Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 2023 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за другим (магістерським) рівнем вищої освіти
на тему
“Дослідження та програмна реалізація системи управління паралельними обчисленнями для наукових досліджень”

Виконав здобувач вищої освіти
ІІ курсу, групи КН-22М-2
ОПП «Комп’ютерні науки»
спеціальності 122 «Комп’ютерні науки»
_____ Деркач Є.А.
«____» _____ 2023 р.

Керівник проекту
кандидат фізико-математичних наук, доцент
_____ Петренюк В.І.
«____» _____ 2023 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Рівень вищої освіти *магістр*

Галузь знань _12 _____*"Інформаційні технології"*_____

Спеціальність 122 *"Комп'ютерні науки"*

Освітньо-професійна (освітньо-наукова) програма _*"Комп'ютерні науки"*_

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ Олексій СМІРНОВ
« 6 » вересня 2023 року

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ДРУГИМ (МАГІСТЕРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Деркачу Євгенію Андрійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження та програмна реалізація системи управління паралельними обчисленнями для наукових досліджень*

2. Керівник роботи *Петренюк Володимир Ілліч, канд. фіз.-мат. наук, доцент*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 33-13 від 04.08.2023 року

3. Строк подання студентом роботи до захисту *10.12.2023 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою розробки є дослідження та програмна реалізація системи управління паралельними обчисленнями для наукових досліджень*

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

1. Призначення та область використання.	*6. Наукова новизна.*
2.Перегляд аналогічних існуючих систем.	*7. Економічна ефективність розробленої програми.*
3. Опис і обґрунтування проектних рішень.	*8. Заходи з охорони праці та техніки безпеки.*
4. Етапи програмування системи.	*9. Висновки.*
5. Впровадження системи в промислову експлуатацію	

6. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Наукова новизна	*1 аркуш*
Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*
Показники економічної ефективності	*1 аркуш*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Савеленко Г.В.	05.10.2023	14.11.2023
Охорона праці	Оришака О.В.	06.10.2023	16.11.2023

7. Дата видачі завдання « 6 » вересня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.10.2023 р.	
2.	Постановка задачі, оформлення ТЗ	15.10.2023 р.	
3.	Розробка моделі компонента	20.10.2023 р.	
4.	Розробка структур даних	25.10.2023 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.10.2023 р.	
6.	Програмування алгоритмів	10.11.2023 р.	
7.	Розрахунок економічної ефективності	13.11.2023 р.	
8.	Розрахунки з охорони праці та техніки безпеки	15.11.2023 р.	
9.	Оформлення ПЗ	17.11.2023 р.	
10.	Попередній захист роботи	10.12.2023 р.	

Дата видачі завдання
« 6 » вересня 2023 р.

Підпис керівника

_____ _____
(прізвище та ініціали)

Завдання прийнято до виконання
« 6 » вересня 2023 р.

Підпис здобувача

_____ _____
(прізвище та ініціали)

АНОТАЦІЯ

Деркач Є.А. Дослідження та програмна реалізація системи управління паралельними обчисленнями для наукових досліджень. 122 Комп'ютерні науки. Центральноукраїнський національний технічний університет. Кропивницький. 2023.

В даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи управління паралельними обчисленнями для наукових досліджень.

Метою розробки є дослідження та програмна реалізація системи управління паралельними обчисленнями для наукових досліджень.

Об'єктом дослідження є процес управління паралельними обчисленнями для наукових досліджень.

Предметом дослідження є методи управління паралельними обчисленнями для наукових досліджень.

Методи дослідження базуються на методах паралелізму, методах математичної статистики, методах розробки програмного забезпечення.

Результат роботи – програмна реалізація системи управління паралельними обчисленнями для наукових досліджень.

В процесі роботи над програмною моделлю виконано аналіз існуючих апаратних та програмних засобів. В повній мірі описані всі компоненти розробленого програмного забезпечення.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

Програма може використовуватися на ПЕОМ архітектури IBM PC з ОС Windows 10/11.

Програму розроблено в середовищі RAD Studio Delphi 10.

Ключові слова: комп'ютерні науки, паралельні обчислення, наукові дослідження

ABSTRACT

Derkach E.A. Research and software implementation of a parallel computing control system for scientific research. 122 Computer Science. Central Ukrainian National Technical University. Kropyvnytskyi. 2023.

In this graduation thesis for the second (master's) level of higher education, software is developed, which is intended for the management system of parallel computing for scientific research.

The purpose of the development is the research and software implementation of the parallel computing control system for scientific research.

The object of research is the process of managing parallel computing for scientific research.

The subject of research is methods of managing parallel computing for scientific research.

Research methods are based on methods of parallelism, methods of mathematical statistics, methods of software development.

The result of the work is the software implementation of the parallel computing control system for scientific research.

In the process of working on the software model, an analysis of existing hardware and software was performed. All components of the developed software are fully described.

A convenient user interface has been developed. Instructions for working with software tools are provided.

The program can be used on PCs of IBM PC architecture with Windows 10/11 OS.

The program was developed in the RAD Studio Delphi 10 environment.

Keywords: computer science, parallel computing, scientific research

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ 3
ВСТУП..... 4
1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ 8
1.1 Призначення системи..... 8
1.2 Область застосування..... 10
2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ 15
2.1 Огляд існуючих систем, технологій, архітектур та програмних рішень за профілем теми випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти..... 15
2.2 Обґрунтування вибору засобів для побудови системи та мови програмування..... 25
2.3 Розгорнута постановка завдання 31
3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ 32
3.1 Опис функціонування системи 32
3.2 Розробка структурної схеми..... 46
3.3 Розробка функціональної схеми 54
3.4 Розробка діаграми процесів..... 57
4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ ТА ПРОГРАМНИХ РІШЕНЬ..... 58
4.1 Розробка блок-схем та опис алгоритмів функціонування системи..... 58
4.2 Захист розробленого програмного забезпечення..... 69
5 ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ..... 72
6 НАУКОВА НОВИЗНА 74

7 ЕКОНОМІЧНА ЕФЕКТИВНІСТЬ РОЗРОБЛЕНОЇ ПРОГРАМИ 75
7.1 Техніко економічне обґрунтування теми випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти 75
7.2 Розрахунок трудомісткості розробки програмної продукції 77
7.3 Визначення чисельності виконавців і планового фонду зарплати 79
7.4 Розрахунок капітальних вкладень та амортизаційних відрахувань у розробника 84
7.5 Визначення собівартості розробки та ціни програмної продукції. 88
7.6 Визначення об'єму капітальних вкладень та експлуатаційних витрат у споживача програмної продукції. 91
7.7 Визначення експлуатаційних витрат 91
7.8 Визначення економічної ефективності програмної продукції. 93
7.9 Висновок 95
8 ЗАХОДИ З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ 96
8.1 Вступ 96
8.2 Аналіз санітарно-гігієнічних умов праці на робочому місці програміста 98
8.3 Розробка заходів з умов поліпшення охорони праці 101
8.4 Розрахунок системи загального штучного освітлення виробничого приміщення де працюють ІТ-фахівці 102
8.5 Висновки до розділу 105
9 ОСНОВНІ ВИСНОВКИ 106
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 108

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

ІС	–	інформаційна система
ОС	–	операційна система
ЦПС	–	цифрова промислова система
DSSS		метод розширення спектра
LAN	–	локальна мережа
RFID	–	радіочастотна ідентифікація
SCADA	–	диспетчерський контроль і накопичення даних
SSID	–	ідентифікатор мережі
Wi-Fi	–	бездротова технологія
WLAN	–	бездротова локальна мережа

ВСТУП

Актуальність теми. Останнім часом активний розвиток одержала нова модель організації ресурсів за назвою GRID – просторово розподілене середовище, що інтегрує безліч ресурсів різних типів (процесори, довгострокова й оперативна пам'ять, сховища файлів, бази даних і мережі), сукупність яких може бути використана для рішення прикладних завдань нового рівня складності. Потенціал технологій GRID уже зараз оцінюється дуже високо: він має стратегічний характер, і в близькій перспективі GRID повинен стати інструментарієм для розвитку високих технологій у різних сферах людської діяльності.

Найбільш розвинений і затребуваний на практиці розглянутий у роботі обчислювальний GRID, що застосовується для проведення різних наукових досліджень, що оперує такими типами ресурсів, як процесори, оперативна й дискова пам'ять, які застосовуються для обробки завдань і зберігання файлів. До теперішнього часу вже розроблені ключові для цього типу GRID протоколи дистанційного запуску завдань і керування файлами.

Ефективність функціонування GRID, що застосовується для проведення наукових досліджень, як середовища з колективною формою обслуговування користувачів визначається в першу чергу погодженістю розподілу наявних ресурсів, що повинне відбуватися автоматично, опираючись на планування обчислювальних процесів в GRID у цілому. Тому однією із ключових функцій, необхідних від програмного забезпечення GRID, що застосовується для проведення наукових досліджень, є функція диспетчеризації, за допомогою якої забезпечується розподіл ресурсів із загального ресурсного пулу між завданнями, доставка програм і даних. Завдання диспетчеризації багато разів успішно вирішувалося для найближчого аналога обчислювального GRID – кластерних систем, однак в умовах GRID вона значно ускладнюється, і для її рішення потрібні нові підходи.

В архітектурі GRID, що застосовується для проведення наукових досліджень, функція диспетчеризації реалізується спеціальними програмними службами, що забезпечують такий рівень інтеграції розподілених ресурсів, при якому GRID представляється у вигляді єдиного операційного середовища обробки запитів (завдань). Сукупність таких служб становлять систему диспетчеризації. Більшість існуючих на сьогодні систем диспетчеризації призначено для обслуговування GRID, що складаються із кластерів, – традиційної форми організації розподілених ресурсів. Використовуваним на практиці системам диспетчеризації властиві досить тверді обмеження по застосуванню, і вони не здатні виключити такі небажані ефекти, як непередбачуваність часу обробки завдань, затримка обробки в ситуаціях, коли є ресурси, що простоюють. Істотним недоліком більшості систем є неможливість обробки паралельних завдань. Основна складність у цьому випадку складається в необхідності планування, що забезпечує нагромадження й потім гарантоване синхронне виділення ресурсів у декількох кластерах (коаллокація ресурсів): це запобігає зависанню завдань, що є наслідком фрагментації ресурсного пулу. Деякі системи здатні вирішити це завдання в спеціальних умовах застосування, коли використовувані ресурси повністю відчужуються в GRID і централізовано управляються.

Магістерська робота присвячена проблемам розробки методів керування паралельними завданнями і їхньою алгоритмічною підтримкою для актуальної форми GRID, що застосовується для проведення наукових досліджень, коли ресурси не відчужуються від власників, а використовуються в GRID разом з ними (невідчужувані ресурси). Рішення завдання в такій постановці відкриває можливість створення високопродуктивних обчислювальних комплексів за допомогою інтеграції просторово розподілених, автономно керованих, не виділених спеціально в GRID багатопроцесорних і кластерних систем у єдине операційне середовище й застосування як засіб міжпроцесорного обміну даними глобальних телекомунікацій.

Мета й завдання дослідження. Метою роботи є дослідження та програмна реалізація системи управління паралельними обчисленнями для наукових досліджень.

Для досягнення поставленої мети визначена програма дослідження, що складається з наступних завдань:

– Огляд існуючих систем управління паралельними обчисленнями для наукових досліджень.

– Дослідження системи управління паралельними обчисленнями для наукових досліджень.

– Програмна реалізація системи управління паралельними обчисленнями для наукових досліджень.

Об'єктом дослідження є процес управління паралельними обчисленнями для наукових досліджень.

Предметом дослідження є методи управління паралельними обчисленнями для наукових досліджень.

Методи дослідження базуються на методах паралелізму, методах математичної статистики, методах розробки програмного забезпечення.

Наукова новизна отриманих результатів. У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

– Удосконалено метод управління паралельними обчисленнями для наукових досліджень.

– Розроблено вітчизняний продукт управління паралельними обчисленнями для наукових досліджень, який має більш широкі можливості, на відміну від існуючих аналогів.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі управління паралельними обчисленнями для наукових досліджень.

Достовірність наукових результатів підтверджена теоретичними викладеннями, даними комп'ютерного моделювання, коректними дослідженнями

параметрів на функціонуючій обчислювальній мережі, а також відповідністю отриманих результатів окремим результатам, наведеним у науковій літературі.

Робота апробована на LVII Науково-технічній конференції здобувачів вищої освіти «Наука – виробництву», 2023, основні положення випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти надруковані у статті збірника праць молодих науковців ЦНТУ, випуск №14.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи управління паралельними обчисленнями для наукових досліджень, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Система призначена для реалізації програмного забезпечення управління паралельними обчисленнями для наукових досліджень. В основу даної системи покладено використання GRID.

Технологія GRID використовується для створення географічно розподіленої обчислювальної інфраструктури, що поєднує ресурси різних типів з колективним доступом до цих ресурсів у рамках віртуальних організацій, що складаються з підприємств і фахівців, що спільно використовує ці загальні ресурси.

Термін Grid (сітка, ґрати) почав використовуватися із середини 90-х років і був обраний за аналогією з мережами передачі й розподілу електроенергії (Power Grids).

Розвиток і впровадження технології GRID носять стратегічний характер. У найближчій перспективі ця технологія дозволить створити принципово новий обчислювальний інструмент для розвитку високих технологій у різних сферах людської діяльності.

Ідейною основою технології GRID є об'єднання ресурсів шляхом створення комп'ютерної інфраструктури нового типу, що забезпечує глобальну інтеграцію інформаційних і обчислювальних ресурсів на основі мережних технологій і спеціального програмного забезпечення проміжного рівня (між базовим і прикладним ПЗ), а також набору стандартизованих служб для забезпечення надійного спільного доступу до географічно розподілених інформаційних і обчислювальних ресурсів: окремим комп'ютерам, кластерам, сховищам інформації й мережам.

Поява технології GRID обумовлено наступними передумовами:

– необхідністю рішення складних наукових, виробничих, інженерних і бізнес-завдань;

– стрімким розвитком мережного транспортного середовища й технологій високошвидкісної передачі даних;

– наявністю в багатьох організаціях обчислювальних ресурсів: суперкомп'ютерів або, що найбільше часто зустрічається, організованих у вигляді кластерів персональних комп'ютерів.

Застосування технології GRID може забезпечити новий якісний рівень, а іноді й реалізувати принципово новий підхід в обробці величезних обсягів експериментальних даних, забезпечити моделювання складних процесів, візуалізацію більших наборів даних, складні бізнес-додатка з більшими обсягами обчислень.

До теперішнього часу вже реалізовані й реалізуються безліч проектів по створенню GRID-систем. Більша частина цих проектів має експериментальний характер. Виходячи з результатів аналізу проектів можна зробити вивід про три напрямки розвитку технології GRID:

– обчислювальний GRID;

– GRID для інтенсивної обробки даних;

– семантичний GRID для оперування даними з різних баз даних.

Метою першого напрямку є досягнення максимальної швидкості обчислень за рахунок глобального розподілу цих обчислень між комп'ютерами. Проект DEISA (www.desia.org) може бути прикладом цього напрямку, у якому вживається спроба об'єднання суперкомп'ютерних центрів.

Метою другого напрямку є обробка величезних обсягів даних відносно нескладними програмами за принципом «одне завдання – один процесор». Доставка даних для обробки й пересилання результатів у цьому випадку являють собою досить складне завдання. Для цього напрямку інфраструктура GRID являє собою об'єднання кластерів. Один із проектів, метою якого і є створення виробничої GRID-системи для обробки наукових даних, є проект EGEE (Enabling Grids for E-scienc), що виконується під егідою Європейського Союзу (www.eu-

egee.org). Учасниками цього проекту є більше 90 наукових і освітніх установ із усього світу, включаючи Україну.

Побудова інфраструктури GRID у рамках проекту EGEE орієнтована, у першу чергу, на застосування в різних галузях наукової діяльності, у тому числі й для обробки даних у фізиці високих енергій учасниками експериментів, проведених на базі створюваного в Європейському центрі ядерних досліджень (CERN, www.cern.ch) прискорювача LHC.

Проект EGEE тісно зв'язаний на даній фазі розвитку із проектом LCG (LHC Computing Grid), що, по суті, і є його технологічною базою. Ведеться активна робота з розширення української інфраструктури GRID.

Незважаючи на досить тісну взаємодію багатьох проектів, конкретні реалізації GRID-систем відрізняються друг від друга, хоча до теперішнього часу з достатньою визначеністю початку спостерігатися тенденція стандартизації більшості компонентів, що означає найважливіший етап формування технології GRID (архітектура, протоколи, сервіси й ін.). Із самих загальних позицій ця технологія характеризується простим набором критеріїв:

– координація використання ресурсів при відсутності централізованого керування цими ресурсами;

– використання стандартних, відкритих, універсальних протоколів і інтерфейсів;

– забезпечення високоякісного обслуговування користувачів.

1.2 Область застосування

Областю застосування системи, яка розробляється у ході виконання магістерської роботи, є наукові дослідження.

Глобальна інфраструктура GRID для науки

У цьому розділі як приклад коротко розглянута сама велика у світі інфраструктура GRID, реалізована в рамках проекту EGEE (Enable Grid E-scienc).

Мета проекту EGEE – об'єднати вже ведуться національної, регіональної й тематичні GRID-розробки в єдину цільну GRID-інфраструктуру для підтримки

наукових досліджень. Ця інфраструктура надає дослідникам, як в академічних колах, так і в різних галузях економіки, цілодобовий доступ до високопродуктивних обчислювальних ресурсів незалежно від їхнього географічного положення. Користуватися інфраструктурою зможуть географічно розподілені співтовариства дослідників, які мають потребу в загальні для них обчислювальних ресурсах, готові об'єднати свої власні обчислювальні інфраструктури й згодні із принципами загального доступу. Проект підтримують в основному установи, що фінансують, ЄС, але призначений він для роботи в усьому світі. Значні засоби надходять від США, України й інших учасників проекту, що не входять у ЄС.

Проект стартував у винятково сприятливих умовах: до його формального початку вже були розміщені основні сервіси й початі розробка проміжного програмного забезпечення й поширення інформації.

Для відпрацьовування початкового рівня впровадження що розвивається GRID-інфраструктури, офіційної оцінки її експлуатаційних якостей і функціональності були обрані дві практичні області. Одна – обробка даних від експериментів на прискорювачі LHC, де GRID-інфраструктура забезпечує зберігання й аналіз петабайтів (10^{15} байтів) реальних і змодельованих даних експериментів по фізиці високих енергій, що ведуться в CERN. Інша – біомедицинські GRID, де декілька коллаборацій вирішують однаково складні завдання, наприклад, пошук у геномних базах даних і індексування лікарняних баз даних, що становить декілька терабайтів у рік для однієї лікарні.

До теперішнього часу десятки додатків використовують цю інфраструктуру, що розвивається, для різних галузей науки: термоядерний синтез, науки про Землю, астрофізика, геофізика, археологія, обчислювальна фізика. Ця інфраструктура відкрита також для індустріальних і соціоекономічних співтовариств. У рамках цієї інфраструктури створено понад 60 віртуальні суспільства. У проекті EGEE беруть участь більше 90 організацій з 32 країн. Ці організації об'єднані в регіональні GRID (федерації). У проекті беруть участь 13

федерацій, у тому числі й федерація «Україна». Сумарна обчислювальна потужність цієї самої великої міжнародної GRID-інфраструктури становить у даний момент понад 20 тисяч процесорів.

Проект EGEE має партнерські відносини з більш ніж 70 учасниками, що не входять у даний проект, у тому числі через ряд інших проектів GRID.

У завдання проекту EGEE входить:

– поширення інформації про технологію GRID;

– залучення нових користувачів, навчання;

– підтримка додатків;

– підтримка й обслуговування інфраструктури GRID і взаємодія з основними провайдерами;

– розробка й інтеграція програмного забезпечення проміжного рівня;

– забезпечення безпеки;

– розробка мережних сервісів.

Далі розглянемо трохи докладніше деякі із завдань проекту.

У рамках проекту працює Служба поширення інформації й розширення кола користувачів (Dissemination and Outreach Activity). Вона тримає зібрані воєдино всі головні відомості про проект разом з посиланнями на відповідні регіональні веб-сайти (www.eu-egee.org). Крім того, ключові користувальницькі групи й групи потенційних користувачів можуть одержувати повідомлення зі спеціалізованих списків розсилання інформації з електронної пошти.

У рамках проекту працює Служба навчання й включення в число користувачів (Training and Induction Activity). Ця служба випускає комплект навчальних матеріалів і курсів англійською мовою. Навчання організоване по регіональному принципу, і ключові матеріали можуть бути переведені на відповідну європейську мову.

Служба розробки й інтеграції проміжного програмного забезпечення для GRID (Grid Middleware Engineering and Integration Activity) підтримує й постійно вдосконалює набір програмних засобів, завдяки якому GRID-сервіси

промислового рівня доступні основному колу користувачів. Діяльність, пов'язана із проміжними програмними засобами EGEE, полягає, головним чином, у їхньому перепроектуванні в плані їхньої функціональності. Проміжне програмне забезпечення розвивається в напрямку сервісно-орієнтованої архітектури, заснованої на стандартах, розроблених у рамках веб-сервісів. Центри перепроектування проміжного програмного забезпечення відповідають за наступні ключові сервіси: доступ до ресурсів (Італія), організація даних (CERN), пошук і збір інформації (Великобританія), брокерські операції з ресурсами й облік ресурсів (Італія), безпека (країни Північної Європи). Безпосереднє відношення до цієї роботи мають групи забезпечення якості й безпеки GRID (Quality Assurance and Grid Security). Центр інтеграції й тестування проміжного програмного забезпечення (Middleware Integration and Testing Center) перебуває в CERN.

Групи, що забезпечують експлуатацію GRID, гарантують високий рівень GRID-сервісів. Основними якостями, що визначають такий рівень GRID-сервісів, є їхня керованість, стійкість до збоїв і різного роду несправностям, єдиний підхід до забезпечення безпеки, а також їхня розширюваність, необхідна для включення в роботу нових ресурсів негайно в міру їхньої появи. Головні завдання цих груп: підтримка базових сервісів інфраструктури, моніторинг і контроль GRID, розміщення проміжного програмного забезпечення, підключення ресурсів, підтримка ресурсів і користувачів, загальне керування GRID й міжнародне співробітництво. Оперативний центр (Operations Management Centre – OMC) в CERN погоджує роботу п'яти центрів базової інфраструктури (Core Infrastructure Centre – CIC), розташованих в CERN, Франції, Італії, України, Великобританії, і восьми регіональних оперативних центрів (Regional Operation Centre – ROC), які, у свою чергу, координують роботу ресурсних центрів (Resource Cerntre – RC).

Робота EGEE для масового користувача заснована на проміжному програмному забезпеченні й сервісах проекту LCG (www.cern.ch/lcg).

Для контролю за функціонуванням цієї інфраструктури розроблені й успішно функціонують різні засоби моніторингу (проходження функціональних тестів, монітори завдань, стану сайтів і інформаційної системи).

Дані моніторингу завдань показують, що щодня на цієї GRID-системі виконуються багато тисяч завдань.

Як транспортне середовище для передачі даних і програми інфраструктури EGEE використовує дослідницьку мережу GEANT і підключені до неї регіональні мережі.

Для включення в GRID-інфраструктуру EGEE нових користувачів і наукових співтовариств діють Служби прийому заявок і підтримки (Application Identification and Support Activity), які ідентифікують і підтримують починаючих користувачів із широкого кола академічних дисциплін і галузей економіки.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи управління паралельними обчисленнями для наукових досліджень, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти

Технологія віртуального GRID GPU

NVIDIA GRID™ vGPU™ забезпечує віртуалізованим рішенням всі переваги апаратно прискореної графіки. Розділяючи ресурси одного GPU між трохи користувачами, ця технологія забезпечує неймовірну графічну продуктивність для віртуальних десктопів, порівнянну із продуктивністю локальних ПК.

NVIDIA GRID™ vGPU™ – це сама просунута в галузі технологія спільного використання апаратного прискорення на GPU декількома віртуальними робочими столами без шкоди для графічних можливостей. Можливості додатків і сумісність такі ж, як і на настільному ПК.

Завдяки технології GRID vGPU графічні команди кожної віртуальної машини передаються прямо в GPU, без трансляції гіпервізором. Це дозволяє планувати GPU із квантуванням часу, щоб забезпечувати максимальну графічну продуктивність у поділюваних віртуалізованих середовищах.

Використовуйте можливості Менеджера віртуального GPU, щоб розподіляти необхідні обсяги пам'яті для задоволення потреб кожного користувача. Кожний віртуальний робочий стіл має дискретну графічну пам'ять, як і на настільному ПК, тому вони завжди мають необхідні ресурси для запуску й роботи додатків. Завдяки Менеджерові віртуального GPU до восьми користувачів можуть спільно використовувати один фізичний GPU. Менеджер рівномірно розподіляє графічні ресурси доступних GPU серед віртуальних машин. Кожна

графічна карта GRID має до чотирьох GPU, завдяки чому одну графічну карту можуть спільно використовувати 32 користувача.

Всі GPU, назва яких закінчується на Q, проходять таку ж серйозну сертифікацію для додатків, як і наші графічні процесори NVIDIA® Quadro® для настільних ПК.

Профілі GRID K100 і K200 GPU створені для роботи з більше простими графічними навантаженнями, які типові для додатків, в основному використовуваних офісними співробітниками й досвідченими користувачами. Сюди ставиться Windows 7 із включеним режимом Aero, перегляд веб контенту, наприклад, Adobe Flash або HTML 5, або просто інтерактивна робота з Microsoft Office (PowerPoint, Excel, Word). Щоб одержати більше докладну інформацію про вимоги ваших додатків до графічного рішення, звернетеся до виробника додатків.

Lxfarm

GRID вузол – комп'ютерна ферма (кластер) Lxfarm. Кластер призначений для проведення розрахунків і зберігання більших обсягів даних, головним чином, в області ядерної фізики, фізик елементарних часток і космічних променів. Крім того, кластер виступає як вузол GRID – всесвітньої розподіленої системи обчислювальних центрів.

Сервіси Lxfarm

Перелічимо коротко сервіси, надавані на кластері Lxfarm. Більш докладно про сервіси Lxfarm і про порядок роботі на комп'ютерах ферми можна прочитати в документі “Робота на кластері Lxfarm.

NIS сервіс забезпечує єдину систему авторизації й автентифікації користувачів на комп'ютерах ферми.

NFS – мережна файлова система – призначена для "експорту" домашніх каталогів користувача, а також каталогів загального програмного забезпечення на всі комп'ютери, де це необхідно. Таким чином, NIS і NFS забезпечують для користувачів єдине робітниче середовище: користувач входить на будь-який комп'ютер, де йому відкритий доступ, з одним ім'ям і паролем і має там ті самі

робочі каталоги і програмне забезпечення. Крім того, система NFS застосовується для "експорту" каталогів, призначених для зберігання більших обсягів даних і каталогів, у які здійснюється резервне копіювання.

AFS мережна файлова система. На відміну від NFS, що є локальною файловою системою, AFS, навпроти, призначена для спільного використання дискового простору комп'ютерами, розкиданими по усьому світі. Обчислювальна ферма підключена як клієнт до AFS-серверів ЦЕРН у зв'язку з тим, що основне співробітництво й по лінії дослідження фундаментальних властивостей матерії й по лінії розвитку GRID-технологій веде в основному із ЦЕРН.

PBS – система керування завданнями (batch система). Суть системи полягає в наступному. Користувач, що бажає виконати завдання на одному з комп'ютерів ферми, становить за деякими правилами спеціальний файл, у якому вказує, яку саме програму він хоче виконати, і які ресурси (вхідні дані, необхідне процесорний час і т.п.) йому для цього необхідні. Система сама вирішує, на якому з комп'ютерів ферми це завдання буде виконуватися з урахуванням поточного завантаження процесорів, їхньої швидкодії, а також запитаних користувачем ресурсів.

Резервне копіювання. Щодня здійснюється резервне копіювання домашніх каталогів користувачів, загального програмного забезпечення, а також деяких найбільш важливих системних файлів. Таке резервування страхує на випадок можливих збоїв на основному файлі-сервері.

Web-сервер надає інформацію про кластер Lxfarm/GRID, містить необхідну документацію й корисні посилання на Інтернет ресурси.

Система електронної пошти призначена, головним чином, для внутрішнього використання: інформування користувачів про новини, зміни в системі й т.п.

Система моніторингу забезпечує інформацію про стан комп'ютерів ферми, їхньому завантаженню, обсягах зайнятого й вільного дискового простору, мережному трафіку т.п.

Програмне забезпечення

Програмне забезпечення загального призначення встановлено на комп'ютері lxfarm99 у каталог /opt, що, експортується системою NFS з lxfarm99 на всі інші комп'ютери кластера. Таким чином, користувачі на будь-якому комп'ютері ферми має той самий набір програмного забезпечення. У даний момент установлені й протестовані наступні пакети програм:

– Бібліотеки CERNLIB. Це великий пакет програм, написаних, головним чином, мовою Фортран.

– Пакет CLHEP, також розроблений у ЦЕРН, містить бібліотеки програм для базових функцій і операцій, написаних на C+. Цей пакет використовується, зокрема в GEANT4.

– Кілька версій пакета ALIROOT експерименту ALICE.

– Пакет програм GEANT4 для моделювання детекторів часток і процесів взаємодії елементарних часток з речовиною. Як програми візуалізації для GEANT4 можна використовувати пакети DAWN і OpenGL.

– Пакет ROOT. ROOT у цей час є одним з основних засобів у фізику часток для обробки експериментальних і модельованого даних, а також для подання отриманих результатів.

– Програма Geant4 VMC являє собою віртуальний Монте-Карло інтерфейс між ROOT і GEANT4.

– Дві програми для моделювання ядро-ядерних взаємодій високих енергій: VENUS і UrQMD.

– Програма Монте-Карло моделювання експерименту CBM – G4CBM.

– Пакет atlsim являє собою інтегроване середовище для моделювання й обробки даних, що включає в себе такі програм з CERNLIB, як PAW і GEANT3.

Програми обновляються в міру появи нових версій. У каталог /opt на прохання користувачів можуть бути встановлені й іншої програми, особливо більші по обсязі займаного дискового простору й такі, котрими користуються відразу трохи користувачів.

Використання Lxfarm

У цей час ферма Lxfarm використовується для проведення високопродуктивних розрахунків, пов'язаних з поруч російських і міжнародних експериментів в області фізики часток високих енергій, ядерної фізики й фізики космічних променів. Ферма використовується групами, що беруть участь у наступних роботах:

– Експеримент ALICE на коллайдері LHC у ЦЕРН.

– Експеримент ATLAS також на коллайдері LHC у ЦЕРН.

– Експеримент CBM на майбутньому прискорювачі ядер в GSI, Німеччина.

– Баксанський Підземний Сцинтиляційний Телескоп.

– Експеримент “НЕВІД” на установці в .

– Супутниковий експеримент “PAMELA”.

– Експеримент “STAR”, що проходить на коллайдері релятивістських ядер RHIC у США.

Сервіси й програмне забезпечення GRID

Основне співробітництво в рамках GRID веде із ЦЕРН. Тому ми встановлюємо програмне забезпечення LCG – галузі GRID, призначеної для моделювання й обробки даних експериментів на коллайдері LHC. На фермі в були встановлені наступні сервіси GRID:

Сервер конфігурацій і програмного забезпечення LCFGng – для зберігання програмного забезпечення LCG, його відновлення в міру необхідності й експорту на інші комп'ютери GRID вузла . Наявність такого сервісу дозволяє в значній мірі автоматизувати процеси установки й відновлення програм на інших комп'ютерах вузла.

Сервер керування обчисленнями CE (Computing Element) – для прийому завдань, що направляються на вузол GRID, як з локальної мережі, так і з інших GRID вузлів, і розподілу цих завдань по вільних ресурсах вузла. Крім того, на цьому ж комп'ютері встановлений GRIS (GRid Information System) – сервіс, що у

режимі реального часу збирає інформацію про наявні й вільні ресурси GRID вузла й робить цю інформацію доступної всім іншим вузлам LCG.

Сервер зберігання даних SE (Storage Element) – для зберігання більших обсягів даних і розподілу їх по запитах на інші вузли. Система GRID припускає реплікацію часто використовуваних файлів на ті SE, які географічно розташовані найбільше близько до вузлів GRID, де виконуються завдання. Це дозволяє значно знизити витрати на передачу більших обсягів даних по мережі.

Інтерфейс користувача UI (User Interface) – комп'ютер з необхідним програмним забезпеченням, з якого користувачі здійснюють постановку завдань на виконання, їх моніторингу й т.п. За допомогою цього ж комп'ютера реалізується система автентифікації користувачів у системі LCG.

Більшість інших комп'ютерів ферми використовуються в якості WN (Working Node), на яких властиво й виконуються завдання, спрямовані на GRID вузол . Для забезпечення оптимального завантаження ферми застосовується описана вище система PBS.

Інші компоненти сервісної інфраструктури містять у собі: сервер керування ресурсами (Resource broker), база даних по ресурсах (DB Information Index) і сервер керування пересиланнями даних (Resource broker). У цей час ці ресурси для забезпечує вузол НДІЯФ МГУ – головної організації проекту LCG у Росії.

Суперкомп'ютерна GRID-система Новосибірського національного технічного університету

Програмно-апаратна система, що інтегрує використання неоднорідних зосереджених розподілених ресурсів (персональних комп'ютерів, робочих станцій, серверів, систем зберігання даних і т.п.) при відсутності централізованого керування ними.

Ресурси GRID-системи доступні будь-якому підрозділу університету й зовнішніх організацій через wide area network або Інтернет.

Сегмент 1. Центральний вузол на факультеті автоматики й обчислювальної техніки (АОТФ).

Лабораторії тонких клієнтів АОТФ.

Лабораторія робочих станцій АОТФ.

Сегмент 2. Центральний вузол на факультеті прикладної математики й інформатики (ФПМІ).

Сегмент 3. Лабораторії на факультеті літальних апаратів (ФЛА).

Програмне забезпечення для реалізації GRID-технологій.

Функції GRID-системи.

Сегмент 1. Центральний вузол на факультеті автоматики й обчислювальної техніки (АОТФ)

Устаткування:

– обчислювальне ядро центрального вузла GRID-системи (8 обчислювальних серверів, 2 інфраструктурних сервери, 2 сервери БД);

– інтелектуальна система зберігання даних (СЗД);

– сервер керування СЗД;

– сервер керування ресурсами кластера (2 шт.);

– сервер для навчального програмного забезпечення (2 шт.);

– сервер для керування дисковим масивом (2 шт.);

– дискові масиви (2 шт.);

– сервер для керуваннями тонкими клієнтами Sun Ray;

– бібліотека архівування на магнітну стрічку;

– комутатор SAN (2 шт.);

– комутатор центрального вузла GRID-системи (2 шт.);

– серверна шафа (4 блоки розподілу живлення, комутатор серверних консолей);

– програмне забезпечення архівування й керування кластерними вузлами;

– робоча станція GRID-системи.

Можливості:

– апаратна й інформаційна платформа для рішення освітніх, науково-дослідних і комерційних завдань;

– організація багатопотокових розподілених і паралельних обчислень;

– середовище для виконання різних прикладних програм;

– забезпечення підтримки використовуваного програмного забезпечення в різних операційних системах;

– забезпечення віртуалізації – можливість міняти системне оточення, включаючи швидку зміну системного й прикладного програмного забезпечення;

– можливість перерозподілу ресурсів системи залежно від потреб користувачів і відповідно до вироблених політиків;

– інструмент ієрархічного моделювання й виконання високопродуктивних обчислень і побудови складних інформаційних систем;

– можливість окремого використання й об'єднання ресурсів різних факультетів НДТУ, а також вітчизняних і закордонних організацій.

Лабораторії тонких клієнтів АОТФ

Устаткування:

– тонкий клієнт Sun Ray 2 з монітором, клавіатурою й маніпулятором (22 шт.);

– комутатор термінальних класів (2 шт.);

– лазерний принтер (2 шт.).

Можливості:

– заняття по дисциплінах «Об’єктно-орієнтоване програмування», «Технологія програмування», «Інтелектуальний аналіз даних», «Інформатика».

Лабораторія робочих станцій АОТФ

Устаткування й програмне забезпечення:

– настільний ПК із монітором і пристроєм безперебійного електропостачання (12 шт.);

– лазерний принтер;

– комутатор термінальних класів;

– програмне забезпечення віддаленої 3D/2D-візуалізації.

Можливості:

– заняття по дисциплінах «Концептуальні основи інформатики», «Інформатика», «Комп'ютерна графіка», «Бази даних», «Операційні системи», «Організація ЕОМ і систем», «Системне програмне забезпечення», «Програмування», «Інтернет і ПК» і ін.

Сегмент 2. Центральний вузол GRID-системи на факультеті прикладної математики й інформатики

Кількість: 4 лабораторії.

Устаткування, отримане в рамках побудови GRID-системи:

– робоча станція HP xw8400 2xQuard Xeon, 32 Gb RAM (2 шт.);

– 2 термінальних класи по 13 комп'ютерів HP dx7400 Core Duo, 3Gb RAM;

– комп'ютер адміністратора/програміста HP dx7400 Core Duo, 3Gb RAM (3 шт.);

– Можливості:;

– наукові розрахунки в рамках хоздоговірних науково-дослідних робіт і досліджень магістрантів і аспірантів;

– проведення занять по дисциплінах, пов'язаним із програмуванням;

– моніторинг і керування мережею.

