НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

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

(повна назва кафедри, циклової комісії)

**КУРСОВА РОБОТА**

з дисципліни «Паралельне програмування»

(назва дисципліни)

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

Студента (ки) 3 курсу \_\_\_\_\_\_ групи

напряму підготовки 050103 «Програмна інженерія»

Кахерський О.І.

(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- 2016 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 6.050103 «Програмна інженерія»

# (шифр і назва)

ЗАВДАННЯ

НА КУРСОВУ РОБОТУ СТУДЕНТУ

Кахерському Олегу Ігоровичу

(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

3. Вхідні дані до роботи

- Засоби роботи з процесами в бібліотеці Win32

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

5. Перелік графічного матеріалу

7. Дата видачі завдання 18.02.2016

# КАЛЕНДАРНИЙ ПЛАН

|  |  |  |
| --- | --- | --- |
| №  з/п | Назва етапів виконання КП | Строк виконання етапів КП |
| 1 | Виконання Розділу 1 | 01.03.2016 |
| 2 | Виконання Розділу 2 | 23.03.2016 |
| 3 | Виконання Розділу 3 |  |
| 4 | Тестування програм ПРГ1 та ПРГ2 |  |
| 7 | Оформлення КР |  |
| 8 | Захист КР |  |

**Студент \_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_**Кахерський О.І.**\_\_\_\_\_**

( підпис ) (прізвище та ініціали)

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

( підпис ) (прізвище та ініціали)

ЗМІСТ

[ЗМІСТ 4](#_Toc446262730)

[ВСТУП 5](#_Toc446262731)

[РОЗДІЛ 1. ЗАСОБИ РОБОТИ З ПРОЦЕСАМИ В БІБЛІОТЕЦІ WIN32 6](#_Toc446262732)

[1.1 Загальні відомості про багатопотоковість 6](#_Toc446262733)

[1.1.1 Поняття багатопотоковості 6](#_Toc446262734)

[1.1.2 Типи реалізації потоків 6](#_Toc446262735)

[1.1.3 Взаємодія потоків 7](#_Toc446262736)

[1.2 Win32. Базові операції над потоками 8](#_Toc446262737)

[1.2.1 Створення нового потоку 8](#_Toc446262738)

[1.2.2 Планування потоків 10](#_Toc446262739)

[1.2.3 Завершення виконання потоків 11](#_Toc446262740)

[1.2.4 Пріорітети потоків 12](#_Toc446262741)

[1.2.5 Функції очікування 13](#_Toc446262742)

[1.2.6 Семафори в Win32 15](#_Toc446262743)

[1.2.7 Взаємні виключення в Win32 17](#_Toc446262744)

[1.2.8 Критичні секції в Win32 19](#_Toc446262745)

[1.2.9 Монітори в Win32. 20](#_Toc446262746)

[1.2.10 Події в Win32 20](#_Toc446262747)

[1.3 Висновки до першого розділу 21](#_Toc446262748)

[РОЗДІЛ 2. РОЗРОБКА ПРОГРАМИ ПРГ1 ДЛЯ ПКС СП 24](#_Toc446262749)

[2.1 Розробка паралельного математичного алгоритму. 24](#_Toc446262750)

[2.2 Розробка алгоритмів задач 25](#_Toc446262751)

[2.3 Розробка схеми взаємодії процесів 27](#_Toc446262752)

[2.4 Розробка програми ПРГ1 29](#_Toc446262753)

[2.5 Тестування програми ПРГ1 30](#_Toc446262754)

[2.6 Висновки до розділу 2 34](#_Toc446262755)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 36](#_Toc446262756)

[ДОДАТКИ 37](#_Toc446262757)

[Додаток А. Лістинг програми ПРГ1 37](#_Toc446262758)

ВСТУП

Паралельним обчисленням в останній час приділяється досить велика увага. Загалом це пов’язане з двома факторами. Перший фактор обумовлений науково-технічним прогресом, в результаті якого з’явилися нові області знань, які потребують ресурсномістких розрахунків, виконання яких можливе лише на базі високовиробничої техніки, за допомогою методів паралельних і розподілених обчислень. Другим суттєвим фактором являється повсюдне розповсюдження паралельних комп’ютерних систем. Паралельні інформаційні технології перетворилися з вузько направленої дисципліни в необхідну складову комплексу знань розробника сучасного програмного забезпечення.

В даній курсовій роботі розглядаються питання розробки програмного забезпечення для паралельних комп’ютерних систем.

В першому розділі «Огляд засобів роботи з процесами в бібліотеці Win32» описані основні принципи роботи бібліотеки Win32 і наведено приклади найголовніших функцій.

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

Лістинги та алгоритми розроблених програм наведено у додатках.

РОЗДІЛ 1. ЗАСОБИ РОБОТИ З ПРОЦЕСАМИ В БІБЛІОТЕЦІ WIN32

* 1. **Загальні відомості про багатопотоковість**
     1. **Поняття багатопотоковості**

Багатопотоковість – властивість операційної системи або програмного застосунку, яка полягає в тому, що процес, створений в операційній системі, може складатися з декількох потоків, що виконуються “паралельно”, тобто без запропонованного порядку в часі. При виконанні деяких задач таке розподілення може досягнути більш ефективного використання ресурсів обчислювальної машини.[2]

Такі потоки в першу чергу називають потоками команд (на відміну від потоків даних) або (рідше) потоками виконання – англ. Execution thread.

Суттю багатопотоковості є квазібагатозадачність на рівні одного процессу, тобто всі потоки виконуються в одному адресному просторі процесу. Крім цього, всі потоки процесу мають не лише єдиний адресний простір, але й спільні дескриптори файлів. Процес, що виконується, має як мінімум один (головний) потік.

Багатопотоковість (як доктрину програмування) не варто плутати ні з багатозадачністю, ні з багатопроцесорністю, не зважаючи на те, що операційні системи, що реалізують багатозадачність, як правило реалізують і багатопотоковість.

До переваг багатопотоковості в програмуванні можна віднести наступне:

1. Спрощення складності програми в деяких випадках за рахунок використання спільного адресного простору;
2. меньші часові витрати на створення нового потоку в порівнянні зі створенням нового процесу;
3. підвищення продуктивності процесу за рахунок розпаралелювання процесорних обчислень та операцій вводу/виводу.
   * 1. **Типи реалізації потоків**

Існує декілька типів реалізації потоків [8]:

1. Потік в просторі користувача.

Кожен процес має таблицю потоків, аналогічну таблиці процесів ядра.

Переваги:

* Можливість реалізації на ядрі, що не підтримує багатопотоковість;
* більш швидке переключення, створення та завершення потоків;
* процес може мати власний алгоритм планування.

Недоліки:

* Відсутність переривання по таймеру в середині одного процесу;
* при використанні блокуючого системного запиту всі інші потоки блокуються;
* складність реалізації.

1. Потік в просторі ядра.

Додатково до таблиці процесів в просторі ядра міститься таблиця потоків.

1. Змішана реалізація.

Потоки працюють в режимі користувача, але при системних викликах переключаються в режим ядра. Переключення в режим ядра та назад є ресурсоємкою операцією та негативно позначається на швидкодії системи. Через це було введено поняття волокна – полегшеного потоку, що виконується виключно в режимі користувача. У кожного потоку може бути декілька волокон. Такий тип багатопотоковості реалізован в ОС Windows.

* + 1. **Взаємодія потоків**

В багатопотоковому середовищі часто виникають проблеми, пов’язані з паралельним використанням потоками одних даних або пристроїв. Це може призвести до конфлікту процесів або некоректної поведінки програми. Тому використання спільних змінних призводить до вирішення проблем, які зводяться розв’язання двох завдань – взаємного виключення та синхронізації процесів. Для цього використовуються такі методи взаємодії потоків, як взаємні виключення (мьютекси), семафори, критичні ділянки та події.[8]

Взаємовиключення (mutex, мьютекс) – це об’єкт синхронізації, який встановлюється в особливий сигнальний стан, коли не зайнятий будь – яким потоком. Тільки один потік володіє цим об’єктом в будь – який момент часу, звідси й назва таких об’єктів (від англійського mutually exclusive access – що взаємно виключає доступ) – одночасний доступ до спільного ресурсу виключається. Після всіх необхідних дій мьютекс звільняється, надаючи іншим потокам доступ до загального ресурсу.

