т1 | Точки синхронізації та критичні ділянки | |
| 1) | Введення MB, MO, alpha |  | |
| 2) | Сигнал про введення в задачі Т1 MB, MO, alpha | S2,..,P-1 | |
| 3) | Чекати введення в задачі Тp MX, MK, beta | Wp-1 | |
| 4) | Обчислення ai=max(MKH) |  | |
| 5) | Обчислення a=max(a, ai) | КД | |
| 6) | Сигнал про обчислення а | S2,..,P-2 | |
| 7) | Чекати обчислення a в Т2..Тр | W2,..,P-2 | |
| 8) | Копіювання alpha1=alpha, beta1=beta | КД | |
| 9) | Копіювання а1=а | КД | |
| 10) | Копіювання МО1=МО | КД | |
| 11) | Обчислення MAH=alpha1\*MBH+beta1\*(MO1\*MXH)\*a1 |  | |
| 12) | Чекати обчислення MA в Т2..Тр | W2,..,P-3 | |
| 13) | Вивід МА |  | |
|  | | Тi (i=2..P-1) | | Точки синхронізації та критичні ділянки |
| 1) | | Чекати введення в задачі Т1 MB, MO, alpha | | W1-1 |
| 2) | | Чекати введення в задачі Тр MX, MK, beta | | Wр-1 |
| 3) | | Обчислення ai=max(MKH) | |  |
| 4) | | Обчислення a=max(a, ai) | | КД |
| 5) | | Сигнал про обчислення а | | S1,..i-1,i+1,..,P-1 |
| 6) | | Чекати обчислення a в Т2..Тр | | W1,..i-1,i+1,..,P-2 |
| 7) | | Копіювання alphai=alpha, betai=beta | | КД |
| 8) | | Копіювання аi=а | | КД |
| 9) | | Копіювання МОi=МО | | КД |
| 10) | | Обчислення MAH=alphai\*MBH+betai\*(MOi\*MXH)\*ai | |  |
| 11) | | Сигнал Т1 про обчислення МА | | S1-2 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | Тp | Точки синхронізації та критичні ділянки | |
| 1) | Введення MX, MK, beta |  |
| 2) | Сигнал про введення в задачі Тp MX, MK, beta | S1,..,P-1 -1 |
| 3) | Чекати введення в задачі Т1 MB, MO, alpha | W1-1 |
| 4) | Обчислення ai=max(MKH) |  |
| 5) | Обчислення a=max(a, ai) | КД |
| 6) | Сигнал про обчислення а | S1,..,P-1 -2 |
| 7) | Чекати обчислення a в Т2..Тр | W1,..,P-1 -2 |
| 8) | Копіювання alphap=alpha, betap=beta | КД |
| 9) | Копіювання аp=а | КД |
| 10) | Копіювання МОp=МО | КД |
| 11) | Обчислення MAH=alphap\*MBH+betap\*(MOp\*MXH)\*ap |  |
| 12) | Сигнал Т1 про обчислення МА | S1-3 |

**Розробка схеми та опис взаємодії процесів**

На рисунку 2.2 зображено розроблену схему взаємодії процесів.

Ti

W1-1

Wp-1

S(1..i-1)V(i+1..P)-1 -1

W(1..i-1)V(i+1..P)-1 -2

S1-2

E1

T1

S2,..,P-1

Wp-1

S2,..,P-2

W2,..,P-2

W2,..,P-3

events[i-1]

E1

events[i-1]

events[0]

S1

events[0]

alpha, beta

CS1

E2

E2

Tp

S1,..,P-1-2

W1,..,P-1-2

S1-3

S1,..,P-1-1

W1-1

events[P-1]

events[P-1]

MO

CS2

a

M1

S1

Рис. 2.2. Схема взаємодії процесів

Опис схеми взаємодії процесів:

E1 – подія по введенню у першому потоці;

E2 – подія по введенню у останньому потоці;

events[0], events[i-1], events[p-1] – події по обчисленню а в потоках;

S1 – семафор для синхронізації по обчисленню МА в потоках 2,..,Р;

CS1 – критична секція для вирішення задачі взаємного виключення для спільних ресурсів alpha та beta;

CS2 – критична секція для вирішення задачі взаємного виключення для спільного ресурсу МО;

M1 – мютекс для вирішення задачі взаємного виключення для спільного ресурсу а.