Сегмент 3. Лабораторії на факультеті літальних апаратів

Кількість: 3 лабораторії

Устаткування:

– сервер ANSYS (HP DL380R05, 16Gb, HD 146Gbx3/8x2.33Hz);

– комп'ютер HP E6550 (12 шт.);

– робоча станція з монітором (Workstation HP xw8400);

– пристрій безперебійного електропостачання Smart-UPS (12 шт.);

– комутатор термінального класу ProCurve Switch 1800-24G;

– принтер HP LaserJet P2015N.

Програмне забезпечення:

– ANSYS (12 робочих місць);

– Remote Graphics (11 робочих місць).

Програмне забезпечення для реалізації GRID-технологій:

– відкрите програмне забезпечення для настроювання системи (Globus Toolkit);

– відкрита бібліотека функцій, призначена для підтримки роботи паралельних процесів у термінах передачі повідомлень (MPI – Message Passing Interface);

– ПЗ для розробки й налагодження додатків (MVS 2005 PE, Intel VFCPE, Intel VPA);

– ПЗ архівування (Windows, Novell, Linux, IBM);

– ПЗ керування кластерами;

– ПЗ швидкого розгортання (Microsoft, SUSE LINUX);

– ПЗ керування серверним устаткуванням (HP-UX, Windows, Linux і ін.);

– Спеціалізоване ПЗ (ANSYS і ін.);

– ПЗ термінальних класів (Windows, Linux, C++, C#, Java, Fortran, різні СУБД і т.д.).

Функції GRID-системи:

– розвиток у вузі робіт в області паралельних обчислювальних технологій;

– забезпечення доступу співробочих НДТУ до будь-якого внутрішнього й зовнішнього інформаційного ресурсів;

– формування в університеті діючого затребуваного єдиного інформаційно-телекомунікаційного середовища;

– навчання викладачів і студентів паралельним технологіям, віртуалізації, різнорідному моделюванню складних структур, процесів і явищ, умінню здійснювати вибір різнорідних апаратних, програмних і математичних засобів і працювати з ними;

– виконання науково-дослідних робіт, підготовка аспірантів і докторантів в області сучасних інформаційно-комунікаційних технологій (ІКТ);

– перепідготовка кадрів і підвищення кваліфікації в області сучасних ІКТ, суперЕОМ і паралельних обчислень;

– надання іншим вузам і господарюючим суб'єктам доступу до сучасних засобів ІКТ шляхом їхньої інтеграції в єдині GRID-системи країни й світу;

– розвиток мережної взаємодії освітніх і наукових установ.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Embarcadero Delphi, раніше Borland Delphi і Codegear Delphi, – інтегроване середовище розробки ПЗ для Microsoft Windows, Mac OS, iOS і Android мовою Delphi (що раніше носила назву Object Pascal), створена спочатку фірмою Borland і на даний момент приналежна й розроблювальна Embarcadero Technologies. Embarcadero Delphi є частиною пакета Embarcadero RAD Studio і поставляється в чотирьох редакціях: Community (поширюється безкоштовно й має обмежену ліцензію на використання в комерційних цілях), Professional, Enterprise і Architect.

Delphi 10.4 Sydney

Випущено 26 травня 2020 року. RAD Studio Delphi 10.4 забезпечує значно поліпшену високопродуктивну нативну підтримку Windows, кращу продуктивність розробки, миттєві підказки code completion, прискорення виконання коду із синтаксисом керованих записів, поліпшення виконання паралельних завдань на сучасних багатоядерних CPU, а також містить більш 1000 виправлень багів, поліпшення продуктивності середовища й бібліотек і багато чого крім того.

Основні можливості Delphi 10.4.1:

− Істотні розширення для Windows: поліпшення для застосунків на моніторах 4K High DPI, інтеграція з новим WebView2 на базі Chromium,

використання розширених title bars, таких же, як в Office, Explorer, Google Chrome.

− Керування пам'яттю в Delphi тепер стандартизоване на всіх підтримуваних платформах – мобільних, настільних і серверних – використовучи класичну реалізацію керування пам'яттю об'єктів.

− Істотне поліпшення Delphi Code Insight (без можливого блокування IDE – в окремому процесі), що допоможе при роботі з великими проектами.

− Тип даних Delphi «record» тепер підтримуть довільні ініціалізацію, фіналізацію й операції копіювання.

− Розширена підтримка бібліотек C++: ZeroMQ, SDL2, SOCI, libSIMDpp і Nematode.

− Відладник Win 64 (на LLDB) і збирач для C++.

− Поліпшення для C++: Включена велика кількість поліпшень STL з Dinkumware.

− Підтримка Metal Driver GPU для macOS і iOS.

− Вбудований Fmxlinux.

− Компонент Twebbrowser для iOS тепер реалізований на Wkwebview API. Реалізація компонента Media Player для macOS тепер використовує Avfoundation. Реалізований заново стилізуємий FMX компонент TMemo на платформі Windows значно поліпшений і тепер має відмінну підтримку IME.

− Численні поліпшення швидкості й стабільності роботи нашої бібліотеки The Parallel Programming Library (PPL).

− Додані оновлені драйвери для FireBird, PostgreSQL і SQLite.

− Клієнтські бібліотеки HTTP і REST Client розширені застосунковими можливостями роботи з HTTPS. Також були розширені можливості підтримки Amazon AWS services

− У технологію Visual LiveBindings внесена безліч поліпшень, у тому числі швидкодії, що стосуться, застосунків на VCL і FireMonkey

RAD Studio 10.4 Короткий огляд:

– Істотні розширення для Windows. Створення застосунків, що чудово виглядають, із чіткими елементами інтерфейсу на 4k моніторах High DPI за допомогою нової гнучкої підтримки стилів елементів керування на екрані. Інтеграція із сучасними, безпечними web-технологіями від Microsoft – новим WebView2 на базі Chromium. Використання сучасних розширених title bars, таких же, як в Office, Explorer, Google Chrome, у своїх проектах. Істотні поліпшення надійності налагодження в новому відладнику для C++ Windows 64-bit.

– Зросла продуктивність розробки. Ріст продуктивності за рахунок миттєвої реакції підказок code completion у середовищі IDE. Краща сумісність із уже наявною кодовою базою, і спрощення програмування за рахунок уніфікованої архітектури керування пам'яттю. Швидке зв'язування даних і візуальних елементів за допомогою розширеної технології Visual LiveBindings з підвищеною швидкодією. Просте використання розповсюджених бібліотек C++, наприклад, ZeroMQ, SDL2, SOCI, libSIMDpp і Nematode. Оновлена підтримка Amazon AWS cloud.

– Поліпшення швидкодії і якості. Більш 1000 поліпшень швидкодії і якості. Краща ефективність коду за допомогою нового синтаксису custom managed records. Більш швидке виконання паралельних завдань на сучасних багатоядерних CPU. Переконаєтеся в прискоренні відображення на екрані з підтримкою Metal API на macOS і iOS. Краща сумісність із уже наявною кодовою базою й спрощення програмування за рахунок уніфікованої архітектури керування пам'яттю.

Істотне поліпшення Delphi Code Insight

Як найбільше й головне поліпшення інструментів програмування Delphi за багато років, в 10.4 Delphi Code Insight реалізований через Language Server Protocol (LSP). LSP – це технологія генерації результатів для code completion, навігації й інших сервісів в окремому процесі. Це значить, що code completion і Code Insight одержать більш точні результати без блокування IDE. 10.4

забезпечує набагато більш високу продуктивність розроблювачів, які працюють із більшими проектами, що містять мільйони рядків коду.

Delphi Custom Managed Records

Ключове розширення мови Delphi: тип даних Delphi «record» тепер підтримуть довільні ініціалізацію, фіналізацію й операції копіювання. Управляйте тем, як ці структури створюються, копіюються й звільняються з допомогу вашого коду, який буде виконуватися у відповідний момент.

Це розширює потужність конструкцій records в Delphi, які використовуться щоб одержати більшу ефективність у порівнянні із класами.

Єдине керування пам'яттю

Керування пам'яттю в Delphi тепер стандартизоване на всіх підтримуваних платформах – мобільних, настільних і серверних – використовучи класичну реалізацію керування пам'яттю об'єктів.

У порівнянні з Automatic Reference Counting (ARC), це дає кращу сумісність із існучим кодом і спрощує написання компонентів, бібліотек і застосунків.

ARC модель керування пам'яттю model залишилася для керування рядками й посиланнями на тип інтерфейсу на всіх платформах. Для C++ це означає, що при створенні й звільненні Delphi-style класів в C++ використовується звичайне керування пам'яттю, як у будь-якого heap-allocated класу C++, що значно знижує складність коду.

Розширена підтримка бібліотек C++

В 10.4 ми портували багато популярних бібліотек C++ у C++Builder.

Забезпечивши оптимізовану підтримку бібліотек ZeroMQ, SDL2, SOCI, libSIMDpp і Nematode, поряд із уже підтримуваними Boost і Eigen, які можуть бути додані за допомогою менеджера пакетів Getit.

Win 64-відладник і збирач для C++

В 10.4 з'явився новий відладник C++ для Windows 64-bit. Відладник заснований на LLDB і показує значне збільшення стабільності при налагодженні

64-bit застосунків поряд з новими відладочними можливостями, такими як перегляд і інспекція типів начебто рядків C++ і Delphi, а також колекцій STL, включаючи std::vector, std::map і інших. Крім того, згенерована для застосунку відладочна інформація має інший внутрішній формат, сприяючи більш стабільному й багатому на можливості процесу налагодження, більш докладним перегляду й інспекції в debug-time.

Підвищення якості й швидкодії інструментів

- Велика кількість поліпшень STL від Dinkumware.
- Поліпшені деякі найважливіші методи й області RTL, на базі поліпшень сумісності з популярними бібліотеками C++.
- Поліпшена підтримка Cmake.
- Велика кількість виправлень для підвищення стабільності і якості.
- Відновлення Windows API – Обновлено й додали безліч декларацій API щоб добитися ще більшої інтеграції із платформою Windows.
- Загальні вдосконалення в бібліотеці доступу до БД FireDAC, включаючи оновлені драйвера для FireBird, PostgreSQL і SQLite. Вибір статичного або динамічного підключення SQLite до застосунку.

Змінені стилі VCL для High DPI

В 10.4, архітектура стилізації VCL була суттєво розширена для підтримки High DPI і 4K моніторів. Тепер усі елементи UI на формі VCL автоматично масштабуться під відповідне до монітора дозвіл для показу форми. Був оновлений API стилізації для підтримки стилів high DPI.

Кожний графічний елемент UI може бути обраний з наборів різних масштабів і масштабований до потрібного DPI, що дає чітке зображення елементів UI на всіх моніторах.

Нові High DPI стилі й стилізація окремих VCL компонент

Обновлено велике число вбудованих і преміальних VCL стилів для підтримки нового режиму стилізації High-dpi. Це дозволяє вам створювати застосунку з відмінним дизайном для всіх моніторів.

Розроблювачі VCL застосунків тепер можуть використовувати трохи VCL стилів на різних формах в одному застосунку або в різних компонентів на одній формі. Це також включає стилізацію компонентів загальною темою для платформи. Крім застосункової гнучкості використання стилів, це дозволяє використовувати нестилізуємі компоненти із зовнішніх бібліотек в VCL застосунках, що використовуть стиль.

Поліпшена кроссплатформеність

- Додана підтримка Metal Driver GPU для macOS і iOS.
- Крім підтримки останнього iOS SDK, в RAD Studio 10.4 розроблювачі можуть задовольнити нові вимоги Apple до набору стартових екранів.
- Реалізований заново стилізуємий FMX компонент TMemo на платформі Windows значно поліпшений і тепер має відмінну підтримку IME.
- Користувачам редакцій Enterprise або Architect доступна повна інтеграція Fmxlinux з IDE для створення клієнтських застосунків Linux з GUI.
- Компонент Twebbrowser для iOS тепер реалізований на Wkwebview API.
- Реалізація компонента Media Player для macOS тепер використовує Avfoundation.

Оновлений менеджер пакетів Getit

Менеджер пакетів Getit в IDE був значно вдосконалений.

Дати випуску релізів пакетів тепер видні, і можливе сортування списку по цих датах; відбір тільки встановлених пакетів, контенту, доступного тільки при наявності підписки, багато чого іншого.

Універсальний інсталятор для установки Online і Offline

В 10.4 включений новий універсальний інсталятор, який використовує технологію на базі Getit. Цей інсталятор підтримує як online, так і offline (з ISO) варіанти установки.

Тепер обоє варіанта установки дозволяють вам указати початковий набір можливостей RAD Studio для установки, наприклад, свою комбінацію мов програмування й цільових платформ, мов інтерфейсу, і додавати до нього або видаляти непотрібне в будь-який момент.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за другим (магістерським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи управління паралельними обчисленнями для наукових досліджень.

В процесі розробки випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти необхідно виконати наступний обсяг роботи:

а) провести аналіз існуючих систем-аналогів для виявлення їх позитивних і негативних якостей. Результати аналізу врахувати в подальших розробках;

б) вибрати та обґрунтувати методику побудови системи контролю роботи технологічного обладнання на виробництві в автоматизованому режимі. Розробити функціональну та структурну схеми системи;

в) розробити програмне забезпечення системи, що дозволить реалізувати поставлену технічним завданням задачу. Побудувати блок-схеми алгоритмів програми та підпрограми;

г) організувати інтерфейс користувача з метою формування та виводу на екран ЕОМ повідомлень про некоректні дії користувача та нестандартні ситуації в роботі технологічного обладнання;

д) розробити рекомендації по організаційних та методичних заходах, які забезпечать впровадження системи в промислову експлуатацію та її подальшу успішну експлуатацію;

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

ж) розробити заходи по охороні праці при впровадженні та експлуатації системи, а також розробити заходи з цивільного захисту;

з) сформувати висновки про виконаний обсяг робіт та одержані результати.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Концепція GRID

GRID є технологією забезпечення гнучкого, безпечного й скоординованого загального доступу до ресурсів. При цьому слово «ресурс» розуміється в дуже широкому змісті, тобто ресурсом може бути апаратура (жорсткі диски, процесори), а також системне й прикладне ПЗ (бібліотеки, додатка).

У термінології GRID сукупність людей і організацій, що вирішують спільно те або інше загальне завдання й які надають друг другу свої ресурси, називається віртуальною організацією. Наприклад, віртуальною організацією може бути сукупність всіх людей, що беруть участь у який-небудь наукової коллаборації. Віртуальні організації можуть розрізнятися по составу, масштабу, часу існування, роду діяльності, цілям, відносинам між учасниками (довірчі, не довірчі) і т.д. Состав віртуальних організацій може динамічно мінятися.

Є два основних критерії, що виділяють GRID-системи серед інших систем, що забезпечують поділюваний доступ до ресурсів:

1. GRID-система координує розрізнені ресурси. Ресурси не мають загального центра керування, а GRID-система займається координацією їхнього використання, наприклад, балансуванням навантаження. Тому проста система керування ресурсами кластера не є системою GRID, тому що здійснює централізоване керування всіма вузлами даного кластера, маючи до них повний доступ. GRID-системи мають лише обмежений доступ до ресурсів, що залежить від політики того адміністративного домену (організації-власника), у якому цей ресурс перебуває.

2. GRID-система будується на базі стандартних і відкритих протоколів, сервісів і інтерфейсів. Не маючи стандартних протоколів, неможливо легко й швидко підключати нові ресурси в GRID-систему, розробляти нові види сервісів і т.д.

Додамо ще кілька властивостей, якими звичайно володіють GRID-системи:

– гнучкість, тобто можливість забезпечення поділюваного доступу потенційно до будь-яких видів ресурсів;

– масштабованість: працездатність GRID-системи при значному збільшенні або зменшенні її состава;

– гнучка й потужна підсистема безпеки: стійкість до атак зловмисників, забезпечення конфіденційності;

– можливість контролю над ресурсами: застосування локальних і глобальних політик і квот;

– гарантії якості обслуговування;

– можливість одночасної, скоординованої роботи з декількома ресурсами.

Хоча сама технологія GRID не прив'язана до певних ресурсів, найбільше часто реалізації GRID-систем забезпечують роботу з наступними типами ресурсів:

– обчислювальні ресурси – окремі комп'ютери, кластери;

– ресурси зберігання даних – диски й дискові масиви, стрічки, системи масового зберігання даних;

– мережні ресурси;

– програмне забезпечення – яке-небудь спеціалізоване ПЗ.

Відзначимо різницю між технологією GRID і реалізаціями GRID-систем. Технологія GRID містить у собі лише найбільш загальні й універсальні аспекти, однакові для будь-якої системи (архітектура, протоколи, інтерфейси, сервіси). Використовуючи цю технологію й наповнюючи її конкретним змістом, можна реалізувати ту або іншу GRID-систему, призначену для рішення того або іншого

класу прикладних завдань.

Не слід змішувати технологію GRID з технологією паралельних обчислень. У рамках конкретної GRID-системи, звичайно, можливо організувати паралельні обчислення з використанням існуючих технологій (PVM, MPI), оскільки GRID-систему можна розглядати як якусь мета-комп'ютер, що має безліч обчислювальних вузлів. Однак технологія GRID не є технологією паралельних обчислень, у її завдання входить лише координація використання ресурсів.

Архітектура GRID

Архітектура GRID визначає системні компоненти, мети й функції цих компонентів і відбиває способи взаємодії компонент один з одним. Архітектура GRID являє собою архітектуру взаємодіючих протоколів, сервісів і інтерфейсів, що визначають базові механізми, за допомогою яких користувачі встановлюють з'єднання з GRID-системою, спільно використовують обчислювальні ресурси для рішення різного роду завдань. Архітектура протоколів GRID розділена на рівні, компоненти кожного з них можуть використовувати можливості компонентів кожного з нижчерозташованих рівнів. У цілому ця архітектура задає вимоги для основних компонентів технології (протоколів, сервісів, прикладних інтерфейсів і засобів розробки ПЗ), не надаючи строгий набір специфікацій, залишаючи можливість їхнього розвитку в рамках прийнятої концепції.

Базовий рівень

Базовий рівень (Fabric Layer) описує служби, що безпосередньо працюють із ресурсами. Ресурс є одним з основних понять архітектури GRID. Ресурси можуть бути досить різноманітними, однак, як уже згадувалося, можна виділити кілька основних типів:

– обчислювальні ресурси;

– ресурси зберігання даних;

– інформаційні ресурси, каталоги;

– мережні ресурси.

Обчислювальні ресурси надають користувачеві GRID-системи (точніше кажучи, завданню користувача) процесорні потужності. Обчислювальними ресурсами можуть бути як кластери, так і окремі робочі станції. При всій розмаїтості архітектур будь-яка обчислювальна система може розглядатися як потенційний обчислювальний ресурс GRID-системи. Необхідною умовою для цього є наявність спеціального програмного забезпечення, називаного ПЗ проміжного рівня (middleware), що реалізує стандартний зовнішній інтерфейс із ресурсом і що дозволяє зробити ресурс доступним для GRID-системи. Основною характеристикою обчислювального ресурсу є продуктивність.

Ресурси пам'яті являють собою простір для зберігання даних. Для доступу до ресурсів пам'яті також використовується програмне забезпечення проміжного рівня, що реалізує уніфікований інтерфейс керування й передачі даних. Як і у випадку обчислювальних ресурсів, фізична архітектура ресурсу пам'яті не принципова для GRID-системи, будь те жорсткий диск на робочій станції або система масового зберігання даних на сотні терабайт. Основною характеристикою ресурсу пам'яті є його обсяг.

Інформаційні ресурси й каталоги є особливим видом ресурсів пам'яті. Вони служать для зберігання й надання метаданих і інформації про інші ресурси GRID-системи. Інформаційні ресурси дозволяють структуровано зберігати величезний обсяг інформації про поточний стан GRID-системи й ефективно виконувати завдання пошуку.

Мережний ресурс є сполучною ланкою між розподіленими ресурсами GRID-системи. Основною характеристикою мережного ресурсу є швидкість передачі даних. Географічно розподілені системи на основі розглянутої технології здатні поєднувати тисячі ресурсів різного типу, незалежно від їхнього географічного положення.

Рівень зв'язку

Рівень зв'язку (Connectivity Layer) визначає комунікаційні протоколи й протоколи автентифікації.

Комунікаційні протоколи забезпечують обмін даними між компонентами базового рівня.

Протоколи автентифікації, ґрунтуючись на комунікаційних протоколах, надають криптографічні механізми для ідентифікації й перевірки дійсності користувачів і ресурсів.

Протоколи рівня зв'язку повинні забезпечувати надійний транспорт і маршрутизацію повідомлень, а також присвоєння імен об'єктам мережі. Незважаючи на існуючі альтернативи, зараз протоколи рівня зв'язку в GRID-системах припускають використання тільки стека протоколів TCP/IP, зокрема: на мережному рівні – IP і ICMP, транспортному рівні – TCP, UDP, на прикладному рівні – HTTP, FTP, DNS, RSVP. З огляду на бурхливий розвиток мережних технологій, у майбутньому рівень зв'язку, можливо, буде залежати й від інших протоколів.

Для забезпечення надійного транспорту повідомлень в GRID-системі повинні використовуватися рішення, що передбачають гнучкий підхід до безпеки комунікацій (можливість контролю над рівнем захисту, обмеження делегування прав, підтримка надійних транспортних протоколів). У цей час ці рішення ґрунтуються як на існуючих стандартах безпеки, споконвічно розроблених для Інтернет (SSL, TLS), так і на нових розробках.

Ресурсний рівень

Ресурсний рівень (Resource Layer) побудований над протоколами комунікації й автентифікації рівня зв'язку архітектури GRID. Ресурсний рівень реалізує протоколи, що забезпечують виконання наступних функцій:

– узгодження політик безпеки використання ресурсу;

– процедура ініціації ресурсу;

– моніторинг стану ресурсу;

– контроль над ресурсом;

– облік використання ресурсу.

Протоколи цього рівня опираються на функції базового рівня для доступу

й контролю над локальними ресурсами. На ресурсному рівні протоколи взаємодіють із ресурсами, використовуючи уніфікований інтерфейс і не розрізняючи архітектурні особливості конкретного ресурсу.

Розрізняють два основних класи протоколів ресурсного рівня:

– інформаційні протоколи, які одержують інформацію про структуру й стан ресурсу, наприклад, про його конфігурацію, поточне завантаженні, політику використання;

– протоколи керування, які використовуються для узгодження доступу до поділюваних ресурсів, визначаючи вимоги й припустимі дії стосовно ресурсу (наприклад, підтримка резервування, можливість створення процесів, доступ до даних). Протоколи керування повинні перевіряти відповідність запитуваних дій політиці поділу ресурсу, включаючи облік і можливу оплату. Вони можуть підтримувати функції моніторингу статусу й керування операціями.

Список вимог до функціональності протоколів ресурсного рівня близький до списку для базового рівня архітектури GRID. Додалася лише вимога єдиної семантики для різних операцій з підтримкою системи оповіщення про помилки.

Колективний рівень

Колективний рівень (Collective Layer) відповідає за глобальну інтеграцію різних наборів ресурсів, на відміну від ресурсного рівня, сфальцьованого на роботі з окремо взятими ресурсами. У колективному рівні розрізняють загальні й специфічні (для додатків) протоколи. До загальних протоколів відносяться, у першу чергу, протоколи виявлення й виділення ресурсів, системи моніторингу й авторизації співтовариств. Специфічні протоколи створюються для різних додатків GRID, (наприклад, протокол архівації розподілених даних або протоколи керування завданнями збереження стану й т.п.).

Компоненти колективного рівня пропонують величезну розмаїтість методів спільного використання ресурсів. Нижче наведені функції й сервіси, реалізовані в протоколах даного рівня:

– сервіси каталогів дозволяють віртуальним організаціям виявляти вільні

ресурси, виконувати запити по іменах і атрибутам ресурсів, таким як тип і завантаження;

– сервіси спільного виділення, планування й розподілу ресурсів забезпечують виділення одного або більше ресурсів для певної мети, а також планування виконуваних на ресурсах завдань;

– сервіси моніторингу й діагностики відслідковують аварії, атаки й перевантаження.

– сервіси дублювання (реплікації) даних координують використання ресурсів пам'яті в рамках віртуальних організацій, забезпечуючи підвищення швидкості доступу до даних відповідно до обраних метрик, такими як час відповіді, надійність, вартість і т.п.;

– сервіси керування робочим завантаженням застосовуються для опису й керування багатокроковими, асинхронними, багатокомпонентними завданнями;

– служби авторизації співтовариств сприяють поліпшенню правил доступу до поділюваних ресурсів, а також визначають можливості використання ресурсів співтовариства. Подібні служби дозволяють формувати політики доступу на основі інформації про ресурси, протоколи керування ресурсами й протоколах безпеки єднального рівня;

– служби обліку й оплати забезпечують збір інформації про використання ресурсів для контролю звертань користувачів;

– сервіси координації підтримують обмін інформацією в потенційно великому співтоваристві користувачів.

Прикладний рівень

Прикладний рівень (Application Layer) описує користувальницькі додатки, що працюють у середовищі віртуальної організації. Додатка функціонують, використовуючи сервіси, визначені на нижчих рівнях. На кожному з рівнів є певні протоколи, що забезпечують доступ до необхідних служб, а також прикладні програмні інтерфейси (Application Programming Interface – API), що відповідають даним протоколам.

Для полегшення роботи із прикладними програмними інтерфейсами користувачам надаються набори інструментальних засобів для розробки програмного забезпечення (Software Development Kit – SDK). Набори інструментальних засобів високого рівня можуть забезпечувати функціональність із одночасним використанням декількох протоколів, а також комбінувати операції протоколів з додатковими викликами прикладних програмних інтерфейсів нижнього рівня.

Оборотний увага, що додатка на практиці можуть викликатися через досить складні оболонки й бібліотеки. Ці оболонки самі можуть визначати протоколи, сервіси й прикладні програмні інтерфейси, однак подібні надбудови не відносяться до фундаментальних протоколів і сервісів, необхідним для побудови GRID-систем.

Поняття про віртуальну організацію

Інфраструктура GRID заснована на наданні ресурсів у загальне користування, з одного боку, і на використанні привселюдно доступних ресурсів, з іншої. У цьому плані ключове поняття інфраструктури GRID – віртуальна організація, у якій кооперуються як споживачі, так і власники ресурсів. Мотиви кооперації можуть бути різними. В існуючих GRID-системах віртуальна організація являє собою об'єднання (коллаборацію) фахівців з деякої прикладної області, які поєднуються для досягнення загальної мети.

Будь-яка віртуальна організація має у своєму розпорядженні певну кількість ресурсів, які надані зареєстрованими в ній власниками (деякі ресурси можуть одночасно належати декільком В). Кожна віртуальна організація самостійно встановлює правила роботи для своїх учасників, виходячи з дотримання балансу між потребами користувачів і наявним обсягом ресурсів, тому користувач повинен обґрунтувати своє бажання працювати з GRID-системою й дістати згоду керуючих органів В.

GRID-система є середовищем колективного комп’ютингу, у якій кожний ресурс має власника, а доступ до ресурсів відкритий у поділюваному за часом і

по просторі режимі безлічі вхідних у віртуальну організацію користувачів. Віртуальна організація може утворюватися динамічно й мати обмежений час існування.

Таким чином, можна визначити GRID-систему як просторово розподілене операційне середовище із гнучким, безпечним і скоординованим поділом ресурсів для виконання додатків у рамках певних віртуальних організацій.

Розподіл ресурсів в GRID

Ефективний розподіл ресурсів і їхня координація є основними завданнями системи GRID, і для їхнього рішення використовується планувальник (брокер ресурсів). Користуючись інформацією про стан GRID-системи, планувальник визначає найбільш підходящі ресурси для кожного конкретного завдання й резервує їх для її виконання. Під час виконання завдання може запросити в планувальника додаткові ресурси або звільнити надлишкові. Після завершення завдання всі відведені для неї обчислювальні ресурси звільняються, а ресурси пам'яті можуть бути використані для зберігання результатів роботи.

Важливою властивістю систем GRID є те, що користувачеві не потрібно знати про фізичне розташування ресурсів, відведених його завданню. Вся робота з керування, перерозподілу й оптимізації використання ресурсів лягає на планувальник і виконується непомітно для користувача. Для користувача створюється ілюзія роботи в єдиному інформаційному просторі, що володіє величезними обчислювальними потужностями й обсягом пам'яті.

GRID є найбільш складним інформаційним середовищем, коли-або створюваної людиною. Для системи такої складності дуже важлива проблема забезпечення надійного функціонування й відновлення при збоях. Людина не здатна встежити за станом тисяч різних ресурсів, що входять в GRID-систему, і із цієї причини завдання контролю над помилками покладає на систему моніторингу, що стежить за станом окремих ресурсів. Дані про стан заносяться в інформаційні ресурси, звідки вони можуть бути прочитані планувальником і іншими сервісами, що дозволяє мати достовірну інформацію, що постійно обновляється, про стан ресурсів.

В GRID-системах використовується складна система виявлення й класифікації помилок. Якщо помилка відбулася з вини завдання, то завдання буде зупинена, а відповідна діагностика спрямована її власникові (користувачеві). Якщо причиною збоїти послужив ресурс, то планувальник зробить перерозподіл ресурсів для даного завдання й запустить знову її.

Збої ресурсів є не єдиною причиною відмов в GRID-системах. Через величезну кількість завдань і постійно мінливої складної конфігурації системи важливо вчасно визначати перевантажені й вільні ресурси, роблячи перерозподіл навантаження між ними. Перевантажений мережний ресурс може стати причиною відмови значної кількості інших ресурсів. Планувальник, використовуючи систему моніторингу, постійно стежить за станом ресурсів і автоматично вживає необхідних заходів для запобігання перевантажень і простою ресурсів.

У розподіленому середовищі, який є GRID-система, життєво важливою властивістю є відсутність так званої єдиної крапки збоїти. Це означає, що відмова будь-якого ресурсу не повинен приводити до збою в роботі всієї системи. Саме тому планувальник, система моніторингу й інші сервіси GRID-системи розподілені й продубльовані. Незважаючи на всю складність, архітектура GRID розроблялася з метою забезпечити максимальну якість сервісу для користувачів. В GRID-системах використовуються сучасні технології передачі даних, забезпечення безпеки й відказостійкості.

Інструментальні засоби GRID (Globus Toolkit)

У цьому розділі буде розглянутий набір інструментальних засобів, використовуваних при реалізації проектів GRID і розроблених у рамках проекту Глобус (Globus Project).

Ці інструментальні засоби утворять набір програмних засобів Globus Toolkit і дозволяють побудувати повнофункціональну GRID-систему. Засоби Globus Toolkit являють собою сукупність програмних компонентів, що реалізують необхідні частини архітектури.

Globus Toolkit складається з наступних основних компонентів:

– GRAM (Globus Resource Allocation Manager), відповідальний за створення/видалення процесів. Цей компонент Globus Toolkit установлюється на обчислювальному вузлі GRID-системи (вузлом може бути як робоча станція, так і обчислювальний кластер). Користувальницькі додатки формують запити до GRAM спеціальною мовою RSL (Resource Specification Language).

– MDS (Monitoring and Discovery Service) забезпечує способи подання інформації про GRID-систему. Ця інформація може бути найрізноманітнішою й містити, наприклад, дані про конфігурацію або стан як всієї системи, так і окремих її ресурсів (тип ресурсу, доступний дисковий простір, кількість процесорів, обсяг пам'яті, продуктивність та інше). Вся інформація логічно організована у вигляді дерева, і доступ до неї здійснюється по стандартному протоколі LDAP (Lightweight Directory Access Protocol).

– GSI (Globus Security Infrastructure) забезпечує захист, що включає шифрування даних, а також автентифікацію (перевірка дійсності, при якій установлюється, що користувач або ресурс дійсно є тим, за кого себе видає) і авторизацію (процедура перевірки, при якій установлюється, що автентифікований користувач або ресурс дійсно має викликані права доступу) з використанням цифрових сертифікатів X.509.

– GASS (Global Access to Secondary Storage) надає можливість зберігання масивів даних у розподіленому оточенні й доступу до цих даних. Визначає різні стратегії розміщення даних.

– Бібліотеки globus_io і Nexus використовуються як прикладними програмами так і компонентами Globus Toolkit для мережної взаємодії вузлів у гетерогенному середовищі.

Далі більш докладно розглянуті деякі із цих компонентів. Слід зазначити, що Globus Toolkit не містить брокера ресурсів, залишаючи завдання його реалізації розроблювачам, що створюють системи GRID на його основі.

Керування ресурсами

Архітектура засобів керування ресурсами (Globus Resource Management Architecture – GRMA) має багаторівневу структуру.

Запити користувальницьких додатків виражаються на RSL і передаються брокерові ресурсів, що відповідає за високорівневу координацію використання ресурсів (балансування завантаження) у певному домені. На основі переданого користувальницьким додатком запиту й політики (права доступу, обмеження по використанню ресурсів) відповідального адміністративного домену брокер ресурсів ухвалює рішення щодо того, на яких обчислювальних вузлах буде виконуватися завдання, який відсоток обчислювальної потужності вузла вона може використовувати й ін.

При виборі обчислювального вузла брокер ресурсів повинен визначити, які вузли доступні в сучасний момент, їхнє завантаження, продуктивність і інші параметри, зазначені в RSL-запиті, вибрати найбільш оптимальний варіант (це може виявитися один обчислювальний вузол або трохи), згенерувати новий RSL-запит (ground RSL) і передати його високорівневому менеджерові ресурсів (co-allocator). Цей запит буде містити вже більше конкретні дані, такі, як імена конкретних вузлів, необхідне кількість пам'яті й др. Основні функції високорівневого менеджера ресурсів перераховані нижче:

– колективне виділення ресурсів;

– додавання/видалення ресурсів до раніше виділеного;

– одержання інформації про стан завдань;

– передача початкових параметрів завданням.

Високорівневий менеджер ресурсів робить декомпозицію запитів ground RSL на безліч більше простих RSL-запитів і передає ці запити GRAM. Далі, при відсутності повідомлень про помилки від GRAM, завдання користувача запускається на виконання. У випадку, якщо один з GRAM повертає помилку, завдання або знімається з виконання, або спроба запуску виробляються повторно.

Менеджер GRAM надає верхнім рівням універсальний API для керування ресурсами вузла GRID. Сам GRAM взаємодіє з локальними засобами керування ресурсами вузла. Вузлом може бути, наприклад, робоча станція або обчислювальний кластер.

Організація доступу до ресурсів

GRAM – досить низькорівневий компонент Globus Toolkit, що є інтерфейсом між високорівневим менеджером ресурсів і локальною системою правління ресурсами вузла. У цей час цей інтерфейс може взаємодіяти з наступними локальними системами керування ресурсами:

– PBS (Portable Batch System) – система керування ресурсами й завантаженням кластерів. Може працювати на великій кількості різних платформ: Linux, FreeBSD, NetBSD, Digital Unix, Tru64, HP-UX, AIX, IRIX, Solaris. У цей час існує вільна й реалізація, що більше володіє широкими можливостями, PBS, називана Torque.

– LSF (Load Sharing Facility) – система, аналогічна PBS. Розроблена компанією Platform Computing. Також здатна працювати на безлічі платформ.

– NQE (Network Queuing Environment) – продукт компанії Cray Research, що використовується найчастіше як менеджер ресурсів на суперкомп'ютерах, кластерах і системах Cray, хоча може працювати й на інших платформах.

– LoadLeveler – продукт компанії IBM, керуючий балансом завантаження великих кластерів. Використовується в основному на кластерах IBM.

– Condor – вільно доступний менеджер ресурсів, розроблений в основному студентами різних університетів Європи й США. Аналогічний перерахованим вище. Працює на різних платформах UNIX і Windows NT.

– Easy-LL – спільна розробка IBM і Cornell Theory Center, призначена для керування великим кластером IBM у цьому центрі. По суті є об'єднанням LoadLeveler і продукту EASY лабораторії Argonne National Lab.

– fork – найпростіший стандартний засіб запуску процесів в UNIX.

Щоб на даному обчислювальному вузлі можна було віддалено запускати на виконання програми, на ньому повинен виконуватися спеціальний процес називаний Gatekeeper. Gatekeeper працює в привілейованому режимі й виконує наступні функції:

– робить взаємну автентифікацію із клієнтом;

– аналізує RSL запит;

– відображає клієнтський запит на обліковий запис деякого локального користувача;

– запускає від імені локального користувача спеціальний процес, називаний Job Manager, і передає йому список ресурсів, що вимагаються.

Після того, як Gatekeeper виконає свою роботу, Job Manager запускає завдання (процес або кілька процесів) і робить його подальший моніторинг, повідомляючи клієнта про помилки й інші події. Gatekeeper запускає тільки один Job Manager для кожного користувача, що управляє всіма завданнями даного користувача. Коли завдань більше не залишається, Job Manager завершує роботу.

Інформаційний сервіс

Всі перераховані компоненти, включаючи користувальницькі додатки, можуть використовувати інформаційний сервіс (Information Service) для одержання всієї необхідної інформації про стан GRID-системи. В Globus Toolkit роль інформаційного сервісу грає MDS. Цей компонент відповідає за збір і надання конфігураційної інформації, інформації про стан GRID-системи і її підсистем, а також забезпечує універсальний інтерфейс одержання необхідної інформації. MDS має децентралізовану, легко масштабовану структуру й працює як зі статичними, так і з динамічно мінливими даними, необхідними користувальницьким додаткам і різним сервісам GRID-системи. Ієрархічна структура MDS представлена на мал. 5.