Семафори – доступні ресурси, які можуть бути надані декількам потокам одночасно, поки пул ресурсів не буде пустим. Тоді додаткові потоки повинні знаходитись в стані блокування, поки необхідна кількість ресурсів не стане наявною. Семафори дуже ефективні, адже вони надають одночасний доступ до ресурсів.

Критичні секції забезпечують синхронізацію подібно мьютексам за винятком того, що об’єкти, які є критичними секціями, наявні в межах одного процесу. Події, мьютекси та семафори також можна використовувати в однопроцесному застосуванні, але критичні секції забезпечують більш швидкий та більш ефективний механізм взаємно – виключної синхронізації. Подібно мьютексам об’єкт, що представляє собою критичну секцію, може бути використаний одним потоком в даний момент часу, що робить їх дуже корисними при обмеженні доступу до спільних ресурсів.

Події корисні в тих випадках, коли необхідно надіслати повідомлення потокові про настання деякої події. Наприклад, при асинхронних операціях вводу/виводу з одного пристрою, система встановлює подію в сигнальний стан, коли закінчується будь-яка з цих операцій. Один потік можу використовувати декілька різних подій в декількох операціях, що перекриваються, а потім очікувати надходження сигналу від будь-якого з них.[8]

* 1. **Win32. Базові операції над потоками**
     1. **Створення нового потоку**

Багатопотокові програми в ОС Windows використовують функції з бібліотеки Win32. Потоки можуть бути створені за допомогою функції CreateThread(). При кожному викликові цієї функції система створює об'єкт ядра (потік). Це компактна структура даних, яка використовується операційною системою для керування потоком та зберігає статистичну інформацію про нього. Система виділяє пам'ять під стек потоку з адресного простору процесу. Новий потік виконується в контексті того ж процесу, що і батьківський потік, тому він отримує доступ до всіх дескрипторів об’єктів ядра, всієї пам’яті та стеків усіх потоків в цьому процесі. За рахунок цього потоки в рамках одного процесу можуть легко взаємодіяти один з одним. [9]

Cтворення нового потоку має вигляд:

Handle thread\_name = CreateThread(

LPSECURITY\_ATTRIBUTES security,

SIZE\_T stackSize,

LPTHREAD\_STERT\_ROUTUNE funcStart,

LPVOID argList,

DWORD initFlags,

LPDWORD threadAddr);

Детальне пояснення аргументів функції:

1. security – атрибут безпеки, значення за умовчуванням – NULL;
2. stackSize – розмір стеку (в байтах). Значення за умовчуванням – 0, що означає, що розмір стеку буде рівним розміру стеку головного потоку;
3. funcStart – адреса функції потоку;
4. argList – передається як аргумент до функції потоку. Це 32-бітний покажчик на структуру даних типу LPVOID.
5. initFlags – флаг відкладеного запуску. Може приймати значення або 0, або CREATE\_SUSPENDED. Значення 0 вказує, що потік повинен почати виконання функції потоку відразу після своєї ініціалізації. Значення CREATE\_SUSPENDED означає відкладене виконання потокової функції. Виконання в цьому виподку почнеться після виклику функції ResumeThread(HANDLE hThread), де в якості вхідного параметру буде переданий thread\_name.

У випадку коректного виконання функції створення нового потоку буде повернутий валідний ідентифікатор потоку, який був присвоєний йому операційною системою, в іншому випадку буде повернуте значення 0.

* + 1. **Планування потоків**

Для виконання всіх потоків операційна система відводить кожному деякий процесорний час. Через це створюється ілюзія одночасного виконання потоків (насправді, для багатопроцесорних комп’ютерів можливий справжній паралелізм). В Windows реалізована система витісняючого планування на основі пріорітетів, в якій завжди виконується потік з найбільшим пріорітетом, що знаходиться в стані готовності до виконання. Обраний для виконання потік виконується на протязі деякого часу, що називається квантом. Квант визначає, скільки часу буде виконуватися потік, поки операційна система не перерве його. По закінченню кванту операційна система перевіряє стан готовності іншого потоку з таким (або більшим) рівнем пріорітету. Якщо таких потоків немає, даному потокові виділяється ще один квант часу. Однак потік може не повністю використати виділений час – як тільки інший потік з більш високим пріорітетом готовий до виконання, поточний потік витісняється, навіть якщо його квант не закінчився.

Квант не вимірюється ні в яких одиницях часу, а виражається цілим числом. Для кожного потоку зберігається поточне значення його кванту. Коли потоку виділяється квант процесорного часу, це означає, що його квант встановлюється в початкове значення. Наприклад, для Win2000 Professional початкове значення кванту дорівнює 6, а для Win2000 Server – 36.[8]

У випадку виникнення переривання від таймеру з кванту потоку віднімається 3 і так до тих пір, поки квант не буде рівним 0. Частота таймеру залежить від апаратної платформи. Наприклад, для більшості однопроцесорних х86 систем від складає 10 мс, а для більшості багатопроцесорних х86 – 15мс.[8]

В будь – якому випадку операційна система повинна визначити, який потік буде виконуватись наступним. Обрав новий потік, операційна система переключає контекст виконання потоку. Ця операція заключається в збереженні параметрів потоку, що виконується (регістри процесору, покажчик на стек ядра та стек користувача, покажчик на адресний простір, в якому виконується потік і т.д.), та завантаженні аналогічних параметрів для іншого потоку, після чого починається виконання нового потоку.[8]

Планування в Windows здійснюється на рівні потоків, а не процесів. Це здається зрозумілим, так як самі процеси не виконуються, а тільки надають ресурси та контекст для виконання потоків. Через це при плануванні потоків, система не звертає уваги на те, якому процесу він належить. Наприклад, якщо процес А має десять готових до виконання потоків, а процес Б – два і всі дванадцять потоків мають однаковий пріорітет, то кожен потік отримає 1/12 процесорного часу.

* + 1. **Завершення виконання потоків**

Потік виконується до тих пір, поки не відбудеться одна з подій :

1. Функція повертає значення потоку;
2. потік викликає функцію ExitThread();
3. інший потік викликає функцію ExitProcess();
4. інший потік викликає функцію TerminateThread() з дескриптором потоку;
5. інший потік викликає функцію TerminateProcess() з дескриптором процесу.

Також потік автоматично припиняє виконання функції потоку у випадку, коли батьківський потік завершує свою роботу. Через це батьківський потік після виконання своєї роботи повинен очікувати завершення роботи всіх потоків – нащадків. Для цього необхідно викликати функцію WaitForSingleObject() або WaitForMultipleObject().[5]

Для перевірки статусу виконання потоку існує функція GetExitCodeThread(), яка приймає дескриптор потоку, а повертає булеве значення: якщо потік був завершений успішно, повертається ненульове значення, у випадку неуспішності завершення потоку повертається 0. Також за допомогою цієї функції можна дізнатися поточний статус потоку, наприклад, якщо потік виконується, то функція повертає значення STILL\_ACTIVE.[9]

* + 1. **Пріорітети потоків**

В Windows існує 32 рівні пріорітетів, від 0 до 31. Вони групуються таким чином: 31 – 16 рівні реального часу; 15 – 1 – динамічні рівні; 0 – системний рівень, зарезервований для потоку анулювання сторінок (zero – page thread).

Процесам може бути назначений клас пріорітету і потокам в процесах також може бути назначений пріорітет, що може бути меншим або більшим за батьківський.

Класи пріорітетів процесів [7]:

1. Real time class (значення 24);
2. High class (значення 13);
3. Above normal class (значення 10);
4. Normal class (значення 8);
5. Below normal class (значення 6);
6. Idle class (значення 4).

Класи пріорітетів потоків [7]:

1. THREAD\_PRIORITY\_LOWEST - два рівні ничже рівня пріорітету процесу;
2. THREAD\_PRIORITY\_BELOW\_NORMAL - один рівень нижче пріорітету процесу;
3. THREAD\_PRIORITY\_NORMAL - пріорітет процесу;
4. THREAD\_PRIORITY\_ABOVE\_NORMAL - один рівень вище пріорітету процесу;
5. THREAD\_PRIORITY\_HIGHEST - два рівні вище рівня пріорітету процесу;
6. THREAD\_PRIORITY\_TIME\_CRITICAL - встановлює базовий пріорітет потоку для Real time класу в 31, для інших – в 15;
7. THREAD\_PRIORITY\_IDLE - встановлює базовий пріорітет потоку для Real time класу в 16, для інших – в 1.