**Розробка паралельної програми з використанням заданих засобів синхронізації, налагодження програми та її виконання**

В результаті розробки алгоритмів задач та схеми їх взаємодії розроблено паралельну програму для вирішення заданої математичної задачі за допомогою засобів бібліотеки Win32. Програму розроблено на мові С++. Для вирішення задачі взаємного виключення було використано критичні секції (CS1 та CS2) та мютекс (М1). Для вирішення задачі синхронізації було використано події (E1, E2, events[1..p]) та семафор (S1). Для створення потоків використовується потокова функція thread(LPVOID void\_tid), якій в якості параметру передається номер потоку, після чого при виконанні дій даним потоком цей номер визначає послідовність дій та параметри обчислень.

Лістинг наведено у додатку А.

# 2.2 Проведення досліджень ефективності розробленої програми в реальній 4-х ядерній системі

Результати проведених досліджень ефективності розробленої програми:

Таблиця 2.1. Час виконання обчислень програмою

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | P=1 | P=2 | P=3 | P=4 |
| 1000 | 6.8 c | 3.3 c | 2.2 c | 1.8 c |
| 2000 | 63.2 c | 32 c | 21.6 c | 16.3 c |
| 3000 | 245.5 c | 122.4 c | 83.6 c | 63.3 c |

Таблиця 2.2. Значення коефіцієнтів прискорення

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | P=1 | P=2 | P=3 | P=4 |
| 1000 | 1 | 2.06 | 3.1 | 3.7 |
| 2000 | 1 | 1.975 | 2.93 | 3.87 |
| 3000 | 1 | 2.01 | 2.94 | 3.88 |

Таблиця 2.3. Значення коефіцієнтів ефективності

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | P=1 | P=2 | P=3 | P=4 |
| 1000 | 100% | 103% | 103.3% | 92.5% |
| 2000 | 100% | 98.75% | 97.6% | 96.75% |
| 3000 | 100% | 100.5% | 98% | 97% |

На основі проведених досліджень побудовано наступні графіки залежностей коефіцієнтів прискорення та ефективності від кількості процесорів (рис. 2.3 та рис. 2.4):

Рис. 2.3. Залежність коефіцієнтів прискорення від кількості процесорів.

Рис. 2.4. Залежність коефіцієнтів ефективності від кількості процесорів.

# Висновки до розділу 2:

1. Було розроблену паралельну програму для обчислення математичного виразу. На основі досліджень отриманої програми можна зробити висновок, що паралелізм програми розроблено коректно, що видно з графіків коефіцієнтів прискорення та ефективності (рис. 2.3 та рис. 2.4).
2. Виявлено, що в деяких випадках коефіцієнти прискорення більші за Р та коефіцієнти ефективності більше 100%. Це можна пояснити тим, що система виділяє процесор потокам за пріоритетами, тобто якийсь інший потік мав більший пріоритет, ніж розроблена програма.
3. Дослідження показують, що коефіцієнт прискорення зростає лінійно, але є деякий розкид прискорення для різних розмірностей елементів математичного виразу.
4. В результаті досліджень можна побачити, що коефіцієнт ефективності зменшується зі зростанням кількості процесорів. Але при цьому при збільшенні кількості ядер він спадає менш інтенсивно, окрім розмірності 1000. Така поведінка графіку для 1000 пояснюється тим, що час виконання програми при такій розмірності малий, але система віддавала перевагу більш пріоритетним задачам при її виконанні.

# РОЗДІЛ 3 Розробка програмного забезпечення для ПКС з ЛП

# 3.1 Розробка програми

**Аналіз вихідної задачі на внутрішній паралелізм з використанням концепції необмеженого паралелізму**

Концепція необмеженого паралелізму складається з трьох принципів:

1. Всі операції виконуються за одиницю часу (t=1);
2. Час передачі даних не враховується (t=0);
3. Кількість процесорів необмежена (P=∞).

Для аналізу задачі на внутрішній паралелізм необхідно визначити кількість операцій при необмеженій кількості процесорів та при рівності часу їх виконання без врахування часу передачі даних.

MА = α\*MB+β\*MO\*MX\*max(MK)

1

max

\*

\*

2

\*

3

\*

4

+

Рис.3.1 Ярусно-паралельна форма виразу

З ярусно-паралельної форми (рис.3.1) виразу видно, що кількість операцій за концепцією необмеженого паралелізму дорівнює n = 4. Оскільки 4≠1, то задача не повністю паралельна.