MDS складається із трьох основних компонентів:

– IP (Information Provider) – є джерелом інформації про конкретний ресурс або частину ресурсу;

– GRIS (Grid Resource Information Service) – надає інформацію про вузол GRID-системи, що може бути як обчислювальним вузлом, так і яким-небудь іншим ресурсом. GRIS опитує індивідуальні IP і поєднує отриману від них інформацію в рамках єдиної інформаційної схеми;

– GIIS (Grid Index Information Service) – поєднує інформацію з різних GRIS або інших GIIS. Для зменшення часу реакції на запит і зниження мережного трафіку GIIS кешує дані. GIIS верхнього рівня містить всю інформація про стан даної системи GRID.

3.2 Розробка структурної схеми

Паралельне завдання визначається як програма, реалізована у вигляді декількох компонентів, кожна з яких запускається на окремому процесорному вузлі й у ході свого виконання може взаємодіяти з іншими компонентами, а також уводиться поняття планування як основного етапу керування паралельними завданнями.

Існують дві групи методів планування, використовуваних у сучасних паралельних архітектурах і кластерних системах:

– Методи першої групи використовують поділ часу процесорів між декількома завданнями. Відзначається основний недолік цих методів, що складає в тому, що компоненти одного завдання можуть бути перервані й перезапущені в різні моменти часу, що приводить до сильного зниження загальної ефективності використання ресурсів.

– Друга група методів планування (FCFS, First-Fit, Backfill) заснована на ідеї поділу простору ресурсів між завданнями, відповідно до якої кожне завдання одержує необхідний обсяг ресурсів на необхідний час в ексклюзивному режимі. Докладно розглядається широко застосовується на практиці метод зворотного заповнення Backfill, що гарантує запуск паралельного завдання й разом з тим

ефективно використовує ресурси завдяки виділенню ресурсів завданням не безпосередньо в момент звільнення, а завчасно.

Розглянемо технології розподіленого комп’ютінгу, зокрема, концепція й базові принципи GRID як способу організації інфраструктури високопродуктивного комп’ютінгу. Сучасний і найпоширеніший підхід до такої організації складається в побудові GRID, що застосовується для проведення наукових досліджень, із просторово розподілених багатопроцесорних установок, таких як масивно-паралельні комп'ютери (MPP) і кластери. Відзначається, що в існуючих проектах (EGEE, Grid2003, NorduGrid і ін.) переважає спосіб створення GRID, коли самі власники не використовують свої ресурси, а віддають їх цілком в GRID. Такий спосіб організації ресурсів, називаний GRID з відчужуваними ресурсами, не може претендувати на універсальність, однак може бути корисний у спеціальних умовах застосування, наприклад, коли власники самі не використовують свої ресурси, а виступають у ролі провайдерів. На практиці представляється більше цікавим спосіб, при якому ресурси використовуються спільно їхніми власниками й користувачами GRID. Цей спосіб організації називають GRID з невідчужуваними ресурсами. Його перевага полягає в тому, що такий GRID можна створити лише на якийсь час, необхідне для рішення конкретного завдання, і без формування спеціальної ресурсної бази: досить лише утягнути вже наявні ресурси, не завантажуючи повністю свої власні.

Розглянемо специфіку керування паралельними завданнями в GRID у порівнянні з паралельними комп'ютерними архітектурами й кластерними системами. Найбільш значимі відмітні властивості GRID, що ускладнюють завдання коаллокації:

– Автономність.

– Гетерогенність.

– Просторова розподіленість.

– Дворівнева організація ресурсів.

Завдання коаллокації – центральне завдання планування паралельних завдань, що складає в розміщенні набору компонент паралельного завдання на безлічі ресурсів. Для умов GRID вона формулюється в такий спосіб. Як ресурси розглядаються обчислювальні ресурси GRID – кластери $R = \{R_1, R_2, \ldots, R_N\}$, кожний з яких являє собою деяку кількість процесорних вузлів – процесорів і пов'язану з кожним з них оперативну й дискову пам'ять. На ці ресурси необхідно розмістити P компонент паралельного завдання $J = \{J_1, J_2, \ldots, J_P\}$, синхронно виділяючи їм ресурси з безлічі ресурсів R на час виконання T .

У результаті рішення завдання визначається час старту завдання й набір аллокацій ресурсів: $A = \{A_x(r_{Ji}, t_0, t_0 + T), \ldots, A_K(r_{JK}, t_0, t_0 + T)\}$, $K = 1, P$, де r_{Ji} – ресурси, використовувані для аллокації такі, що:

$$r_{Ji} \subseteq R_j, i = 1,K, r_{J1} + r_{J2} + \ldots + r_{Jдо} = P,$$

де r_{Ji} – кількість аллоційованих процесорних вузлів на ресурсі. Всі аллокації починаються в один час і мають однакову тривалість. Крім того, забезпечується виконання двох умов:

– характеристики виконавчих ресурсів відповідають вимогам компонентів завдання;

– виконавчі ресурси в період $[t_0, t_0 + T]$ вільні, і політика поділу ресурсів не перешкоджає їхньому виділенню для завдання.

Перераховані вище властивості GRID роблять можливість побудови точних аллокацій $A_1, \ldots, A_K$, $K = 1,P$, у яких визначається безліч виконавчих процесорів $r_h, \ldots, r_{JK}$ і часовий інтервал $[t_0, t_0 + T]$, на який вони приділяються цьому завданню, проблематично. Замість цього в багатьох практичних реалізаціях результатом планування є аллокації, у яких час початку й ресурси точно не визначені. Для паралельних завдань це спричиняє серйозні дефекти при їхній обробці, такі як «зависання» завдань у черзі кластера.

Робиться вивід про те, що для запобігання небажаних ефектів планування в GRID, що застосовується для проведення наукових досліджень, повинне бути детермінованим, тобто повинне будувати точні аллокації, і, крім того, запуск

завдань на виконавчих ресурсах повинен здійснюватися відповідно до побудованого аллокаціями.

Обґрунтуємо вибір на користь пріоритетного планування для застосування в GRID, що застосовується для проведення наукових досліджень, і розглянемо класифікацію методів такого планування.

Як показує практика, методи поділу часу важко застосовні в умовах GRID. Крім того, ці методи спрямовані на поліпшення середнього часу обробки завдань, тоді як в GRID планування в першу чергу повинне мати на меті забезпечення якості обслуговування окремих користувачів. Для цього більше підходять методи поділу простору ресурсів, які в цей час переважно використовуються в GRID, що застосовується для проведення наукових досліджень. Серед них виділяються методи пріоритетного планування й методи, засновані на інтегральних критеріях. Останні здатні оптимізувати завантаження ресурсів або загальний час рахунку пакета завдань. Представляється, що планування в GRID у першу чергу повинне бути спрямоване на справедливий поділ ресурсів між завданнями, забезпечуючи при цьому можливість керування порядком виділення ресурсів для окремих завдань. Для таких цілей найбільш підходящим є пріоритетне планування.

Спосіб рішення завдання коаллокації залежить значною мірою від того, яка інформація про ресурси для планування є в наявності. За цим критерієм сучасні методи пріоритетного планування розділяються на два класи. До першого класу належать методи, засновані на використанні черг завдань, основний принцип яких полягає у виділенні ресурсів для вартих у черзі завдань, виходячи з поточного стану ресурсів.

Застосування цих методів для паралельних завдань в GRID неефективно по двох причинах:

– По-перше, нагромадження ресурсів, необхідних для запуску завдання, може бути здійснено лише за допомогою блокування вільних ресурсів, що приводить до їхнього простою протягом невизначеного часу.

– По-друге, механізм попереднього резервування, призначений для забезпечення детермінованого планування, важко застосуємо через відсутність інформації про час звільнення необхідного завданню обсягу ресурсів.

Другий клас включає методи, що використовують для розподілу ресурсів повноцінний план на майбутнє або розклад. До них ставиться метод зворотного заповнення (Backfill). З їхньою допомогою автоматично забезпечується нагромадження ресурсів і природно реалізується механізм попереднього резервування ресурсів, що дозволяє використовувати методи цього класу для обслуговування паралельних завдань в GRID.

Опишемо запропонований метод керування паралельними завданнями за допомогою випереджального планування для практично важливої форми організації GRID, що застосовується для проведення наукових досліджень, з невідчужуваними ресурсами, коли вони не виділяються в GRID повністю, а використовуються в режимі поділу з їхніми власниками. У цих умовах завдання надходять не тільки з GRID (глобальний рівень), але й від користувачів кластера (локальний рівень) (рис. 3.1).

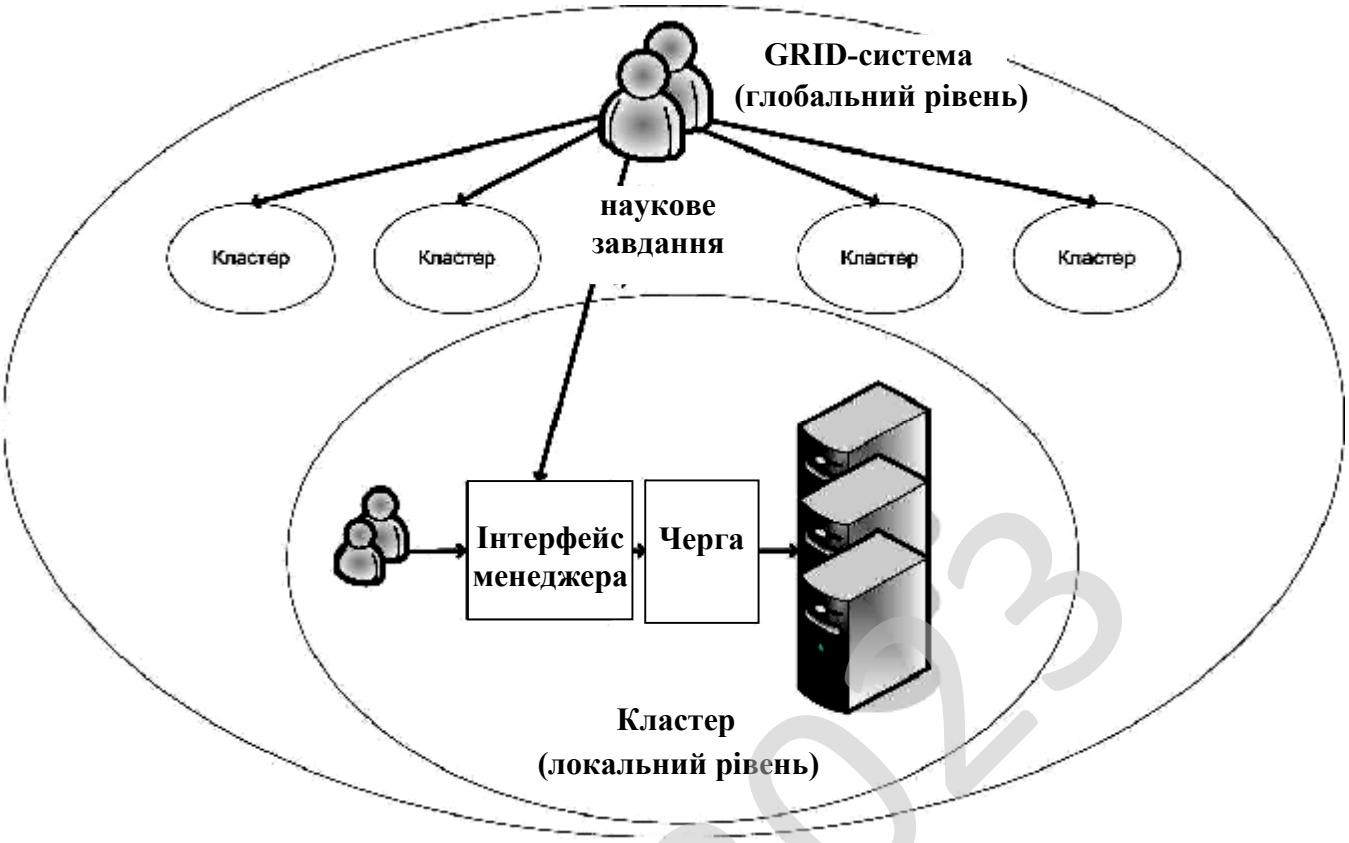

Рисунок 3.1 – Структурна схема системи

Особливістю запропонованого рішення є те, що ресурси розподіляються не в момент їхнього звільнення, а будується розклад їхнього використання на деякий період часу у вигляді безлічі тимчасових слотів, де кожний слот відповідає початку й кінцю деякого завдання або вільній ділянці розкладу. Виконання розкладу забезпечується за допомогою попереднього резервування ресурсів. Завдяки цьому планування в умовах невідчуждаємості ресурсів є детермінованим.

Розглянемо економічний підхід до планування в GRID, що застосовується для проведення наукових досліджень, для керування паралельними завданнями. Відповідно до цього підходу процес розподілу ресурсів організується на базі моделі ринку: власники ресурсів виступають як продавці, а користувачі – як покупці цих ресурсів. Реалізуючи такий підхід, пропонуються дві угоди по поділу ресурсів і способи їхньої підтримки при плануванні.

Перша угода стосується поділу ресурсів між користувачами. Основна вимога полягає в тому, що користувач повинен мати можливість управляти швидкістю одержання ресурсів при плануванні. Для цього їм установлюється плата за виконання кожного завдання в рамках виділеного йому бюджету. При збільшенні плати шанси швидше виконати завдання підвищуються, однак при цьому обмеження бюджету не дозволяє користувачеві монополізувати весь ресурсний пул.

Для поділу ресурсів між двома потоками завдань, що надходять із локального й глобального рівнів, пропонується друга угода про поділ ресурсів: глобальне завдання може зайняти ресурси за умови, що плата, призначена користувачем за виконання завдання, не менше їхньої вартості. Така угода дозволяє здійснювати динамічне балансування потоків завдань, забезпечуючи конкуренцію глобальних і локальних завдань, і є ключовим для GRID з невідчужуваними ресурсами. Відповідно до цього підходу глобальні завдання можуть розміщатися не тільки на вільних ресурсах, але й на ресурсах, на якому претендують локальні завдання, за умови, що плата за ці завдання не нижче вартості ресурсів. Таким чином, планування паралельних завдань полягає в підборі безлічі «підходящих» слотів, які:

– можна синхронно виділити в кількості, необхідній завданню;

– задовольняють ресурсному запиту завдання;

– підходять по вартості.

Приведемо алгоритм планування, що вирішує завдання коаллокації ресурсів з урахуванням угод по їхньому поділу, а також проведемо його обґрунтування й експериментальну оцінку, проведену на розробленому прототипі диспетчера завдань. Один із кращих на сьогоднішній день методів планування паралельних завдань – метод зворотного заповнення (Backfill) – в умовах невідчуждаємості ресурсів має квадратичну складність від загальної кількості слотів. Для цих умов у роботі пропонується алгоритм планування лінійної складності і його варіант, що реалізує критерій якнайшвидшого завершення завдань.

Алгоритм розрахований на наступні умови:

– Усі компоненти паралельного завдання мають ті самі вимоги до ресурсів комп'ютера, виражені в ресурсному запиті.

– Плата за кожний компонент завдання становить RC / P, де P – кількість компонентів, а RC – вартість завдання в цілому. Таким чином, відповідно до принципу поділу ресурсів, завданню можуть бути виділені тільки ті з них, які мають вартість, меншу RC / P .

Алгоритм підбирає слоти для одного завдання у два етапи й заснований на спільному використанні двох подань розкладу у вигляді списків L_x і L_2, у першому з яких слоти відсортовані по зростанню часу початку, а в другому – по зростанню часу кінця. Це сортування виконується один раз для всієї черги завдань перед початком роботи алгоритму.

На першому етапі алгоритму підраховується кількість підходящих слотів у кожний момент часу. Зміна цього числа відбувається тільки в точках часової осі, що відповідають початку або кінцю якого-небудь слоту. Для підрахунку виконується паралельний прохід по списках L_1 і L_2. При переході до нового слоту зі списку L_1, час початку аллокації t установлюється рівним початку цього слоту. Здійснюється перевірка, чи є слот підходящим для запуску завдання. Якщо це так, то кількість підходящих слотів збільшується. Після цього проходяться слоти другого списку, кінець яких менше $t + T$, і відповідно зменшується кількість підходящих слотів. Як тільки набирається необхідна кількість підходящих слотів P, цей етап роботи алгоритму завершується, і виходить час $l_0 = t$, починаючи з якого для завдання можна синхронно виділити необхідну кількість слотів.

Якщо на першому етапі необхідна кількість слотів знайдена, то всі слоти, що лежать на інтервалі $[t_0, t_0 + T]$ і які є підходящими, можуть бути використані для його запуску. Для їхнього відбору відбувається ще один прохід по першому списку (другий етап). Отриманий набір слотів (або їхніх частин) є шуканим набором аллокацій.

Представлений алгоритм визначає самий ранній час старту завдання. Якщо в системі є процесорні вузли з різною продуктивністю, то більше корисним

є критерій якнайшвидшого завершення завдання. Для реалізації цього критерію пропонується модифікація алгоритму, що намагається в першу чергу підібрати слоти на найбільш продуктивних процесорах, щоб виконати завдання якнайшвидше.

Для аналізу ефективності розробленого алгоритму планування реалізований планувальник, розглянутий далі і підготовлене експериментальне середовище, заснована на подійному моделюванні. З його допомогою отримані показники, використовувані для оцінки ефективності алгоритмів планування: середній час очікування, час рахунку пакета завдань і ступінь завантаження ресурсів, а також побудована залежність середнього часу планування від кількості завдань у системі. На основі отриманих результатів проведене порівняння розробленого алгоритму планування з алгоритмом FCFS в умовах GRID з невідчужуваними ресурсами, що показав істотно більш високі показники ефективності планування першого алгоритму. Також досліджений вплив зміни вартості ресурсів на час відгуку й масштабованість планувальника.

3.3 Розробка функціональної схеми

Розглянемо функціональну схему системи диспетчеризації паралельних завдань в GRID, що застосовується для проведення наукових досліджень, в основу якої покладені розроблені в магістерській роботі метод і алгоритм. Система ґрунтується на сучасному підході до побудови GRID-систем – Відкритій архітектурі GRID-служб (OGSA), реалізованої за допомогою концепції веб-служб (Web Service Architecture – WSA) і відповідних стандартних протоколів. В основу реалізації покладене проміжне програмне забезпечення, визнане стандартом де-факто, – інструментарій Globus Toolkit 4. Особливістю системи є те, що для планування диспетчер використовує прогноз використання ресурсів на майбутнє, що дозволяє ефективно розподіляти завдання по ресурсах.

Опишемо функціональну схему системи, що включає в себе три основних компоненти:

– Диспетчер.

– Ресурсний агент.

– Користувальницький інтерфейс.

На рис. 3.2 сірим кольором виділені розроблені блоки, що є частиною системи, а компоненти, реалізовані у вигляді веб-служб (WS-компоненти), укладені в жирну рамку. Кожному компоненту присвячений окремий розділ.

Розглянемо диспетчер, що представляється набором GRID-служб, що реалізують базові компоненти, необхідні для розподілу завдань по ресурсах.

Склад диспетчера включає:

– службу прийому команд від користувачів;

– службу прийому інформації про ресурси;

– базу даних планування;

– службу планування;

– службу керування запуском, що включає в себе менеджер резервування, менеджер запуску й менеджер доставки даних.

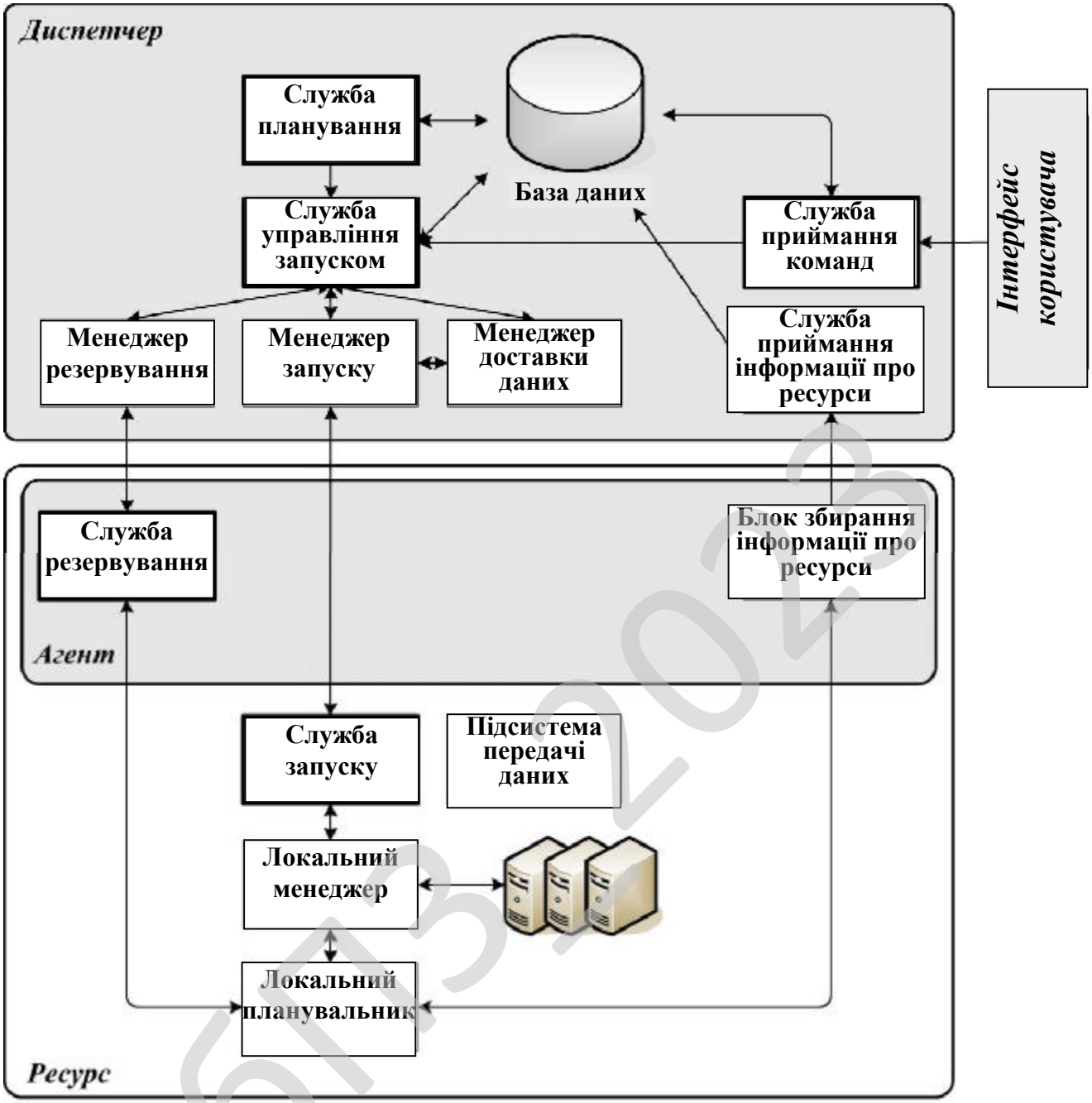

Рисунок 3.2 – Функціональна схема системи

Робота служби планування – основного компонента диспетчера – є циклічною: на кожному циклі будується план розподілу завдань по ресурсах на основі інформації, що перебуває в базі даних планування. Під час роботи циклу всі завдання, що надходять від служби прийому завдань, і відновлення кластерних розкладів, що поставляються ресурсними агентами, буферізуються

засобами СУБД і не враховуються на поточному циклі. По закінченню циклу вони актуалізуються в базі даних для використання на наступних циклах. Після побудови плану визначається підмножина завдань, час початку аллокацій яким досить близько до моменту початку виконання. Для цих завдань за допомогою служби попереднього резервування виконується резервування ресурсів, що гарантує їхнє виділення відповідно до побудованого планувальника аллокаціями, і ініціюється доставка завдань у кластери засобами Globus Toolkit.

Опишемо ресурсний агент, основною функцією якого є збір інформації про стан ресурсу, зокрема, побудову прогнозу, і передача цієї інформації диспетчерові. Крім того, на ресурсного агента покладена функція резервування ресурсів, необхідна для обслуговування паралельних завдань. Реалізація ресурсного агента, заснована на спеціальному режимі SIMULATION кластерного планувальника Maui. Цей режим дозволяє моделювати процес розміщення завдань у кластері в майбутньому часі, забезпечуючи генерацію локальних розкладів. Для відстеження зміни стану кластера запропонований спосіб, заснований на «прослуховуванні» повідомлень, які локальний менеджер посилає планувальникові Maui. Таким чином, ресурсний агент довідається про події, що відбуваються в кластері, і управляє моделюванням, у результаті якого будується розклад. Також запропонований метод динамічного керування кроком планування, що дозволяє істотно скоротити час побудови прогнозу. На основі засобів Maui, що дозволяють зарезервувати локальні ресурси, розроблений і задіяний у диспетчері механізм попереднього резервування ресурсів, у реалізації якого вирішена важливе завдання закріплення резервування за конкретним завданням GRID, виконання якого заплановане на відповідних ресурсах.

Користувальницький інтерфейс надає можливість запускати завдання й одержувати інформацію про його стан.

3.4 Розробка діаграми процесів

Діаграма процесів розробленої системи зображена на рисунку 3.3. На рисунку представлена реалізація як серверної так і клієнтської частини ПЗ. Розглянемо розроблене серверне ПЗ яке зображено з лівого боку схеми. Після початку роботи розробленого ПЗ ми потрапляємо до головного вікна серверного ПЗ звідки можемо перейти до БД системи, блоку перевірки доступу до мережі, модулю захисту ПЗ, налаштування ПЗ. Далі через обробник помилок та менеджер запуску можемо потрапити до менеджеру резервування з проведенням отримання та передачі даних. Розглянемо розроблене клієнтське ПЗ зображено з правого боку схеми. Після початку роботи розробленого ПЗ ми також потрапляємо до головного вікна клієнтського ПЗ, далі проводиться перевірка доступу до мережі, робота з локальним планувальником, локальним менеджером та через службу запуску та розрахунковий модуль потрапляємо до підсистеми передачі даних.

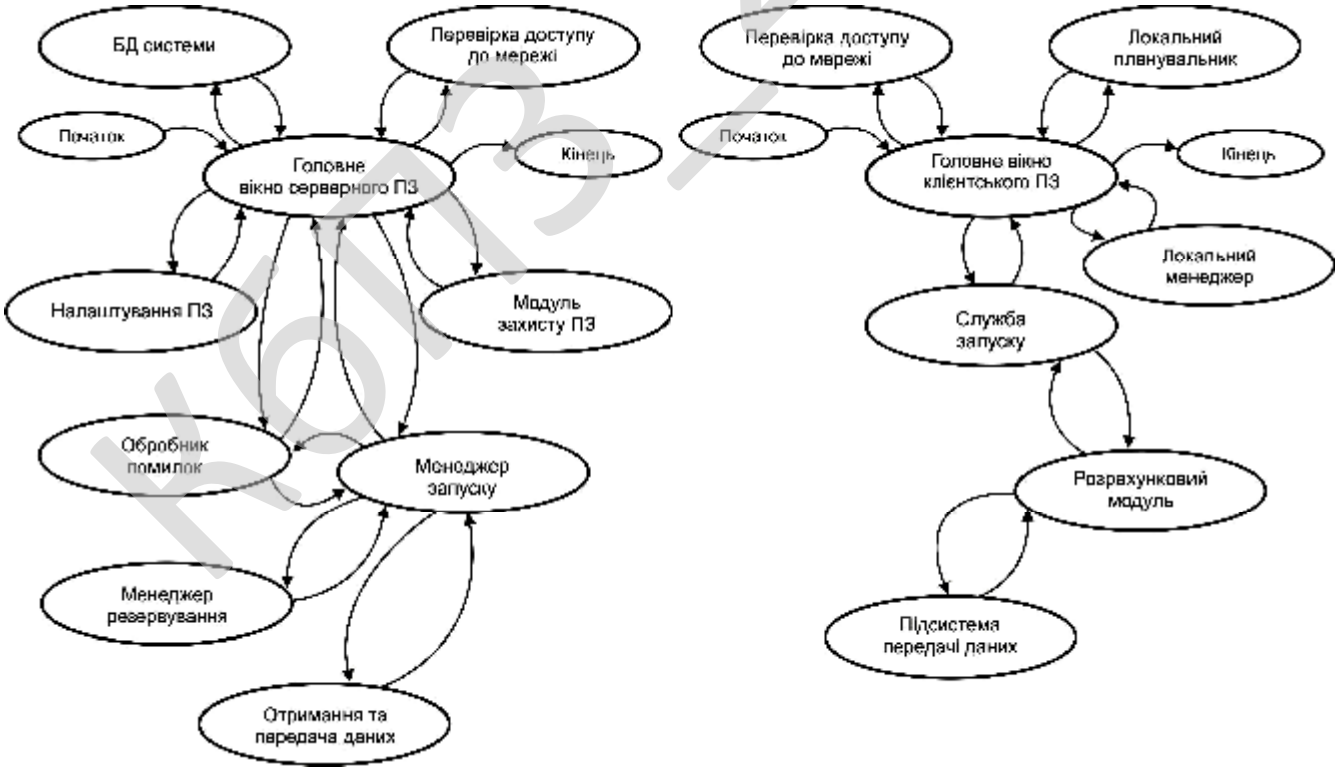

Рисунок 3.3 – Діаграма взаємодії процесів

4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ПРАВИЛЬНІСТЬ ПРОЕКТНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

Розглянемо алгоритм роботи серверної частини програми. Його блок-схема зображена на рисунку 4.1. З рисунку видно, що після запуску серверної частини програми спочатку відбувається виділення пам'яті ПЗ. Потім здійснюється:

– Ініціалізація ресурсів ПЗ.

– Підключення файлу налаштувань.

– Перевірка доступу до БД.

– Запит з БД повернуто?

– Виведення головної форми ПЗ.

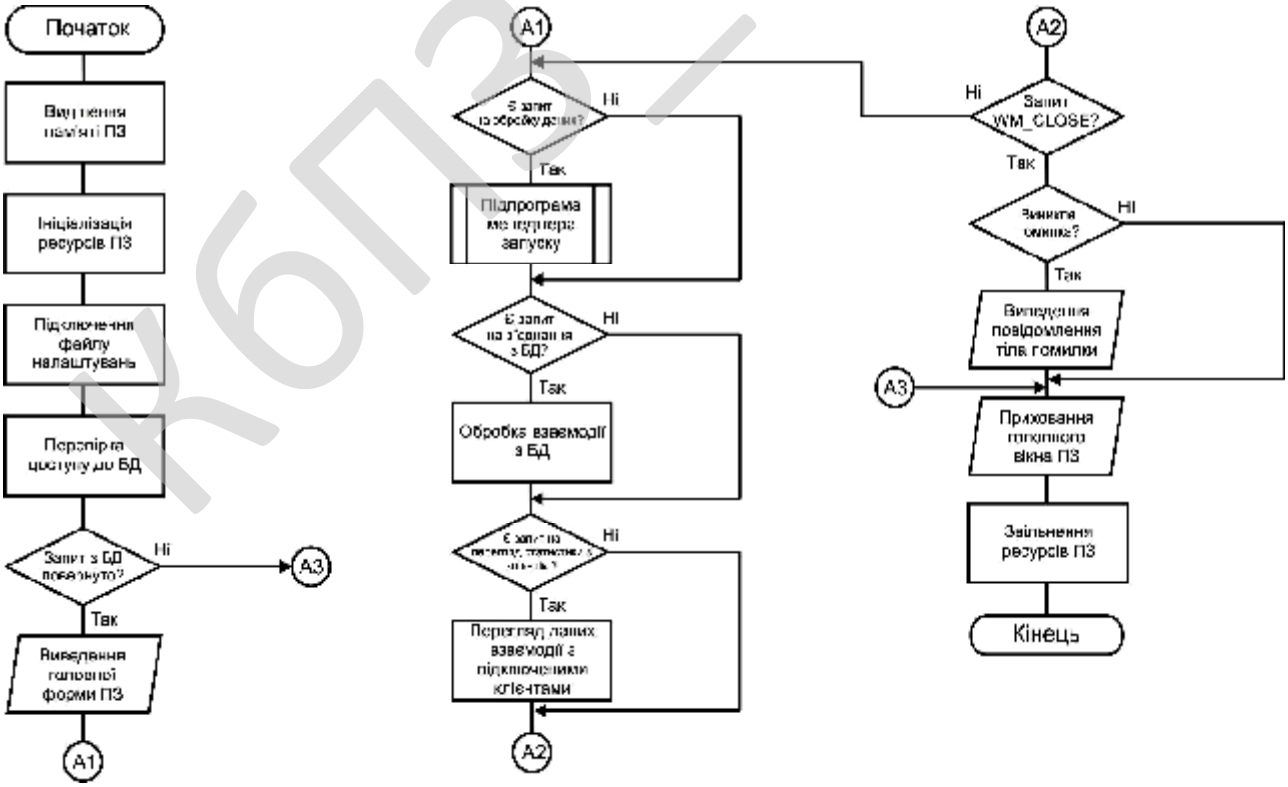

Рисунок 4.1 – Блок-схема серверної програми

– Є запит на обробку даних?

– Підпрограма менеджера запуску.

– Є запит на з’єднання з БД?

– Обробка взаємодії з БД.

– Є запит на перегляд статистики з клієнтів ?

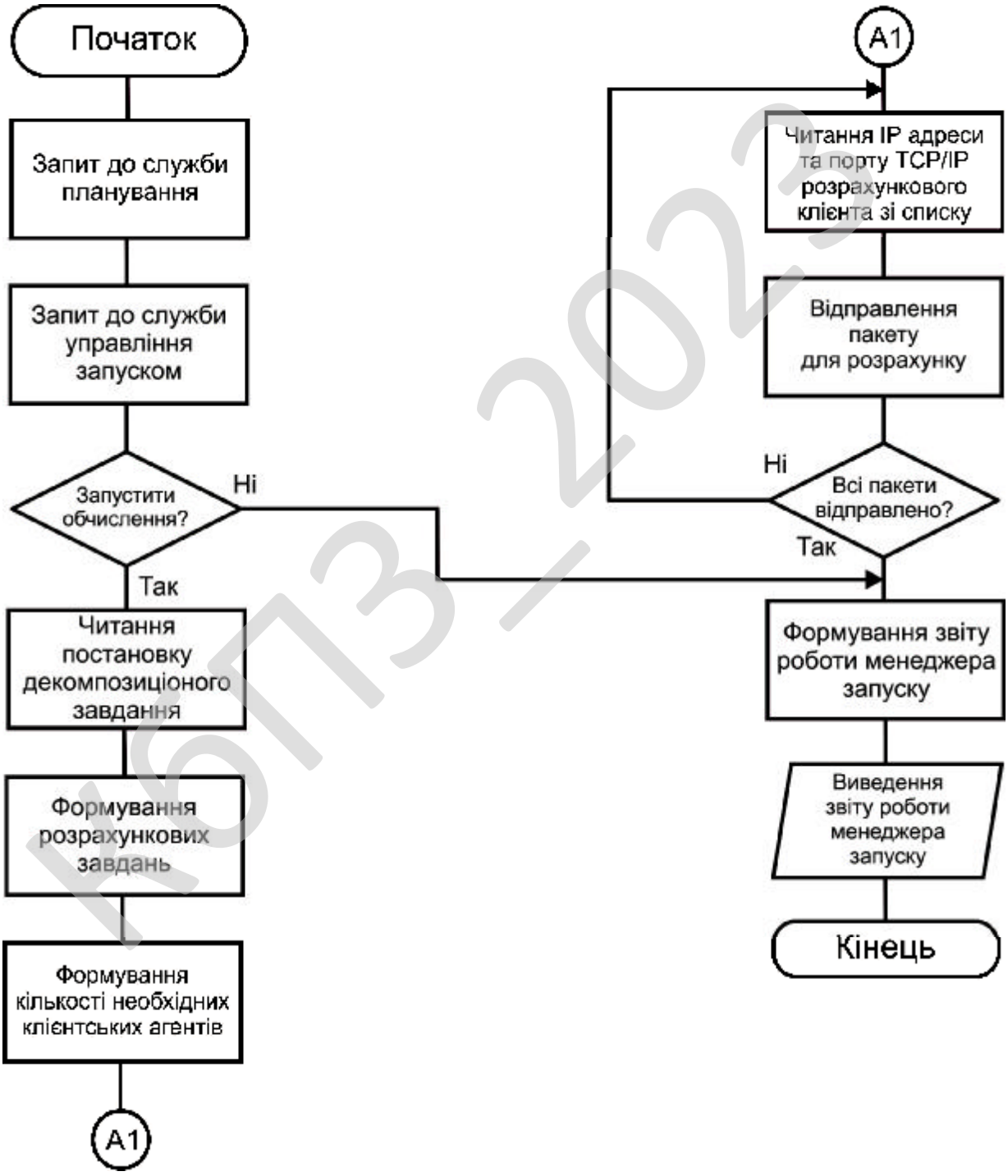

Рисунок 4.2 – Блок-схема підпрограми менеджера запуску

– Перегляд даних взаємодії з підключеними клієнтами.

– Запит WM_CLOSE?

– Виникла помилка (запит).

– Виведення повідомлення тіла помилки.

– Приховання головного вікна ПЗ.

– Звільнення ресурсів ПЗ.

На рисунку 4.2 зображено роботу підпрограми менеджера запуску розробленого серверного ПЗ. У підпрограмі виконуються наступні дії:

– Запит до служби планування.

– Запит до служби управління запуском.

– Запустити обчислення?

– Читання постановку декомпозиціоного завдання.

– Формування розрахункових завдань.

– Формування кількості необхідних клієнтських агентів.

– Читання IP адреси та порту TCP/IP розрахункового клієнта зі списку.

– Відправлення пакету для розрахунку.

– Всі пакети відправлено?

– Формування звіту роботи менеджера запуску.

– Виведення звіту роботи менеджера запуску.