Перевірити пріорітет потоку можна за допомогою функції GetThreadPriority(), яка приймає в якості аргументу дескриптор потоку. Встановити пріорітет можна за допомогою функції SetThreadPriority(), яка приймає в якості аргументів дескриптор потоку та назву пріорітету.

* + 1. **Функції очікування**

Існюють три типи функцій очікування, які дозволяють потоку блокувати своє виконання залежно від результатів виконання перевірки стану об’єкта синхронізації:

1. Одиночні;
2. множинні;
3. попереджувальні

Одиночні функції синхронізації SignalObjectAndWait, WaitForSingleObject, WaitForSingleObjectEx використовують ідентифікатор (типу HANDLE) об’єкта синхронізації і блокують процес, якщо об’єкт синхронізації знаходиться в забороненому стані; функції закінчують своє виконання, якщо:

1. указаний об’єкт знаходиться в дозволеному стані;
2. вичерпаний час очікування. Час очікування може бути встановлений в INFINITE, що означає необмежений час очікування.

Функція SignalObjectAndWait дозволяє потоку, з якого відбувся виклик, встановити стан об’єкта в дозволений і чекати дозволеного стану іншого об’єкту:

Bool SignalObjectAndWait(

HANDLE hObjectToSignal, //ідентифікатор сигнального об’єкту

HANDLE hObjectToWaitOn, //ідентифікатор об’єкта – сигналу

DWORD dwMilliseconds, // тайм – аут в мілісекундах

BOOL bAlertable); //прапор раннього виконання

Сигнатура функції очікування конкретного потоку: WaitForSingleObject(HANDLE hHandle, DWORD wdMilliseconds). Перший параметр – дескриптор потоку, другий – час очікування в мілісекундах. Значення другого параметру INFINITE вказує на відсутність часового проміжку очікування – даний потік буде очікувати сигналу завершення роботи іншого потоку. [9]

DWORD WaitForSingleObjectEx(

HANDLE hHANDLE,

DWORD dwMilliseconds,

BOOL bAllertable);

Перший параметр вказує на ідентифікатор об’єкта, від якого очікується сигнал, другий параметр вказує на час тайм – ауту, третій є прапором раннього виконання (для операцій вводу/виводу).

Множинні функції WaitForMultipleObjects, WaitForMultipleObjectsEx, MsgWaitForMultipleObjects, MsgWaitForMultipleObjectsEx дозволяють потоку, з якого вони були викликані, визначати масив, який містить один або декілька ідентифікаторів об’єктів синхронізації; функції закінчують своє виконання, якщо:

1. стан одного або кількох (усіх) об’єктів дозволений – можна вказувати об’єкти для виклику функцій;
2. вичерпаний час очікування. Час очікування може бути встановлений в INFINITE, що означає необмежений час очікування.

Функції MsgWaitForMultipleObjects, MsgWaitForMultipleObjectsEx дозволяють визначити об’єкт події введенням масивів ідентифікаторів об’єктів, для чого потрібно визначити тип уведення в черзі введення потоку. Наприклад, потік може використати MsgWaitForMultipleObjects для власного блокування до тих пір, поки стан об’єкта не буде встановлено і дозволено введення з маніпулятора в черзі введення потоку. Потік може використати GetMessage і PeekMessage для оброблення введення.[9]

Формати виклику цих функцій:

DWORD WaitForMultipleObjects(

DWORD nCount,

CONST HANDLE \*lpHandles,

BOOL bWaitAll,

DWORD dwMilliseconds);

Сигнатура функції WaitForMultipleObjects має два додаткових параметри порівняно з WaitForSingleObject: другий параметр – масив дескрипторів потоків, завершення роботи яких повинен очікувати даний потік, третій параметр – булеве значення, якщо TRUE – даний потік буде очікувати завершення виконання всіх потоків, FALSE – очікування завершення виконання будь-якого одного потоку.

DWORD WaitForMultipleObjectsEx(

DWORD nCount,

CONST HANDLE \*lpHandles,

BOOL bWaitAll,

DWORD dwMilliseconds,

BOOL bAlertable);

DWORD MsgWaitForMultipleObjects(

DWORD nCount,

LPHANDLE pHandles,

BOOL fWaitAll,

DWORD dwMilliseconds,

DWORD dwWakeMask) //тип події уведення для очікування

DWORD MsgWaitForMultipleObjects(

DWORD nCount,

LPHANDLE pHandles,

BOOL fWaitAll,

DWORD dwMilliseconds,

DWORD dwWakeMask);

Попереджувальні функції: MsgWaitForMultipleObjectsEx, SignalObjectAndWait, WaitForMultipleObjectsEx, WaitForSingleObjectEx можуть додатково виконувати попереджувальні операції. Вони також можуть закінчуватися з досягненням деякої умови, але також і в разі появи в системній черзі введення – виведення інформації, або віддаленого виклику процедур.[9]

Функції очікування можуть змінювати стан об’єкта синхронізації, причому змін зазнають тільки ті об’єкти, від яких залежить вихід з функції очікування і розблокування потоку відповідно[9]. Таким чином, функції змінюють такі типи об’єктів синхронізації:

1. лічильник семафора зменшується на одиницю і досягає забороненого стану, якщо дорівнює нулю;
2. об’єкту мьютексу, подія автоскидання встановлюються в заборонені;
3. стан таймера синхронізації заборонений.
   * 1. **Семафори в Win32**

В бібліотеці Win32 семафори є багатозначними, реалізовані за допомогою спеціального типу та декількох функцій. Функції WaitForSingleObject() та ReleaseSemaphore() є аналогами P() та V(), відповідно. Початкове та максимальне значення лічильника вказуються при створенні об’єкта семафора. Початкове значення має бути більшим або дорівнювати нулю та меншим або дорівнює максимальному значенню. Максимальне значення лічильника має бути більшим за нуль. Значення семафора повинно бути в діапазоні від нуля до максимального значення.[5]

Функція створення семафора:

HANDLE CreateSemaphore(

LPSECURITY\_ATTRIBUTES lpSemaphoreAttributes,

LONG initialCount,

LONG lMaximumCount,

LPCTSTR lpName);

Параметри:

1. lpSemaphoreAttributes – покажчик на структуру SECURITY\_ATTRIBUTES, яка визначає, чи може ідентифікатор, що повертається, бути успадкованим дочірнім процесом. Якщо покажчик установлено в NULL, то покажчик не може бути успадкованим;
2. initialCount – початкове значення лічильника семафора. Не може бути більшим за максимальне значення або меншим за нуль. Якщо покажчик дорівнює нулю, то семафор знаходиться в забороненому стані;
3. lMaximumCount – максимальне значення семафора, повинно бути більшим за нуль;
4. lpName – визначає ім’я об’єктау вигляді рядка, що має закінчуватись нулем. Якщо параметр – NULL, то семафор не має ім’я. Інші процеси можуть використовувати це ім’я для отримання управління через виклики CreateSemaphore() та OpenSemaphore().

Не вважається, що потік володіє семафором – один потік може виконувати WaitForSingleObject() на цьому семафорі, а інший потік може викликати ReleaseSemaphore().[4]

Функція OpenSemaphore() повертає ідентифікатор створеного семафора.

HANDLE OpenSemaphore(

DWORD dwDesiredAccess, //прапор доступу

BOOL bInheritHandle, // прапор наслідування

LPSTSTR lpName) // ім’я семафора

Функція ReleaseSemaphore() збільшує лічильник семафора на вказане значення. Отже, багато потоків можуть бути розблокованими в результаті одного виклику даної функції.

BOOL ReleaseSemaphore(

HANDLE hSemaphore, //ідентифікатор семафора

LONG lReleaseCount,//інкремент лічильника

LPLONG lpPreviousCount); //адреса попереднього лічильника

Якщо нове значення семафора після операції інкременту буде більше за максимальне значення семафора, то функція повертає значення FALSE, значення семафора залишається незмінним. Останній аргумент є адресою на значення типу long, що вказує на попереднє значення лічильника. [4]

Для звільнення семафору необхідно викликати функцію CloseHandle(hSemaphore). Об’єкт семафору буде знищений, коли останній handle буде закритим. Закриття hanlde не впливає на лічильник семафору. Отже, функція ReleaseSemaphore() повинна бути викликана обов’язково перед закриттям handle або перед завершенням процесу. В іншому випадку, неменуче очікування на невизначений час або тайм – аут, якщо він був встановлений.