**Розробка паралельного математичного алгоритму вихідної математичної задачі з виділенням спільних ресурсів**

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 шматочки Н, кожен з яких обробляється своїм завданням.



Таким чином, отримано алгоритм виконання математичної задачі:

1. ai=max(MKH), i=(1..P);
2. a=max(a, ai), I=(1..P);
3. MAH=α\*MBH+β\*(MO\*MXH)\*a

Загальні ресурси: α, МО, β, а.

**Розробка алгоритмів кожного паралельного процесу на основі моделі, заснованої на посилці повідомлень**

|  |  |
| --- | --- |
|  | Т0 |
| 1) | Введення MB, MO, alpha, MX, MK, beta |
| 2) | Пересилка MB2H, MO, alpha, MX2H, MK2H, beta у задачі Ti, i=(2, 4, ..,2\*P) |
| 3) | Обчислення a=max(MKH) |
| 4) | Отримання аі від задач Ti2 |
| 5) | Обчислення a=max(a, ai) |
| 6) | Пересилка а у задачі Ti2 |
| 7) | Обчислення MAH=alpha\*MBH+beta\*(MO\*MXH)\*a |
| 8) | Отримання MA2H від задач Ti, i=(2, 4,..,2\*P) |
| 9) | Вивід МА |

|  |  |
| --- | --- |
|  | Ti2 |
| 1) | Отримання MB2H, MO, alpha, MX2H, MK2H, beta від задачі Т0 |
| 2) | Пересилка MBH, MO, alpha, MXH, MKH, beta у задачу Ti1 |
| 3) | Обчислення a=max(MKH) |
| 4) | Отримання аі від задачі Ti1 |
| 5) | Обчислення a=max(a, ai) |
| 6) | Пересилка а у задачу T0 |
| 7) | Отримання а від задачі Т0 |
| 8) | Пересилка а у задачу Ті1 |
| 9) | Обчислення MAH=alpha\*MBH+beta\*(MO\*MXH)\*a |
| 10) | Отримання MAH від задачі Ti1 |
| 11) | Пересилка MA2H у задачу Т0 |

|  |  |
| --- | --- |
|  | Ti1 |
| 1) | Отримання MBH, MO, alpha, MXH, MKH, beta від задачі Ті2 |
| 2) | Обчислення a=max(MKH) |
| 3) | Посилка а у задачу Ті2 |
| 4) | Отримання а від задачі Ті2 |
| 5) | Обчислення MAH=alpha\*MBH+beta\*(MO\*MXH)\*a |
| 6) | Пересилка MAH у задачу Ті2 |

**Розробка схеми та опис взаємодії процесів**

На рисунку 3.2 зображено розроблену схему взаємодії процесів.

MAH🡪

MA2H🡪

a🡨

MO, MBH, MKH, MXH, alpha, beta 🡨

res

Max

Data

MO, MBH, MKH, MXH, alpha, beta, a

Тi1

Max

Т0

MO, MB, MK, MX, alpha, beta, a

MO, MB2H, MK2H, MX2H, alpha, beta 🡨

ai🡪

res

Max

Data

MO, MB2H, MK2H, MX2H, alpha, beta, a

Тi2

ai🡪

a🡨

Max2

a🡪

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.

**Розробка паралельної програми з використанням заданих засобів передачі даних, налагодження програми та її виконання**

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.

# 3.2 Проведення досліджень ефективності розробленої програми в реальній 4-х ядерній системі

Результати проведених досліджень ефективності розробленої програми:

Таблиця 3.1. Час виконання обчислень програмою

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | P=1 | P=2 | P=3 | P=4 |
| 1000 | 3.896 c | 3.057 c | 2.799 c | 2.917 c |
| 2000 | 30.963 c | 24.342 c | 23.774 c | 22.464 c |
| 3000 | 104.53 c | 82.392 c | 78.109 c | 78.078 c |

Таблиця 3.2. Значення коефіцієнтів прискорення

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | P=1 | P=2 | P=3 | P=4 |
| 1000 | 1 | 1.27 | 1.39 | 1.33 |
| 2000 | 1 | 1.27 | 1.3 | 1.38 |
| 3000 | 1 | 1.27 | 1.34 | 1.34 |

Таблиця 3.3. Значення коефіцієнтів ефективності

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | P=1 | P=2 | P=3 | P=4 |
| 1000 | 100% | 63.5% | 46.35% | 33.25% |
| 2000 | 100% | 63.5% | 43.41% | 34.46% |
| 3000 | 100% | 63.5% | 44.6% | 33.5% |

На основі проведених досліджень побудовано наступні графіки залежностей коефіцієнтів прискорення та ефективності від кількості процесорів (рис. 3.3 та рис. 3.4):

Рис. 3.3. Залежність коефіцієнтів прискорення від кількості процесорів.