На рисунку 4.2 зображено роботу розробленого клієнтського ПЗ. У програмі виконуються наступні дії:

– Виділення пам’яті ПЗ.

– Ініціалізація початкових змінних ПЗ.

– Виведення головного вікна ПЗ.

– Читання налаштувань та встановлення режиму взаємодії з сервером.

– Відкриття портів TCP/IP для передачі даних.

– Читання налаштувань та встановлення адреси серверу БД.

– Підключення до БД.

– БД підключено (запит).

– Очікування команди серверу.

– Отримана команда (запит).

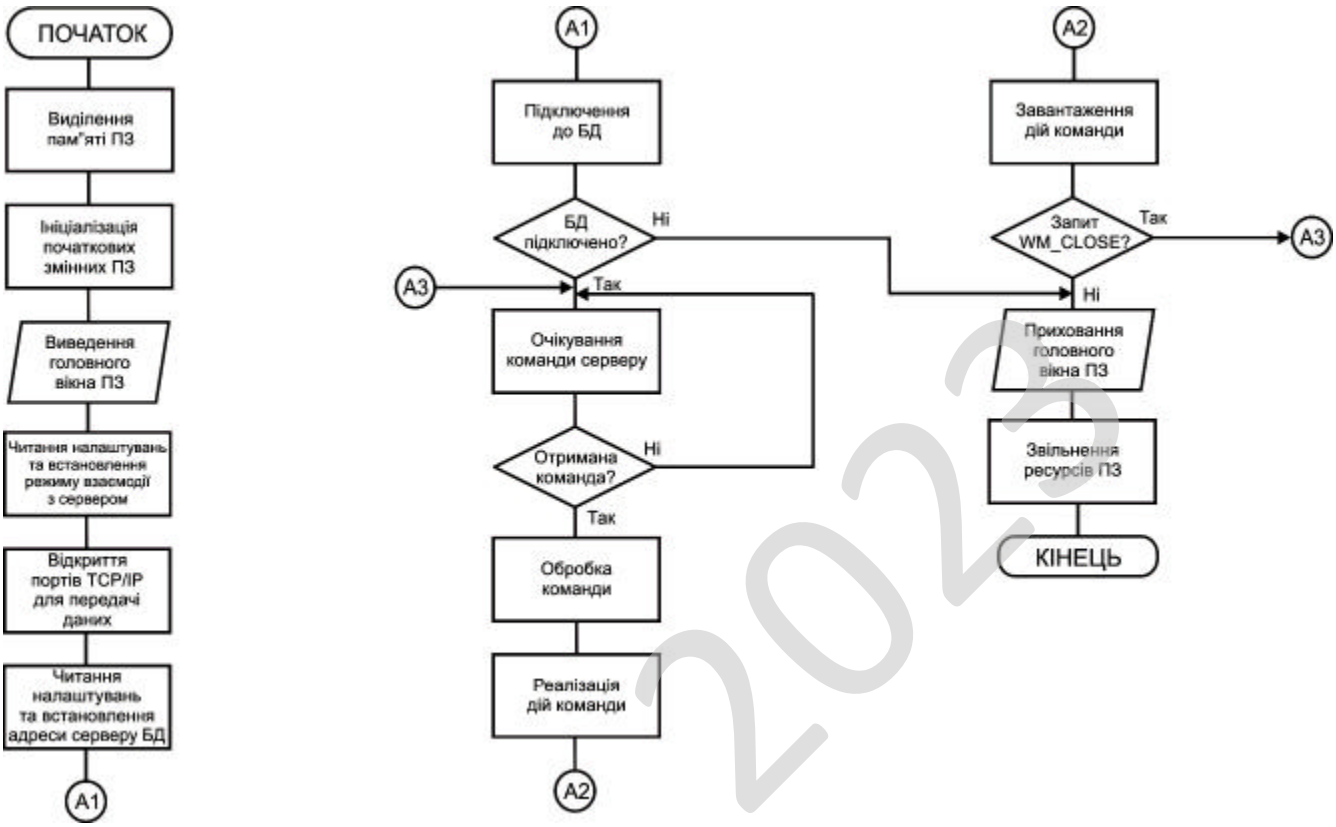

Рисунок 4.3 – Блок-схема клієнтської програми

– Обробка команди.

– Реалізація дій команди.

– Завантаження дій команди.

– Запит WM_CLOSE?

– Звільнення ресурсів ПЗ.

Опис алгоритмів функціонування системи

Розглянемо технологію клієнт-сервер, яку я реалізував у магістерській роботі. Алгоритм роботи сокетного сервера.

Сервер, заснований на сокетном протоколі, дозволяє обслуговувати відразу декількох клієнтів. Причому, обмеження на їхню кількість задає програма (або взагалі прибрати це обмеження, як це зроблене за замовчуванням).

Для кожного підключеного клієнта сервер відкриває окремий сокет, по якому проводиться обмін даними із клієнтом. Також створює для кожного підключення окремий процес (Thread).

Розберемо схему докладніше:

1. Визначення властивостей Port і Servertype – щоб до сервера могли нормально підключатися клієнти, потрібно, щоб порт, використовуваний сервером, точно збігався з портом, використовуваним клієнтом (і навпаки). Властивість Servertype визначає тип підключення;

2. Відкриття сокету – відкриття сокету й зазначеного порту. Тут виконується автоматичний початок очікування приєднання клієнтів (Listen);

3. Підключення клієнта й обмін даними з ним – тут підключається клієнт і проходить обмін даними з ним;

4. Відключення клієнта – тут клієнт відключається й закривається його сокетне з'єднання із сервером;

5. Закриття сервера й сокету – По команді адміністратора сервер завершує свою роботу, закриваючи всі відкриті сокетні канали й припиняючи очікування підключень клієнтів.

Пункти 3-4 повторюються багаторазово, тобто ці пункти виконуються для кожного нового підключення клієнта.

Опис використовуваного компонента TServerSocket

Розглянемо основні властивості, методи й події компонента Tserversocket які використовувалися при написанні програми.

Властивості що використовувались.

Socket – клас Tserverwinsocket, через який я одержував доступ до відкритих сокетних каналів.

Servertype – тип сервера. Може приймати одне із двох значень: stnonblocking – синхронна робота із клієнтськими сокетами.

При такому типі сервера можна працювати із клієнтами через події Onclientread і Onclientwrite. stthreadblocking – асинхронний тип.

Для кожного клієнтського сокетного каналу створюється окремий процес (Thread). Тип: Tservertype;

Threadcachesize – кількість клієнтських процесів (Thread), які будуть керуватися сервером. Тут необхідно підбирати середнє значення залежно від завантаженості сервера.

Керування відбувається для того, щоб не створювати щораз окремий процес і не знищувати закритий сокет, а залишити їх для подальшого використання. Тип: Integer;

Active – показник того, активний у даних момент сервер, чи ні, фактично, значення True указує на те, що сервер працює й готів до приймання клієнтів, а False – сервер виключений.

Щоб запустити сервер, потрібно просто привласнити цій властивості значення True. Тип: Boolean;

Port – номер порту для встановлення з'єднань із клієнтами. Порт у сервера й у клієнтів повинні бути однаковими. Застосовувалися значення від 1025 до 65535, тому що від 1 до 1024 – можуть бути зайняті системою. Тип: Integer;

Service – рядок, що визначає службу (ftp, http, pop, і т.д.), порт якої буде використаний. Це своєрідний довідник відповідності номерів портів різним стандартним протоколам. Тип: string.

Методи що використовувались

Open – Запускає сервер. По суті, ця команда ідентична присвоєнню значення True властивості Active;

Close – Зупиняє сервер. По суті, ця команда ідентична присвоєнню значення False властивості Active.

Події що використовувались

Onclientconnect – виникає, коли клієнт установив сокетне з'єднання й чекає відповіді сервера (Onaccept).

Onclientdisconnect – виникає, коли клієнт від'єднався від сокетного каналу.

Onclienterror – виникає коли поточна операція завершилася невдало, тобто відбулася помилка.

Onclientread – виникає, коли клієнт передав серверу які-небудь дані. Доступ до цих даних можна одержати через переданий параметр Socket: Tcustomwinsocket.

Onclientwrite – виникає, коли сервер може відправляти дані клієнтові по сокету.

Ongetsocket – в оброблювачі цієї події можна відредагувати параметр Clientsocket.

Ongetthread – в оброблювачі цієї події можна визначити унікальний процес (Thread) для кожного окремого клієнтського каналу, привласнивши параметру Socketthread потрібне підзавдання Tserverclientthread.

Onthreadstart, Onthreadend – виникає, коли підзавдання (процес, Thread) запускається або зупиняється, відповідно.

Onaccept – виникає, коли сервер ухвалює клієнта або відмовляє йому в з'єднанні.

Onlisten – виникає, коли сервер переходить у режим очікування приєднання клієнтів.

Tserversocket.Socket (Tserverwinsocket)

Спілкуватися із клієнтом можна через параметр Clientsocket (Tcustomwinsocket).

Activeconnections (Integer) – кількість підключених клієнтів.

Activethreads (Integer) – кількість працюючих процесів.

Connections (array) – масив, що складається з окремих класів Tclientwinsocket для кожного підключеного клієнта.

Наприклад, така команда:

```
Serversocket1.Socket.Connections[0].Sendtext('Hello!');
```

Відсилає першому підключеному клієнтові повідомлення 'Hello!'. Команди для роботи з елементами цього масиву – також (Send/Receive) (Text,Buffer, Stream).

Idlethreads (Integer) – кількість вільних процесів. Такі процеси керуються сервером.

Local address, Localhost, Localport – відповідно – локальний Ip-адреса, хост, порт.

Remot eaddress, Remote host, Remote port – відповідно віддалений Ip-адрес, хост- ім'я, порт.

Методи Lock і Unlock – відповідно, блокування й розблокування сокету.

Розглянемо приклад роботи з Tserversocket який реалізований у програмі, демонструється протоколювання всіх важливих подій і можливість приймати й відсилати текстові повідомлення.

Протоколювання й вивчення роботи сервера, посилка/приймання повідомлень через сокети:

```
{Тут іде заголовок файлу й визначення форми Tform1 і її екземпляра Form1 }
 procedure Tform1.Button1Click(Sender: Tobject);
 begin
     {Визначаємо порт і запускаємо сервер}
       Serversocket1.Port := 1025;
     {Метод Insert вставляє рядок у масив у зазначену позицію}
       Memo2.Lines.Insert(0,'Завантаження серверу');
       Serversocket1.Open;
 end;

 procedure Tform1.Button2Click(Sender: Tobject);
 begin
       {Зупиняємо сервер}
       Serversocket1.Active := False;
       Memo2.Lines.Insert(0,'Зупинка серверу');
 end;

 procedure Tform1.Serversocket1Listen(Sender: Tobject;
```

```
      Socket: Tcustomwinsocket);
begin
{Тут сервер "прослуховує" сокет на наявність клієнтів}
Memo2.Lines.Insert(0,'Listening on port '+Inttostr( Serversocket1. Port));
end;

procedure Tform1.Serversocket1Accept(Sender: Tobject;
      Socket: Tcustomwinsocket);
begin
      {Тут сервер приймає клієнта}
      Memo2.Lines.Insert(0,'Client connection accepted');
end;

procedure Tform1.Serversocket1Clientconnect(Sender: Tobject;
      Socket: Tcustomwinsocket);
begin
      {Тут клієнт приєднується}
      Memo2.Lines.Insert(0,'Client connected');
end;

procedure Tform1.Serversocket1Clientdisconnect( Sender:
Tobject;
      Socket: Tcustomwinsocket);
    begin
      {Тут клієнт від'єднується}
      Memo2.Lines.Insert(0,'Client disconnected');
end;

procedure Tform1.Serversocket1Clienterror(Sender: Tobject;
      Socket: Tcustomwinsocket; Errorevent: Terrorevent;
      var Errorcode: Integer);
begin
{Відбулася помилка – виводимо її код}
Memo2.Lines.Insert(0,'Client error. Code = '+Inttostr(
Errorcode));
end;

procedure Tform1.Serversocket1Clientread(Sender: Tobject;
      Socket: Tcustomwinsocket);
begin
  {Від клієнта отримане повідомлення – виводимо його в Memo1}
      Memo2.Lines.Insert(0,'Message received from client');
```

```
      Memo1.Lines.Insert(0,'> '+Socket.Receivetext);
end;

procedure Tform1.Serversocket1Clientwrite(Sender: Tobject;
      Socket: Tcustomwinsocket);
begin
      {Тепер можна відіслати дані в сокет}
      Memo2.Lines.Insert(0,'Now can write to socket');
end;

procedure Tform1.Serversocket1Getsocket(Sender: Tobject;
Socket: Integer;
      var Clientsocket: Tserverclientwinsocket);
begin
      Memo2.Lines.Insert(0,'Get socket');
end;

procedure Tform1.Serversocket1Getthread(Sender: Tobject;
      Clientsocket: Tserverclientwinsocket;
      var Socketthread: Tserverclientthread);
begin
      Memo2.Lines.Insert(0,'Get Thread');
end;

procedure Tform1.Serversocket1Threadend(Sender: Tobject;
      Thread: Tserverclientthread);
begin
      Memo2.Lines.Insert(0,'Потік зупинено');
end;

procedure Tform1.Serversocket1Threadstart(Sender: Tobject;
      Thread: Tserverclientthread);
begin
      Memo2.Lines.Insert(0,'Потік завантажено');
end;

procedure Tform1.Button3Click(Sender: Tobject);
     var i: Integer;
begin
      {Посилаємо всім клієнтам повідомлення з Edit1}
for i := 0 to Serversocket1.Socket.Activeconnections-1 do
begin
```

```
    Serversocket1.Socket.Connections[i].Sendtext(Edit1.Text);
end;
    Memo1.Lines.Insert(0,'< '+Edit1.Text);
end;
```

При роботі програми необхідно зберігати унікальні дані для кожного клієнта. Потрібно зберігати інформацію для кожного клієнта (ім'я, і ін.), причому із прив'язкою цієї інформації до сокету даного клієнта.

У деяких випадках робити все це вручну (прив'язка до handle сокету, масиви клієнтів, і т.д.) не дуже зручно. Тому для кожного сокету існує спеціальна властивість – Data.

Насправді, Data – це всього-на-всього покажчик. Тому, записуючи дані клієнта в цю властивість, я додержувався правил роботи з покажчиками (виділення пам'яті, визначення типу, і т.д.).

Опис реалізації посилки файлів через сокет

Відкривається файл як файловий потік (Tfilestream) і відбувається відправлення його через сокет (Sendstream).

```
{Посилка файлу через сокет}
  procedure Sendfilebysocket(filename: string);
   var srcfile: Tfilestream;
  begin
    {Відкриваємо файл filename}
    srcfile := Tfilestream.Create(filename,fmopenread);
    {Посилаємо його першому підключеному клієнтові}
    Serversocket1.Socket.Connections[0].Sendstream(srcfile);
    {Закриваємо файл}
    srcfile.Free;
end;
```

Метод Sendstream використовується не тільки сервером, але й клієнтом (Clientsocket1.Socket.Sendstream(srcfile)), кілька блоків при передачі можуть поєднуватися в один.

Сокет це звичайний потік, він передає дані повільніше, саме тому дві команди:

```
Serversocket1.Socket.Connections[0].Sendtext('Hello, ');
Serversocket1.Socket.Connections[0].Sendtext('world!');
```

Цілком ідентичні однієй команді:

```
Serversocket1.Socket.Connections[0].Sendtext('Hello, world!');
```

Тому в програмі завжди використовується Sendtext одним рядком коду. Для збору блоків файлу при прийманні даних використовується файловий потік – Tfilestream (або потік у пам'яті – Tmemorystream), через подію Onread (Onclientread), використовуючи універсальний метод Receivebuf.

Визначити розмір отриманого блоку можна методом Receivelength. Також можна скористатися сокетним потоком.

```
{Приймання файлу через сокет}
procedure Tform1.Clientsocket1Read(Sender: Tobject;
    Socket: Tcustomwinsocket);
   var l: Integer;
       buf: Pchar;
       src: Tfilestream;
begin
    {Записуємо в l розмір отриманого блоку}
    l := Socket.Receivelength;
    {Замовляємо пам'ять для буфера}
    Getmem(buf,l+1);
    {Записуємо в буфер отриманий блок}
    Socket.Receivebuf(buf,l);
    {Відкриваємо тимчасовий файл для запису}
    src := Tfilestream.Create('myfile.tmp',fmopenreadwrite);
    {Ставимо позицію в кінець файлу}
    src.Seek(0,sofromend);
    {Записуємо буфер у файл}
    src.Writebuffer(buf,l);
    {Закриваємо файл}
    src.Free;
    {Звільняємо пам'ять}
    Freemem(buf);
end;
```

4.2 Захист розробленого програмного забезпечення

Для захисту розробленого програмного забезпечення запропоновано використовувати алгоритм Madryga. Алгоритм Madryga складається із двох

вкладених циклів. Зовнішній цикл повторюється вісім разів (для гарантії надійності число циклів можна збільшити) і полягає в застосуванні внутрішнього циклу до відкритого тексту. Внутрішній цикл перетворює відкритий текст у шифртекст і виконується однократно над кожним 8-бітовим блоком (байтом) відкритого тексту. Таким чином, весь відкритий текст послідовно вісім разів обробляється алгоритмом.

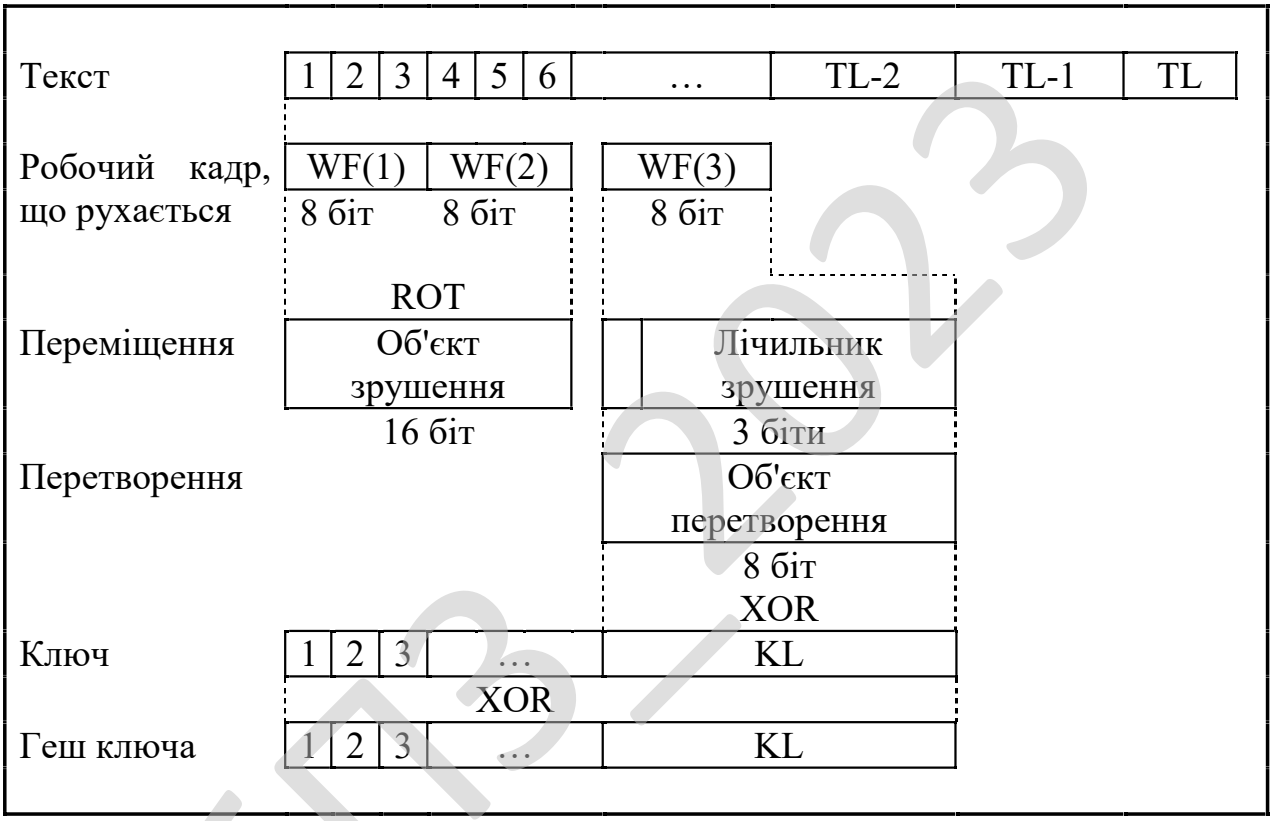

Рисунок 4.4 – Одна ітерація алгоритму Madryga

Ітерація внутрішнього циклу оперує з 3-байтовим вікном даних, називаним робочим кадром (рисунок 4.4). Це вікно зрушується на 1 байт за ітерацію. (При роботі з останніми 2 байтами дані покладаються циклічно замкнутими). Перші два байти робочого кадру циклічно зрушуються на змінне число позицій, а для останнього байта виконується операція XOR з декількома бітами ключа. У міру переміщення робочого кадру всі байти послідовно циклічно зрушуються й піддаються операції XOR із частинами ключа. Послідовні циклічні

зрушення перемішують результати попередніх операцій XOR і циклічного зрушення, причому на циклічне зрушення впливають результати XOR. Завдяки цьому процес у цілому оборотний.

Оскільки кожний байт даних впливає на два байти ліворуч і на один байт праворуч від себе, після восьми проходів кожний байт шифртексту залежить від 16 байтів ліворуч і 8 байтів праворуч.

При шифруванні кожна ітерація внутрішнього циклу встановлює робочий кадр на передостанній байт відкритого тексту й циклічно переміщає його до третього з кінця байту відкритого тексту. Спочатку весь ключ піддається операції XOR з випадковою константою й потім циклічно зрушується вправо на 3 біти (ключ і дані рухаються в різних напрямках, щоб мінімізувати надлишкові операції з бітами ключа). Молодші три біти молодшого байта робочого кадру зберігаються, вони визначають циклічне зрушення інших двох байтів. Далі конкатенація двох старших байтом циклічно зрушується вліво на змінне число біт (від 0 до 7). Потім над молодшим байтом робочого кадру виконується операція XOR з молодшим байтом ключа. Нарешті робочий кадр зміщається вправо на один байт і весь процес повторюється.

Випадкова константа призначена для перетворення ключа в псевдовипадкову послідовність. Довжина константи повинна бути рівній довжині ключа. При обміні даними абоненти повинні користуватися однієї й тією же константою. Для 64-бітового ключа Madryga рекомендує константу 0x0fle2d3c4b5a6978.

При розшифруванні процес повторюється у зворотному порядку. У кожній ітерації внутрішнього циклу робочий кадр установлюється на байт, третій ліворуч від останнього байта шифртексту, і циклічно зрушується у зворотному напрямку до байта, розташованого на 2 байти уліво відносно останнього байта шифртексту. 2 байти шифртексту в процесі циклічно зрушуються вправо, а ключ – уліво. Після циклічних зрушень виконується операція XOR.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Програма має простий та інтуїтивно зрозумілий інтерфейс, який зображений на рисунку 5.1. На рисунку зображено головне вікно серверної програми. З нього видно, що інтерфейс користувача програми складається з таких логічних блоків: 1. Терміналу роботи. де відображуються поточні дії та знаходяться функції: Редагувати; Сканувати; Запит. 2. Меню: Налаштування служб; довідка. 3. Функції: Вікно служби планування; Вікно менеджера резервування; Вікно менеджера доставки даних; Вікно служби приймання інформації про ресурси; Перегляд таблиць БД; Налаштування.

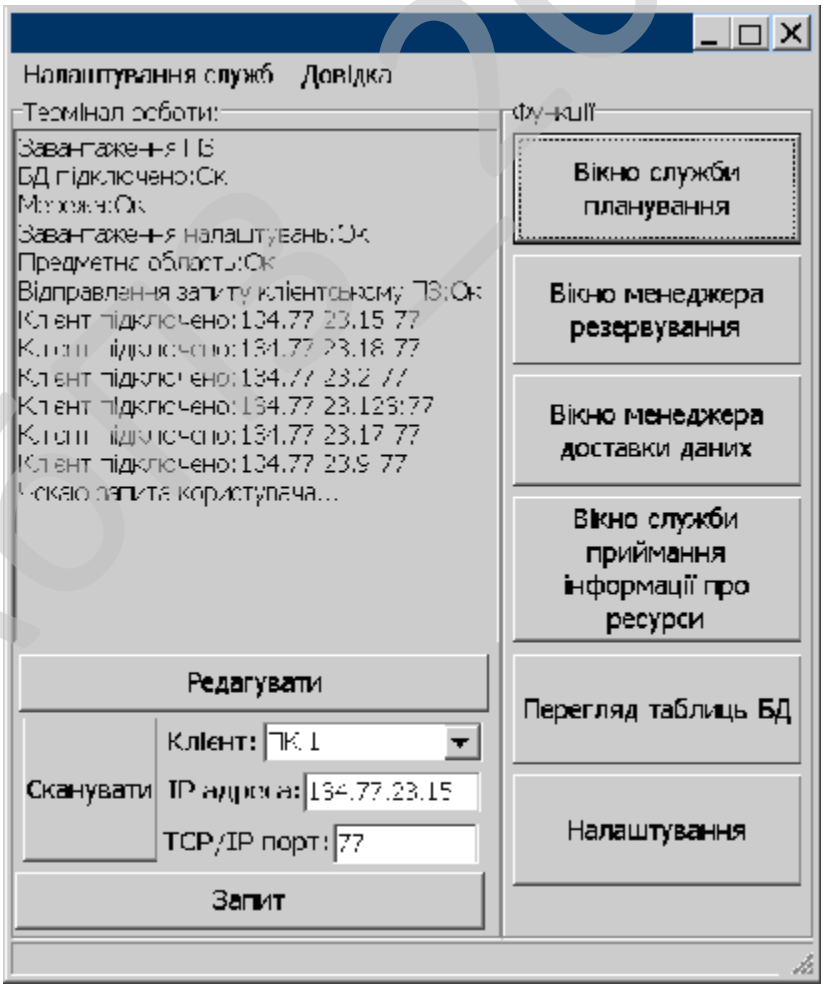

Рисунок 5.1 – Головне вікно програми

На рисунку 5.2 зображено форму авторського права, де відображені дані розробника.

Було обрано Shareware умову розповсюдження. Під умовно-безплатним програмним забезпеченням можна розуміти спосіб або метод розповсюдження комерційного ПЗ на ринку (тобто на шляху до кінцевого користувача), при якому випробувачеві пропонується обмежена за можливостями (неповнофункціональна або демонстраційна версія), терміном дії (тріал версія) або версія з вбудованим набридливим нагадуванням про необхідність оплати використання програми.

В угоді про використання (ліцензії для кінцевого користувача, EULA) також може бути обумовлена заборона на комерційне або професійне (не тестове) її використання.

Основний принцип умовно-безплатного ПЗ – «спробуй, перш ніж купити» (try before you buy). ПЗ що поширюється як умовно-безплатний, надається користувачам безоплатно.

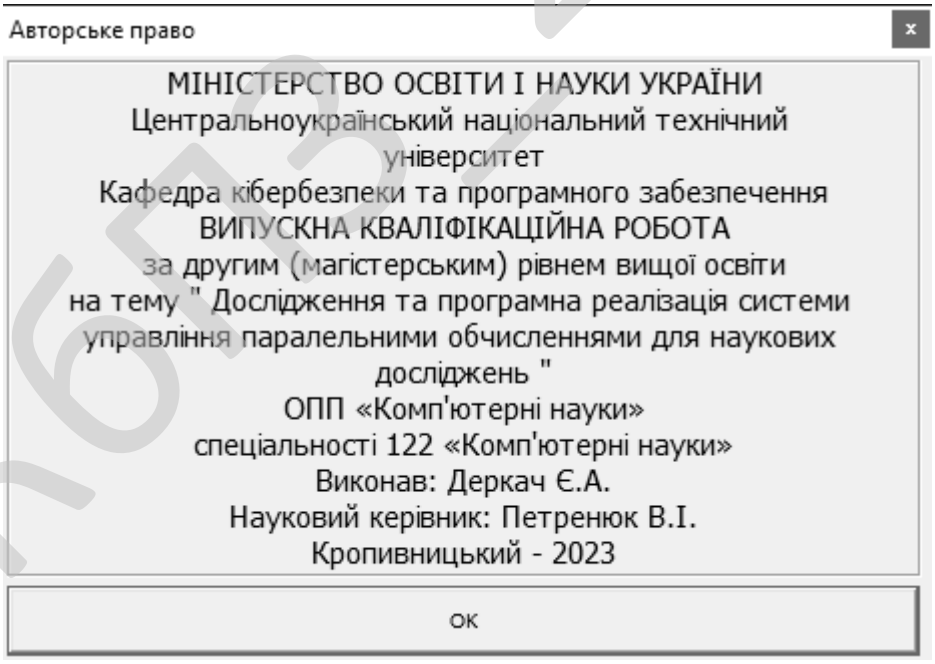

Рисунок 5.2 – Довідка

6 НАУКОВА НОВИЗНА

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи управління паралельними обчисленнями для наукових досліджень.

Метою розробки є дослідження та програмна реалізація системи управління паралельними обчисленнями для наукових досліджень.

Об'єктом дослідження є процес управління паралельними обчисленнями для наукових досліджень.

Предметом дослідження є методи управління паралельними обчисленнями для наукових досліджень.

Методи дослідження базуються на методах паралелізму, методах математичної статистики, методах розробки програмного забезпечення.

Наукова новизна отриманих результатів. У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

– Удосконалено метод управління паралельними обчисленнями для наукових досліджень.

– Розроблено вітчизняний продукт управління паралельними обчисленнями для наукових досліджень, який має більш широкі можливості, на відміну від існуючих аналогів.

7 ДАНІ ПРО ЕКОНОМІЧНУ ЕФЕКТИВНІСТЬ РОЗРОБЛЕНОЇ ПРОГРАМИ

7.1 Техніко-економічне обґрунтування теми випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти

Після ознайомлення з підприємством та засобами розробки програмної продукції був розроблений план розробки програми. Був підрахований необхідний час для розробки та впровадження програми. Цей час склав 60 днів (три місяці). В магістерській роботі було проведено дослідження та виконана програмна реалізація системи управління паралельними обчисленнями для наукових досліджень.

Розроблене програмне забезпечення має достатню надійність і задовольняє усім поставленим умовам, а саме:

а) невеликий розмір;

б) невеликі системні потреби;

в) незалежність від встановлених на комп’ютері баз даних;

г) зручність у користуванні та надійність.

Таблиця 7.1 – Початкові дані

Показники	Позначення	Характеристика або величина
1	2	3
1. Кількість розроблених програм період, шт.	N	1
2. Кількість екземплярів програм, шт.	Ne	24
3. Запланований термін розробки, днів	Fpq	60 (3 місяці)
4. Група задачі підсистеми управління (1-6)	–	1
5. Ступінь новизни задачі (А, Б, В, Г)	–	Б
6. Складність алгоритму (1, 2, 3)	–	2

Продовження таблиці 7.1

1	2	3
7. Кількість макетів вхідної інформації	–	3
8. Кількість форм вихідної інформації	–	4
9. Мова програмування (1-6)	–	2
10. Попередній досвід (1-6)	–	3
11. Гнучкість проекту ПП (1-6)	–	3
12. Детальність проекту ПП (1-6)	–	2
13. Рівень спрацьованості колективу (1-6)	–	2
14. Ступінь вимірності процесів (1-6)	–	3
15. Необхідна надійність програмного забезпечення (1-6)	–	2
16. Розмір бази даних (порівняно з розміром програми) (1-6)	–	2
17. Складність кінцевого програмного продукту (1-6)	–	2
18. Необхідний рівень забезпечення повторного використання (1-6)	–	2
19. Документованість відповідно до планованого життєвого циклу (1-6)	–	2
20. Вимоги до швидкодії ПП (1-6)	–	2
21. Обмеження на розміри основного сховища даних (1-6)	–	2
22. Різноманітність використовуваних обчислювальних платформ (1-6)	–	2
23. Професійний рівень аналітиків (1-6)	–	2
24. Професійний рівень програмістів (1-6)	–	2
25. Постійність складу команди розробників (1-6)	–	2
26. Досвід розробки додатків (1-6)	–	2
27. Досвід роботи з обчислювальною платформою (1-6)	–	2

Продовження таблиці 7.1

1	2	3
28. Досвід роботи з мовою і інструментами середовища розробки (1-6)	–	2
29. Досвід роботи з програмними інструментами розробки (1-6)	–	3
30. Розробка ПЗ для декількох серверів одночасно (1-6)	–	2
31. Вимоги до дотримання встановленого графіка робіт (1-6)	–	2
32. Вартість ПЗ у розробника (НМА), грн.	–	24000
33. Норматив додаткової зарплати, % :	$Н_д$	10
34. Норматив відрахувань у соціальні фонди, %	Нс	22
35. Норматив загальногосподарських витрат, %	Нг	15
36. Норматив витрат на освоєння нових мов програмування, %	Нп	15
37. Рівень рентабельності програмної продукції, %	Ре	50
38. Ставка податку на додану вартість, %	Ндв	20

7.2 Розрахунок трудомісткості розробки програмної продукції

Значення трудомісткості розробки програмного забезпечення для стадій ТЗ, ЕК, ТП та ВП визначаємо по типовим нормам часу приведеним в додатках МВ. Стадія РП є найбільш тривалою і трудомісткою, що робить значний вплив на інші стадії проекту.

Визначимо трудомісткість розробки ПЗ для стадії РП.

Обчислюємо номінальні трудовитрати, люд-міс.:

$$T_{ном} = A \, \text{Size}^B, \qquad (7.1)$$

де: A – коефіцієнт Боема, A = 2,45;

Size – загальний об'єм відлагодженого програмного коду, тис. рядків;

B – показник ступеня, що визначається співвідношенням:

$$B = 1{,}01 + 0{,}001\sum W_i, \qquad (7.2)$$

де: Wi – сумарне значення п'яти показників (МВ, додаток 2), що відображають особливості розробки проекту програмного продукту (ПП) і колективу розробників.

$$B = 1{,}01+0{,}001(2{,}43+3{,}64+3{,}38+3{,}95+2{,}73) = 1{,}027.$$

$$T_{ном} = 2{,}45 \cdot 2{,}7^{1{,}026} = 6{,}78 \text{ люд-міс.}$$

Визначаємо уточнені (з урахуванням приведених в МВ додатку 3 сімнадцяти додаткових коефіцієнтів) трудовитрати, люд-міс.:

$$T_{уточн} = T_{ном}\, ПВ_j, \qquad (7.3)$$

де: $ПВ_j$ – добуток сімнадцяти додаткових коефіцієнтів, приведених в МВ додатку 3.

$$T_{уточн} = 6{,}78 \cdot (0{,}88 \cdot 0{,}93 \cdot 0{,}88 \cdot 0{,}91 \cdot 0{,}95 \cdot 1 \cdot 1 \cdot 0{,}87 \cdot 1{,}22 \cdot 1{,}16 \cdot 1{,}1 \cdot 1{,}1 \cdot$$

$$\cdot 1{,}12 \cdot 1{,}1 \cdot 1 \cdot 1{,}1 \cdot 1{,}1) = 9{,}37 \text{ люд-міс.}$$

Ці коефіцієнти дозволяють диференційовано оцінювати результати роботи програмістів, беручи до уваги швидкодію програми, використання різноманітних обчислювальних платформ і інструментів розробки, взаємодію декількох серверів, вимоги до об'ємів баз даних і ін.

Визначаємо підсумкові трудовитрати по стадії робочий проект, люд-дні:

$$T_{РП} = 0{,}3 C T_{уточн}^{0{,}33+0{,}2(B-1{,}01)} S, \qquad (7.4)$$

де: C – визначений емпірично коефіцієнт, запропонований авторами методики, (МВ, додаток 4);

S – коефіцієнт стиснення (або подовження) графіка робіт %, що дозволяє коректувати терміни розробки ПЗ згідно встановленим вимогам. Вибираємо в межах (25…350)%.

$$T_{РП} = 0{,}3 \cdot 2{,}66 \cdot 9{,}37^{0{,}33+0{,}2(1{,}026-1{,}01)} \cdot 58 = 98 \text{ люд/день.}$$

Для зручності визначення загальної трудомісткості на розробку програмного забезпечення результати розрахунків по стадіям зводимо до таблиці 7.2.

Таблиця 7.2 – Визначення трудомісткості розробки програмного забезпечення

Стадії розробки	Трудомісткість за типовими нормами та розрахунками	
	Величина, люд/дні	Підстава
Технічне завдання	9	Д5
Ескізний проект	10	Д6
Технічний проект	9	Д7
Робочий проект	98	Ф 7.1-7.4
Впровадження	13	Д13
Всього	139	–

7.3 Визначення чисельності виконавців і планового фонду зарплати

Чисельність ставок інженерів-програмістів для розробки програмного забезпечення визначається за формулою:

$$Ч = \frac{T_{пз} N}{F_{рq} - H_{ев}}, \quad (7.5)$$

де: $F_{рq}$ – плановий фонд робочого часу одного спеціаліста, днів;

$T_{пз}$ – трудомісткість розробки програмного забезпечення люд-дні.

$$Ч = \frac{139 \cdot 1}{60 - 5} = 2,5 \text{ ставки.}$$

Чисельність інженерів-електронщиків для проведення технічного обслуговування та ремонту комп'ютерних мереж визначається в залежності від наявності технічних засобів і норм витрат часу на виконання профілактичних робіт на протязі року.

Визначаємо затрати часу на виконання профілактичних робіт по обслуговуванню обладнання за період розробки. Результати розрахунку зводимо до таблиці 7.3.