* + 1. **Взаємні виключення в Win32**

Об’єкт взаємного виключення – це об’єкт синхронізації, який встановлено в сигнальний стан, коли жоден потік не володіє ним, та в несигнальний, коли є потік, що володіє ним. Лише один потік одночасно може володіти об’єктом мьютексу, завдання якого заключається в координуванні ексклюзивного доступу до загального ресурсу. Наприклад, для запобігання запису в спільну пам’ять двума потоками в один момент часу, кожен потік очікує захоплення мьютексу перед виконанням коду для доступу до пам’яті. Після запису до розподіленої пам’яті потік звільняє об’єкт мьютексу. [5]

Потік використовує функції CreateMutex та CreateMutexEx для створення нового мьютексу. Створюючий потік також може виконати запит на негайне захоплення мьютексу та також може присвоїти йому ім’я.[3]

Сигнатура функції CreateMutex:

CreateMutex(

LPSECURITY\_ATTRIBUTES lpMutexAttributes,

BOOL bInitialOwner,

LPCTSTR lpName);

lpMutexAttributes вказує на структуру SECURITY\_ATTRIBUTES. Якщо параметр – NULL, то дескриптор мьютексу не може бути наслідуваним підпроцесами.

bInitialOwner – булеве значення, що вказує на початкове захоплення мьютексу

потоком, що створив його. Якщо FALSE, то потік, який створив мьютекс, не захоплює його.[6]

lpName – назва мьютексу.

Функція повертає значення дескриптору мьютексу, якщо помилка – NULL. Для перегляду розширеної інформації про помилку необхідно викликати функцію GetLastError.[6]

Сигнатура функції CreateMutexEx має вигляд:

CreateMutexEx(

LPSECURITY\_ATTRIBUTES lpMutexAttributes,

LPCTSTR lpName,

DWORD dwFlags,

DWORD dwDesiredAccess);

Параметри lpMutexAttributes, lpName аналогічні параметрам функції CreateMutex. dwDesiredAccess – маска доступу до мьютекса.

Потоки в інших процесах можуть відкрити доступ до дескриптору мьютекса за допомогою функції OpenMutex з його іменем в якості параметру. Для отримання доступу до безіменного мьютексу з іншого процесу, необхідно використовувати функцію DuplicateHandle або механізм наслідування дескрипторів. [3]

Будь – який потік з дескриптором мьютексу може використовувати будь – яку з функцій очікування (див. п. 1.2.5) для запиту на захоплення об’єкту мьютекса. Якщо в цей момент мьютекс захоплений іншим потоком, то даний потік блокується до звільнення мьютексу володіючим потоком за допомогою функції ReleaseMutex.

Якщо потік терміново переривається без звільнення мьютексу, то припускається, що об’єкт мьютексу є неконтрольованим.

Очікуючий потік може отримати доступ до неконтрольованого об’єкту мьютекса, але функція очікування поверне значення WAIT\_ABANDONED для повідомлення неконтрольованості мьютекса. Неконтрольований статус мьютекса сповіщає про виникнення помилки та про недетермінований стан спільного ресурсу, що був захищений даним мьютексом.

* + 1. **Критичні секції в Win32**

Об’єкт критичної секції надає механізм синхронізації, аналогічний до механізму, що реалізується мьютексом, за виключенням того, що критична секція може використовуватись потоками лише зі спільного процесу. Події, мьютекси, семафори також можуть бути використані в однопроцесових програмних застосунках, але критичні секції надають більш швидкий апаратно – програмний механізм вирішення завдання взаємного виключення. Подібно об’єкту мьютекса, критичною секцією може володіти лише один потік в кожен момент часу, що робить її корисною для захисту спільних ресурсів від неконтрольованого доступу. На відміну від мьютексів, немає можливості сповістити, що критична секція є недійсною.

Процес відповідальний за виділення пам’яті для використання критичною секцією. Зазвичай, цього можна досягти просто задекларувавши змінну типу CRITICAL\_SECTION. Перед її використанням потоками процесу, необхідно ініціалізувати критичну секцію, викликавши функцію InitializeCriticalSection чи InitializeCriticalSectionAndSpinCount. [1]

Сигнатура функції InitializeCriticalSection має вигляд:

InitializeCriticalSection(

\_Out\_ LPCRITICAL\_SECTION lpCriticalSection

);

Функція приймає один вихідний параметр, який є посиланням на об’єкт критичної секції.

Сигнатура функції InitializeCriticalSectionAndSpinCount має вигляд:

BOOL WINAPI InitializeCriticalSectionAndSpinCount(

\_Out\_ LPCRITICAL\_SECTION lpCriticalSection,

\_In\_  DWORD              dwSpinCount);

Функція має додатковий вхідний параметр, що вказує на кількість спінів для об’єкту критичної секції. В однопроцесорних системах, кількість спінів ігнорується та параметр встановлюється в нуль. В багатопроцесорних системах, якщо критична секція є недоступною, потік, що викликав функцію, входить в цикл, який ітерується певну кількість разів, що вказана в параметрі dwSpinCount, перевіряючи, чи критична секція звільнилась. Якщо до закінчення ітерацій критична секція не є звільненою, потік блокується до звільнення критичної секції. [5]

Потік використовує функції EnterCriticalSection або TryEnterCriticalSection для запиту на захоплення критичної секції. Функція LeaveCriticalSection використовується для звільнення захоплення критичної секції потоком. Якщо критична секція є захопленою на момент виклику функції EnterCriticalSection, то поток блокується на невизначений проміжок часу (до звільнення критичної секції іншим потоком). Для порівняння, коли мьютекс використовується для вирішення завдання взаємного виключення, функції очікування визначають інтервал очікування. Функція TryEnterCriticalSection виконує спробу увійти до критичної секції без блокування потоку, що викликав функцію. [1]

Коли потік знаходиться в критичній секції, він може робити додаткові виклики функцій EnterCriticalSection, TryEnterCriticalSection без блокування виконання. Це попереждує блокування самого себе для очікування свого виходу з критичної секції. Для виходу з критичної секції, потік має викликати функцію LeaveCriticalSection для кожного входу в критичну секцію. Також слід зазначити, що не існує детермінованого порядку доступу до критичної секції заблокованими потоками. [5]

Кожен потік процесу може викликати функцію видалення критичної секції DeleteCriticalSection для звільнення системних ресурсів, що були виділені для даного об’єкту критичної секції. Після виклику цієї функції, критична секція не може бути використана для синхронізації.

* + 1. **Монітори в Win32.**

Механізм моніторів відсутній в бібліотеці Win32.

* + 1. **Події в Win32**

Механізм подій призначений виключно для синхронізації потоків. Створення події виконується зі змінною типу HANDLE за допомогою функції CreateEvent():

HANDLE CreateEvent(

LPSECURITY\_ATTRIBUTES //адреса атрибутів захисту

BOOL //прапор ручної встановки події  
BOOL //прапор початкового стану

LPCTSTR); //адреса об’єкта події

У разі використання механізму подій очікування події реалізуються за допомогою однієї з функцій очікування, а сигнал про настання події – SetEvent():

BOOL SetEvent(

HANDLE подія);

Для роботи з подіями використовують такі функції:

1. OpenEvent – повертає значення існуючого об’єкта – події;
2. PulseEvent – забезпечує встановлення стану події в сигнальний і наступне перемикання його в несигнальний стан після реалізації посилання сигналу очікуваним потокам;
3. ResetEvent – виконує скидання події (встановлює стан події в несигнальний).
   1. **Висновки до першого розділу**
4. Багатопотоковість дозволяє спростити складність програми в деяких випадках за рахунок використання спільного адресного простору, зменшити часові витрати на створення нового потоку в порівнянні зі створенням нового процесу, підвищити продуктивність процесу за рахунок розпаралелювання процесу обчислень та операцій вводу/виводу.
5. В ОС Windows реалізована схема витісняючого планування, що базується на пріорітетах потоків. Планування на рівні потоків, тому процес лише надає адресний простір та ресурси для виконання потоків.
6. ОС виконує квантування часу виконання потоків, що створює ілюзію одночасного виконання декількох задач в однопроцесорній системі.
7. Бібліотека Win32 дозволяє створювати нові потоки в контексті батьківських потоків, встановлювати атрибути безпеки, передавати посилання на потокову функцію та структуру даних, що містить параметри потокової функції, встановлювати відкладений запуск.
8. У бібліотеці Win32 реалізовано декілька механізмів для припинення виконання потоку: після завершення виконання потокової функції; у разі припинення виконання батьківського потоку; при викликові ExitThread та TerminateThread з потоку; при викликові ExitProcess з потоку іншого процесу. Також бібліотека надає функції для перевірки статусу виконання потоку.
9. Потокам може бути назначений пріорітет виконання відносно батьківського пріорітету (або пріорітету процесу, якщо потік не має батьківського). Встановити пріорітет можна за допомогою функцій Win32 SetPrioroty, а дізнатись пріорітет процесу – за допомогою GetPriority. Механізм пріорітетів дозволяє більш гнучко керувати послідовністю виконання потоків, застосовувати різні алгоритми планування виконання задач та реалізовувати черги з пріорітетами з мешними затратами.
10. Бібліотека Win32 реалізує функції очікування, які здатні блокувати потік, що їх викликав, до отримання сигналу від об’єкту синхронізації (або від декількох об’єктів у випадку множинної функції очікування). Також можна вказати тайм – аут очікування, після якого потік буде розблокованим, а у випадку множинної функції очікування – очікування сигналу від всіх вказаних об’єктів синхронізації або від одного з них.
11. У Win32 реалізовані множинні семафори, що дозволяє з меншими ресурсними затратами та простішою логікою вирішувати завдання синхронізації та взаємного виключення у випадку кількості потоків, що більша за два, а також надає функції для створення, збільшення лічильника, звільнення.
12. Win32 надає можливість використовувати мьютекси та критичні секції для вирішення завдання взаємного виключення (мьютекси – в межах одного процесу та в багатопроцесному середовищі, критичні секції – в межах одного процесу). Критичні секції надають більш швидкий апаратно – програмний механізм захисту спільних ресурсів та є більш високорівневими, ніж події, семафори з точки зору застосування в програмах. На відміну від інших об’єктів синхронізації блокування при вході в критичну секцію не має тайм – ауту, тому бібліотека надає функції, що не блокують потік при вході в критичну секцію, якщо вона зайнята іншим потоком.
13. Win32 реалізує об’єкти-події Event для вирішення завдання синхронізації, надає методи для створення, встановлення та скидання сигнального стану події, а також його встановлення з наступним скиданням в несигнальний стан. Також при створенні події можна встановити флаг ручного скидання в несигнальний стан, що дозволяє синхронізувати виконання більше двух потоків, що, в свою чергу, зменшує складність програми та кількість ресурсів, що виділяються ОС.
14. На основі огляду бібліотек Win32, можна зробити висновок, що дана бібліотека має всі необхідні засоби для створення паралельних програм будь-якої складності.

РОЗДІЛ 2. РОЗРОБКА ПРОГРАМИ ПРГ1 ДЛЯ ПКС СП

Математична задача:

Структура ПКС СП зображена на рисунку 2.1

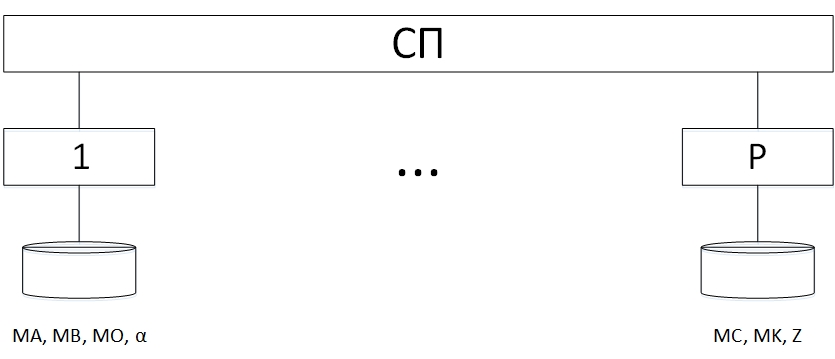


Рис. 2.1. Структура ПКС СП для ПРГ 1

**2.1 Розробка паралельного математичного алгоритму.**

Паралельний математичний алгоритм відповідно до рекомендованої методичної літератури [9] можна подати у вигляді наступних двох етапів:

1. *zi = min(ZH),*
2. *minZ = min(minZ,zi), i = ;*

СР: MB, , minZ

Пояснення до констант:

1. N – розмірність векторів та матриць;
2. P – кількість ядер;
3. H = N/P – кількість рядків матриць та векторів, які будуть використані кожною задачею;
4. *i =* – номер задачі.
   1. **Розробка алгоритмів задач**

**Задача Т1**

|  |  |
| --- | --- |
| Крок алгоритму | ТС/КД |
| Ввід MO, MB, |  |
| Сигнал задачам Т2...Tp про завершення вводу даних |  |
| Очікування сигналу від Тp про завершення вводу даних |  |
| Обчислення z1 = min(ZH) |  |
| Обчислення minZ = min(minZ, z1) | КД |
| Сигнал задачам Т2…Tp про завершення обчислення minZ |  |
| Очікування сигналів від T2…Tp про завершення обчислення minZ |  |
| Копіювати | КД |
| Обчислення виразу |  |
| Очікування сигналів від T2…Tp про завершення обчислення |  |
| Вивід МА |  |

**Задача Ti (i = )**

|  |  |
| --- | --- |
| Крок алгоритму | ТС/КД |
| Очікування сигналу від Т1 про завершення вводу даних |  |
| Очікування сигналу від Тp про завершення вводу даних |  |
| Обчислення zi = min(ZH) |  |
| Обчислення minZ = min (minZ, zi) | КД |
| Сигнал задачам Т1…Т(i-1), T(i+1)..Tp про завершення обчислення minZ |  |
| Очікування сигналів від Т1…Т(i-1), T(i+1)..Tp про завершення обчислення minZ |  |
| Копіювати | КД |
| Обчислення виразу |  |
| Сигнал задачі Т1 про завершення обчислення |  |

**Задача Тp**

|  |  |
| --- | --- |
| Крок алгоритму | ТС/КД |
| Ввід MC, Z, MK |  |
| Сигнал задачам Т1…Т(p-1) про завершення вводу даних |  |
| Очікування сигналу від Т1 про завершення вводу даних |  |
| Обчислення zp = min(ZH) |  |
| Обчислення minZ = min(minZ, zp) | КД |
| Сигнал задачам Т1…Т(p-1) про завершення обчислення minZ |  |
| Очікування сигналів від Т1…Т(p-1) про завершення обчислення minZ |  |
| Копіювати | КД |
| Обчислення виразу |  |
| Сигнал задачі Т1 про завершення обчислення |  |

* 1. **Розробка схеми взаємодії процесів**

На основі алгоритму для всіх задач, розглянутого в попередньому розділі, було розроблена структурна схема взаємодії задач (рис. Рисунок 2.2). На схемі зображено захищений модуль ProtectedModule, за допомогою якого виконується вся взаємодія між задачами (англ. task). На схемі позначені потрібні для вирішення задач синхронізації та взаємного виключення захищені функції, процедури та входи.

На структурній схемі взаємодії задач введено такі засоби для вирішення задач взаємного виключення та синхронізації:

Входи (вирішення завдання синхронізації):

1. waitInput – очікування завершення вводу даних;
2. waitMinZCalcFinish – очікування завершення обчислення min(Z);
3. readyForOuptut – очікування задачею завершення обчислення математичного виразу задачами .

Захищені функції, що дозволяють копіювати спільний ресурс:

1. readAlpha – копіювання спільного ресурсу – α;
2. readMB – копіювання спільного ресурсу – матриці MB;
3. readMinZ – копіювання спільного ресурсу – мінімального значення вектора Z.