Рис. 3.4. Залежність коефіцієнтів ефективності від кількості процесорів.

# Висновки до розділу 3:

1. Було розроблену паралельну програму для обчислення математичного виразу. На основі досліджень отриманої програми можна зробити висновок, що паралелізм програми розроблено коректно, що видно з графіків коефіцієнтів прискорення та ефективності (рис. 3.3 та рис. 3.4).
2. Виявлено, що коефіцієнти прискорення та ефективності дуже малі. Це можна пояснити недостачею ресурсів системи. При тестуванні згідно з особливостями системи створювалися програми з 3, 5, 7, 9 потоками відповідно (1, 2, 3, 4 гілки процесорів). А реальна система мала лише 4 ядра, що негативно вплинуло на коефіцієнти.
3. Дослідження показують, що коефіцієнт прискорення зростає лінійно, але є деякий розкид прискорення для різних розмірностей елементів математичного виразу.
4. В результаті досліджень можна побачити, що коефіцієнт ефективності зменшується зі зростанням кількості процесорів.

# ВИСНОВКИ

1. В даному курсовому проекті було розглянуто реалізації семафорів в мовах Ада, C#, Java та бібліотеці Win32. З даних оглядів реалізацій можна зробити висновок, що найзручніша реалізація в мовах Java та C#, тому що в них семафорі можуть бути багатозначними та мають методи для додавання та віднімання декількох одиниць від лічильників.
2. Не зручною реалізацією є реалізація в мові Ада, тому що в ній семафори можуть бути лише двійковими, що незручно при декількох процесах.
3. В розділі два було розроблено програму з загальною пам’яттю за допомогою бібліотеки Win32 для обчислення математичного виразу. На основі цієї програми було проведено тестування, яке показало ефективність та правильність цієї програми.
4. В розділі три було розроблено програму з локальною пам’яттю за допомогою механізму рандеву мови Ада для обчислення математичного виразу. На основі цієї програми було проведено тестування, яке показало, що ця програма має гарні показники тільки при кількості процесорів в P\*2+1 разів більше, ніж потоків, оскільки P в даній програмі показує не кількість потоків, а кількість гілок в системі. А в кожній гілці по два процесори, окрім них ще є кореневий процесор.
5. Тестування по розділах два та три показало, що коефіцієнти прискорення збільшуються лінійно та мають розбіг при збільшенні кількості процесорів, а коефіцієнти ефективності спадають, при збільшені кількості процесорів повільніше.

# Скорочення

ЗП – загальна пам’ять.

КЕ - коефіцієнт ефективності.

КП – коефіцієнт прискорення.

КР – курсова робота.

ЛП – локальна пам’ять.

ОС – операційна система.

ПЗ – програмне забезпечення.

ПКС – паралельна комп’ютерна система.

ТЗ – технічне завдання.

# Література:

1. І. А. Жуков, О. В. Корочкін Паралельні та розподілені обчислення – Київ: «Корнійчук» 2005. – 224 с.
2. <http://docs.oracle.com/javase/1.5.0/docs/api/java/util/concurrent/Semaphore.html>
3. <http://msdn.microsoft.com/en-us/library/z6zx288a(v=VS.110).aspx>
4. <http://msdn.microsoft.com/en-us/library/system.threading.semaphoreslim(v=vs.110).aspx>
5. <http://msdn.microsoft.com/en-us/library/system.threading.semaphore(v=vs.110).aspx>
6. <http://www.ada-ru.org/files/17_27.pdf>
7. Корочкин А.В. Ада 95: Введение в программирование. - Киев; Свит, 1998. - 260 с.
8. <http://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D0%BC%D0%B0%D1%84%D0%BE%D1%80_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)>
9. <http://msdn.microsoft.com/en-us/library/aa450749.aspx>
10. <http://www.infeig.unige.ch/support/ada/gnatlb/a-sytaco.html>
11. <http://www.e-reading.org.ua/chapter.php/92786/200/Bah_-_Arhitektura_operacionnoii_sistemy_UNIX.html>
12. <http://www.ada-ru.org/V-0.4w/toc_ru.html>
13. <http://msdn.microsoft.com/ru-ru/library/system.threading.semaphorefullexception(v=vs.110).aspx>
14. <http://www.bsu.by/Cache/pdf/194693.pdf>
15. http://www.ngpedia.ru/id627124p3.html