Таблиця 7.3 – Затрати часу на виконання профілактичних робіт по обслуговуванню обладнання за розрахунковий період

Найменування обладнання	Профілактичне обслуговування			
	Кількість хв. на один. обл.	Кількість облад-нання	Затрати часу в хв.	Затрати часу в год.
Системний блок ПК	90	7	630	10,5
Монітор	60	7	420	7
Клавіатура	30	7	210	3,5
Маніпулятор «мишка»	30	7	210	3,5
Принтер матричний	60	0	0	0,0
Принтер лазерний	120	1	120	2
Принтер струминний	60	1	60	1
Сканер	20	1	20	0,33
Концентратор-маршрутизатор	30	2	60	1
Кабельні господарства ЛОМ на 1 м.п.	2,5	200	500	8,33
Копіювальний апарат	140	1	140	2,33
Усього за рік:			$З_ч$	39,49

Час на профілактику обладнання в загальному балансі робочого часу інженерів-електронщиків неповинен складати більше 10%.

Виходячи з цього фонд робочого часу інженерів-електронщиків складає:

$$Ф_{др}^{с} = \frac{З_ч \cdot n_{тіс}}{1,2}, \quad (7.6)$$

$$Ф_{др}^{с} = \frac{39 \cdot 3}{1,2} = 97,5 \text{ год.}$$

Визначаємо необхідну кількість ставок штатного персоналу сектора ТО:

$$Ч_{ел} = \frac{Ф_{др}^{с}}{F_{др} \cdot T_{зм}}, \quad (7.7)$$

$$Ч_{ел} = 97,5/(60 \cdot 8) = 0,2 \text{ ставки.}$$

Для забезпечення нормального технічного обслуговування засобів ТО та мереж, необхідно прийняти найбільше ціле значення розрахункової чисельності інженерів-електронщиків.

Чисельність інженерів-системотехніків, адміністраторів мережі, дизайнерів WEB вузлів, системних програмістів (аналітиків), бухгалтерів-економістів визначається за потребою в залежності від функціональних обов'язків. Після визначення чисельності персоналу складається штатний розклад.

Таблиця 7.4 – Розрахунок чисельності штатного персоналу сектору системного та адміністративного обслуговування засобів ОТ та комп'ютерних мереж

Посада	Вид роботи	Час	К-ть штатних одиниць
Адміністратор загальної мережі, аналітик	Адміністрування локальної мережі, поштового та серверу DNS (ОС FreeBSD), маршрутизатора Cisco, доменного контролеру Windows Server, серверу доступу ADSL (ОС Linux), налаштування ADSL, VPN PPPoE, Frame Relay, Wi-Fi	2	0,5
	Налаштування і конфігурування базової станції безпровідного зв'язку (CMTS)	0,5	
	Розробка та впровадження проектів з організації зв'язку між віддаленими об'єктами, ЛОМ	0,5	
	Забезпечення цілодобової роботи зв'язку клієнтів до мережі Інтернет	1	
Всього		4	

Продовження таблиці 7.4

Посада	Вид роботи	Час	К-ть штатних одиниць
Продакт-менеджер	Презентації нової продукції, пошук каналів збуту	1	0,25
	Підтримка постійних клієнтів	0,5	
	Оформлення договорів, ведення тендерів	0,25	
	Контроль взаєморозрахунків з постачальниками	0,25	
Всього		2	
Дизайнер WEB	Розробка концепції оформлення та інтерфейсу сайту, оптимізація дизайну існуючих, проектує їх структуру та навігацію	1	0,25
	Створення графічних і стилістичних елементів сайту	0,5	
	Оформлення банерів і промо-сторінок	0,25	
	Розміщення графіки і контенту на Інтернет сторінках	0,25	
Всього		2	
Інженер верстальник	Розробка та верстка макетів рекламної продукції та технічної документації	1	0,25
	Верстка друкованих видань	0,5	
	Додрукова підготовка макетів	0,25	
	Розміщення графіки і контенту на Інтернет сторінках	0,25	
Всього		2	

Складемо штатний розклад виконавців.

Таблиця 7.5 – Штатний розклад виконавців

Посада	Кількість ставок	Середньомісячний оклад, грн.	Всього за період розробки, грн.
Керівник (ІТ-менеджер)	1	19038	57114
Продакт-менеджер	0,25	18000	13500
Інженер-програміст	2,5	19000	142500
Інженер-електронщик	0,2	16500	9900
Інженер-системотехнік	0,25	16500	12375
Адміністратор мережі	0,5	16500	24750
Системний програміст	0,25	16500	12375
Дизайнер WEB	0,25	16500	12375
Інженер-верстальник	0,25	16500	12375
Бухгалтер-економіст	0,5	16500	24750
Всього за період розробки	$R_{сп}$ = 5,95	-	$Ф_{роб}$ = 322014

Розрахуємо середньоденну зарплату одного виконавця:

$$З_{сд} = \frac{Ф_{роб}}{R_{сп} F_{рд}}, \quad (7.8)$$

де: $Ф_{роб}$ – загальна сума зарплати за плановий період, грн.

$$З_{сд} = \frac{322014}{5,95 \cdot 60} = 902 \; грн.$$

7.4 Розрахунок капітальних вкладень та амортизаційних відрахувань у розробника

Балансова вартість будівель визначається з урахуванням кількості робочих місць виконавців, питомої площі на одне робоче місце, та вартості одного квадратного метра виробничої площі:

$$Б_{уд} = R_{сп}^{1} S_{у} Ц_{пл}, \quad (7.9)$$

де: $R_{сп}^{1}$ – кількість робочих місць виконавців, шт. Приймаємо 8 робочих місць;

$S_{у}$ – питома площа на одне робоче місце, $м^2$;

$Ц_{пл}$ – вартість одного квадратного метра площі, грн.

Згідно даних інтернет ресурсу DOM.RIA (https://dom.ria.com) ціна одного квадратного метра площі, вік якої не перевищує 30 років, по місту складає 500…1600 $у.о./м^2$. Враховуючи, що курс складає 1 у.о. = 38 грн. приймаємо для розрахунку вартість одного метра квадратного рівною 20000 грн./м^2. На кожне робоче місце у середньому потрібно 8 $м^2$. З урахуванням цього:

$$Б_{уд} = 8 \cdot 8 \cdot 20000 = 1280000 \text{ грн.}$$

Вартість передавальних пристроїв складає 10% від вартості будівель, і у даному випадку вона складе: 128000 грн.

Балансова вартість інвентарю розраховується за нормою 3500 грн. на одне робоче місце. Тобто:

$$І_{нв} = R_{сп}^{1} \cdot Ц_{м}, \quad (7.10)$$

де: $Ц_{м}$ – ціна меблів для одного робочого місця, грн.

$$І_{нв} = 8 \cdot 3500 = 28000 \text{ грн.}$$

Балансова вартість обчислювальної техніки визначається по оптовим цінам постачальника з врахуванням витрат на транспортування.

Специфікація на обчислювальну техніку наведена в таблиці 7.7.

Дані по оптовій ціні на обладнання та комплектуючі вибирались за прайсом Інтернет-магазину Компбест за 17.10.23 – джерело https://compbest.com.ua.

Таблиця 7.6 – Специфікація

Найменування комплектуючої або обладнання	Тип	Оптова ціна
Персональний комп’ютер		11771

Продовження таблиці 7.6

Найменування комплектуючої або обладнання	Тип	Оптова ціна
Системний блок		7771
Процесор	Intel Core i5-6400 (4 ядра по 2.7 – 3.3 GHz), MB Smart Cache	-
Системна плата	HP 2B47, 3x USB 2.0, 3x USB 3.0, 1x DVI, 1 HDMI, 1x DisplayPort, 5x Audio, 1x LAN (RJ-45	-
Жорсткий диск	240 GB SSD	-
Оперативна пам’ять	8 GB DDR4	-
Відеоадаптер	nVidia GeForce GTX 950, 2 GB GDDR5, 128 bit	
DVD-привод	DVD±RW ASUS DRW-24B5ST Black Bulk	-
Корпус	HP Pavilion 560 Tower, 350W	-
Кардрідер внутрішній	Transcend TS-RDF8K USB 3.0	-
інше	Клавіатура, мишка	-
Монітор	Монітор BenQ GL2450HM Black	2600
Принтер лазерний	Canon i-SENSYS LBP6030W	2700
Принтер струминний	Epson Stylus Photo P50 (C11CA45341) + USI cable	5500
Сканер	Epson Perfection V37	2800
Копіювальний апарат	Canon i-SENSYS MF217W with Wi-Fi	5965
Пристрій безперебійног живлення	Powercom BNT-600AP USB	1400

Витрати на транспорт, монтаж та випробування можуть бути прийняті в межах до 10% від оптової ціни.

Для визначення необхідної кількості капітальних вкладень складемо таблицю 7.8.

Таблиця 7.7 – Балансова вартість обчислювальної техніки

Найменування обчислювальної техніки	Кількість, шт.	Ціна за одиницю, грн.	Витрати на транспортування, монтаж та випробовування.	Загальна вартість, грн.
Персональні комп'ютери	8	11771	9416,8	103584,8
Принтер лаз.	2	2700	540	5940
Принтер струм.	1	5500	550	6050
Сканери	1	2800	280	3080
Копіюв. апарат	1	5965	596,5	6561,5
Всього	–	–	–	125216,3

Таблиця 7.8 – Вартість основних фондів та амортизаційні відрахування розробника

Групи та види основних фондів	Балансова вартість, грн.	Амортизація	
		Норма, %	Відрахування, грн.
1	2	3	4
Група 3			
1. Будівлі	1280000	-	-
2. Передавальні пристрої	128000	-	-
Всього по групі	1408000	5	70400

Продовження таблиці 7.8

1	2	3	4
Група 4			
3. Обчислювальна техніка	125216	-	-
Всього по групі	125216	50	62608
Група 5,6			
4. Вимірювальні пристрої	5190	20	-
5. Транспортні засоби	143000	25	
6. Господарський інвентар	28000	20	-
Всього по групам 5, 6	176190	-	35238
7. Нематеріальні активи	24000	10	2400
Разом	$К_р$= 1733406		$А_р$= 170646

Примітка: вартість автомобіля Volkswagen Fox 2005 взята за даним сайту «Авто-РІА», джерело https://auto.ria.com/uk/auto_volkswagen_fox_33599812.html, складає 143000 грн.

7.5 Визначення собівартості розробки та ціни програмної продукції

Визначимо основну зарплату виконавців:

$$З_о = \frac{З_{сд} \cdot Т_{пз}}{N_е}, \quad (7.11)$$

де: $N_е$ – кількість екземплярів програм, шт.

$$З_о = 902 \cdot 139/24 = 5225 \text{ грн.}$$

Визначимо додаткову зарплату (оплата відпусток, виконання державних та суспільних обов'язків) на рівні 10%:

$$З_д = З_о \cdot Н_q \cdot 0{,}01, \quad (7.12)$$

де: $Н_q$ – норматив додаткової зарплати, %.

$$З_д = 5225 \cdot 10 \cdot 0{,}01 = 523 \text{ грн.}$$

Відрахування на соціальні потреби за нормативом $Н_с = 22\%$ від суми основної та додаткової зарплати:

$$С_{оц} = 0{,}01 \cdot Н_с (З_о + З_д), \quad (7.13)$$

де: $Н_с$ – відрахування на соціальні потреби, %.

$$С_{оц} = 0{,}01 \cdot 22(5225+523) = 1265 \text{ грн.}$$

Визначимо загальногосподарські витрати (електроенергію, ремонт і утримання приміщень і т.д) за нормативом $Н_г = 15\%$ від основної зарплати:

$$Г_{осп} = З_о \cdot Н_г \cdot 0{,}01, \quad (7.14)$$

де: $Н_г$ – загальногосподарські витрати, %.

$$Г_{осп} = 5225 \cdot 15 \cdot 0{,}01 = 784 \text{ грн.}$$

Визначимо витрати на матеріали для розробки програмної продукції за нормами споживання та діючими цінами за одиницю виміру:

$$З_М = (З_{М1} + З_{М2} + З_{М3})/N_е, \quad (7.15)$$

де: $З_{М1}$ – вартість паперу, грн.;

$З_{М2}$ – вартість запам’ятовуючих пристроїв, грн.;

$З_{М3}$ – вартість фарби, картриджів, тонеру, грн.;

$N_е$ – кількість екземплярів програм, шт.

Згідно прийнятих норм на підприємстві $n_{вит}$ приймаємо 1,5 пачки паперу на період розробки. Тоді, враховуючи, що вартість пачки паперу складає $Ц_п$=210 грн., визначаємо вартість паперу за період розробки:

$$З_{М1} = Ц_п \cdot N_м. \quad (7.16)$$

$$З_{М1} = 210 \cdot 1{,}5 = 315 \text{ грн.}$$

Згідно прийнятих норм по комплектації до вартості запам’ятовуючих пристроїв входить вартість CD/DVD дисків. Їх кількість дорівнює кількості коробочних версій запропонованого продукту (приймаємо 10):

$$З_{М2} = \sum Ц_д \,, \tag{7.17}$$

де: $Ц_д$ – вартість дисків CD/DVD: CDR box – 23,6 грн./шт., DVD-R box – 30 грн./шт.

$$З_{М2} = 30 \cdot 10 = 300 \text{ грн.}$$

Згідно норм одноразовій заправці підлягають усі друкуючі пристрої і становить:

$$З_{М3} = \sum Ц_з., \tag{7.18}$$

де: $Ц_з$ – вартість розхідних матеріалів друкуючих пристроїв: відновлення та заправка картриджу для Canon i-SENSYS LBP6030W – 574 грн.; картридж для Epson Stylus Photo P50 – 558 грн.; відновлення картриджу для MF217W – 570 грн.

$$З_{М3} = 574+558+570 = 1702 \text{ грн.}$$

$$З_М = (315 + 300 + 1702)/24 = 97 \text{ грн.}$$

Визначимо витрати на освоєння нових мов програмування або операційних систем за нормативом ($Н_п$ = 15%) від основної зарплати виконавців:

$$О_п = З_о \cdot Н_п \cdot 0{,}01, \tag{7.19}$$

де: $Н_п$ – норматив витрат на освоєння нових мов програмування, %.

$$О_п = 5225 \cdot 15 \cdot 0{,}01 = 784 \text{ грн.}$$

Визначимо витрати на амортизацію основних фондів з урахуванням загальної річної суми амортизаційних відрахувань та кількості екземплярів програм ($N_е$ = 24 прим.):

$$А_м = \frac{А_р \cdot N_{міс}}{N_е \cdot 12}, \tag{7.20}$$

де: $А_р$ – загальна річна сума амортизаційних відрахувань, грн.

$$А_м = 170646 \cdot 3/(24 \cdot 12) = 1777 \text{ грн.}$$

Повна собівартість ПЗ визначається як сума витрат за попередніми статтями калькуляції:

$$С_п = З_о + З_д + С_{оц} + Г_{осп} + З_м + О_п + А_м. \tag{7.21}$$

$$С_п = 5225+523+1265+784+97+784+1777 = 10455 \text{ грн.}$$

Визначимо плановий прибуток за рівнем рентабельності (P_n) програмної продукції, яка залежить від складності програми та ступеня новизни задачі.

Для даного програмного забезпечення рівень рентабельності складає 50%.

$$П_р = 0{,}01 \cdot P_n \cdot C_n, \quad (7.22)$$

де: P_n – рівень рентабельності, %.

$$П_р = 0{,}01 \cdot 50 \cdot 10455 = 5228 \text{ грн.}$$

Величини ціна підприємства, податок на додану вартість, відпускна ціна програмної продукції визначаються за формулами, приведеними в таблиці 7.9

Таблиця 7.9 – Нормативна калькуляція собівартості розробки програмного забезпечення задачі

Найменування статей витрат	Позначення	Величина, грн
1	2	3
1. Основна зарплата виконавців	$З_о$	5225
2. Додаткова зарплата виконавців	$З_д$	523
3. Відрахування на соціальні потреби	$С_{оц}$	1265
4. Загальногосподарські витрати	$Г_{осп}$	784
5. Витрати на матеріали	$З_М$	97
6. Освоєння нових операційних систем, мов програмування	$О_п$	784
7. Амортизація основних фондів	$А_м$	1777
8. Повна собівартість програмного забезпечення	$С_п$	10455
9. Плановий прибуток	$П_р$	5228
10.Ціна підприємства $Ц_п = С_п + П_р$	$Ц_п$	15683
11.Податок на додану вартість $ПДВ = 0.01 \cdot Н_{дв} \cdot Ц_п$	$ПДВ$	3136,6
12.Відпускна ціна програмної продукції $Ц = Ц_п + ПДВ$	$Ц$	18819,6

7.6 Визначення об’єму капітальних вкладень у споживача програмної продукції

Об’єм капітальних вкладень у споживача програмної продукції визначаємо на основі балансової вартості основних фондів, яка враховує ціну, транспортно-заготівельні витрати, вартість будівель, монтажних та пусконалагоджувальних робіт, а також витрати на випробування у виробничих умовах. Результати розрахунків зводимо у таблицю 7.10.

Таблиця 7.10 – Розрахунок об’єму капітальних вкладень у споживача програмної продукції

Найменування капітальних вкладень	Сума за варіантами, грн.	
	Базовий	Новий
Вартість програмної продукції	–	18820
Всього капітальних витрат	–	18820

7.7 Визначення експлуатаційних витрат

Експлуатаційні витрати у споживача програмної продукції визначаємо при умові роботи підсистеми на протязі року. Результати зводимо до таблиці 7.11.

$$З_о = Ч_е О_е М_е (1 + 0{,}01 Н_q)(1 + 0{,}01 Н_c), \qquad (7.23)$$

де $Ч_е$ – чисельність працівників, чол.;

$О_е$ – заробітна плата, грн./год;

$М_е$ – час, що витрачається на нарахування ЗП.

Після купівлі нового програмного забезпечення час на нарахування зменшився з 630 годин на рік до 393 годин на рік, тому витрати на технічне обслуговування зменшилися з:

$$З_{о\ баз} = 1 \cdot 63 \cdot 630 \cdot 1{,}1 \cdot 1{,}22 = 53264\ грн,$$

до:

$$З_{о\ нов} = 1 \cdot 63 \cdot 393 \cdot 1{,}1 \cdot 1{,}22 = 33227\ грн.$$

Таблиця 7.11 – Розрахунок експлуатаційних витрат у споживача програмної продукції

Найменування статей витрат	Позначення	Сума витрат за варіантами, грн.	
		Базовий	Новий
1. Заробітна плата(основна, додаткова, відрахування в соціальні фонди)	$З_р$	53264	33227
2. Витрати на електроенергію	$З_{ел}$	9639	8925
3. Витрати на амортизацію	$З_{ам}$	0	4705
Всього витрат за рік	$І$	62903	46857

Витрати на електроенергію визначаються з урахуванням споживаємої потужності ($П_{ел}$) в кіловатах, часу експлуатації технічних засобів ($Т_р$) в годинах та ціни однієї кіловат-години ($Ц_{ел}$):

$$З_{ел} = П_{ел} \cdot Т_р \cdot Ц_{ел}. \quad (7.24)$$

$$З_{ел\ баз} = 0{,}54 \cdot 8500 \cdot 2{,}1 = 9639 \text{ грн.}$$

$$З_{ел\ нов} = 0{,}50 \cdot 8500 \cdot 2{,}1 = 8925 \text{ грн.}$$

Витрати по амортизації визначаються на основі норм амортизаційних відрахувань, вартості програмної продукції і основних фондів. Для розрахунку складаємо таблицю 7.12.

Таблиця 7.12 – Розрахунок амортизаційних відрахувань

Групи основних фондів	Норма амортизації %	Балансова вартість, грн., за варіантами		Сума відрахувань, грн за варіантами	
		Базовий	Новий	Базовий	Новий
Програмна продукція	25	–	18820	–	4705
Всього відрахувань	-	–	18820	–	4705

7.8 Визначення економічної ефективності програмної продукції

Економічна ефективність програмного забезпечення визначається для виготовлювача і споживача за такими показниками.

Величина економічного ефекту при виготовленні програмної продукції, розраховуємо за формулою:

$$E_в = (Ц_п - C_п) \cdot N_е - \sum_{i=1}^{m} E_{p_m} \cdot K_{p_m}, \quad (7.25)$$

де: $K_р$ – балансова вартість основних фондів розробника, грн.; $E_р$ – розрахунковий коефіцієнт капіталовкладень.

$$Ев = (15683 - 10455) \cdot 24 - (0,05 \cdot 1408000 + 0,5 \cdot 125216 +$$
$$+0,25 \cdot 33190 + 0,2 \cdot 143000 + 0,1 \cdot 24000) \cdot 3/12 = 82396 \text{ грн.}$$

Визначимо період окупності додаткових капітальних вкладень у виробника програмної продукції:

$$T_в = \frac{K_р^*}{(Ц_п - C_п) \cdot N_е}, \quad (7.26)$$

де: $K_р^*$ – балансова вартість основних фондів розробника без врахування вартості ОФ третьої групи, так як їх строк служби на порядок більший ніж період розробки ПЗ.

$$T_в = \frac{325406}{(15683 - 10455) \cdot 24 \cdot 12 / 3} = 0,7 \text{ років}.$$

Визначимо величину економічного ефекту у користувача програмної продукції за формулою:

$$E_{сп} = (I_б - I_н) - E_н(K_н - K_б), \quad (7.27)$$

де: $I_б$, $I_н$ – величина експлуатаційних витрат за базовим и новим варіантом відповідно;

$K_б$, $K_н$ – об'єм капітальних вкладень за варіантами, що порівнюються.

$$E_{сп} = (62903\text{-}46857)\text{-}0,25 \cdot 18820 = 11341 \text{ грн.}$$

Показники економічної ефективності програмної продукції зводимо до таблиці 7.13.

Таблиця 7.13 – Показники економічної ефективності програмної продукції

Найменування показників	Одиниця виміру	Величина
1. Кількість екземплярів програми	Прим.	24
2. Повна собівартість розробленої програми	Грн.	10455
3. Ціна розробленої програми	Грн.	15683
4. Плановий прибуток від реалізації розробленої програми	Грн.	5228
5. Рентабельність програмної продукції	%	50
6. Об’єм додаткових капітальних вкладень у виробника програмної продукції	Грн.	1733406
7. Загальний прибуток від реалізації програмної продукції	Грн.	125472
8. Величина економічного ефекту при виготовлені програмної продукції	Грн.	82396
9. Період окупності додаткових капітальних вкладень у виробника програмної продукції	Років	0,7
10.Об’єм додаткових капітальних вкладень у споживача програмної продукції	Грн.	18820
11.Величина економічного ефекту у користувача програмної продукції	Грн.	11341
12.Період окупності додаткових капітальних вкладень у користувача програмної продукції	Рік	1,2

Визначимо період окупності додаткових капітальних вкладень у споживача програмної продукції за рахунок зниження експлуатаційних витрат:

$$T_{сп} = \frac{K_н - K_б}{I_б - I_н}, \qquad (7.28)$$

$$T_{сп} = \frac{18820}{62903 - 46857} = 1,2 \text{ років}.$$

7.9 Висновки

Розроблена програма економічно вигідна. За рахунок впровадження програмного забезпечення досягається скорочення часу обробки інформації, підвищується культура праці, підвищення якості приймаючих управлінських рішень.

8 ЗАХОДИ З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

8.1 Вступ

В охорону праці включають санітарно-гігієнічні, лікувально-профілактичні та організаційно-технічні системи правових і соціально-економічних заходів.

В кожній ІТ компанії є трудові відносини з працівниками. Згідно закону України “Про охорону праці” [3] кожна компанія впроваджує заходи з охорони праці. Реалізується трудові відносини з вживанням необхідних засобів з охорони праці та розробки відповідних документів:

– Інструкцій з охорони праці по кожній професії і загальні.

– Положення про охорону праці.

– Накази з охорони праці.

– Журнали реєстрації та інструктажу.

Роботодавець створює відділ який працює відповідно до типового положення, яку затверджується центральним органом виконавчої влади і забезпечує виконання вимог державної політики у сфері охорони праці.

За недотриманням вимог, керівники ІТ компаній можуть бути притягнуті до відповідальності, яка виглядає у виді накладання штрафу. Якщо в результаті порушення умов охорони праці є постраждалі працівники то керівні особи ІТ компаній притягуються до кримінальної відповідальності.

Законом України “Про охорону праці” [3] регламентуються загальні положення державної політики в галузі охорони праці, а конкретизуються ці положення нормативно-правовими актами про охорону праці, зокрема Наказом Міністерства соціальної політики України 14.02.2018 № 207, який зареєстровано в Міністерстві юстиції України 25 квітня 2018 р. за №508/31960 «Про затвердження Вимог щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями» [5], яким затверджено нормативно-правовий акт з охорони праці

НПАОП 0.00-7.15-18, «Правила охорони праці під час експлуатації електронно-обчислювальних машин», та «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» ДСанПіН 3.3.2-007-98.

Програмісти у процесі роботи мають негативний вплив на органи зору, а також мають значну розумову напругою і нервово-емоційне навантаження. Руки (суглоби польців та м'язи рук) при роботі з клавіатурою мають теж істотне навантаженням. До шкідливих факторів, які впливають на робитників галузі інформаційних технологій (ІТ) спеціалісти відносять високочастотні електромагнітні коливання (випромінювання) роботи апаратної частини ЕОМ та виділення шкідливих газів.

Ці шкідливі фактори можуть привести до професійних захворювань.

Розглянемо шкідливі чинники роботи програмістів керуючись наступними нормативно-правовими актами: «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» ДСанПіН 3.3.2-007-98 [5], та «Вимоги щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями» НПАОП 0.00-7.15-18.

Умови праці програміста включають наступні фактори:

– параметри повітряного середовища в приміщенні;

– вентиляція приміщення;

– освітлення приміщення;

– параметри повітряного середовища в приміщенні, тощо.

Щоб запропонувати заходи щодо зменшення негативного впливу комп’ютера на організм людини визначемо фактори, які можуть викликати професійне захворювання і впливають на працездатність програміста.

8.2 Аналіз санітарно-гігієнічних умов праці на робочому місці програміста

Розглянемо умови праці у приміщенні, в якому працюють програмісти. Геометричні розміри приміщення наведено у таблиці 8.1.

Таблиця 8.1 – Розміри приміщення

Найменування	Значення, м
Ширина	7
Довжина	8
Висота	3,3

Таблиця 8.2 – Площа та обсяг приміщення, на одного працюючого*

Геометрична характеристика	Одиниця виміру	Нормативне значення*	Фактичне значення
Площа, S	$м^2$	не менше 6.0	6,2
Об'єм, V	$м^3$	не менше 20.0	20,5

* Згідно ДСанПіН 3.3.2.007-98 (Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин).

У зазначеному приміщенні працюють 9 людей. За даними, які наведено у табл. 8.1, та табл. 8.2, можна зробити висновок, що площа та об’єм приміщення у розрахунку на одно робоче місце програміста не відповідають нормативним вимогам ДСанПіН 3.3.2-007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» [2], але відповідають нормативним вимогам Наказу Міністерства соціальної політики України № 207, від 14.02.2018 «Про затвердження Вимог щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями» та НПАОП 0.00-1.28-10 «Правила охорони праці під час експлуатації електронно-обчислювальних

машин»). Таним чином можна зробити висновок, що санітарно-гігієнічні умови праці на робочому місці програміста відповідають вимогам.

Температура повітря в приміщенні визначається впливом температури зовнішнього повітря і тепловою енергією, яка виділяється всередині приміщення. Джерелами виділення теплоти в даному приміщенні є електроустаткування, освітлювальні прилади, а також люди. У світлий час доби джерелом надлишкового тепла є сонячна радіація. Згідно Постанови № 42 від 01.12.1999 Головного державного санітарного лікаря України, робота, виконувана в даному приміщенні, відноситься до категорії Іа. В цьому випадку людина витрачає енергії до 120 ккал у годину. Вологість повітря в приміщенні визначається впливом багатьох факторів, серед яких: вологість атмосферного повітря, виділення вологи людьми (при диханні та випарами з поверхні шкіри).

Мікроклімат повітряного середовища в приміщенні характеризується запиленістю та загазованістю повітря. Мікроклімат приміщення визначається діючим на організм людини поєднанням, вологості, температури, швидкості руху повітря та інтенсивності теплового випромінювання. Аналіз мікроклімату складається з визначення зазначених вище факторів і порівняння результатів із встановленими нормами.

У таблиці 8.3 наведено оптимальні та фактичні значення параметрів мікроклімату як для категорії ваги робіт Іа, так і розглянутого приміщення. У приміщеннях, де встановлено ЕОМ, рекомендується застосування тільки оптимальних значень показників мікроклімату.

Таблиця 8.3 – Оптимальні і фактичні значення параметрів мікроклімату

Пора року	Оптимальні для Іа			Фактичні		
	Температура, °С	Воло-гість,%	Швидкість повітря, м/с	Температура, °С	Воло-гість%	Швидкість повітря, м/с
Холодна	22-24	40-60	0,1	22-24	40-55	0,1
Тепла	23-25	50-70	0,1	24-25	50-65	0,9

Проведений аналіз показує, що показники мікроклімату в приміщенні відповідають установленим нормам. Штучне опалення застосовується у холодний період року.

В літню пору застосовується кондиціонер.

Для боротьби з пилом робляться регулярні провітрювання та вологі прибирання приміщенні.

У приміщенні знаходяться наступні джерела шуму: принтер Xerox WorkCentre 3025BI (3025VBI), електродвигуни вентиляторів ЕОМ.

Одним з найважливіших факторів, які впливають на ефективність трудової діяльності людини, та попереджають травматизм і професійні захворювання програмістів є освітлення на робочому місці.

З 2019 року діють Державні будівельні норми україни "Природне і штучне освітлення" – ДБН В.2.5-28:2018 [1], у яких прописані вимоги до використання всіх освітлювальних приладів, у т.ч. світлодіодних.

Працю працівника, який постійно працює за комп'ютером, згідно ДБН В.2.5-28:2018 [1], можна віднести до роботи з малою точністю (найменший розмір об'єкта розрізнення від 1 до 5 мм) V-го розряду зорової роботи, з великою контрастністю об'єкта розрізнення (символів на екрані дисплея), з темним тлом (під розряд зорової роботи В). Приміщення можна віднести до 1-ої групи приміщень, у яких проводиться розрізнення об'єктів зорової роботи при фіксованому напрямку лінії зору того, що працює на робочу поверхню. Для такого типу приміщень і розряду зорової роботи нормоване значення коефіцієнта природної освітленості (КПО) робочої поверхні (при поєднаному, спільному освітленні), повинен становити не більше 1,5%, освітленість при штучному висвітленні повинна становити 300 Лк. [1], Крім того все поле зору повинне бути освітлено достатньо рівномірно – ця основна гігієнічна вимога. Так як яскраве світло на ділянці периферійного зору значно збільшує напруженість очей і, як наслідок, призводить до їх швидкої стомлюваності, ступінь освітлення приміщення і яскравість екрану комп'ютера повинні бути приблизно однаковими.

8.3 Розробка заходів з умов поліпшення охорони праці

Згідно аналізу умов праці в розглянутому приміщенні, ми одержали наступні результати:

– розмірі приміщення, у розрахунку на одному працюючого, відповідають нормативам;

– мікроклімат відповідає нормативному значенню;

– акустичні умови роботи не перевищують нормативних значеннь;

Таким чином можна припустити, що основною причиною можнивого зниження працездатності програміста є психофізіологічний фактор, тому основна пропозиція буде така: дотримання позитивної психологічної атмосфери в колективі та регламентованого режиму праці та відпочинку, організація робочого місця з урахуванням ергономічних вимог.

Рекомендовані заходи: регулярні періодичні наочні огляди персоналом шляхів для евакуації людей із приміщення, відповідно до плану евакуації (який повинен розташовуватись на видному місці у приміщенні), включення до колективного договору мінімально можливого вмісту аптечок з обов'язковою наявністю масок-клапанів, або іншого спорядження для штучного дихання. Регулярна періодична перевірка параметрів заземлення та занулення (вимірювання опору ланцюга) [9].

Регулярна наочне знайомство персоналу із шляхами для евакуації людей із приміщення відповідно до плану евакуації, забезпечення розподільних щитів спеціальними розетками з заземлюючими контактами; організація заземлення всіх приладів і пристроїв, які працюють при напрузі вище 36 В.

Так як при ураженні електричним струмом у людини може статися фібриляція шлуночків серця, в організації бажано мати дефібрилятор і підготовлений персонал для роботи з ним.

8.4 Розрахунок системи загального штучного освітлення виробничого приміщення де працюють ІТ – фахівці

Приміщення з ПК повинні мати природне і штучне освітлення, яке відповідало б вимогам ДБН В.2.5-28-2006 «Природне і штучне освітлення», ДСАНПІН 3.3.2.007-98 «Гігієнічні вимоги до організації роботи з візуальними дисплейними терміналами електронно-обчислювальних машин». Приміщення для роботи із ПЕОМ повинні мати природне й штучне освітлення. Віконні прорізи повинні бути орієнтовані на північ або на північний схід, забезпечувати коефіцієнт природної освітленості (К.П.О.) не менш 1,5% і мати жалюзі або штори. Віконні прорізи повинні мати регульовані пристрої для відкривання, а також жалюзі, завіски, зовнішні козирки тощо. Приміщення із ПЕОМ повинні бути обладнані системою загального рівномірного освітлення. У виробничих і адміністративно-суспільних 130 приміщеннях, де переважно ведеться робота з документами, допускається комбінована система штучного освітлення. Штучне освітлення має здійснюватися системою загального рівномірного освітлення, яка включає суцільні або такі, що перериваються лінії світильників, розташованих збоку робочих місць (переважно ліворуч), паралельно лінії зору користувачів ПК. Світильники повинні мати розсіювачі світла. У світильниках місцевого освітлення можна використовувати лампи накалювання. При розміщенні ПК по периметру приміщення лінії світильників штучного освітлення повинні розміщуватися локально над робочими місцями. Система освітлення робочого місця користувача ПК має відповідати наступним вимогам (рис. 8.1).

Рисунок 8.1 – Вимоги до системи освітлення робочого місця користувача ПК

Освітленість на робочому столі користувача в зоні розташування документів має бути в межах 300-500 лк. Якщо цей рівень освітленості неможливо забезпечити системою загального освітлення то допускається застосування світильників місцевого освітлення, але при цьому не повинно бути відблисків на поверхні екрану (яскравість відблисків не повинна перевищувати 40 кд/м^2) та перевищення його освітленості більше ніж 300 лк. Яскравість світильників загального освітлення, а також яскравість стелі при застосуванні системи відбитого освітлення не повинна перевищувати 200 кд/м^2. Величина коефіцієнта пульсації освітленості не повинна перевищувати 5%. Що стосується розподілу яскравості в полі зору працюючих за дисплеями ПК, то відношення значень яскравості робочих поверхонь не повинно перевищувати 3:1

Проводемо розрахунок штучного освітлення за методом коефіцієнта використання світлового потоку для приміщення ширина якого складає 6 м, довжина – 7 м, висота – 2,9 м. У зазначеному приміщенні працює 7 людей.

Для того, щоб визначити потрібну кількість світильників, які повинні забезпечити нормований рівень освітленості, визначимо світловий потік, що падає на робочу поверхню за формулою [1]:

$$F=ESKZ/n,$$

де: *F* – світловий потік, що розраховується, Лм;

E – нормована мінімальна освітленість, Лк; *E = 300 Лк;*

S – площа освітлюваного приміщення (у нашому випадку *S*=6 х 7 = 42 $м^2$);

K – коефіцієнт запасу, що враховує зменшення світлового потоку лампи в результаті забруднення світильників в процесі експлуатації (його значення залежить від типу приміщення і характеру робіт, що проводяться в ньому, в нашому випадку *K = 1,5);*

Z – відношення середньої освітленості до мінімальної (зазвичай приймається рівним 1.1... 1.2, в нашому випадку *Z = 1,1*);

n – коефіцієнт використання світлового потоку, (відношення світлового потоку, що падає на розрахункову поверхню, до сумарного потоку всіх ламп і обчислюється в долях одиниці [8]; залежить від характеристик світильника, розмірів приміщення, забарвлення стін і стелі, що характеризуються коефіцієнтами відбиття від стін **($\rho_{стін.}$)** і стелі **($\rho_{стелі}$),** значення коефіцієнтів дорівнюють $\rho_{стін}$ = 50% і $\rho_{стелі}$ = 50%.

Обчислимо індекс приміщення за формулою:

$$i=S/(h(A+B)),$$

де: *S* – площа приміщення, *S* = 42 $м^2$;

h – розрахункова висота підвісу, *h* = 2,9 *м (співпадає з в*исотою стелі, т.я. лампи освітлення закріплюються на стелі)*;*

A – ширина приміщення, *A = 6 м;*

B – довжина приміщення, *B* = 7 *м.*

Підставимо всі значення у формулу та визначемо індекса приміщення: і=1,4.

Знаючи індекс приміщення, за знаходимо *n = 0,29 (з табличних даних коефіцієнтів використання світлового потоку (n) світильників з відповідним типом лампам) [8].* Підставимо всі значення у формулу, визначемо світловий потік: *F=71689 Лм.*

Для розрахунку дудемо використовувати *світлодіодні стельові панелі Delux LED Panel 41 44Вт.*, світловий потік яких $F_л = 3600$ Лм.

Число ламп визначається по формулі:

$$N=F/F_л$$

де: F – світловий потік,

$F_л$ – світловий потік однієї лампи.