Захищені процедури для посилання сигналів та запису спільних ресурсів:

1. inputFinishSignal – сигнал про завершення вводу даних;
2. finishCalcsSignal – сигнал про завершення обчислення математичного виразу;
3. finishMinZCalcSignal – сигнал про завершення обчислення мінімального значення Z;
4. setMinZ – запис мінімального значення Z в змінну minZ;
5. initMB – запис матриці MB;
6. initAlpha – запис значення константи α.

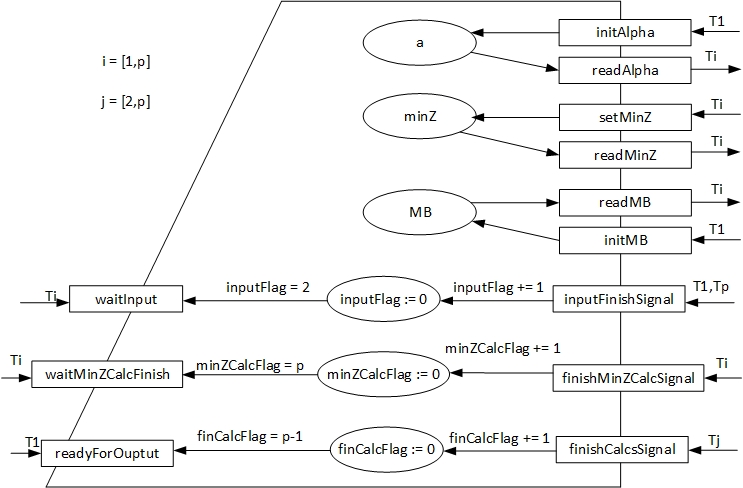


Рис. 2.2. Структурна схема взаємодії задач для ПРГ1

* 1. **Розробка програми ПРГ1**

Програма ПРГ1 згідно технічного завдання розроблення на мові програмування Ада. Для взаємодії процесів використовується захищений модуль**.** Програма складається з трьох файлів: main.adb – головний файл програми, в якому описується головна процедура main; matrixoperations.adb – generic-модуль, який приймає значення N – розмірність матриць та векторів, а також містить тіла процедур та функцій для роботи з матрицями та векторами в рамках завдання; matrixoperations.ads – файл специфікації generic-модуля, який містить сигнатури функцій та процедур, тіла яких описані в matrixoperations.adb, специфікацію типів Matrix, Vector, підтипу Index.

Головний файл програми має опис тіла головної процедури, яка складається з:

1. Визначення змінних N(розмірність векторів та матриць), P(кількість задач), H(кількість рядків матриць та векторів, які будуть оброблені кожною задачею), StartTime (початковий момент часу виконання обчислень), DiffTime (час виконання завдання);
2. процедури Start;
3. тіла процедури main, в якій виконується ввід даних з консолі (N та P), обчислення H, ініціалізація StartTime, а також виклик процедури Start;

Процедура Start – процедура, яка містить:

1. специфікацію та реалізацію захищеного модулю ProtectedModule;
2. специфікацію та реалізацію задачного типу ThreadTask;
3. визначення змінних матриць та векторів, що не є спільними ресурсами;
4. екземпляр generic-пакету, який ініціалізується з параметром N;
5. специфікацію типу ThreadTaskPointer, що є посиланням на задачу ThreadTask;
6. специфікацію типу TaskArray, що є масивом елементів типу ThreadTaskPointer;
7. тіло процедури compareWithMinZ, яка порівнює вхідний параметр зі значенням minZ та змінює minZ на значення вхідного параметру у випадку, якщо той є меншим за поточне значення minZ;
8. тіло процедури calcMatrixEquation, яка обчислює значення виразу згідно завданню.
   1. **Тестування програми ПРГ1**

Метою проведення тестування є оцінка коефіцієнтів прискорення і коефіцієнтів ефективності для розроблених програм при їх виконанні на реальній паралельній обчислювальній системі. Для визначення вищевказаних коефіцієнтів проведений ряд експериментів із різними розмінностями операндів (N = 1000, 2000, 3000) і різною кількістю працюючих ядер ( P = 1, 2, 3, 4, 5, 6).

Для виміру часу використовується стандартна процедура Clock мови програмування Ада пакету Ada.Real\_Time, що повертає поточний час системи. Різниця між часом системи після завершення обчислення і часом при запуску програми дає час виконання обчислення, що вимірюється в секундах.

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 табл. 2.1 – 2.3.

Таблиця 2.1. Час виконання програми ПРГ1.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| N | Т1,с | Т2, с | Т3,с | Т4,с | Т5, с | Т6, с |
| 1000 | 18,2 | 9,91 | 7,95 | 6,02 | 5,11 | 6,1 |
| 2000 | 189,76 | 95,13 | 71,93 | 54,27 | 43,84 | 50,56 |
| 3000 | 644,26 | 322,3 | 245,33 | 184,38 | 148,87 | 167,58 |

На основі даних із таблиці 2.1 виконано розрахунок значень коефіцієнтів прискорення, які наведені в таблиці 2.2.

Таблиця 2.2. Коефіцієнти прискорення для програми ПРГ1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| N | Кількість процесорів (P) | | | | | |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 1000 | 1,00 | 1,84 | 2,29 | 3,02 | 3,56 | 2,98 |
| 2000 | 1,00 | 1,99 | 2,64 | 3,50 | 4,33 | 3,75 |
| 3000 | 1,00 | 2,00 | 2,63 | 3,49 | 4,33 | 3,84 |

Коефіцієнти ефективності (таблиця 2.3) обчислено за даними таблиці 2.2.

Таблиця 2.3. Коефіцієнти ефективності для програми ПРГ1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| N | Кількість процесорів | | | | | |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 1000 | 100,00 | 91,83 | 76,31 | 75,58 | 71,23 | 49,73 |
| 2000 | 100,00 | 99,74 | 87,94 | 87,41 | 86,57 | 62,55 |
| 3000 | 100,00 | 99,95 | 87,54 | 87,35 | 86,55 | 64,07 |

Використовуючи таблиці Таблиця 2.1 – 2.3 побудовано графіки зміни коефіцієнтів прискорення і ефективності, а також зміни часу роботи програми в залежності від *N* і .

Рис. 2.3. Графіки часу роботи ПРГ1 в залежності від N та P

Рис. 2.4. Графіки зміни коефіцієнту прискорення при різних N в залежності від кількості ядер P

Рис. 2.5. Графіки зміни коефіцієнту ефективності при різних N в залежності від кількості ядер P

* 1. **Висновки до розділу 2**

Виконано розробку програми ПРГ1 для ПСК СП з використанням мови Ада та захищених модулів. Тестування програми показало наступне:

* Використання багатоядерної ПКС та програми ПРГ1 забезпечує скорочення часу обчислення заданої математичної задачі. Значення К*П* лежить в межах від 1,84 до 4,33.
* Максимальне значення К*П* =4,33 забезпечує ПКС з Р = 5 та N = 2000 i N = 3000.
* Мінімальне значення К*П* = 1,84 виявлено у ПКС з Р = 2 та N = 1000.
* Найефективніше програма ПРГ1 використовує ПКС з Р = 2 та N = 3000. При цьому Ке = 99,95%.
* Зі зростанням Р від 1 до 6 Ке  спадає від 100,00% до 49,73% при N = 1000.
* Стрімке спадання Ке (близько 20%) спостерігається при N = 1000, кількість ядер змінюється з 2 до 3, з 5 до 6; при N = 2000 та 3000, кількість ядер змінюється з 5 до 6.
* Найшвидший час виконання обчислень для N = 1000, 2000, 3000 спостерігається при p = 5, при цьому коефіцієнт ефективності зменшується максимум на 30%.
* При збільшенні кількості потоків (більше ніж ядер) показники Кп та Ке різко зменшуються.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Critical Sections Objects [Електронний ресурс]. – Режим доступу: https://msdn.microsoft.com/en-us/library/windows/desktop/ms682530(v=vs.85).aspx – дата звернення 27.02.2016 – Назва з екрану.

# Multithreading with C and Win32 [Електронний ресурс]. – Режим доступу:

https://msdn.microsoft.com/en-us/library/aa270957(v=vs.60).aspx – дата звернення 28.02.2016 – Назва з екрану.