# Додаток А. Лістинг програми для ПКС з ЗП.

**//**Косейкіна Ганна Сергіївна, ІО-91

//Курсова робота

// MА = α\*MB+β\*MO\*MX\*max(MK)

#include <iostream>

#include <fstream>

#include <Windows.h>

using std::cout;

using std::endl;

int const N = 3000;

int const P = 4;

int const H = N/P;

struct Matrix{

int mas[N][N];

};

int alpha, beta;

int a = 0;

Matrix MA, MB, MO, MX, MK;

//events

HANDLE E1, E2;

HANDLE events[P];

//mutex

HANDLE M1;

//semaphore

HANDLE S1;

//CS

CRITICAL\_SECTION CS1, CS2;

//threads

HANDLE hThread[P];

//thread id

DWORD Tid[P];

//thread function

void thread(LPVOID void\_tid){

int tid = \*(int\*) void\_tid;

std:: cout << "Task "<<tid+1<< " started" << std::endl;

int ai = 0;

int alphai, betai;

Matrix MOi;

//input T1

if (tid ==0) {

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

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

MB.mas[i][j] = 1;

MO.mas[i][j] = 1;

}

}

alpha = 1;

//signal about input T1

SetEvent(E1);

//wait for input Tp

WaitForSingleObject(E2, INFINITE);

}

//input Tp

if(tid == P-1){

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

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

MX.mas[i][j] = 1;

MK.mas[i][j] = 1;

}

}

beta = 1;

//signal about input Tp

SetEvent(E2);

//wait for input T1

WaitForSingleObject(E1, INFINITE);

}

//wait for input T1 and Tp

if ((tid != 0)&&(tid !=P-1)){

WaitForSingleObject(E1, INFINITE);

WaitForSingleObject(E2, INFINITE);

}

//count ai

if (tid == 0){

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

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

if (ai < MK.mas[i][j]){

ai = MK.mas[i][j];

}

}

}

}else{

for (int i = tid\*H; i<(tid+1)\*H; i++){

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

if (ai < MK.mas[i][j]){

ai = MK.mas[i][j];

}

}

}

}

//count a

WaitForSingleObject (M1, INFINITE);

if (a<ai){

a = ai;

}

ReleaseMutex(M1);

//signal about count a

SetEvent (events[tid]);

//wait for count a

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

WaitForSingleObject (events[i], INFINITE);

}

//copy a, alpha, beta, MO

WaitForSingleObject (M1, INFINITE);

ai = a;

ReleaseMutex(M1);

EnterCriticalSection (&CS1);

alphai = alpha;

betai = beta;

LeaveCriticalSection (&CS1);

EnterCriticalSection (&CS2);

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

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

MOi.mas[i][j] = MO.mas[i][j];

}

}

LeaveCriticalSection (&CS2);

//count MAh

for(int i=tid\*H; i<(tid+1)\*H; i++){

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

MA.mas[i][j]=0;

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

MA.mas[i][j] += MX.mas[i][k] \* MOi.mas[k][j];

}

MA.mas[i][j] = alphai\*MB.mas[i][j]+betai\*ai\*MA.mas[i][j];

}

}

//T1 wait, Ti signal

if (tid != 0){

ReleaseSemaphore (S1, 1, NULL);

}else

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

WaitForSingleObject(S1, INFINITE);

}

//output MA

if (tid == 0){

std::cout << MA.mas[0][0] << std::endl;

// Sleep(1000);

std:: cout << "Task "<<tid+1<< " finished" << std::endl;

}else

std:: cout << "Task "<<tid+1<< " finished" << std::endl;

}

int main (int argc, char\* argv[])

{

//#pragma comment (linker, "/STACK:599999")

int tid[P];

E1 = CreateEvent(NULL,true,false,NULL);

E2 = CreateEvent(NULL,true,false,NULL);