Підставимо всі значення у формулу та визначемо індекса приміщення: *N= 71689 / 3600=19,9 шт.*

Приймаємо необхідну кількість *світлодіодних світильників* 20 шт.

8.5 Висновки до розділу

Дотримання всіх необхідних умов праці не лише сприяє збереженню здоров'я працівників, а також підвищує ефективність виробництва вцілому.

З цих міркувань було здійснено аналіз умов праці на робочому місці ІТ-фахівця, призначеного для праці програмістів, проведено розгляд небезпечних та шкідливих факторів, що негативно впливають на програмістів під час роботи. Виконано розрахунок штучного освітлення, як одного з ключових факторів впливу на працездатність та здоров'я програміста. Розроблено заходи з умов поліпшення охорони праці.

9 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти, призначено для системи управління паралельними обчисленнями для наукових досліджень.

В межах України в недостатній мірі представлені вітчизняні розробки в цій області.

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти наведені теоретичне узагальнення й рішення наукового завдання дослідження методів управління паралельними обчисленнями для наукових досліджень.

Рішення даного завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем управління паралельними обчисленнями для наукових досліджень.

– Досліджена система управління паралельними обчисленнями для наукових досліджень.

– На основі отриманих результатів досліджень створена програмна реалізація системи управління паралельними обчисленнями для наукових досліджень.

Розроблені під час виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання управління паралельними обчисленнями для наукових досліджень.

Проведено аналіз предметної галузі в ході якого були виявлені об'єкти, взаємодія яких носить істотний характер для функціональної діяльності предметної галузі, і їхні основні характеристики; побудована алгоритм і вибраний середовище розробки.

Розроблене програмне забезпечення має простий, дружній та зручний інтерфейс користувача, що забезпечує легкість у освоєнні роботи програмного продукту, зручність у використанні, і не потребує особливих спеціальних знань.

При створені програмного забезпечення було використано об'єктно-орієнтований підхід, що відповідає сучасним тенденціям у галузі розробки комерційних програмних систем.

Програма реалізована на мові високого рівня RAD Studio Delphi 10. Дана мова програмування дозволяє найбільш ефективно обробляти дані. Це дозволило мінімізувати строк розробки програмного забезпечення, і, як слід, зменшити витрати на його розробку. Запропоноване програмне забезпечення ділиться на загальне програмне забезпечення, що поставляється із засобами обчислювальної техніки й спеціальне програмне забезпечення, що спеціально розроблене для даної конкретної системи й включає програми, що реалізують її функції.

Програма призначена для виконання під управлінням багатозадачної операційної системи Windows 10/11.

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм Madryga.

В цілому створене програмне забезпечення підтверджує правильність використаних проектних рішень та повністю відповідає вимогам технічного завдання. Створене програмне забезпечення має потенційну можливість для подальшого вдосконалення і застосування у різних галузях.

Розроблена програма має реальний економічний ефект від її впровадження у виробництво у сумі 11341 грн. З урахуванням вартості розробки програми та обладнання, строк окуплення становить 1,2 роки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Деркач Є.А. Дослідження та програмна реалізація системи управління паралельними обчисленнями для наукових досліджень // Збірник праць молодих науковців ЦНТУ. – Вип. 14. – Кропивницький: ЦНТУ, 2023.

2. I. Foster, C. Kesselman. The Grid: Blueprint for a New Computing Infrastructure. – Morgan Kaufmann Publishers, 1998.

3. I. Foster, C. Kesselman, S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. – International J. Supercomputer Applications, 15(3), 2001, http://www.globus.org/research/papers/anatomy.pdf

4. I. Foster, C. Kesselman, S. Tuecke, J. M. Nick. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. – Morgan Kaufmann Publishers, 2002.

5. I. Foster, H. Kishimoto, A. Savva, D. Berry et al. The Open Grid Services Architecture. – Global Grid Forum, 2005, http://www.ggf.org/documents/GFD.30.pdf

6. I. Foster, D. Gannon, H. Kishimoto et al. Open Grid Services Architecture Use Cases. – Global Grid Forum, 2004, http://www.ggf.org/documents/GFD.29.pdf

7. J. Treadwell, M. Behrens, D. Berry et al. Open Grid Services Architecture Glossary of Terms. – Global Grid Forum, 2005, http://www.ggf.org/documents/GFD.44.pdf

8. I Foster, C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. ftp://ftp.globus.org/pub/globus/papers/globus.pdf

9. I. Foster, C. Kesselman. The Globus Project: A Status Report. ftp://ftp.globus.org/pub/globus/papers/globus – hcw98.pdf

10. Overview of the Grid Security Infrastructure, http://www.globus.org/security/overview.html

11. RSL Specification, http://www – fp.globus.org/gram/rsl_spec1.html

12. Aggarwal C.C. Neural Networks and Deep Learning: A Textbook 1st ed. 2018 Edition. – Springer, 2018. – 520 p

13. Aho A.V., Hopcroft J.E., Ullman J.D. Data Structures and Algorithms. – Pearson, 2001. – 620 c.

14. Brink H., Richards J., Fetherolf M. Real-World Machine Learning. – Manning, 2016. – 474 p.

15. Cormen T.H., Leiserson C.E., Rivest R.L., Stein C. Introduction to Algorithms, 3rd Edition (The MIT Press) 3rd Edition – The MIT Press, 2019. – 1292 p.

16. Fenner M. Machine Learning with Python for Everyone (Addison-Wesley Data & Analytics Series) 1st Edition, Kindle Edition. – Addison-Wesley Professional, 2019. – 586 p.

17. Foreman J.W. Data Smart: Using Data Science to Transform Information into Insight 1st Edition. – Wiley, 2013. – 432 p.

18. Hurbans R. Grokking Artificial Intelligence Algorithms. – Manning, 2020. – 631 p.

19. Gusfield D. Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology 1st Edition. – Cambridge University Press, 2008. – 556 p.

20. Kotu V., Deshpande B. Data Science: Concepts and Practice. – Elsevier Science, 2018. – 953 p.

21. Knowledge Base A Complete Guide – 2021 Edition // The Art of Service – Knowledge Base Publishing, 2020. – 306 p.

22. Knuth D. The Art of Computer Programming, Vol. 1: Fundamental Algorithms, 3rd Edition 3rd Edition. – Addison-Wesley Professional, 2019. – 672 p.

23. Mattmann C. Machine Learning with TensorFlow, Second Edition. – Manning, 2020. – 1124 p.

24. Mueller J.P., Massaron L. Machine Learning For Dummies. – Wiley, 2016. – 714 p.

25. Teofili T. Deep Learning for Search. – Manning, 2019. – 695 p.

26. Rungta K. TensorFlow in 1 Day: Make your own Neural Network. – Publishdrive, 2019. – 587 p.

27. Weidman S. Deep Learning from Scratch: Building with Python from First Principles. – O'Reilly. 2019. – 252 p.

28. Rajasekaran S., Vijayalakshmi Pai G.A. Neural networks, fuzzy logic, and genetic algorithms: synthesis and applications (with cd-rom) Kindle Edition. – PHI, 2013. – 628 p.

29. Smirnov, O., Karapetyan, A., Fedorov, E., «Creating Neural Network and Single Solution Human-Based Metaheuristic Methods of Solving the Traveling Salesman Problem». CEUR Workshop Proceedings, Volume 3312, 2022, pp. 47-58.

30. Smirnov, O., Neskorodieva, T., Fedorov, E., Rudakov, K., Neskorodieva, A. «Method Detection Audit Data Anomalies on Basis Restricted Cauchy Machine» CEUR Workshop Proceedings, Volume 3187, 2022, pp. 1-12.

31. Smirnov O., Smirnova T., Anas M. Al-Oraiqat, Drieiev O., Polishchuk L., Sheroz Khan, Yassin M. Y. Hasan, Aladdein M. Amro, Hazim S. AlRawashdeh «Method for Determining Treated Metal Surface Quality Using Computer Vision Technology». Sensors (Basel, Switzerland) Volume 22, Issue 16, 6223, 2022.

32. Smirnov, O., Lakhno, V., Akhmetov, B., Chubaievskyi, V., Khorolska, K., Bebeshko, B. «Selection of a Rational Composition of İnformation Protection Means Using a Genetic Algorithm». In: Rajakumar, G., Du, KL., Vuppalapati, C., Beligiannis, G.N. (eds) Intelligent Communication Technologies and Virtual Mobile Networks. Lecture Notes on Data Engineering and Communications Technologies, vol 131. 2023. Springer, Singapore. pp. 21-34.

33. Kuznetsov, A., Oleshko, I., Chernov, K., Bagmut, M., Smirnova, T. «Biometric authentication using convolutional neural networks». Lecture Notes in Networks and Systems. Volume 152, 2021, Pages 85-98.

34. Smirnov O., Kuznetsov A., Kryvinska N., Kiian A., Kuznetsova K. «Full Non-Binary Constant-Weight Codes». SN Computer Science, Vol 2, 337, 2021. https://doi.org/10.1007/s42979-021-00739-w.

35. Smirnov O., Neskorodieva T., Fedorov E., Rymar P. «Neural Network Modeling Method of Transformations Data of Audit Production with Returnable Waste». CEUR Workshop Proceedings Volume 3101, 2021, Pages 192-207.

36. Smirnov, O., Kuznetsov, A., Potii, O., Poluyanenko, N., Stelnyk, I., Mialkovsky, D. «Combining and filtering functions in the framework of nonlinear-feedback shift register». International Journal of Computing; 2020, Volume 19, Issue 2 – Research Institute for Intelligent Computer Systems – 2020. – P. 247-256.

37. Smirnov O., Kuznetsov A., Kiian A., Kuznetsova T. «Non-binary constant weight coding technique». CEUR Workshop Proceedings. Volume 2740, 2020, Pages 102-114.

38. Smirnov O., Kuznetsov A., Kiian A., Cherep A., Kanabekova M., Chepurko I. «Testing of code-based pseudorandom number generators for post-quantum application». 2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT), Ukraine, Kyiv, May 14-18. 2020. P. 172-177.

39. Smirnov O., Kuznetsov A., Pushkar'ov A., Serhiienko R., Babenko V., Kuznetsova T., «Representation of Cascade Codes in the Frequency Domain». In: Radivilova T., Ageyev D., Kryvinska N. (eds) Data-Centric Business and Applications. Lecture Notes on Data Engineering and Communications Technologies, vol 48. Springer, Cham. 2021. pp 557-587.

40. Smirnov, O., Drieieva, H., Drieiev, O., Polishchuk, Y., Brzhanov, R., Aleksander, M. «Method of fractal traffic generation by a model of generator on the graph». CEUR Workshop Proceedings Volume 2616, 2020, Pages 366-379.

41. Smirnov, O., Drieieva, H., Drieiev, O., Simakhin, V., Bondar, S., Odarchenko, R. «Managing multifractal properties of the binary sequence generated with the Markov chains», CEUR Workshop Proceedings Volume 2608, 2020, Pages 633-645.

42. Smirnov O. Kuznetsov A., Zaicenko Yu., Pastukhov M., Oleshko O., Kuznetsova K., «Formation of Discrete Signals with Special Correlation Properties».

International Conference on Information and Telecommunication Technologies and Radio Electronics, UkrMiCo 2019; Odessa; Ukraine; 9-13 September 2019. P.22-28.

43. Smirnov, O., Kuznetsov, A., Kolovanova, I., Kuznetsova, T., «Noise immunity of the algebraic geometric codes». International Journal of Computing; 2019, Volume 18, Issue 4 – Research Institute for Intelligent Computer Systems – 2019. – P. 393-407.

44. Smirnov, O., Kuznetsov, A., Reshetniak, O., Ivko, N., Katkova, T., Kuznetsova, T., «Generators of Pseudorandom Sequence with Multilevel Function of Correlation». 2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T), Kyiv, Ukraine, 8 – 11 October 2019 . P.517-522.

45. Smirnov, O., Ulichev, O., Meleshko, Y., Khokh, V., Goncharenko, I. «Method of Choosing Objects for Informational Influence in Social Networks during Information Campaign Based on the Analytic Hierarchy Process». CEUR Workshop Proceedings, Vol 2588, P. 215-227, 2019.

46. Smirnov, O., Krasnobayev, V., Yanko, A., Kuznetsova, T. «Methods of nulling numbers in the system of residual classes». CEUR Workshop Proceedings, Vol 2588, P. 90-106, 2019.

47. Smirnov, O., Kuznetsov, A., Kovalchuk, D., Averchev, A., Pastukhov, M., Kuznetsova, K., «Formation of Pseudorandom Sequences with Special Correlation Properties», 2019 3rd International Conference on Advanced Information and Communications Technologies, AICT -2019/ Lviv, Ukraine, 2-6 July, 2019, P. 395-399.

48. Smirnov, O., Kuznetsov, A., Kiian, A., Zamula, A., Rudenko, S., Hryhorenko, V., «Variance Analysis of Networks Traffic for Intrusion Detection in Smart Grids», 2019 IEEE 6th International Conference On Energy Smart Systems (2019 IEEE ESS), Kyiv, Ukraine April 17-19, 2019 P. 353-358.

49. Smirnov, O., Kuznetsov, A., Kavun, S., Babenko, B., Nakisko, O., Kuznetsova, K., «Malware Correlation Monitoring in Computer Networks of Promising

Smart Grids», 2019 IEEE 6th International Conference On Energy Smart Systems (2019 IEEE ESS), Kyiv, Ukraine April 17-19, 2019 P. 347-352.

50. Smirnov, O., Kuznetsov, A., Kovalchuk, D., Pastukhov, M., Kuznetsova, K., Prokopovych-Tkachenko, D., «Discrete Signals with Special Correlation Properties», CEUR Workshop Proceedings Volume 2353, CEUR Workshop Proceedings 2019, Pages 618-629.

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування.....2
2 Підстава для розробки.....2
3 Мета та призначення розробки.....2
4 Джерела розробки.....2
5 Технічні вимоги.....2
5.1 Вміст проекту.....2
5.2 Показники призначення.....3
5.3 Вимоги до функціональних характеристик.....3
5.4 Вимоги до архітектури.....3
5.5 Вимоги до надійності.....3
5.6 Умови експлуатації.....4
5.7 Вимоги до складу та параметрів технічних засобів.....4
5.8 Вимоги до інформаційної і програмної сумісності.....4
5.8.1 Обладнання.....4
5.8.2 Мова програмування.....4
5.8.3 Вхідні дані.....5
5.8.4 Вихідні дані.....5
6 Вимоги до програмної документації.....5
7 Економічні вимоги.....5
8 Вимоги щодо охорони праці.....5
9 Перелік документів, що розробляються.....6
10 Етапи розробки.....6
11 Порядок контролю та приймання.....6

1 Найменування та область застосування

Це технічне завдання розповсюджується на дослідження та програмну реалізацію системи управління паралельними обчисленнями для наукових досліджень.

2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу за другим (магістерським) рівнем вищої освіти, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 33-13 від 04.08.2023 року).

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти є дослідження та програмна реалізація системи управління паралельними обчисленнями для наукових досліджень.

4 Джерела розробки

Джерелом цієї випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти є стосовна до теми література і існуючі аналоги.

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

– розробка програмної частин системи, а також розробка взаємодії системи з ОС та з користувачем;

– техніко-економічне обґрунтування доцільності прийнятого до розробки програмного забезпечення;

– аналіз умов праці;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– програмну реалізацію системи управління паралельними обчисленнями для наукових досліджень;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинно мати обмежень на версію драйверів та операційної системи.

5.4 Вимоги до архітектури

Компонент, що розробляється повинен використовувати системні засоби та апаратні засоби, що на даному етапі розвитку обчислювальної техніки найбільше поширені.

5.5 Вимоги до надійності

Програмні модулі написані по всім правилам, які стосуються стандартних викликів процедур, функцій, методів і форм, визначених технічною документацією на середовище розробки.

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 19-20 град. по Цельсію;

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на ПЕОМ архітектури IBM PC, працювати в ОС Windows 10/11 і з сумісними з цією платформою пристроями і прикладним програмним забезпеченням.

5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows 10/11.

5.8.1 Обладнання

Комп’ютер Intel® Celeron/8 Mb/1.2 Gb/SVGA 14” 1Mb або сумісні з ним.

5.8.2 Мова програмування

Середовище RAD Studio Delphi 10.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у виді опису структури даних, схем та опису алгоритму, а також текстів вихідних модулів програмного забезпечення згідно ЄСПД .

7 Економічні вимоги

7.1 Для ПЗ необхідно виробити функціонально-вартісний аналіз варіантів розробки.

7.2 Виконати розрахунок витрат показників економічного ефекту з урахуванням цін на 3 вересня 2023 року.

8 Вимоги щодо охорони праці

В частині охорони праці випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти повинен бути розглянутий розрахунок системи загального штучного освітлення виробничого приміщення де працюють ІТ-фахівці.

9 Перелік документів, що розробляються

– Наукова новизна – 1 аркуш.
– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 2 аркуша.
– Показники економічної ефективності – 1 аркуш.
– Пояснювальна записка – 113 аркушів.

10 Етапи розробки

10.1 Збір і обробка інформації по темі випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти. Постановка задачі на виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти (складання ТЗ).

10.2 Проведення досліджень або експериментальних робіт для уточнення основних положень випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти.

10.3 Розробка функціональних схем, блок схем алгоритмів роботи програмного забезпечення.

10.4 Побудова схем взаємодії даних.

10.5 Створення прототипу ПЗ.

10.6 Віднаходження ПЗ, аналіз отриманих результатів.

10.7 Робота над питанням охорони праці і техніки безпеки.

10.8 Розрахунок з техніко-економічного обґрунтування.

10.9 Оформлення пояснювальної записки і виконання робіт по графічній частині.

11 Порядок контролю та приймання

11.1 Подання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти на попередній захист 10.12.2023 р.

11.2 Подання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти на захист 11.12.2023 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти

_____ Петренюк В.І.

Дослідження та програмна реалізація системи управління паралельними обчисленнями для наукових досліджень

Лістинг програми

Код документу 12

Носій: CD/DVD-диск / USB-флеш-накопичувач

Загальна кількість аркушів: 53

Літера: РП

Кропивницький – 2023 року

ФАЙЛ ПРОЕКТУ КЛІЄНТСЬКОЇ ЧАСТИНИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

```
program Project_Client;
{
Міністерство освіти і науки України
Центральноукраїнський національний технічний університет
Факультет механіко-технологічний
Кафедра кібербезпеки та програмного забезпечення
Вихідний код до магістерської роботи
на тему: Дослідження та програмна реалізація системи
управління паралельними обчисленнями для наукових досліджень
Виконав: студент  6 курсу,
групи КН-22М-2, Деркач Євгеній Андрійович
Керівник: Петренюк В.І.
2023 рік
}
uses // підключення бібліотек
  Forms,
  Unit1 in 'Unit1.pas' {Form1}; // підключення форм
{$R *.res} // ресурси
begin
  Application.Initialize; // ініціалізація
  Application.CreateForm(TForm1, Form1); // створення форми
  Application.Run; // завантаження та работа
end.
```

ФАЙЛ ПРОЕКТУ СЕРВЕРНОЇ ЧАСТИНИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

```
program Project_Server;
{
Міністерство освіти і науки України
Центральноукраїнський національний технічний університет
Факультет механіко-технологічний
Кафедра кібербезпеки та програмного забезпечення
Вихідний код до магістерської роботи
на тему: Дослідження та програмна реалізація системи
управління паралельними обчисленнями для наукових досліджень
Виконав: студент  6 курсу,
групи КН-22М-2, Деркач Євгеній Андрійович
Керівник: Петренюк В.І.
2023 рік
}

uses // підключення бібліотек
Forms,
SysUtils,
frmMAIN in 'frmMAIN.pas' {Form1},
frmSettings in 'frmSettings.pas' {Form2},
frmGEN in 'frmGEN.pas' {Form3},
frmLOG in 'frmLOG.pas' {Form4},
frmSTAT in 'frmSTAT.pas' {Form5},
frmSPLASH in 'frmSPLASH.pas' {U_Form_Splash},
frmAbout in 'frmAbout.pas' {AboutBox};
{$R *.res} // htcehcb
begin
try // спроба виконання з можливою помилкою
U_Form_Splash:=TU_Form_Splash.Create(Application);
U_Form_Splash.Show;
U_Form_Splash.Update;
U_Form_Splash.Label2.Caption:='Підключення модулів розрахунку паралельного
обчислення';
U_Form_Splash.Update;
U_Form_Splash.Label2.Caption:='Налаштування';
U_Form_Splash.Update;
Application.HintPause:=200;//ms
Application.HintHidePause:=7000;//ms
Application.HintShortPause:=25;//ms
Application.Initialize;
Application.CreateForm(TForm1, Form1);
Application.CreateForm(TForm2, Form2);
Application.CreateForm(TForm3, Form3);
Application.CreateForm(TForm4, Form4);
Application.CreateForm(TForm5, Form5);
Application.CreateForm(TAboutBox, AboutBox);
finally
U_Form_Splash.free;
end;
Application.Run;
end.
```

ФАЙЛ АВТОРСЬКОГО ПРАВА

```
unit frmAbout;
{
Міністерство освіти і науки України
Центральноукраїнський національний технічний університет
Факультет механіко-технологічний
Кафедра кібербезпеки та програмного забезпечення
Вихідний код до магістерської роботи
на тему: Дослідження та програмна реалізація системи
управління паралельними обчисленнями для наукових досліджень
Виконав: студент  6 курсу,
групи КН-22М-2, Деркач Євгеній Андрійович
Керівник: Петренюк В.І.
2023 рік
}

interface

uses // підключення бібліотек
Windows, SysUtils, Classes, Graphics, Forms, Controls, StdCtrls,
  Buttons, ExtCtrls, jpeg;

type
  TAboutBox = class(TForm)
    Panel1: TPanel;
    ProgramIcon: TImage;
    ProductName: TLabel;
    Version: TLabel;
    Copyright: TLabel;
    Comments: TLabel;
    OKButton: TButton;
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  AboutBox: TAboutBox;

implementation

{$R *.dfm}

end.
```

ФАЙЛ ТЕРМІНАЛУ РОБОТИ

```
unit frmTerminal

{
Міністерство освіти і науки України
Центральноукраїнський національний технічний університет
Факультет механіко-технологічний
Кафедра кібербезпеки та програмного забезпечення
Вихідний код до магістерської роботи
на тему: Дослідження та програмна реалізація системи
управління паралельними обчисленнями для наукових досліджень
Виконав: студент  6 курсу,
групи КН-22М-2, Деркач Євгеній Андрійович
Керівник: Петренюк В.І.
2023 рік
}

interface

uses // підключення бібліотек
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ExtCtrls;

type
  TForm4 = class(TForm)
    Panel1: TPanel;
    Panel2: TPanel;
    Memo1: TMemo;
    Button1: TButton;
    Button2: TButton;
    SaveDialog1: TSaveDialog;
    procedure FormShow(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form4: TForm4;

implementation

uses // підключення бібліотек
frmMAIN;

{$R *.dfm}

procedure TForm4.FormShow(Sender: TObject);
begin
Memo1.Clear;
Memo1.Lines.Assign(Form1.Memo1.Lines);
end;

procedure TForm4.Button1Click(Sender: TObject);
begin
form4.Hide;
Form1.Show;
end;

procedure TForm4.Button2Click(Sender: TObject);
begin

if SaveDialog1.Execute then
begin
Memo1.Lines.SaveToFile(SaveDialog1.FileName);
```

```
end;


end;
end.
```

ФАЙЛ ПЕРШОЇ ФОРМИ

```
unit frmMAIN;
{
Міністерство освіти і науки України
Центральноукраїнський національний технічний університет
Факультет механіко-технологічний
Кафедра кібербезпеки та програмного забезпечення
Вихідний код до магістерської роботи
на тему: Дослідження та програмна реалізація системи
управління паралельними обчисленнями для наукових досліджень
Виконав: студент  6 курсу,
групи КН-22М-2, Деркач Євгеній Андрійович
Керівник: Петренюк В.І.
2023 рік
}

interface

uses // підключення бібліотек
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, Menus, StdCtrls, ExtCtrls,ScktComp, Buttons;
const
PORT_NUM=5555;//Користуемий порт до доступу
type
TForm1 = class(TForm)
Panel1: TPanel;
MainMenu1: TMainMenu;
Fil1: TMenuItem;
N1: TMenuItem;
N2: TMenuItem;
N3: TMenuItem;
N4: TMenuItem;
Panel2: TPanel;
Memo1: TMemo;
Panel3: TPanel;
Panel5: TPanel;
BitBtn1: TBitBtn;
BitBtn2: TBitBtn;
BitBtn3: TBitBtn;
BitBtn4: TBitBtn;
BitBtn5: TBitBtn;
BitBtn6: TBitBtn;
BitBtn7: TBitBtn;
BitBtn8: TBitBtn;
BitBtn9: TBitBtn;
BitBtn10: TBitBtn;
BitBtn11: TBitBtn;
BitBtn12: TBitBtn;
BitBtn13: TBitBtn;
BitBtn14: TBitBtn;
ListBox2: TListBox;
procedure FormCreate(Sender: TObject);
procedure FormDestroy(Sender: TObject);
procedure BitBtn9Click(Sender: TObject);
procedure BitBtn1Click(Sender: TObject);
procedure BitBtn13Click(Sender: TObject);
procedure BitBtn12Click(Sender: TObject);
procedure BitBtn10Click(Sender: TObject);
procedure N4Click(Sender: TObject);

private
CLIENT_CONNECT:integer;
A:TServerSocket;
public
procedure Add_Memo(s:string);
procedure Listen(Sender : TObject;Socket: TCustomWinSocket);
procedure Accept(Sender: TObject;Socket: TCustomWinSocket);
procedure ClientConnect(Sender: TObject; Socket: TCustomWinSocket);
```

```
procedure ClientDisconnect(Sender: TObject;Socket: TCustomWinSocket);
procedure ClientError(Sender: TObject;Socket: TCustomWinSocket; ErrorEvent:
TErrorEvent; var ErrorCode: Integer);
procedure ClientRead(Sender: TObject; Socket: TCustomWinSocket);
procedure ClientWrite(Sender: TObject;Socket: TCustomWinSocket);
procedure GetThread(Sender: TObject;ClientSocket: TServerClientWinSocket;var
SocketThread: TServerClientThread);
procedure ThreadEnd(Sender: TObject;Thread: TServerClientThread);
procedure ThreadStart(Sender: TObject; Thread: TServerClientThread);
procedure ADD_CLIENT_DATA(A:string);
procedure PACKED_RASBOR(s:string);
procedure PACKED_processing;
end;

var
Form1: TForm1;

implementation

uses // підключення бібліотек
frmLOG, frmSTAT;
var
Login,Pass,Excl:string;
COL:integer;
{$R *.dfm}

procedure TForm1.Accept(Sender: TObject; Socket: TCustomWinSocket);
begin

end;

procedure TForm1.ADD_CLIENT_DATA(A: string);
begin

end;

procedure TForm1.ClientConnect(Sender: TObject; Socket: TCustomWinSocket);
begin
Memo1.lines.add('Клієнт:'+Socket.RemoteAddress+'Хост:'+Socket.RemoteHost+' ->
CONNECT');
ListBox2.Items.Add('Підключення клієнту:');
ListBox2.Items.Add('            Клієнт:'+Socket.RemoteAddress);
ListBox2.Items.Add('            Хост:'+Socket.RemoteHost);
end;

procedure TForm1.ClientDisconnect(Sender: TObject;
Socket: TCustomWinSocket);
begin
Memo1.lines.add(' Клієнт:'+Socket.RemoteAddress+'Хост:'+Socket.RemoteHost+' ->
DISCONNECT');
ListBox2.Items.Add('            Хост:'+Socket.RemoteHost);
end;

procedure TForm1.ClientError(Sender: TObject; Socket: TCustomWinSocket;
ErrorEvent: TErrorEvent; var ErrorCode: Integer);
begin

end;

procedure TForm1.ClientRead(Sender: TObject; Socket: TCustomWinSocket);
begin
PACKED_RASBOR(Socket.ReceiveText);
PACKED_processing;
{Add_Memo('Name='+Login);
Add_Memo('Pass='+Pass);
Add_Memo('Excl='+Excl);
Add_Memo('Dlinna='+inttostr(COL));}
end;
```

```
procedure TForm1.ClientWrite(Sender: TObject; Socket: TCustomWinSocket);
begin
//
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
A:=TServerSocket.Create(self);
A.ServerType:=stNonBlocking;
A.OnListen:=Listen;
A.OnAccept:=Accept;
A.OnClientConnect:=ClientConnect;
A.OnClientDisconnect:=ClientDisconnect;
A.OnClientError:=ClientError;
A.OnClientRead:=ClientRead;
A.OnClientWrite:=ClientWrite;
A.OnGetThread:=GetThread;
A.OnThreadEnd:=ThreadEnd;
A.OnThreadStart:=ThreadStart;
A.Port:=PORT_NUM;
A.Open;
CLIENT_CONNECT:=0;
Form1.Caption:='Сервер обробки даних (ПОРТ'+inttostr(PORT_NUM)+')';
Add_Memo('Сервер');
end;

procedure TForm1.GetThread(Sender: TObject;
ClientSocket: TServerClientWinSocket;
var SocketThread: TServerClientThread);
begin

end;

procedure TForm1.Listen(Sender: TObject; Socket: TCustomWinSocket);
begin

end;

procedure TForm1.ThreadEnd(Sender: TObject; Thread: TServerClientThread);
begin

end;

procedure TForm1.ThreadStart(Sender: TObject; Thread: TServerClientThread);
begin

end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
A.Close;
A.Destroy;
end;

procedure TForm1.Add_Memo(s: string);
begin
Memo1.lines.Add(DateTimeToStr(now)+' : '+s);
end;

procedure TForm1.PACKED_RASBOR(s: string);
var
A:string;
Point:integer;
begin
A:=s;
MessageDlg(A,mtInformation,[mbOK],0);
if (s<>'') then
begin
Point:=pos('*',A);
```

```
Login:=copy(A,0,point-1);
MessageDlg(Login,mtInformation,[mbOK],0);
A:=copy(A,point+1,Length(A)-point);

Point:=pos('*',A);
Pass:=copy(A,0,point-1);;
//MessageDlg(Pass,mtInformation,[mbOK],0);

A:=copy(A,point+1,Length(A)-point);
Point:=pos('*',A);
Excl:=copy(A,0,point-1);;
//MessageDlg(Excl,mtInformation,[mbOK],0);

A:=copy(A,point+1,Length(A)-point);
COL:=strtoint(A);
end;

procedure TForm1.BitBtn9Click(Sender: TObject);
begin
form1.Close;
end;

procedure TForm1.PACKED_processing;
begin
A.Socket.Connections[0].SendText('Hello!');
{
A.Socket.SendText('OK');
A.Socket.SendText('OK');
A.Socket.SendText('OK');;}
{Login
Pass
Excl
COL }
end;

procedure TForm1.BitBtn1Click(Sender: TObject);
begin
Form4.show;
Form1.Hide;
end;

procedure TForm1.BitBtn13Click(Sender: TObject);
begin
Memo1.Lines.Clear;
ListBox2.Items.Clear;
end;

procedure TForm1.BitBtn12Click(Sender: TObject);
begin
Form5.ListBox1.Items.Clear;
end;

procedure TForm1.BitBtn10Click(Sender: TObject);
begin
MessageDlg('Ok',mtInformation,[mbOK],0);
end;

procedure TForm1.N4Click(Sender: TObject);
begin
MessageDlg('Betta ver 1.0',mtInformation,[mbOK],0);
end;

end.
```

ФАЙЛ ФОРМИ НАЛАШТУВАННЯ

```
unit frmSettings;
{
Міністерство освіти і науки України
Центральноукраїнський національний технічний університет
Факультет механіко-технологічний
Кафедра кібербезпеки та програмного забезпечення
Вихідний код до магістерської роботи
на тему: Дослідження та програмна реалізація системи
управління паралельними обчисленнями для наукових досліджень
Виконав: студент  6 курсу,
групи КН-22М-2, Деркач Євгеній Андрійович
Керівник: Петренюк В.І.
2023 рік
}

interface

uses // підключення бібліотек
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ComCtrls, ExtCtrls, StdCtrls;

type
  TForm2 = class(TForm)
    Panel2: TPanel;
    Panel3: TPanel;
    Button1: TButton;
    Button2: TButton;
    Panel4: TPanel;
    PageControl2: TPageControl;
    TabSheet6: TTabSheet;
    GroupBox4: TGroupBox;
    TabSheet7: TTabSheet;
    GroupBox1: TGroupBox;
    GroupBox2: TGroupBox;
    TabSheet8: TTabSheet;
    GroupBox3: TGroupBox;
    CheckBox1: TCheckBox;
    CheckBox2: TCheckBox;
    CheckBox3: TCheckBox;
    CheckBox4: TCheckBox;
    CheckBox5: TCheckBox;
    TabSheet9: TTabSheet;
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form2: TForm2;

implementation
{$R *.dfm}
end.
```

ФАЙЛ ВСПЛИВАЮЧОЇ ФОРМИ

```
unit frmSPLASH;
{
Міністерство освіти і науки України
Центральноукраїнський національний технічний університет
Факультет механіко-технологічний
Кафедра кібербезпеки та програмного забезпечення
Вихідний код до магістерської роботи
на тему: Дослідження та програмна реалізація системи
управління паралельними обчисленнями для наукових досліджень
Виконав: студент  6 курсу,
групи КН-22М-2, Деркач Євгеній Андрійович
Керівник: Петренюк В.І.
2023 рік
}

interface

uses // підключення бібліотек
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ComCtrls, StdCtrls, ExtCtrls;

type
  TU_Form_Splash = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    Label2: TLabel;
    Animate1: TAnimate;
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  U_Form_Splash: TU_Form_Splash;

implementation

{$R *.dfm}

end.
```