1. Mutex Objects [Електронний ресурс]. – Режим доступу: https://msdn.microsoft.com/en-us/library/windows/desktop/ms684266(v=vs.85).aspx – дата звернення 27.02.2016 – Назва з екрану.
2. Semaphore Objects [Електронний ресурс]. – Режим доступу: https://msdn.microsoft.com/en-us/library/windows/desktop/ms685129(v=vs.85).aspx – дата звернення 27.02.2016 – Назва з екрану.
3. Richard H. Carver, Kuo-Chung Tai. Modern Multithreading. Implementing, Testing, Debugging Multithreaded Java and C++/Pthreads/Win32 Programs – К.: A JOHN WILEY & SONS, INC., PUBLICATION, 2006 – 465 с.
4. Win32 multithreading and synchronization [Електронний ресурс]. – Режим доступу: http://faq.programmerworld.net/programming/win32-multithreading-and-synchronization.html - дата звернення 26.02.2016 – Назва з екрану.
5. Windows Application Interface(WinAPI) [Електронний ресурс].− Режим доступу: https://msdn.microsoft.com/en-us/library/windows/desktop/ff818516− дата звернення 29.02.2016. – Назва з екрану.
6. Современные операционные системы. Планирование заданий, процессов и потоков [Електронний ресурс]. – Режим доступу: http://www.intuit.ru/studies/courses/631/487/lecture/11055?page=4 – дата звернення 29.02.2016. – Назва з екрану
7. Жуков І.А., Корочкін О.В. Паралельні та розподілені обчислення: Навч. посібник. – К.: Корнійчук, 2005. – 284 с. - ISBN 996-7599-36-1.

ДОДАТКИ

**Додаток А. Лістинг програми ПРГ1**

GNAT 4.6.4

Copyright 1992-2010, Free Software Foundation, Inc.

Compiling: main.adb (source file time stamp: 2016-03-19 20:32:57)

1. with Ada.Text\_IO, Ada.Integer\_Text\_IO,Ada.Float\_Text\_IO, Ada.Synchronous\_Task\_Control, Ada.Real\_Time, MatrixOperations;

2. use Ada.Text\_IO, Ada.Integer\_Text\_IO,Ada.Float\_Text\_IO, Ada.Synchronous\_Task\_Control, Ada.Real\_Time;

3.

4.

5. ------------------------------------------------------------------

6. -- --

7. -- Програмування паралельних компьютерних систем --

8. -- Курсова робота. ПРГ1. Ада. Захищені модулі --

9. -- --

10. -- Файл: parallelCalcs.ada --

11. -- Завдання: --

12. -- MA = MB\*(MC+MO)\*a + min(Z)\*MK --

13. -- --

14. -- --

15. -- --

16. -- Автор: Кахерський Олег, група IП-31 --

17. -- Дата: 12.03.2016 --

18. -- --

19. ------------------------------------------------------------------

20.

21. procedure Main is

22. n: Integer; --розмірність матриць та векторів

23. p: Integer; --кількість задач

24. h: Integer; --кількість рядків матриць та векторів на одну задачу

25. StartTime: Time; --початок обчислень

26. DiffTime : Time\_Span; --різниця між поточним часом(завершення обчислень) та початковим часом

27. procedure Start is

28. package MatrixOperationsN is new MatrixOperations(N);

29. use MatrixOperationsN;

30. MA,MO,MC,MK: Matrix;

31. Z: Vector;

32. task type ThreadTask(TaskNumber: Integer);

33. protected ProtectedModule is

34. entry waitInput; --бар'єр для синхронізації вводу даних

35. entry waitMinZCalcFinish; --бар'єр для синхронізації обчислень min(Z)

36. entry readyForOuptut; --очікування сигналу від T2-Tp задачею Т1

37. function readAlpha return Integer; --захищене зчитування alpha

38. function readMinZ return Integer; --захищене зчитування min(Z)

39. function readMB return Matrix; --захищене зчитування readMB

40. procedure inputFinishSignal; --надсилання сигналу про завершення вводу даних

41. procedure finishCalcsSignal; --надсилання сигналу про завершення обчислень

42. procedure finishMinZCalcSignal; --надсилання сигналу про завершення обчислень min(z,zi)

43. procedure setMinZ(b: Integer); --встановлення значення minZ

44. procedure initMB(Buff: Matrix); --ініціалізація MB

45. procedure initAlpha(const: Integer);--ініціалізація alpha

46. private

47. inputFlag: integer := 0; --флаг для синхронізації вводу

48. minZCalcFlag : integer := 0; --флаг для синхронізації завершення обчислень min(Z)

49. finCalcFlag : integer := 0; --флаг для синхронізації завершення обчислень мат. виразу

50. --спільні ресурси

51. a, minZ: Integer;

52. MB : Matrix;

53. end ProtectedModule;

54.

55. protected body ProtectedModule is

56.

57. --розблокування задач при inputFlag = 2 (завершення вводу даних в T1, Tp)

58. entry waitInput when inputFlag = 2 is

59. begin

60. null;

61. end waitInput;

62.

63. --розблокування задач при minZCalcFlag = p (завершення обчислення min(Z))

64. entry waitMinZCalcFinish when minZCalcFlag = p is

65. begin

66. null;

67. end waitMinZCalcFinish;

68.

69. --розблокування Т1 при завершенні обчислень в задачах T2..Tp

70. entry readyForOuptut when finCalcFlag = p-1 is

71. begin

72. null;

73. end readyForOuptut;

74.

75. function readAlpha return Integer is

76. begin

77. return a;

78. end readAlpha;

79.

80. function readMinZ return Integer is

81. begin

82. return minZ;

83. end readMinZ;

84.

85. function readMB return Matrix is

86. begin

87. return MB;

88. end readMB;

89.

90. procedure inputFinishSignal is

91. begin

92. inputFlag := inputFlag + 1;

93. end inputFinishSignal;

94.

95. procedure finishCalcsSignal is

96. begin

97. finCalcFlag := finCalcFlag + 1;

98. end finishCalcsSignal;

99.

100. procedure finishMinZCalcSignal is

101. begin

102. minZCalcFlag := minZCalcFlag + 1;

103. end finishMinZCalcSignal;

104.

105. procedure setMinZ(b: in Integer) is

106. begin

107. minZ := b;

108. end setMinZ;

109.

110. procedure initAlpha(const: Integer) is

111. begin

112. a := const;

113. end initAlpha;

114.

115. procedure initMB(Buff: Matrix) is

116. begin

117. MB := Buff;

118. end initMB;

119.

120. end ProtectedModule;

121.

122. procedure compareWithMinZ(const:Integer) is

123. begin

124. Put\_Line("Task is ready for calc minZ");

125. New\_line;

126. if ProtectedModule.readMinZ >= const then

127. Put\_Line("Task is changing minZ");

128. New\_line;

129. ProtectedModule.setMinZ(const);

130. end if;

131. end compareWithMinZ;

132.

133. procedure calcMatrixEquation(a,zi,startIndex,endIndex : Integer; MB: Matrix) is

134. Buf1: Matrix;

135. begin

136. Matrix\_Matrix\_Add(MC,MO,startIndex,endIndex,1,Buf1); --Buf1\_h = MOh+MCh

137. Matrix\_Matrix\_Multiply(Buf1, MB, startIndex, endIndex, a, MA); --MA\_h = Buf1\_h\*MB

138. Matrix\_Matrix\_Add(MA,MK,startIndex,endIndex,zi,MA);

139. end calcMatrixEquation;

140.