InitializeCriticalSection(&CS1);

InitializeCriticalSection(&CS2);

M1 = CreateMutex(NULL, false, NULL);

S1 = CreateSemaphore(NULL, 0, P-1, NULL);

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

{

tid[i] = i;

hThread[i] = CreateThread(NULL, 268435456, (LPTHREAD\_START\_ROUTINE) thread, &tid[i], 0, &Tid[i]);

events[i] = CreateEvent(NULL,true,false,NULL);

}

int p;

std::cin>>p;

return 0;

}

# Додаток Б. Лістинг програми для ПКС з ЛП

--Косейкіна Ганна Сергіївна, ІО-91

--Курсова робота

-- MА = α\*MB+β\*MO\*MX\*max(MK)

with Ada.TEXT\_IO; use Ada.TEXT\_IO;

with Ada.INTEGER\_TEXT\_IO; use Ada.INTEGER\_TEXT\_IO;

with Ada.Calendar; use Ada.Calendar;

with Ada.float\_text\_io; use Ada.float\_text\_io;

PROCEDURE Cursach IS

N: INTEGER := 3000;

P: INTEGER := 9;

H: INTEGER := N/P;

TYPE VECTOR IS ARRAY (INTEGER RANGE <>) OF INTEGER;

TYPE MATRIX IS ARRAY (INTEGER RANGE <>) OF VECTOR(1..N);

task type Ti2 (id: INTEGER) is

ENTRY res(MA1: out MATRIX);

ENTRY Max(a1: IN Integer);

ENTRY Data(alfa1: in integer; beta1: in integer; MB1: IN MATRIX; MO1: IN MATRIX; MX1: IN MATRIX; MK1: IN MATRIX);

end Ti2;

type massT is array (1..P) of Ti2;

gg: Time;

Jj: Time;

Tx: Duration;

task type Ti1 (id: INTEGER) is

ENTRY res(MA1: out MATRIX);

ENTRY Max(a1: IN Integer);

entry max2(a1: out integer);

ENTRY Data(alfa1: in integer; beta1: in integer; MB1: IN MATRIX; MO1: IN MATRIX; MX1: IN MATRIX; MK1: IN MATRIX);

end Ti1;

task body Ti2 is

alfa, beta, a, a1: INTEGER;

MA: MATRIX(1..H\*2);

MB: MATRIX(1..H\*2);

MO: MATRIX(1..N);

MX: MATRIX(1..H\*2);

MK: MATRIX(1..H\*2);

Ti: Ti1=new Ti1(id-1);

BEGIN

PUT\_LINE(id+"STARTED ");

--2

ACCEPT data(alfa1: in integer; beta1: in integer; MB1: IN MATRIX; MO1: IN MATRIX; MX1: IN MATRIX; MK1: IN MATRIX) DO

alfa:= alfa1;

beta:= beta1;

MB(1..H\*2):= MB1(id\*H+1..H\*(id+1))&MB1((id-1)\*H+1..H\*id);

MO:= MO1;

MX(1..H\*2):= MX1(id\*H+1..H\*(id+1))&MX1((id-1)\*H+1..H\*id);

MK(1..H\*2):= MK1(id\*H+1..H\*(id+1))&MK1((id-1)\*H+1..H\*id);

END data;

--3

Ti.DATA(alfa, beta, MB, MO, MX, MK);

a:= -99999999;

for i in 1..H loop

for j in 1..N loop

if MK(i)(j) > a then

a:= MK(i)(J);

end if;

end loop;

end loop;

Ti.max2(a1);

if a1 > a then

a := a1;

end if;

T0.max(a);

--4

ACCEPT max(a1: IN INTEGER) DO

if a1 > a then

a := a1;

end if;

END max;

--5

Ti.max(a);

--6

FOR I IN 1..H LOOP

FOR J IN 1..N LOOP

MA(i)(J):= 0;

FOR K IN 1..N LOOP

MA(i)(J):= MA(i)(J) +(MO(i)(J)\*MX(I)(K));

END LOOP;

MA(i)(J) := alfa\*MB(i)(j)+beta\*a\*MA(i)(J);

END LOOP;

END LOOP;

--7

Ti.res(MA);

ACCEPT res(MA1: out matrix) DO

MA1((id-1)\*H+1..(id+1)\*H):= MA(1..H\*id);

END res;

PUT\_LINE(id+" FINISHED ");

end Ti2;