РОЗРОБЛЕНІ БІБЛІОТЕКИ, ЩО ВИКОРИСТОВУЮТЬСЯ ЯК У КЛІЄНТСЬКІЙ ТАК І У СЕРВЕРЕРНІЙ ЧАСТИНІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

```
unit Logics;
{
Міністерство освіти і науки України
Центральноукраїнський національний технічний університет
Факультет механіко-технологічний
Кафедра кібербезпеки та програмного забезпечення
Вихідний код до магістерської роботи
на тему: Дослідження та програмна реалізація системи
управління паралельними обчисленнями для наукових досліджень
Виконав: студент  6 курсу,
групи КН-22М-2, Деркач Євгеній Андрійович
Керівник: Петренюк В.І.
2023 рік
}

interface

uses
Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
glLogics,
glSBox, StdCtrls, glGrBox, ExtCtrls, ComCtrls, dsgnintf, ToolWin, ImgList,
richEdit,
glPage, Tabs;

type
TchGroupBoxPlus = class;

TchLogicsComponentEditor = class(TComponentEditor)
procedure ExecuteVerb(Index: Integer); override;
function GetVerb(Index: Integer): string; override;
function GetVerbCount: Integer; override;
private
procedure ShowEditor(LogicProducer: TLogicProducer);
end;

TchLogicsEditor = class(TForm)
SB: TchScrollBox;
Panel1: TPanel;
iPKey: TImage;
iFKey: TImage;
Label2: TLabel;
cbMode: TComboBox;
SBar: TStatusBar;
iLink: TImage;
cbNext: TComboBox;
Label4: TLabel;
eStepName: TEdit;
Label1: TLabel;
ImageList1: TImageList;
ToolBar1: TToolBar;
ToolButton1: TToolButton;
tbNew: TToolButton;
ToolButton3: TToolButton;
ToolButton4: TToolButton;
cbNextFalse: TComboBox;
Label5: TLabel;
Image3: TImage;
Label7: TLabel;
Label8: TLabel;
Shape1: TShape;
ImageList: TImageList;
ToolButton2: TToolButton;
ToolButton5: TToolButton;
ToolButton6: TToolButton;
ToolButton7: TToolButton;
cbIgnoreSpaces: TCheckBox;
```

```
ToolButton8: TToolButton;
pLeft: TPanel;
pLog: TPanel;
Splitter1: TSplitter;
tbStop: TToolButton;
Panel2: TPanel;
Splitter2: TSplitter;
reReslt: TRichEdit;
PC: TchPageControl;
tsLog: TTabSheet;
mLog: TMemo;
tsDictionary: TTabSheet;
mDictionary: TMemo;
Shape2: TShape;
Shape3: TShape;
Shape4: TShape;
Shape5: TShape;
Shape6: TShape;
Shape7: TShape;
Shape8: TShape;
Shape9: TShape;
Shape10: TShape;
Shape11: TShape;
Shape12: TShape;
Shape13: TShape;
TabSet1: TTabSet;
procedure SBEraseBkgndEvent(Sender: TObject; DC: HDC);
procedure cbNextChange(Sender: TObject);
procedure cbModeChange(Sender: TObject);
procedure tbNewClick(Sender: TObject);
procedure cbNextFalseChange(Sender: TObject);
procedure eStepNameChange(Sender: TObject);
procedure FormShow(Sender: TObject);
procedure ToolButton5Click(Sender: TObject);
procedure ToolButton7Click(Sender: TObject);
procedure cbIgnoreSpacesClick(Sender: TObject);
procedure pLeftDockOver(Sender: TObject; Source: TDragDockObject; X,
Y: Integer; State: TDragState; var Accept: Boolean);
procedure pLeftUnDock(Sender: TObject; Client: TControl;
NewTarget: TWinControl; var Allow: Boolean);
procedure pLeftDockDrop(Sender: TObject; Source: TDragDockObject; X,
Y: Integer);
procedure ToolButton8Click(Sender: TObject);
procedure tbStopClick(Sender: TObject);
procedure TabSet1Change(Sender: TObject; NewTab: Integer;
var AllowChange: Boolean);
private
FActiveBox: TchGroupBoxPlus;
LogicProducer: TLogicProducer;
Logics: TLogics;

procedure MouseDown_(Sender: TObject; Button: TMouseButton; Shift: TShiftState;
X, Y: Integer);
procedure MouseMove_(Sender: TObject; Shift: TShiftState; X, Y: Integer);
procedure MouseUp_(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
procedure DblClick_(Sender: TObject);
procedure SetActiveBox(const Value: TchGroupBoxPlus);
procedure UpdateView;
procedure AddBox(LogicElement_: TLogicElement);
procedure AddShape(CommentArea: TCommentArea);

procedure OnTraceMessage(Sender: TLogics; fStepResult: boolean; const
StepResult, ParsedResult, Msg: string);
public
function Execute(LogicProducer: TLogicProducer): boolean;
property ActiveBox: TchGroupBoxPlus read FActiveBox write SetActiveBox;
end;
```

```
TchGroupBoxPlus = class(TchGroupBox)
public
pt: TPoint;
Selected: boolean;
LogicElement: TLogicElement;
fAsLogical: boolean;
constructor Create(AOwner: TComponent); override;
//    destructor Destroy; override;
procedure Paint; override;
end;

TchShapePlus = class(TShape)
public
pt: TPoint;
Selected: boolean;
CommentArea: TCommentArea;
procedure Paint; override;
end;

var
glLogicsEditor: TchLogicsEditor;

implementation
uses // підключення бібліотек
glTypes, glUtils, geLogicItemEditor, clipbrd;
{$R *.DFM}

{ TchLogicsEditor }

function TchLogicsEditor.Execute(LogicProducer: TLogicProducer): boolean;
var
i: integer;
begin
fLogicItemEditor := TfLogicItemEditor.Create(nil);

mDictionary.Lines.Assign(LogicProducer.Dictionary);
try // спроба виконання з можливою помилкою


self.LogicProducer := LogicProducer;
self.Logics := LogicProducer.Logics;

Logics.OnTraceMessage := OnTraceMessage;

cbNext.Items.Clear;
cbNextFalse.Items.Clear;
cbNext.Items.Add('');
cbNextFalse.Items.Add('');

for i := 0 to Logics.Count-1 do AddBox(Logics[i]);

for i := 0 to LogicProducer.CommentAreas.Count-1 do
AddShape(LogicProducer.CommentAreas[i]);

Result := ShowModal = mrOK;
finally
fLogicItemEditor.Free;
end;

pLog.Dock(pLeft, rect(1,1,10,10));
pLog.Dock(pLeft, rect(1,1,10,10));
pLeft.Dock(pLog, rect(1,1,1,1));
end;

procedure TchLogicsEditor.AddBox(LogicElement_: TLogicElement);
var
Box: TchGroupBoxPlus;
begin
Box := TchGroupBoxPlus.Create(self);
```

```
with Box do
begin
Parent := SB;
Left := LogicElement_.Left;
Top := LogicElement_.Top;
Width := 100;
Height := 50;
Caption := LogicElement_.Caption;
Options := Options - [fgoCanCollapse];
CaptionAlignment := fcaWidth;
CaptionBorder.Inner := bvRaised;
CaptionBorder.Outer := bvLowered;
Colors.Caption := clBtnShadow;
Colors.TextActive := clBtnHighlight;
OnMouseDown := MouseDown_;
OnMouseMove := MouseMove_;
OnMouseUp := MouseUp_;
OnDblClick := DblClick_;
Border.Inner := bvRaised;
Border.Outer := bvNone;
Colors.Client := clWhite;
Colors.ClientActive := clWhite;
Box.LogicElement := LogicElement_;

cbNext.Items.AddObject(LogicElement_.Caption, LogicElement);
cbNextFalse.Items.AddObject(LogicElement_.Caption, LogicElement);

end;
end;

procedure TchLogicsEditor.AddShape(CommentArea: TCommentArea);
var
Shape: TchShapePlus;
begin
Shape := TchShapePlus.Create(self);
Shape.CommentArea := CommentArea;
with Shape do
begin
Parent := SB;
Left := CommentArea.Left;
Top := CommentArea.Top;
Width := CommentArea.Width;
Height := CommentArea.Height;
Pen.Style := psDashDot;
Brush.Style := bsClear;

OnMouseDown := MouseDown_;
OnMouseMove := MouseMove_;
OnMouseUp := MouseUp_;
OnDblClick := DblClick_;
end;

end;

procedure TchLogicsEditor.MouseDown_(Sender: TObject; Button: TMouseButton;
Shift: TShiftState; X, Y: Integer);
var i: integer;
begin
if Sender is TchShapePlus then with Sender as TchShapePlus do
begin
pt.X := X; pt.Y := Y;
pt := ClientToScreen(pt);
Selected := true;
Tag := 1;
if (X>=Width-5)and(X<Width)and(Y>=Height-5)and(Y<Height) then Tag := 2;
exit;
end;

with TchGroupBoxPlus(Sender) do
```

```
begin
pt.X := X; pt.Y := Y;
pt := ClientToScreen(pt);
pt.Y := pt.Y;
// + SB.VertScrollBar.ScrollPos;
Tag := 1;
Options := Options + [fgoDelineatedText];
Colors.Caption := clBtnHighlight;
Colors.TextActive := clBlack;
Font.Style := [fsBold];
Selected := true;
ActiveBox := TchGroupBoxPlus(Sender);
end;

for i:=0 to SB.ControlCount-1 do
if (SB.Controls[i] is TchGroupBoxPlus) then with TchGroupBoxPlus(SB.Controls[i])
do
begin
if ActiveBox = TchGroupBoxPlus(SB.Controls[i]) then continue;
Options := Options - [fgoDelineatedText];
Colors.Caption := clBtnShadow;
Colors.TextActive := clBtnHighlight;
Font.Style := [];
Selected := false;
Repaint;
end;
end;

procedure TchLogicsEditor.MouseMove_(Sender: TObject; Shift: TShiftState; X, Y:
Integer);
var
pt_new: TPoint;
begin
if Sender is TchShapePlus then with Sender as TchShapePlus do
begin
case Tag of
1:
begin
pt_new.X := X; pt_new.Y := Y;
pt_new := ClientToScreen(pt_new);
Left := Left + pt_new.X - pt.X;
Top := Top + pt_new.Y - pt.Y;
CommentArea.Left := Left + SB.HorzScrollBar.ScrollPos;
CommentArea.Top := Top + SB.VertScrollBar.ScrollPos;
end;
2:
begin
pt_new.X := X; pt_new.Y := Y;
pt_new := ClientToScreen(pt_new);
Width := Width + pt_new.X - pt.X;
Height := Height + pt_new.Y - pt.Y;
if Width < 50 then Width := 50; if Height < 50 then Height := 50;
CommentArea.Width := Width;
CommentArea.Height := Height;
end;
end;
pt.X := pt_new.X; pt.Y := pt_new.Y;
exit;
end;

with TchGroupBoxPlus(Sender) do
begin
if bool(Tag) then
begin
pt_new.X := X; pt_new.Y := Y;
pt_new := ClientToScreen(pt_new);
pt_new.Y := pt_new.Y;
// + SB.VertScrollBar.ScrollPos;
Left := Left + pt_new.X - pt.X;
```

```
Top := Top + pt_new.Y - pt.Y;

LogicElement.Left := Left + SB.HorzScrollBar.ScrollPos;
LogicElement.Top := Top + SB.VertScrollBar.ScrollPos;

SBar.SimpleText := IntToStr(Left) + ':' + IntToStr(Top);

UpdateView;
end;
pt.X := pt_new.X; pt.Y := pt_new.Y;
end;
end;

procedure TchLogicsEditor.UpdateView;
var
DC: HDC;
begin
DC := GetDC(SB.Handle);
SendMessage(SB.Handle, WM_EraseBkgnd, WPARAM(DC),0);
ReleaseDC(SB.Handle, DC);
end;

procedure TchLogicsEditor.MouseUp_(Sender: TObject; Button: TMouseButton; Shift:
TShiftState; X, Y: Integer);
var i: integer;
begin
TControl(Sender).Tag := 0;
for i := 0 to SB.ControlCount-1 do
if (SB.Controls[i] is TchShapePlus) then (SB.Controls[i] as TchShapePlus).Paint;
end;

procedure TchLogicsEditor.DblClick_(Sender: TObject);
var str: string;
begin
TControl(Sender).Tag := 0;
if Sender is TchShapePlus then with Sender as TchShapePlus do
begin
str := CommentArea.Text;
if InputQuery('Caption', 'Comments', str) then CommentArea.Text := str;
PostMessage(TWinControl(Parent).Handle, WM_LBUTTONUP, 1, 1);
exit;
end;

fLogicItemEditor.Execute(Logics, TchGroupBoxPlus(Sender).LogicElement);
PostMessage(TWinControl(Sender).Handle, WM_LBUTTONUP, 1, 1);
end;


procedure TchLogicsEditor.SetActiveBox(const Value: TchGroupBoxPlus);
var
i, Index: integer;
Box: TchGroupBoxPlus;
begin
FActiveBox := Value;

Index := cbNext.Items.IndexOfObject(Value.LogicElement.NextElement);
if Index <> -1 then cbNext.ItemIndex := Index else cbNext.ItemIndex := 0;

Index := cbNextFalse.Items.IndexOfObject(Value.LogicElement.NextFalseElement);
if Index <> -1 then cbNextFalse.ItemIndex := Index else cbNextFalse.ItemIndex :=
0;

cbMode.ItemIndex := integer(FActiveBox.LogicElement.IsFirst);
eStepName.Text := FActiveBox.LogicElement.Caption;
end;

procedure TchLogicsEditor.OnTraceMessage(Sender: TLogics; fStepResult: boolean;
const StepResult, ParsedResult, Msg: string);
begin
```

```
mLog.Lines.Add(Msg);
if reReslt.Text = '' then reReslt.Tag := 0;

if length(ParsedResult) = 0 then exit;
tag := 1 - tag;

reReslt.Text := reReslt.Text + ParsedResult;

reReslt.SelStart := length(reReslt.Text) - length(ParsedResult);
reReslt.SelLength := length(ParsedResult);
if tag = 0 then reReslt.SelAttributes.Color := clRed else
reReslt.SelAttributes.Color := clGreen;
reReslt.SelAttributes.Color := RGB(100+Random(100), 100+Random(100),
100+Random(100));

reReslt.SelLength := 0;
reReslt.Lines.EndUpdate;
end;

{ TchGroupBoxPlus }

procedure TchGroupBoxPlus.Paint;
var
i: integer;
R: TRect;
str: string;
begin
inherited;
ChangeBitmapColor((Owner as TchLogicsEditor).iLink.Picture.Bitmap,
GetPixel((Owner as TchLogicsEditor).iLink.Picture.Bitmap.Canvas.Handle, 0,0),
IIF(LogicElement.NextElement<>nil, clGreen, clRed));
//  BitBlt(Canvas.handle, 100-14-3, 3, 14, 13, (Owner as
TchLogicsEditor).iLink.Picture.Bitmap.Canvas.Handle, 0, 0, SRCCOPY);

Canvas.Font.Color := clTeal;//clBlue;
Canvas.Font.Style := [];
str := LogicElement.Expression + ' ' + LogicRuleLabels[LogicElement.Rule] + ' '
+ LogicElement.Value;
R := Bounds(3, 20, Width-6, Height-22);
DrawText(Canvas.handle, PChar(str), length(str), R, DT_WORDBREAK or
DT_END_ELLIPSIS or DT_MODIFYSTRING);
end;

procedure TchLogicsEditor.SBEraseBkgndEvent(Sender: TObject; DC: HDC);
var
Canvas: TCanvas;
LogicElement: TLogicElement;
i: integer;
PenFalse, Pen, PenTrue, OldPen, PenGrid: HPen;
Brush, OldBrush: HBrush;
NextBox, PrevBox, PrevFalseBox: TchGroupBoxPlus;
bmp: TBitmap;

function FindBox(LogicElement: TLogicElement): TchGroupBoxPlus;
var
i: integer;
begin
Result := nil;
if LogicElement = nil then exit;
for i := 0 to SB.ControlCount-1 do
if SB.Controls[i] is TchGroupBoxPlus then
if TchGroupBoxPlus(SB.Controls[i]).LogicElement = LogicElement then Result :=
TchGroupBoxPlus(SB.Controls[i]);
end;
procedure Line(X, Y, X2, Y2: integer; isTrueLine: boolean);
const R = 5;
begin
if isTrueLine then SelectObject(DC, PenTrue) else SelectObject(DC, PenFalse);
MoveToEx(DC, X, Y, nil); LineTo(DC, X2, Y2);
```

```
SelectObject(DC, Pen);
if (abs(X2-X) < 500)and(abs(Y2-Y) < 500) then
MoveToEx(DC, X, Y+1, nil); LineTo(DC, X2, Y2+1);
Ellipse(DC, X-R-2, Y-R-2, X+R*2-2, Y+R*2-2);
Ellipse(DC, X2-R-2, Y2-R-2, X2+R*2-2, Y2+R*2-2);
end;
procedure DrawGrid;
var i, j: integer;
const step = 14;
begin
FillRect(DC, SB.ClientRect, Brush);
for i:=1 to SB.Width div step do
begin
j := i*14;
MoveToEx(DC, j, 0, nil); LineTo(DC, j, SB.Height);
end;
for i:=1 to SB.Height div step do
begin
j := i*14;
MoveToEx(DC, 0, j, nil); LineTo(DC, SB.Width, j);
end;
end;
begin
try // спроба виконання з можливою помилкою

Brush := CreateSolidBrush(clWhite);
Pen := CreatePen( PS_SOLID, 1, clBlack );
PenGrid := CreatePen( PS_SOLID, 1, $E0E0E0 );
//PenLong := CreatePen( PS_DASHDOT, 1, $E0E0E0 );
PenTrue := CreatePen( PS_SOLID, 1, $FF9090 );
PenFalse := CreatePen( PS_SOLID, 1, $009090 );
OldPen := SelectObject( DC, PenGrid );
OldBrush := SelectObject( DC, Brush );

DrawGrid;

for i := 0 to SB.ControlCount-1 do
begin
if not(SB.Controls[i] is TchGroupBoxPlus) then continue;
LogicElement := TchGroupBoxPlus(SB.Controls[i]).LogicElement;
if LogicElement = nil then exit;

if LogicElement.IsFirst then
begin
MoveToEx(DC, 0, 0, nil);
LineTo(DC, SB.Controls[i].Left, SB.Controls[i].Top);
end;

PrevBox := FindBox(LogicElement.NextElement);
if Assigned(PrevBox) then
begin
Line(SB.Controls[i].Left + SB.Controls[i].Width, SB.Controls[i].Top,
PrevBox.Left, PrevBox.Top, true);
DeleteObject( SelectObject( DC, OldPen ) );
end;

PrevFalseBox := FindBox(LogicElement.NextFalseElement);
if Assigned(PrevFalseBox) then
begin
Line(SB.Controls[i].Left + SB.Controls[i].Width, SB.Controls[i].Top +
SB.Controls[i].Height, PrevFalseBox.Left, PrevFalseBox.Top, false);
DeleteObject( SelectObject( DC, OldPen ) );
end;

bmp := TBitmap.Create;
if LogicElement.NextElement <> nil then
begin
ImageList.GetBitmap(0, bmp);
```

```
BitBlt(DC, SB.Controls[i].Left + SB.Controls[i].Width-3, SB.Controls[i].Top,
bmp.width, bmp.height, bmp.Canvas.Handle, 0, 0, SRCCOPY);
end;

if LogicElement.NextFalseElement <> nil then
begin
ImageList.GetBitmap(1, bmp);
BitBlt(DC, SB.Controls[i].Left + SB.Controls[i].Width-3, SB.Controls[i].Top +
SB.Controls[i].Height-17, bmp.width, bmp.height, bmp.Canvas.Handle, 0, 0,
SRCCOPY);
end;

if LogicElement.IsFirst then
begin
ImageList.GetBitmap(2, bmp);
BitBlt(DC, SB.Controls[i].Left - 20, SB.Controls[i].Top, bmp.width, bmp.height,
bmp.Canvas.Handle, 0, 0, SRCCOPY);
end;

bmp.Free;

end;

finally
SelectObject(DC, Pen);
DeleteObject(SelectObject(DC, OldPen));
DeleteObject(PenTrue);
DeleteObject(PenFalse);
DeleteObject(PenGrid);
DeleteObject(Brush);
end;

//  for i := 0 to SB.ControlCount-1 do
//    if (SB.Controls[i] is TchShapePlus) then (SB.Controls[i] as
TchShapePlus).Paint;

end;

procedure TchLogicsEditor.cbNextChange(Sender: TObject);
begin
if FActiveBox = nil then exit;
if FActiveBox.LogicElement <> cbNext.items.Objects[cbNext.ItemIndex] then
begin
FActiveBox.LogicElement.NextElement :=
TlogicElement(cbNext.items.Objects[cbNext.ItemIndex]);
end;

UpdateView;
end;

procedure TchLogicsEditor.cbModeChange(Sender: TObject);
begin
if FActiveBox = nil then exit;
FActiveBox.LogicElement.IsFirst := cbMode.ItemIndex = 1;
end;

{ TchLogicsComponentEditor }

procedure TchLogicsComponentEditor.ExecuteVerb(Index: Integer);
begin
inherited;
ShowEditor(TLogicProducer(Component));
end;

function TchLogicsComponentEditor.GetVerb(Index: Integer): string;
begin
case Index of
0: Result := 'Edit component...';
end;
```

```
end;

function TchLogicsComponentEditor.GetVerbCount: Integer;
begin
Result := 1;
end;

procedure TchLogicsComponentEditor.ShowEditor(LogicProducer: TLogicProducer);
var
glLogicsEditor: TchLogicsEditor;
Logics: TLogics;
begin
Logics := LogicProducer.Logics;
{  with Logics.Add do
begin
Left := 10; Top := 10;
IsFirst := true;
end;

with Logics.Add do
begin
Left := 200; Top := 30;
PrevElementID := Logics[0].ID;
end;

Logics[0].NextElementID := Logics[1].ID;

with Logics.Add do
begin
Left := 200; Top := 100;
end;
}
try // спроба виконання з можливою помилкою

glLogicsEditor := TchLogicsEditor.Create(nil);
glLogicsEditor.Execute(LogicProducer);
finally
FreeAndNil(glLogicsEditor);
end;
end;


procedure TchLogicsEditor.tbNewClick(Sender: TObject);
var
LogicElement: TLogicElement;
begin
LogicElement := Logics.Add;
with LogicElement do
begin
Left := SB.Width div 2; Top := SB.Height div 2;
AddBox(LogicElement);
end;
end;

procedure TchLogicsEditor.cbNextFalseChange(Sender: TObject);
begin
if FActiveBox = nil then exit;
if FActiveBox.LogicElement <> cbNextFalse.items.Objects[cbNextFalse.ItemIndex]
then
begin
FActiveBox.LogicElement.NextFalseElement :=
TlogicElement(cbNextFalse.items.Objects[cbNextFalse.ItemIndex]);
end;
UpdateView;
end;

procedure TchLogicsEditor.eStepNameChange(Sender: TObject);
begin
if not Assigned(ActiveBox) then exit;
```

```
ActiveBox.LogicElement.Caption := eStepName.Text;
ActiveBox.Caption := eStepName.Text;
end;

procedure TchLogicsEditor.FormShow(Sender: TObject);
begin
SB.BufferedDraw := true;
end;

procedure TchLogicsEditor.ToolButton5Click(Sender: TObject);
var i: integer;
begin
pLog.Visible := true;

mLog.Lines.Clear;
reReslt.Text := '';

Logics.Analyze;

for i := 0 to SB.ControlCount-1 do
if (SB.Controls[i] is TchGroupBoxPlus) then
if TchGroupBoxPlus(SB.Controls[i]).LogicElement.IsTrue then
TchGroupBoxPlus(SB.Controls[i]).Colors.Caption := clGreen;

end;

procedure TchLogicsEditor.ToolButton7Click(Sender: TObject);
var
CommentArea: TCommentArea;
begin
CommentArea := LogicProducer.CommentAreas.Add;
with CommentArea do
begin
Left := SB.Width div 2; Top := SB.Height div 2;
Width := 100; Height := 100;
AddShape(CommentArea);
end;
end;

{ TchShapePlus }

procedure TchShapePlus.Paint;
var
i: integer;
R: TRect;
str: string;
begin
inherited;

Canvas.Font.Color := clBlue;
Canvas.Font.Style := [fsBold];
str := CommentArea.Text;
R := Bounds(3, 2, Width, Height);
DrawText(Canvas.handle, PChar(str), length(str), R, DT_WORDBREAK);
end;

procedure TchLogicsEditor.cbIgnoreSpacesClick(Sender: TObject);
begin
LogicProducer.IgnoreSpaces := cbIgnoreSpaces.Checked;
end;

procedure TchLogicsEditor.pLeftDockOver(Sender: TObject; Source:
TDragDockObject; X, Y: Integer; State: TDragState; var Accept: Boolean);
begin
Accept := true;
end;

procedure TchLogicsEditor.pLeftUnDock(Sender: TObject; Client: TControl;
NewTarget: TWinControl; var Allow: Boolean);
```

```
begin
(Sender as TWinControl).Height := 6;
end;

procedure TchLogicsEditor.pLeftDockDrop(Sender: TObject; Source:
TDragDockObject; X, Y: Integer);
begin
(Sender as TWinControl).Height := 100;
SBar.Top := 1500;
end;

procedure TchLogicsEditor.ToolButton8Click(Sender: TObject);
var i: integer;
begin

if Logics.TraceItem = nil then
begin
Logics.StartAnalyze;
reReslt.Text := '';
end;
tbStop.Enabled :=true;

Logics.AnalyzeStep;

for i := 0 to SB.ControlCount-1 do
if (SB.Controls[i] is TchGroupBoxPlus) then
if TchGroupBoxPlus(SB.Controls[i]).LogicElement.IsTrue then
TchGroupBoxPlus(SB.Controls[i]).Colors.Caption := clGreen;

ShowMessage(Logics.Result);

end;

procedure TchLogicsEditor.tbStopClick(Sender: TObject);
var i: integer;
begin
Logics.TraceItem := nil;
tbStop.Enabled :=false;

for i := 0 to SB.ControlCount-1 do
if (SB.Controls[i] is TchGroupBoxPlus) then
if TchGroupBoxPlus(SB.Controls[i]).LogicElement.IsTrue then
TchGroupBoxPlus(SB.Controls[i]).Colors.Caption := clBtnShadow;
end;

procedure TchLogicsEditor.TabSet1Change(Sender: TObject; NewTab: Integer; var
AllowChange: Boolean);
begin
PC.ActivePageIndex := NewTab;
end;

end.
```