141. task body ThreadTask is

142. i: Integer := TaskNumber;

143. ai, zi, startIndex, endIndex: Integer;

144. MBi : Matrix;

145. begin

146. Put\_Line("Task " & Integer'Image(i) & " started");

147. New\_line;

148. startIndex := h\*(i - 1) + 1;

149. endIndex := h\*i;

150. if i = p AND n rem p = 1 then

151. endIndex := endIndex +1;

152. end if;

153. if i = 1 then

154. Put\_Line("Task "& Integer'Image(i) & " is initing data");

155. New\_line;

156. Put\_Line("Task "& Integer'Image(i) & " inited MB");

157. New\_line;

158. ProtectedModule.initMB(Input\_Matrix);

159. Put\_Line("Task "& Integer'Image(i) & " inited MO");

160. New\_line;

161. MO := Input\_Matrix;

162. Put\_Line("Task "& Integer'Image(i) & " inited A");

163. New\_line;

164. ProtectedModule.initAlpha(Input\_Constant);

165. --сигнал задачам о завершении ввода

166. Put\_Line("Task "& Integer'Image(i) & " finished input and send Signal to All");

167. New\_line;

168. ProtectedModule.inputFinishSignal;

169. end if;

170. if i = p then

171. Put\_Line("Task "& Integer'Image(i) & " is initing data");

172. New\_line;

173. Put\_Line("Task "& Integer'Image(i) & " inited MC");

174. New\_line;

175. MC := Input\_Matrix;

176. Put\_Line("Task "& Integer'Image(i) & " inited MK");

177. New\_line;

178. MK := Input\_Matrix;

179. Put\_Line("Task "& Integer'Image(i) & " inited Z");

180. New\_line;

181. Z := Input\_Vector;

182. ProtectedModule.setMinZ(100000000);

183. Put\_Line("Task "& Integer'Image(i) & " finished input and send Signal to All");

184. New\_line;

185. ProtectedModule.inputFinishSignal;

186. else

187. Put\_Line("Task "& Integer'Image(i) & " waits for inputting data in T1, Tp");

188. New\_line;

189. end if;

190. ProtectedModule.waitInput;

191. zi := Min(Z,startIndex, endIndex);

192. Put\_Line("Task "& Integer'Image(i) & " finished calcs minZ and sent signal of finishing");

193. compareWithMinZ(zi);

194. Put\_Line("Task "& Integer'Image(i) & " is copying shared resources");

195. New\_line;

196. --копіювання СР

197. ai := ProtectedModule.readAlpha;

198. zi := ProtectedModule.readMinZ;

199. MBi := ProtectedModule.readMB;

200. Put\_Line("Task "& Integer'Image(i) & " is calculating matrix equation");

201. New\_line;

202. calcMatrixEquation(ai,zi,startIndex,endIndex,MBi);

203. if i = 1 then

204. Put\_Line("Task "& Integer'Image(i) & " is waiting for signal from all(finishing calc matrix equation)");

205. New\_line;

206. ProtectedModule.readyForOuptut;

207. DiffTime := Clock - StartTime;

208. Put\_Line("Calculating time = ");

209. Put(Float(To\_Duration(DiffTime)));

210. New\_line;

211. if n <= 8 then

212. Put\_Line("Result:");

213. New\_line;

214. Output\_Matrix(MA);

215. end if;

216. Get(H);

217. else

218. Put\_Line("Task "& Integer'Image(i) & " is sending signal to T1 (finishing calc matrix equation)");

219. New\_line;

220. ProtectedModule.finishCalcsSignal;

221. end if;

222. Put\_Line("Task " & Integer'Image(i) & " finished");

223. New\_line;

224. end ThreadTask;

225. type ThreadTaskPointer is access ThreadTask;

226. type TaskArray is array (1 .. p) of ThreadTaskPointer;

227. tArray: TaskArray;

228. begin

229. for I in 1 .. p loop

230. tArray(I) := new ThreadTask(I);

231. end loop;

232. end Start;

233.

234. begin

235. Put("Program started");

236. New\_line;

237. Put("Input N");

238. New\_line;

239. Get(n);

240. Put("Input P");

241. New\_line;

242. Get(p);

243. StartTime := Clock;

244. h := n/p;

245. Start;

246. end Main;

246 lines: No errors

GNAT 4.6.4

Copyright 1992-2010, Free Software Foundation, Inc.

Compiling: matrixoperations.adb (source file time stamp: 2016-03-19 18:55:56)

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

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

3. ------------------------------------------------------------------

4. -- File: matrixOperations.adb --

5. -- --

6. -- Автор: Олег Кахерський, група ІП-31 --

7. -- Дата: 12.03.2016 --

8. ------------------------------------------------------------------

9. package body MatrixOperations is

10.

11. procedure Output\_Matrix (MA : Matrix) is

12. begin

13. New\_Line;

14. for I in Index loop

15. for J in Index loop

16. Put(Item => MA(i)(j), Width => 5);

17. end loop;

18. New\_line;

19. end loop;

20. end Output\_Matrix;

21.

22. procedure Matrix\_Matrix\_Multiply

23. (Left : Matrix;

24. Right : Matrix;

25. Start\_Index: Integer;

26. End\_Index: Integer;

27. const: Integer;

28. Result\_Matrix: out Matrix) is

29. begin

30. for i in Start\_Index..End\_Index loop

31. for J in Index loop

32. Result\_Matrix(I)(J) := 0;

33. for K in Index loop

34. Result\_Matrix(I)(J) := Result\_Matrix(I)(J) + Left(I)(K) \* Right(K)(J);

35. end loop;

36. Result\_Matrix(I)(J) := Result\_Matrix(I)(J) \* const;

37. end loop;

38. end loop;

39. end Matrix\_Matrix\_Multiply;

40.

41. procedure Matrix\_Matrix\_Add

42. (Left: Matrix;

43. Right: Matrix;

44. Start\_Index: Integer;

45. End\_Index: Integer;

46. const: Integer;

47. Result\_Matrix: out Matrix) is

48. begin

49. for I in Start\_Index .. End\_Index loop

50. for J in Index loop

51. Result\_Matrix(I)(J) := Left(I)(J) + Right(I)(J)\*const;

52. end loop;

53. end loop;

54. end Matrix\_Matrix\_Add;

55.

56. function Min

57. (Vect: Vector;

58. Start\_Index: Integer;

59. End\_Index: Integer) return Integer is

60. min: Integer;

61. begin

62. min := Vect(Start\_Index);

63. for I in Start\_Index .. End\_Index loop

64. if min > Vect(I) then

65. min := Vect(I);

66. end if;

67. end loop;

68. return min;

69. end Min;

70.

71. function Input\_Vector return Vector is

72. V: Vector;

73. begin

74. for I in Index loop

75. V(I) := 1;

76. end loop;

77. return V;

78. end Input\_Vector;

79.

80. function Input\_Constant return Integer is

81. begin

82. return 1;

83. end Input\_Constant;

84.

85. function Input\_Matrix return Matrix is

86. Result\_Matrix: Matrix;

87. begin

88. for I in Index loop

89. for J in Index loop

90. Result\_Matrix(I)(J) := 1;

91. end loop;

92. end loop;

93. return Result\_Matrix;

94. end Input\_Matrix;

95.

96. end MatrixOperations;

Compiling: matrixoperations.ads (source file time stamp: 2016-03-15 20:00:35)

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

2. -- File: matrixOperations.ads --

3. -- --

4. -- Автор: Олег Кахерський, група ІП-31 --

5. -- Дата: 12.03.2016 --

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

7. generic

8. N: in Integer; --розмірність масивів та матриць

9. package MatrixOperations is

10. type Vector is private;

11. type Matrix is private;

12.

13. --Вивід матриці на екран

14. procedure Output\_Matrix (MA : Matrix);

15.

16. --Множення матриці на матрицю(діапазон рядків StartIndex..EndIndex),

17. --результат множення множиться на константу

18. procedure Matrix\_Matrix\_Multiply

19. (Left : Matrix;

20. Right : Matrix;

21. Start\_Index: Integer;

22. End\_Index: Integer;

23. const: Integer; --константа, на яку множиться результат множення

24. Result\_Matrix: out Matrix);

25.

26. --Додавання матриць, Right(I)(J) перед додаванням множиться на константу

27. procedure Matrix\_Matrix\_Add

28. (Left: Matrix;

29. Right: Matrix;

30. Start\_Index: Integer;

31. End\_Index: Integer;

32. const: Integer;

33. Result\_Matrix: out Matrix);

34.

35. --Знаходження мінімального значення в діапазоні StartIndex, EndIndex

36. function Min

37. (Vect: Vector;

38. Start\_Index: Integer;

39. End\_Index: Integer) return Integer;

40.

41. --Ввід вектора

42. function Input\_Vector return Vector;

43.

44. --Ввід константи

45. function Input\_Constant return Integer;

46.

47. --Ввід матриці

48. function Input\_Matrix return Matrix;

49.

50. private

51. subtype Index is Integer range 1..N;

52. type Vector is array (Index) of Integer;

53. type Matrix is array (Index) of Vector;

54. end MatrixOperations;

96 lines: No errors