TASK BODY Ti1 IS

alfa, beta, a: INTEGER;

MA: MATRIX(1..H);

MB: MATRIX(1..H);

MO: MATRIX(1..N);

MX: MATRIX(1..H);

MK: MATRIX(1..H);

BEGIN

PUT\_LINE(id+" STARTED ");

--2

ACCEPT data(alfa1: in integer; beta1: in integer; MB1: IN MATRIX; MO1: IN MATRIX; MX1: IN MATRIX; MK1: IN MATRIX) DO

alfa:= alfa1;

beta:= beta1;

MB(1..H):= MB1(1+H..H\*2);

MO:= MO1;

MX(1..H):= MX1(1+H..H\*2);

MK(1..H):= MK1(1+H..H\*2);

END data;

--3

a:= -99999999;

for i in 1..H loop

for j in 1..N loop

if MK(i)(j) > a then

a:= MK(i)(J);

end if;

end loop;

end loop;

accept max2(a1: out integer) do

if a>a1 then

a1 = a;

end if;

end max2;

accept max(a1: in integer) do

a = a1;

end max;

--6

FOR I IN 1..H LOOP

FOR J IN 1..N LOOP

MA(i)(J):= 0;

FOR K IN 1..N LOOP

MA(i)(J):= MA(i)(J) +(MO(i)(J)\*MX(I)(K));

END LOOP;

MA(i)(J) := alfa\*MB(i)(j)+beta\*a\*MA(i)(J);

END LOOP;

END LOOP;

--7

ACCEPT res(MA1: out matrix) DO

MA1(H+1..H\*2):= MA(1..H);

END res;

PUT\_LINE(id+" FINISHED ");

END Ti1;

--========================================================================

TASK T0 IS

ENTRY Max(a1: IN Integer);

END T0;

--========================================================================

TASK BODY T0 IS

alfa, beta, a: INTEGER;

MA: MATRIX(1..N);

MB: MATRIX(1..N);

MO: MATRIX(1..N);

MX: MATRIX(1..N);

MK: MATRIX(1..N);

Ti: massT(1..P);

BEGIN

Gg:= clock;

PUT\_LINE(" T0 STARTED ");

--1

FOR I IN 1..N LOOP

FOR J IN 1..N LOOP

MB(I)(J) := 1;

MO(I)(J) := 1;

MX(I)(J) := 1;

MK(I)(J) := 1;

END LOOP;

END LOOP;

ALFA:= 1;

beta:= 1;

for i in 1..P loop

Ti(i)=new Ti2(i\*2);

end loop;

--2

for i in 1..P loop

Ti(i).data(alfa, beta, MB, MO, MX, MK);

end loop;

--3

a:= -99999999;

for i in 1..H loop

for j in 1..N loop

if MK(i)(j) > a then

a:= MK(i)(J);

end if;

end loop;

end loop;

--4

for i in 1..P loop

ACCEPT max(a1: IN INTEGER) DO

if a1 > a then

a := a1;

end if;

END max;

end loop;

--5

for i in 1..P loop

Ti(i).max(a);

end loop;

--6

FOR I IN 1..H LOOP

FOR J IN 1..N LOOP

MA(i)(J):= 0;

FOR K IN 1..N LOOP

MA(i)(J):= MA(i)(J) +(MO(i)(J)\*MX(I)(K));

END LOOP;

MA(i)(J) := alfa\*MB(i)(j)+beta\*a\*MA(i)(J);

END LOOP;

END LOOP;

--7

for i in 1..P loop

Ti(i).res(MA);

end loop;

--8

IF N < 10 THEN

FOR i IN 1..N LOOP

FOR J IN 1..N LOOP

PUT(MA(i)(j));

PUT(" ");

end loop;

NEW\_LINE;

END LOOP;

ELSE

PUT(MA(1)(1));

NEW\_LINE;

END IF;

PUT\_LINE(" T0 FINISHED ");

jj:= clock;

Tx := Jj-Gg;

put (Float(Tx), 4, 3, 0);

END T0;

BEGIN

NULL;

END Cursach;

# Додаток В. Алгоритм головної програми для ПКС з ЗП

# Додаток Г. Алгоритм процесів для ПКС з ЗП

# Додаток Ґ. Алгоритм головної програми для ПКС з ЛП

# 

# Додаток Д. Алгоритм процесів для ПКС з ЛП