ФАЙЛ БІБЛІОТЕКИ ПАРАЛЕЛЬНОГО ОБЧИСЛЕННЯ

```
unit AsyncCalls;
{
Міністерство освіти і науки України
Центральноукраїнський національний технічний університет
Факультет механіко-технологічний
Кафедра кібербезпеки та програмного забезпечення
Вихідний код до магістерської роботи
на тему: Дослідження та програмна реалізація системи
управління паралельними обчисленнями для наукових досліджень
Виконав: студент  6 курсу,
групи КН-22М-2, Деркач Євгеній Андрійович
Керівник: Петренюк В.І.
2023 рік
}
interface

uses // підключення бібліотек
  Windows, Messages, SysUtils, Classes, Contnrs, ActiveX, SyncObjs;

type
    INT_PTR = Integer;
  IInterface = IUnknown;
  INT_PTR = Integer;

  TAsyncIdleMsgMethod = procedure of object;

  TCdeclFunc = Pointer; // function(Arg1: Type1; Arg2: Type2; ...); cdecl;
  TCdeclMethod = TMethod; // function(Arg1: Type1; Arg2: Type2; ...) of object;
cdecl;
  TLocalAsyncProc = function: Integer;
  TLocalVclProc = function(Param: INT_PTR): INT_PTR;
  TLocalAsyncProcEx = function(Param: INT_PTR): INT_PTR;
  TLocalAsyncForLoopProc = function(Index: Integer;
                                    SyncLock: TCriticalSection): Boolean;
  TAsyncCallArgObjectProc = function(Arg: TObject): Integer;
  TAsyncCallArgIntegerProc = function(Arg: Integer): Integer;
  TAsyncCallArgStringProc = function(const Arg: string): Integer;
  TAsyncCallArgWideStringProc = function(const Arg: WideString): Integer;
  TAsyncCallArgInterfaceProc = function(const Arg: IInterface): Integer;
  TAsyncCallArgExtendedProc = function(const Arg: Extended): Integer;
  TAsyncCallArgVariantProc = function(const Arg: Variant): Integer;
  TAsyncCallArgObjectMethod = function(Arg: TObject): Integer of object;
  TAsyncCallArgIntegerMethod = function(Arg: Integer): Integer of object;
  TAsyncCallArgStringMethod = function(const Arg: string): Integer of object;
  TAsyncCallArgVariantMethod = function(const Arg: Variant): Integer of object;
  TAsyncCallArgObjectEvent = procedure(Arg: TObject) of object;
  TAsyncCallArgIntegerEvent = procedure(Arg: Integer) of object;
  TAsyncCallArgStringEvent = procedure(const Arg: string) of object;
  TAsyncCallArgWideStringEvent = procedure(const Arg: WideString) of object;
  TAsyncCallArgInterfaceEvent = procedure(const Arg: IInterface) of object;
  TAsyncCallArgExtendedEvent = procedure(const Arg: Extended) of object;
  TAsyncCallArgVariantEvent = procedure(const Arg: Variant) of object;
  TAsyncCallArgRecordProc = function(var Arg: TRecordType}): Integer;
  TAsyncCallArgRecordMethod = function(var Arg: TRecordType}): Integer of
object;
  TAsyncCallArgRecordEvent = procedure(var Arg: TRecordType}) of object;
  EAsyncCallError = class(Exception);

  {Інтерфейси}
  IAsyncCallEx = interface
    ['{A31D8EE4-17B6-4FC7-AC94-77887201EE56}']
    function GetEvent: THandle;
    function SyncInThisThreadIfPossible: Boolean;
  end;

  IAsyncRunnable = interface
```

```
    ['{1A313BBD-0F89-43AD-8B57-BBA3205F4888}']
    procedure AsyncRun;
procedure SetMaxAsyncCallThreads(MaxThreads: Integer);
function GetMaxAsyncCallThreads: Integer;
function AsyncCall(Proc: TAsyncCallArgObjectProc; Arg: TObject): IAsyncCall;
overload;
function AsyncCall(Proc: TAsyncCallArgIntegerProc; Arg: Integer): IAsyncCall;
overload;
function AsyncCall(Proc: TAsyncCallArgStringProc; const Arg: string):
IAsyncCall; overload;
function AsyncCall(Proc: TAsyncCallArgWideStringProc; const Arg: WideString):
IAsyncCall; overload;
function AsyncCall(Proc: TAsyncCallArgInterfaceProc; const Arg: IInterface):
IAsyncCall; overload;
function AsyncCall(Proc: TAsyncCallArgExtendedProc; const Arg: Extended):
IAsyncCall; overload;
function AsyncCallVar(Proc: TAsyncCallArgVariantProc; const Arg: Variant):
IAsyncCall; overload;
function AsyncCall(Method: TAsyncCallArgObjectMethod; Arg: TObject): IAsyncCall;
overload;
function AsyncCall(Method: TAsyncCallArgIntegerMethod; Arg: Integer):
IAsyncCall; overload;
function AsyncCall(Method: TAsyncCallArgStringMethod; const Arg: string):
IAsyncCall; overload;
function AsyncCall(Method: TAsyncCallArgWideStringMethod; const Arg:
WideString): IAsyncCall; overload;
function AsyncCall(Method: TAsyncCallArgInterfaceMethod; const Arg: IInterface):
IAsyncCall; overload;
function AsyncCall(Method: TAsyncCallArgExtendedMethod; const Arg: Extended):
IAsyncCall; overload;
function AsyncCallVar(Method: TAsyncCallArgVariantMethod; const Arg: Variant):
IAsyncCall; overload;

function AsyncCall(Method: TAsyncCallArgObjectEvent; Arg: TObject): IAsyncCall;
overload;
function AsyncCall(Method: TAsyncCallArgIntegerEvent; Arg: Integer): IAsyncCall;
overload;
function AsyncCall(Method: TAsyncCallArgStringEvent; const Arg: string):
IAsyncCall; overload;
function AsyncCall(Method: TAsyncCallArgWideStringEvent; const Arg: WideString):
IAsyncCall; overload;
function AsyncCall(Method: TAsyncCallArgInterfaceEvent; const Arg: IInterface):
IAsyncCall; overload;
function AsyncCall(Method: TAsyncCallArgExtendedEvent; const Arg: Extended):
IAsyncCall; overload;
function AsyncCallVar(Method: TAsyncCallArgVariantEvent; const Arg: Variant):
IAsyncCall; overload;

function AsyncCall(Runnable: IAsyncRunnable): IAsyncCall; overload;

procedure AsyncExec(Method: TNotifyEvent; Arg: TObject; IdleMsgMethod:
TAsyncIdleMsgMethod);
function LocalAsyncCall(LocalProc: TLocalAsyncProc): IAsyncCall;
function LocalAsyncCallEx(LocalProc: TLocalAsyncProcEx; Param: INT_PTR):
IAsyncCall;
procedure LocalAsyncExec(Proc: TLocalAsyncProc; IdleMsgMethod:
TAsyncIdleMsgMethod);
procedure LocalVclCall(LocalProc: TLocalVclProc; Param: INT_PTR = 0);
function LocalAsyncVclCall(LocalProc: TLocalVclProc; Param: INT_PTR = 0):
IAsyncCall;

function AsyncCallEx(Proc: TAsyncCallArgRecordProc; var Arg: TRecordType}):
IAsyncCall; overload;
function AsyncCallEx(Method: TAsyncCallArgRecordMethod; var Arg: TRecordType}):
IAsyncCall; overload;
function AsyncCallEx(Method: TAsyncCallArgRecordEvent; var Arg: TRecordType}):
IAsyncCall; overload;
{
```

```
Підтримка типів даних:
  Integer      :  Arg: Integer
  Boolean      :  Arg: Boolean
  Char         :  Arg: AnsiChar
  WideChar     :  Arg: WideChar
  Int64        :  [const] Arg: Int64
  Extended     :  [const] Arg: Extended
  Currency     :  [const] Arg: Currency
  String       :  [const] Arg: ShortString
  Pointer      :  [const] Arg: Pointer
  PChar        :  [const] Arg: PChar
  Object       :  [const] Arg: TObject
  Class        :  [const] Arg: TClass
  AnsiString   :  [const] Arg: AnsiString
  UnicodeString:  [const] Arg: UnicodeString
  PWideChar    :  [const] Arg: PWideChar
  WideString   :  [const] Arg: WideString
  Interface    :  [const] Arg: IInterface
  Variant      :  const Arg: Variant

Приклад:
  procedure Test(const S: string; I: Integer; E: Extended; Obj: TObject); cdecl;
  begin
  end;

  AsyncCall(@Test, ['Hallo', 10, 3.5, MyObject]);
}
function AsyncCall(Proc: TCdeclFunc; const Args: array of const): IAsyncCall;
overload;
function AsyncCall(Proc: TCdeclMethod; const Args: array of const): IAsyncCall;
overload;

const
  MAXIMUM_ASYNC_WAIT_OBJECTS = MAXIMUM_WAIT_OBJECTS - 3;

function AsyncMultiSync(const List: array of IAsyncCall; WaitAll: Boolean =
True;
  Milliseconds: Cardinal = INFINITE): Cardinal;
function AsyncMultiSyncEx(const List: array of IAsyncCall; const Handles: array
of THandle;
  WaitAll: Boolean = True; Milliseconds: Cardinal = INFINITE): Cardinal;
function MsgAsyncMultiSync(const List: array of IAsyncCall; WaitAll: Boolean;
  Milliseconds: Cardinal; dwWakeMask: DWORD): Cardinal;
function MsgAsyncMultiSyncEx(const List: array of IAsyncCall; const Handles:
array of THandle;
  WaitAll: Boolean; Milliseconds: Cardinal; dwWakeMask: DWORD): Cardinal;

implementation

uses // підключення бібліотек
  Forms; // AllocateHWnd

const
  WM_VCLSYNC = WM_USER + 12;

var
  SyncEvent: THandle;

type
  TThread = class(Classes.TThread)
    private
    class procedure WakeMainThread(Sender: TObject);
    public
    class procedure StaticSynchronize(AThread: TThread; AMethod: TThreadMethod);
  end;

class procedure TThread.StaticSynchronize(AThread: TThread; AMethod:
TThreadMethod);
var
```

```
  Obj: TThread;
begin
  if GetCurrentThreadId = MainThreadId then
    AMethod
  else if AThread <> nil then
    AThread.Synchronize(AMethod)
  else
  begin
        Obj := TThread.Create(True);
        try // спроба виконання з можливою помилкою

      Obj.Synchronize(AMethod);
    finally
      Obj.Free;
    end;
  end;
end;

procedure StaticSynchronize(AMethod: TThreadMethod);
begin

  TThread.Synchronize(nil, AMethod);
  TThread.StaticSynchronize(nil, AMethod);
end;

function CheckSynchronize(Timeout: Integer = 0): Boolean;
begin
  Result := False;
end;

function AcquireExceptionObject: Pointer;
type
  PRaiseFrame = ^TRaiseFrame;
  TRaiseFrame = record
    NextRaise: PRaiseFrame;
    ExceptAddr: Pointer;
    ExceptObject: TObject;
    ExceptionRecord: PExceptionRecord;
  end;
begin
  if RaiseList <> nil then
  begin
    Result := PRaiseFrame(RaiseList)^.ExceptObject;
    PRaiseFrame(RaiseList)^.ExceptObject := nil;
  end
  else
    Result := nil;
end;
var
  OrgWakeMainThread: TNotifyEvent;

class procedure TThread.WakeMainThread(Sender: TObject);
begin
  if Assigned(OrgWakeMainThread) then
    OrgWakeMainThread(Sender);
  SetEvent(SyncEvent);
end;

procedure HookWakeMainThread;
begin
  OrgWakeMainThread := Classes.WakeMainThread;
  Classes.WakeMainThread := TThread.WakeMainThread;
end;

procedure UnhookWakeMainThread;
begin
  Classes.WakeMainThread := OrgWakeMainThread;
end;
```

```
function InterlockedCompareExchange(var Destination: Integer; Exchange: Integer;
Comparand: Integer): Integer;
asm // використання asm вставок
       XCHG    EAX, ECX
  LOCK CMPXCHG [ECX], EDX
end;

type
  TAsyncCallThread = class(TThread)
  private
    FTaskCount: Integer;
    FTaskTime: Int64;
    protected
    procedure Execute; override;
  public
    constructor Create(ACreateSuspended: Boolean);
  end;

{TThreadPool містить пул потоків, які припинені}

  TThreadPool = class(TObject)
  private
    FWakeUpEvent: THandle;
    FThreadTerminateEvent: THandle;
    FSleepingThreadCount: Integer;
    FMaxThreads: Integer;
    FDestroying: Boolean;

    FThreadCount: Integer;
    FThreads: array[0..255] of TAsyncCallThread;

    FAsyncCallsCritSect: TRTLCriticalSection;
    FAsyncCallHead, FAsyncCallTail: TInternalAsyncCall;
    FAutoDeleteAsyncCalls: TInternalAsyncCall;

    FNumberOfProcessors: Cardinal;
    FSyncExecutedCount, FAsyncExecutedCount: Integer;

    FMainThreadSyncEvent: THandle;
    FMainThreadVclHandle: HWND;
    procedure MainThreadWndProc(var Msg: TMessage);
    procedure ProcessMainThreadSync;

    procedure AllocThread;
    function GetNextAsyncCall: TInternalAsyncCall; // called from the threads

    procedure WakeUpThread;
    procedure Sleep;
    procedure ReleaseAutoDeleteAsyncCalls;
    procedure CheckAutoDelete(Call: TInternalAsyncCall);
  public
    constructor Create;
    destructor Destroy; override;

    procedure CheckDestroying;
    procedure SendVclSync(Call: TInternalAsyncCall);

    procedure AddAsyncCall(Call: TInternalAsyncCall);
    function RemoveAsyncCall(Call: TInternalAsyncCall): Boolean;
    procedure ForgetAsyncCall(Call: TInternalAsyncCall);

    property MaxThreads: Integer read FMaxThreads;
    property NumberOfProcessors: Cardinal read FNumberOfProcessors;
    property MainThreadSyncEvent: THandle read FMainThreadSyncEvent;
  end;

{ ---------------------------------------------------------------------- }

  TAsyncCallArgObject = class(TInternalAsyncCall)
```

```
  private
    FProc: TAsyncCallArgObjectProc;
    FArg: TObject;
  protected
    function ExecuteAsyncCall: Integer; override;
  public
    constructor Create(AProc: TAsyncCallArgObjectProc; AArg: TObject);
  end;

  TAsyncCallArgString = class(TInternalAsyncCall)
  private
    FProc: TAsyncCallArgStringProc;
    FArg: string;
  protected
    function ExecuteAsyncCall: Integer; override;
  public
    constructor Create(AProc: TAsyncCallArgStringProc; const AArg: string);
  end;

  TAsyncCallArgWideString = class(TInternalAsyncCall)
  private
    FProc: TAsyncCallArgWideStringProc;
    FArg: WideString;
  protected
    function ExecuteAsyncCall: Integer; override;
  public
    constructor Create(AProc: TAsyncCallArgWideStringProc; const AArg:
WideString);
  end;

  TAsyncCallArgInterface = class(TInternalAsyncCall)
  private
    FProc: TAsyncCallArgInterfaceProc;
    FArg: IInterface;
  protected
    function ExecuteAsyncCall: Integer; override;
  public
    constructor Create(AProc: TAsyncCallArgInterfaceProc; const AArg:
IInterface);
  end;

  TAsyncCallArgExtended = class(TInternalAsyncCall)
  private
    FProc: TAsyncCallArgExtendedProc;
    FArg: Extended;
  protected
    function ExecuteAsyncCall: Integer; override;
  public
    constructor Create(AProc: TAsyncCallArgExtendedProc; const AArg: Extended);
  end;

  TAsyncCallArgVariant = class(TInternalAsyncCall)
  private
    FProc: TAsyncCallArgVariantProc;
    FArg: Variant;
  protected
    function ExecuteAsyncCall: Integer; override;
  public
    constructor Create(AProc: TAsyncCallArgVariantProc; const AArg: Variant);
  end;

{ ---------------------------------------------------------------------------- }

  TAsyncCallLocalProc = class(TInternalAsyncCall)
  private
    FProc: TLocalAsyncProc;
    FBasePointer: Pointer;
  protected
    function ExecuteAsyncCall: Integer; override;
```

```
  public
    constructor Create(AProc: TLocalAsyncProc; ABasePointer: Pointer);
  end;

  TAsyncCallLocalProcEx = class(TInternalAsyncCall)
  private
    FProc: TLocalAsyncProc;
    FBasePointer: Pointer;
    FParam: INT_PTR;
  protected
    function ExecuteAsyncCall: Integer; override;
  public
    constructor Create(AProc: TLocalAsyncProc; AParam: INT_PTR; ABasePointer:
Pointer);
  end;

  TAsyncVclCallLocalProc = class(TInternalAsyncCall)
  private
    FProc: TLocalVclProc;
    FBasePointer: Pointer;
    FParam: INT_PTR;
  protected
    function ExecuteAsyncCall: Integer; override;
  public
    constructor Create(AProc: TLocalVclProc; AParam: INT_PTR; ABasePointer:
Pointer);
  end;

{ ---------------------------------------------------------------------------- }

  TAsyncCallMethodArgObject = class(TInternalAsyncCall)
  private
    FProc: TAsyncCallArgObjectMethod;
    FArg: TObject;
  protected
    function ExecuteAsyncCall: Integer; override;
  public
    constructor Create(AProc: TAsyncCallArgObjectMethod; AArg: TObject);
  end;

  TAsyncCallMethodArgString = class(TInternalAsyncCall)
  private
    FProc: TAsyncCallArgStringMethod;
    FArg: string;
  protected
    function ExecuteAsyncCall: Integer; override;
  public
    constructor Create(AProc: TAsyncCallArgStringMethod; const AArg: string);
  end;

  TAsyncCallMethodArgWideString = class(TInternalAsyncCall)
  private
    FProc: TAsyncCallArgWideStringMethod;
    FArg: WideString;
  protected
    function ExecuteAsyncCall: Integer; override;
  public
    constructor Create(AProc: TAsyncCallArgWideStringMethod; const AArg:
WideString);
  end;

  TAsyncCallMethodArgInterface = class(TInternalAsyncCall)
  private
    FProc: TAsyncCallArgInterfaceMethod;
    FArg: IInterface;
  protected
    function ExecuteAsyncCall: Integer; override;
  public
```

```
    constructor Create(AProc: TAsyncCallArgInterfaceMethod; const AArg:
IInterface);
  end;

  TAsyncCallMethodArgExtended = class(TInternalAsyncCall)
  private
    FProc: TAsyncCallArgExtendedMethod;
    FArg: Extended;
  protected
    function ExecuteAsyncCall: Integer; override;
  public
    constructor Create(AProc: TAsyncCallArgExtendedMethod; const AArg:
Extended);
  end;

  TAsyncCallMethodArgVariant = class(TInternalAsyncCall)
  private
    FProc: TAsyncCallArgVariantMethod;
    FArg: Variant;
  protected
    function ExecuteAsyncCall: Integer; override;
  public
    constructor Create(AProc: TAsyncCallArgVariantMethod; const AArg: Variant);
  end;

{ ---------------------------------------------------------------------------- }

  TAsyncCallArgRecord = class(TInternalAsyncCall)
  private
    FProc: TAsyncCallArgRecordProc;
    FArg: Pointer;
  protected
    function ExecuteAsyncCall: Integer; override;
  public
    constructor Create(AProc: TAsyncCallArgRecordProc; AArg: Pointer);
  end;

  TAsyncCallMethodArgRecord = class(TInternalAsyncCall)
  private
    FProc: TAsyncCallArgRecordMethod;
    FArg: Pointer;
  protected
    function ExecuteAsyncCall: Integer; override;
  public
    constructor Create(AProc: TAsyncCallArgRecordMethod; AArg: Pointer);
  end;

  TAsyncCallArrayOfConst = class(TInternalAsyncCall)
  private
    FProc: function: Integer register;
    FArgs: array of TVarRec;
  protected
    procedure CopyVarRec(const Data: TVarRec; var Result: TVarRec);
    function ExecuteAsyncCall: Integer; override;
  public
    constructor Create(AProc: Pointer; const AArgs: array of const); overload;
    constructor Create(AProc: Pointer; MethodData: TObject; const AArgs: array
of const); overload;
    destructor Destroy; override;
  end;

{ ---------------------------------------------------------------------------- }
var
  ThreadPool: TThreadPool;

procedure SetMaxAsyncCallThreads(MaxThreads: Integer);
begin
  if MaxThreads >= Length(ThreadPool.FThreads) then
    MaxThreads := Length(ThreadPool.FThreads);
```

```
  if MaxThreads >= 0 then
    ThreadPool.FMaxThreads := MaxThreads;
end;

function GetMaxAsyncCallThreads: Integer;
begin
  Result := ThreadPool.FMaxThreads;
end;

{ ------------------------------------------------------------------------- }

function AsyncCall(Proc: TAsyncCallArgObjectProc; Arg: TObject): IAsyncCall;
begin
  if ThreadPool.MaxThreads = 0 then
    Result := TSyncCall.Create(Proc(Arg))
  else
    Result := TAsyncCallArgObject.Create(Proc, Arg).ExecuteAsync;
end;

function AsyncCall(Proc: TAsyncCallArgIntegerProc; Arg: Integer): IAsyncCall;
begin
  Result := AsyncCall(TAsyncCallArgObjectProc(Proc), TObject(Arg));
end;

function AsyncCall(Proc: TAsyncCallArgStringProc; const Arg: string):
IAsyncCall;
begin
  if ThreadPool.MaxThreads = 0 then
    Result := TSyncCall.Create(Proc(Arg))
  else
    Result := TAsyncCallArgString.Create(Proc, Arg).ExecuteAsync;
end;

function AsyncCall(Proc: TAsyncCallArgWideStringProc; const Arg: WideString):
IAsyncCall;
begin
  if ThreadPool.MaxThreads = 0 then
    Result := TSyncCall.Create(Proc(Arg))
  else
    Result := TAsyncCallArgWideString.Create(Proc, Arg).ExecuteAsync;
end;

function AsyncCall(Proc: TAsyncCallArgInterfaceProc; const Arg: IInterface):
IAsyncCall;
begin
  if ThreadPool.MaxThreads = 0 then
    Result := TSyncCall.Create(Proc(Arg))
  else
    Result := TAsyncCallArgInterface.Create(Proc, Arg).ExecuteAsync;
end;

function AsyncCall(Proc: TAsyncCallArgExtendedProc; const Arg: Extended):
IAsyncCall;
begin
  if ThreadPool.MaxThreads = 0 then
    Result := TSyncCall.Create(Proc(Arg))
  else
    Result := TAsyncCallArgExtended.Create(Proc, Arg).ExecuteAsync;
end;

function AsyncCallVar(Proc: TAsyncCallArgVariantProc; const Arg: Variant):
IAsyncCall;
begin
  if ThreadPool.MaxThreads = 0 then
    Result := TSyncCall.Create(Proc(Arg))
  else
    Result := TAsyncCallArgVariant.Create(Proc, Arg).ExecuteAsync;
end;
```

```
{ ------------------------------------------------------------------------- }

function AsyncCall(Method: TAsyncCallArgObjectMethod; Arg: TObject): IAsyncCall;
begin
  if ThreadPool.MaxThreads = 0 then
    Result := TSyncCall.Create(Method(Arg))
  else
    Result := TAsyncCallMethodArgObject.Create(Method, Arg).ExecuteAsync;
end;

function AsyncCall(Method: TAsyncCallArgIntegerMethod; Arg: Integer):
IAsyncCall;
begin
  Result := AsyncCall(TAsyncCallArgObjectMethod(Method), TObject(Arg));
end;

function AsyncCall(Method: TAsyncCallArgStringMethod; const Arg: string):
IAsyncCall;
begin
  if ThreadPool.MaxThreads = 0 then
    Result := TSyncCall.Create(Method(Arg))
  else
    Result := TAsyncCallMethodArgString.Create(Method, Arg).ExecuteAsync;
end;

function AsyncCall(Method: TAsyncCallArgWideStringMethod; const Arg:
WideString): IAsyncCall;
begin
  if ThreadPool.MaxThreads = 0 then
    Result := TSyncCall.Create(Method(Arg))
  else
    Result := TAsyncCallMethodArgWideString.Create(Method, Arg).ExecuteAsync;
end;

function AsyncCall(Method: TAsyncCallArgInterfaceMethod; const Arg: IInterface):
IAsyncCall;
begin
  if ThreadPool.MaxThreads = 0 then
    Result := TSyncCall.Create(Method(Arg))
  else
    Result := TAsyncCallMethodArgInterface.Create(Method, Arg).ExecuteAsync;
end;

function AsyncCall(Method: TAsyncCallArgExtendedMethod; const Arg: Extended):
IAsyncCall;
begin
  if ThreadPool.MaxThreads = 0 then
    Result := TSyncCall.Create(Method(Arg))
  else
    Result := TAsyncCallMethodArgExtended.Create(Method, Arg).ExecuteAsync;
end;

function AsyncCallVar(Method: TAsyncCallArgVariantMethod; const Arg: Variant):
IAsyncCall;
begin
{Виконати функцію SYNCHRON якщо немає пулу потоків}
  if ThreadPool.MaxThreads = 0 then
    Result := TSyncCall.Create(Method(Arg))
  else
    Result := TAsyncCallMethodArgVariant.Create(Method, Arg).ExecuteAsync;
end;

{ ------------------------------------------------------------------------- }

function AsyncCall(Method: TAsyncCallArgObjectEvent; Arg: TObject): IAsyncCall;
begin
  Result := AsyncCall(TAsyncCallArgObjectMethod(Method), Arg);
end;
```

```
function AsyncCall(Method: TAsyncCallArgIntegerEvent; Arg: Integer): IAsyncCall;
begin
  Result := AsyncCall(TAsyncCallArgIntegerMethod(Method), Arg);
end;

function AsyncCall(Method: TAsyncCallArgStringEvent; const Arg: string):
IAsyncCall;
begin
  Result := AsyncCall(TAsyncCallArgStringMethod(Method), Arg);
end;

function AsyncCall(Method: TAsyncCallArgWideStringEvent; const Arg: WideString):
IAsyncCall;
begin
  Result := AsyncCall(TAsyncCallArgWideStringMethod(Method), Arg);
end;

function AsyncCall(Method: TAsyncCallArgInterfaceEvent; const Arg: IInterface):
IAsyncCall;
begin
  Result := AsyncCall(TAsyncCallArgInterfaceMethod(Method), Arg);
end;

function AsyncCall(Method: TAsyncCallArgExtendedEvent; const Arg: Extended):
IAsyncCall;
begin
  Result := AsyncCall(TAsyncCallArgExtendedMethod(Method), Arg);
end;

function AsyncCallVar(Method: TAsyncCallArgVariantEvent; const Arg: Variant):
IAsyncCall;
begin
  Result := AsyncCallVar(TAsyncCallArgVariantMethod(Method), Arg);
end;

function AsyncCallRunnable(const Arg: IInterface): Integer;
begin
  IAsyncRunnable(Arg).AsyncRun;
  Result := 0;
end;

function AsyncCall(Runnable: IAsyncRunnable): IAsyncCall;
begin
  Result := AsyncCall(AsyncCallRunnable, IInterface(Runnable));
end;

{ ---------------------------------------------------------------------------- }

procedure AsyncExec(Method: TNotifyEvent; Arg: TObject; IdleMsgMethod:
TAsyncIdleMsgMethod);
var
  Handle: IAsyncCall;
begin
  Handle := AsyncCall(Method, Arg);
  if Assigned(IdleMsgMethod) then
  begin
    Handle.ForceDifferentThread;
    IdleMsgMethod;
    while MsgAsyncMultiSync([Handle], False, INFINITE, QS_ALLINPUT or
QS_ALLPOSTMESSAGE) = 1 do
      IdleMsgMethod;
  end;
end;

{ ---------------------------------------------------------------------------- }
function InternLocalAsyncCall(LocalProc: TLocalAsyncProc; BasePointer: Pointer):
IAsyncCall;
begin
  Result := TAsyncCallLocalProc.Create(LocalProc, BasePointer).ExecuteAsync;
```

```
end;

function LocalAsyncCall(LocalProc: TLocalAsyncProc): IAsyncCall;
asm // використання asm вставок
  mov ecx, edx // interface повертає address
  mov edx, ebp
  jmp InternLocalAsyncCall
end;

function InternLocalAsyncCallEx(LocalProc: TLocalAsyncProc; Param: INT_PTR;
BasePointer: Pointer): IAsyncCall;
begin
  Result := TAsyncCallLocalProcEx.Create(LocalProc, Param,
BasePointer).ExecuteAsync;
end;

function LocalAsyncCallEx(LocalProc: TLocalAsyncProcEx; Param: INT_PTR):
IAsyncCall;
asm // використання asm вставок
  push ecx // interface повертає address

  mov ecx, ebp
  call InternLocalAsyncCallEx
end;

procedure InternLocalAsyncExec(LocalProc: TLocalAsyncProc; IdleMsgMethod:
TAsyncIdleMsgMethod; BasePointer: Pointer);
var
  Handle: IAsyncCall;
begin
  Handle := TAsyncCallLocalProc.Create(LocalProc, BasePointer).ExecuteAsync;
  if Assigned(IdleMsgMethod) then
  begin
    Handle.ForceDifferentThread;
    IdleMsgMethod;
    while MsgAsyncMultiSync([Handle], False, INFINITE, QS_ALLINPUT or
QS_ALLPOSTMESSAGE) = 1 do
      IdleMsgMethod;
  end;
end;

procedure LocalAsyncExec(Proc: TLocalAsyncProc; IdleMsgMethod:
TAsyncIdleMsgMethod);
asm // використання asm вставок
 // метод
  pop ebp // видалення stackframe
  mov edx, ebp
  jmp InternLocalAsyncExec
end;
{ ------------------------------------------------------------------------- }
function AsyncCallEx(Proc: TAsyncCallArgRecordProc; var Arg: TRecordType}):
IAsyncCall;
begin
{Виконати функцію SYNCHRON якщо немає пулу потоків}
  if ThreadPool.MaxThreads = 0 then
    Result := TSyncCall.Create(Proc(Arg))
  else
    Result := TAsyncCallArgRecord.Create(Proc, @Arg).ExecuteAsync;
end;

function AsyncCallEx(Method: TAsyncCallArgRecordMethod; var Arg{: TRecordType}):
IAsyncCall;
begin

{Виконати функцію SYNCHRON якщо немає пулу потоків}
  if ThreadPool.MaxThreads = 0 then
    Result := TSyncCall.Create(Method(Arg))
  else
    Result := TAsyncCallMethodArgRecord.Create(Method, @Arg).ExecuteAsync;
```

```
end;
{ ------------------------------------------------------------------------------ }

function AsyncCall(Proc: TCdeclFunc; const Args: array of const): IAsyncCall;
overload;
var
  Call: TInternalAsyncCall;
begin
  Call := TAsyncCallArrayOfConst.Create(Proc, Args);
  if ThreadPool.MaxThreads = 0 then
  begin
    Call.InternExecuteSyncCall;
    Result := TAsyncCall.Create(Call);
  end
  else
    Result := Call.ExecuteAsync;
end;

function AsyncCall(Proc: TCdeclMethod; const Args: array of const): IAsyncCall;
overload;
var
  Call: TInternalAsyncCall;
begin
  Call := TAsyncCallArrayOfConst.Create(Proc.Code, TObject(Proc.Data), Args);
  if ThreadPool.MaxThreads = 0 then
  begin
    Call.InternExecuteSyncCall;
    Result := TAsyncCall.Create(Call);
  end
  else
    Result := Call.ExecuteAsync;
end;

{ ------------------------------------------------------------------------------ }

function InternalAsyncMultiSync(const List: array of IAsyncCall; const Handles:
array of THandle;
  WaitAll: Boolean; Milliseconds: Cardinal; MsgWait: Boolean; dwWakeMask:
DWORD): Cardinal;

  function InternalWait(const List: array of IAsyncCall; const Handles: array of
THandle;
    WaitAll: Boolean; Milliseconds: Cardinal; MsgWait: Boolean; dwWakeMask:
DWORD): Cardinal;
  var
    WaitHandles: array of THandle;
    Mapping: array of Integer;
    I: Integer;
    Count: Cardinal;
    EventIntf: IAsyncCallEx;
    SignalState: Cardinal;
  begin
    SetLength(WaitHandles, Length(List) + Length(Handles));
    SetLength(Mapping, Length(WaitHandles));
    Count := 0;
    for I := 0 to High(List) do
    begin
      if (List[I] <> nil) and Supports(List[I], IAsyncCallEx, EventIntf) then
      begin
        WaitHandles[Count] := EventIntf.GetEvent;
        if WaitHandles[Count] <> 0 then
        begin
          Mapping[Count] := I;
          Inc(Count);
        end;
      end
      else
      if not WaitAll then
      begin
```

```
        Result := I;
        Exit;
      end;
    end;

    for I := 0 to High(Handles) do
    begin
      WaitHandles[Count] := Handles[I];
      Mapping[Count] := Length(List) + I;
      Inc(Count);
    end;
{Зачекайте асинхронних викликів}
    if Count > 0 then
    begin
      if GetCurrentThreadId = MainThreadID then
      begin
        SignalState := WaitForMultipleObjectsMainThread(Count, WaitHandles,
WaitAll, Milliseconds, MsgWait, dwWakeMask);
        if SignalState = Count then // повідомлення
        begin
          Result := SignalState;
          Exit;
        end;
      end
      else
      begin
        if MsgWait then
        begin
          SignalState := MsgWaitForMultipleObjects(Count, WaitHandles[0],
WaitAll, Milliseconds, dwWakeMask);
          if SignalState = Count then // повідомлення
          begin
            Result := SignalState;
            Exit;
          end;
        end
        else
          SignalState := WaitForMultipleObjects(Count, @WaitHandles[0], WaitAll,
Milliseconds);
      end;
      if (SignalState >= WAIT_OBJECT_0) and} (SignalState < WAIT_OBJECT_0 +
Count) then Result := WAIT_OBJECT_0 + Mapping[SignalState - WAIT_OBJECT_0]
      else if (SignalState >= WAIT_ABANDONED_0) and (SignalState <
WAIT_ABANDONED_0 + Count) then
        Result := WAIT_ABANDONED_0 + Mapping[SignalState - WAIT_ABANDONED_0]
      else
        Result := SignalState;
    end
    else
      Result := WAIT_OBJECT_0; // Синхронізація
  end;

  function InternalWaitAllInfinite(const List: array of IAsyncCall; const
Handles: array of THandle): Cardinal;
  var
    I: Integer;
  begin

{Зачекайте асинхронних викликів}
    for I := 0 to
High(List) do
      if List[I] <> nil then
        List[I].Sync;

    if Length(Handles) > 0 then
    begin
      if GetCurrentThreadId = MainThreadID then
        WaitForMultipleObjectsMainThread(Length(Handles), Handles, True,
INFINITE, False, 0)
```

```
      else
        WaitForMultipleObjects(Length(Handles), @Handles[0], True, INFINITE);
    end;
    Result := WAIT_OBJECT_0;
  end;

var
  Count: Integer;
begin
  Count := Length(List) + Length(Handles);
  if (Count > 0) and (Count <= MAXIMUM_ASYNC_WAIT_OBJECTS) then
  begin
    if WaitAll and (Milliseconds = INFINITE) and not MsgWait and
(GetCurrentThreadId <> MainThreadId) then
      Result := InternalWaitAllInfinite(List, Handles)
    else
      Result := InternalWait(List, Handles, WaitAll, Milliseconds, MsgWait,
dwWakeMask);
  end
  else
    Result := WAIT_FAILED;
end;

function AsyncMultiSync(const List: array of IAsyncCall; WaitAll: Boolean;
  Milliseconds: Cardinal): Cardinal;
begin
  Result := InternalAsyncMultiSync(List, [], WaitAll, Milliseconds, False, 0);
end;

function AsyncMultiSyncEx(const List: array of IAsyncCall; const Handles: array
of THandle;
  WaitAll: Boolean = True; Milliseconds: Cardinal = INFINITE): Cardinal;
begin
  Result := InternalAsyncMultiSync(List, Handles, WaitAll, Milliseconds, False,
0);
end;

function MsgAsyncMultiSync(const List: array of IAsyncCall; WaitAll: Boolean;
  Milliseconds: Cardinal; dwWakeMask: DWORD): Cardinal;
begin
  Result := InternalAsyncMultiSync(List, [], WaitAll, Milliseconds, True,
dwWakeMask);
end;

function MsgAsyncMultiSyncEx(const List: array of IAsyncCall; const Handles:
array of THandle;
  WaitAll: Boolean; Milliseconds: Cardinal; dwWakeMask: DWORD): Cardinal;
begin
  Result := InternalAsyncMultiSync(List, Handles, WaitAll, Milliseconds, True,
dwWakeMask);
end;

procedure NotFinishedError(const FunctionName: string);
begin
  if FunctionName <> '' then
    OutputDebugString(PChar(FunctionName));
  raise EAsyncCallError.Create(RsAsyncCallNotFinished);
end;

procedure UnknownVarRecType(VType: Byte);
begin
  raise EAsyncCallError.CreateFmt(RsAsyncCallUnknownVarRecType, [VType]);
end;

{ ---------------------------------------------------------------------------- }
{ TThreadPool }

constructor TThreadPool.Create;
var
```

```
  SysInfo: TSystemInfo;
begin
  inherited Create;
  FMainThreadVclHandle := AllocateHWnd(MainThreadWndProc);
  FMainThreadSyncEvent := CreateEvent(nil, False, False, nil);
  FWakeUpEvent := CreateEvent(nil, False, False, nil);
  FThreadTerminateEvent := CreateEvent(nil, True, False, nil);
  InitializeCriticalSectionAndSpinCount(FAsyncCallsCritSect, 4000);

  GetSystemInfo(SysInfo);
  FNumberOfProcessors := SysInfo.dwNumberOfProcessors;
  FMaxThreads := SysInfo.dwNumberOfProcessors * 4 - 2 {main thread};
  if FMaxThreads > Length(FThreads) then
    FMaxThreads := Length(FThreads);
end;


procedure TThreadPool.CheckDestroying;
begin
  if FDestroying then
    raise EAsyncCallError.CreateRes(@RsNoVclSyncPossible);
end;

procedure TThreadPool.CheckAutoDelete(Call: TInternalAsyncCall);
var
  AutoDelete: Boolean;
begin
  EnterCriticalSection(FAsyncCallsCritSect);
  try // спроба виконання з можливою помилкою
    AutoDelete := Call.FAutoDelete;
  finally
    LeaveCriticalSection(FAsyncCallsCritSect);
  end;
  if AutoDelete then
  begin
    try // спроба виконання з можливою помилкою
      Call.Sync;
    except
      if Assigned(ApplicationHandleException) then
        ApplicationHandleException(Self);
    end;
    Call.Free;
  end;
end;

procedure TThreadPool.ReleaseAutoDeleteAsyncCalls;
var
  ItemP: ^TInternalAsyncCall;
  Next: TInternalAsyncCall;
begin
  EnterCriticalSection(FAsyncCallsCritSect);
  try // спроба виконання з можливою помилкою
    ItemP := @FAutoDeleteAsyncCalls;
    while ItemP^ <> nil do
    begin
      Next := ItemP^.FNext;
      try
        ItemP^.Sync;
      finally
        ItemP^.Free;
        ItemP^ := Next;
      end;
    end;
  finally
    LeaveCriticalSection(FAsyncCallsCritSect);
  end;
end;

function TThreadPool.RemoveAsyncCall(Call: TInternalAsyncCall): Boolean;
```

```
var
  Item: TInternalAsyncCall;
begin
  Result := False;
  EnterCriticalSection(FAsyncCallsCritSect);
  try // спроба виконання з можливою помилкою
    Item := FAsyncCallHead;
    if Item = Call then
    begin
      FAsyncCallHead := Call.FNext;
      if FAsyncCallHead = nil then
        FAsyncCallTail := nil;
      Result := True;
    end
    else
    begin
      while (Item <> nil) and (Item.FNext <> Call) do
        Item := Item.FNext;
      if Item <> nil then
      begin
        Item.FNext := Call.FNext;
        if Call = FAsyncCallTail then
          FAsyncCallTail := Item;
        Result := True;
      end;
    end;
  finally
    LeaveCriticalSection(FAsyncCallsCritSect);
  end;
end;

procedure TThreadPool.AddAsyncCall(Call: TInternalAsyncCall);
begin
  { Enqueue }
  EnterCriticalSection(FAsyncCallsCritSect);
  if FAsyncCallTail = nil then
  begin
    FAsyncCallHead := Call;
    FAsyncCallTail := Call;
  end
  else
  begin
    FAsyncCallTail.FNext := Call;
    FAsyncCallTail := Call;
  end;
  LeaveCriticalSection(FAsyncCallsCritSect);

{Всі потоки зайняті, необхідно створити ще один потік, якщо це можливо}
  if FSleepingThreadCount = 0 then
  begin
    if FThreadCount < MaxThreads then
      AllocThread;
  end;

  {Запустити один з потоків}
  WakeUpThread;
end;

procedure TThreadPool.ForgetAsyncCall(Call: TInternalAsyncCall);
var
  Item: TInternalAsyncCall;
begin
   Assert(Call.FRefCount > 0);
  EnterCriticalSection(FAsyncCallsCritSect);
  try
    Item := FAsyncCallHead;
    while (Item <> nil) and (Item <> Call) do
      Item := Item.FNext;
    if Item <> nil then
```

```
      Call.FAutoDelete := True // безпечно встановлювати FAutoDelete
    else
    begin
      Call.FNext := FAutoDeleteAsyncCalls;
      FAutoDeleteAsyncCalls := Call;
    end
  finally
    LeaveCriticalSection(FAsyncCallsCritSect);
  end;
end;

procedure TThreadPool.AllocThread;
var
  Index: Integer;
begin
  { Increment FThreadCount якщо <  FMaxThreads }
  repeat
    Index := FThreadCount;
  until (Index = FMaxThreads) or (InterlockedCompareExchange(FThreadCount, Index
+ 1, Index) = Index);

  if Index < FMaxThreads then
    FThreads[Index] := TAsyncCallThread.Create(False);
end;

procedure TThreadPool.SendVclSync(Call: TInternalAsyncCall);
begin
  CheckDestroying;

  if not PostMessage(FMainThreadVclHandle, WM_VCLSYNC, 0, LPARAM(Call)) then
    Call.Quit(0)
  else
    SetEvent(FMainThreadSyncEvent);
end;

procedure TThreadPool.WakeUpThread;
begin
  SetEvent(FWakeUpEvent);
end;

procedure TThreadPool.Sleep;
var
  Handles: array[0..1] of THandle;
begin
  Handles[0] := FWakeUpEvent;
  Handles[1] := FThreadTerminateEvent;

  InterlockedIncrement(FSleepingThreadCount);

  SetThreadPriority(GetCurrentThread, THREAD_PRIORITY_ABOVE_NORMAL);
  WaitForMultipleObjects(2, @Handles, False, INFINITE);
  SetThreadPriority(GetCurrentThread, THREAD_PRIORITY_NORMAL);  done in
TAsyncThread.Execute

  InterlockedDecrement(FSleepingThreadCount);
end;

procedure TThreadPool.MainThreadWndProc(var Msg: TMessage);
begin
  case Msg.Msg of
    WM_VCLSYNC:
      TInternalAsyncCall(Msg.LParam).InternExecuteSyncCall;
  else
    Msg.Result := DefWindowProc(FMainThreadVclHandle, Msg.Msg, Msg.WParam,
Msg.LParam);
  end;
end;

procedure TThreadPool.ProcessMainThreadSync;
```

```
var
  Msg: TMsg;
begin
  Assert(GetCurrentThreadId = MainThreadId);
  while PeekMessage(Msg, FMainThreadVclHandle, 0, 0, PM_REMOVE) do
  begin
    TranslateMessage(Msg);
    DispatchMessage(Msg);
  end;
end;

{ ---------------------------------------------------------------------------- }
{ TSyncCall }

function TSyncCall.Canceled: Boolean;
begin
  Result := False;
end;

procedure TSyncCall.CancelInvocation;
begin
end;

constructor TSyncCall.Create(AReturnValue: Integer);
begin
  inherited Create;
  FReturnValue := AReturnValue;
end;

function TSyncCall.Finished: Boolean;
begin
  Result := True;
end;

procedure TSyncCall.ForceDifferentThread;
begin
end;

procedure TSyncCall.Forget;
begin
end;

function TSyncCall.ReturnValue: Integer;
begin
  Result := FReturnValue;
end;

function TSyncCall.Sync: Integer;
begin
  Result := FReturnValue;
end;

{ ---------------------------------------------------------------------------- }
{ TInternalAsyncCall }

constructor TInternalAsyncCall.Create;
begin
  inherited Create;
  FEvent := CreateEvent(nil, True, False, nil);
end;

destructor TInternalAsyncCall.Destroy;
begin
  if FEvent <> 0 then
  begin
    try // спроба виконання з можливою помилкою
      Sync;
    finally
      CloseHandle(FEvent);
```

```
      FEvent := 0;
    end;
  end;
  inherited Destroy;
end;

function TInternalAsyncCall.Finished: Boolean;
begin
  if FCanceled or (FCancelInvocation and not FExecuted) then
    Result := True
  else
    Result := (FEvent = 0) or FFinished or (WaitForSingleObject(FEvent, 0) =
WAIT_OBJECT_0);
end;

procedure TInternalAsyncCall.ForceDifferentThread;
begin
  FForceDifferentThread := True;
end;

procedure TInternalAsyncCall.Forget;
begin
  ForceDifferentThread;
  ThreadPool.ForgetAsyncCall(Self);
end;

function TInternalAsyncCall.Canceled: Boolean;
begin
  Result := FCanceled;
end;

procedure TInternalAsyncCall.CancelInvocation;
begin
  FCancelInvocation := True;
end;

function TInternalAsyncCall.GetEvent: THandle;
begin
  Result := FEvent;
end;

procedure TInternalAsyncCall.InternExecuteAsyncCall;
var
  Value: Integer;
begin
  {$IFDEF DEBUG_THREADSTATS}
  InterlockedIncrement(ThreadPool.FAsyncExecutedCount);
  {$ENDIF DEBUG_THREADSTATS}
  Value := 0;
  try // спроба виконання з можливою помилкою
    if not FCancelInvocation then
    begin
      FExecuted := True;
      Value := ExecuteAsyncCall;
    end
    else
      FCanceled := True;
  except
    FFatalErrorAddr := ExceptAddr;
    FFatalException := AcquireExceptionObject;
  end;
  Quit(Value);
end;

procedure TInternalAsyncCall.InternExecuteSyncCall;
var
  Value: Integer;
begin
  {$IFDEF DEBUG_THREADSTATS}
```

```
  InterlockedIncrement(ThreadPool.FSyncExecutedCount);
  {$ENDIF DEBUG_THREADSTATS}
  Value := 0;
  try // спроба виконання з можливою помилкою
    if not FCancelInvocation then
    begin
      FExecuted := True;
      Value := ExecuteAsyncCall();
    end
    else
      FCanceled := True;
  finally
    Quit(Value);
  end;
end;

procedure TInternalAsyncCall.Quit(AReturnValue: Integer);
begin
  FReturnValue := AReturnValue;
  FFinished := True;
  SetEvent(FEvent);
end;

function TInternalAsyncCall.ReturnValue: Integer;
var
  E: Exception;
begin
  if not Finished then
    NotFinishedError('IAsyncCall.ReturnValue');
  Result := FReturnValue;

  if FFatalException <> nil then
  begin
    E := FFatalException;
    FFatalException := nil;
    raise E at FFatalErrorAddr;
  end;
end;

function TInternalAsyncCall.Sync: Integer;
var
  E: Exception;
begin
  if not Finished then
  begin
    if not SyncInThisThreadIfPossible then
    begin
      if GetCurrentThreadId = MainThreadID then
      begin
        if WaitForSingleObjectMainThread(FEvent, INFINITE) <> WAIT_OBJECT_0 then
          NotFinishedError('IAsyncCall.Sync');
      end
      else
      if WaitForSingleObject(FEvent, INFINITE) <> WAIT_OBJECT_0 then
        NotFinishedError('IAsyncCall.Sync');
    end;
  end;
  Result := FReturnValue;

  if FFatalException <> nil then
  begin
    E := FFatalException;
    FFatalException := nil;

    raise E at FFatalErrorAddr;
  end;
end;

function TInternalAsyncCall.SyncInThisThreadIfPossible: Boolean;
```

```
begin
  if not Finished then
  begin
    Result := False;
    if not FForceDifferentThread or ThreadPool.FDestroying then
    begin

{Якщо жоден потік не був призначений на цей асинхронний виклик,
видалити його}
if ThreadPool.RemoveAsyncCall(Self) then
      begin
        InternExecuteSyncCall;
        Result := True;
      end;
    end;
  end
  else
    Result := True;
end;

function TInternalAsyncCall.ExecuteAsync: TAsyncCall;
begin
  ThreadPool.AddAsyncCall(Self);
  Result := TAsyncCall.Create(Self);
end;

{ ---------------------------------------------------------------------------- }
{$IFDEF SUPPORT_LOCAL_FUNCTIONS}
{ TAsyncCallArrayOfConst }

constructor TAsyncCallArrayOfConst.Create(AProc: Pointer; const AArgs: array of
const);
var
  I: Integer;
begin
  inherited Create;
  FProc := AProc;
  SetLength(FArgs, Length(AArgs));
  for I := 0 to High(AArgs) do

    CopyVarRec(AArgs[I], FArgs[I]);
end;

constructor TAsyncCallArrayOfConst.Create(AProc: Pointer; MethodData: TObject;
const AArgs: array of const);
var
  I: Integer;
begin
  inherited Create;
  FProc := AProc;
  SetLength(FArgs, 1 + Length(AArgs));

  // insert "Self"
  FArgs[0].VType := vtObject;
  FArgs[0].VObject := MethodData;

  for I := 0 to High(AArgs) do
    CopyVarRec(AArgs[I], FArgs[I + 1]);
end;

destructor TAsyncCallArrayOfConst.Destroy;
var
  I: Integer;
  V: PVarRec;
begin
  for I := 0 to High(FArgs) do
  begin
    V := @FArgs[I];
    case V.VType of
```

```
      vtAnsiString: AnsiString(V.VAnsiString) := '';
      vtWideString: WideString(V.VWideString) := '';

      {$IFDEF UNICODE}
      vtUnicodeString: UnicodeString(V.VUnicodeString) := '';
      {$ENDIF UNICODE}
      vtInterface : IInterface(V.VInterface) := nil;

      vtString    : Dispose(V.VString);
      vtExtended  : Dispose(V.VExtended);
      vtCurrency  : Dispose(V.VCurrency);
      vtInt64     : Dispose(V.VInt64);
      vtVariant   : Dispose(V.VVariant);
    end;
  end;
  inherited Destroy;
end;

procedure TAsyncCallArrayOfConst.CopyVarRec(const Data: TVarRec; var Result:
TVarRec);
begin
  if (Data.VPointer <> nil) and
     (Data.VType in [vtString, vtAnsiString, vtWideString,
                     {$IFDEF UNICODE}vtUnicodeString,{$ENDIF} vtExtended,
                     vtCurrency, vtInt64, vtVariant, vtInterface]) then
  begin
    Result.VType := Data.VType;
    Result.VPointer := nil;
    case Result.VType of
      vtAnsiString: AnsiString(Result.VAnsiString) :=
AnsiString(Data.VAnsiString);

      vtWideString: WideString(Result.VWideString) :=
WideString(Data.VWideString);
      vtUnicodeString: UnicodeString(Result.VUnicodeString) :=
UnicodeString(data.VUnicodeString);
      vtInterface : IInterface(Result.VInterface) :=
IInterface(Data.VInterface);

      vtString    : begin New(Result.VString);   Result.VString^ :=
Data.VString^; end;
      vtExtended  : begin New(Result.VExtended); Result.VExtended^ :=
Data.VExtended^; end;
      vtCurrency  : begin New(Result.VCurrency); Result.VCurrency^ :=
Data.VCurrency^; end;

      vtInt64     : begin New(Result.VInt64);    Result.VInt64^ := Data.VInt64^;
end;
      vtVariant   : begin New(Result.VVariant);  Result.VVariant^ :=
Data.VVariant^; end;
    end;
  end
  else
    Result := Data;
end;

function TAsyncCallArrayOfConst.ExecuteAsyncCall: Integer;
var
  I: Integer;
  V: ^TVarRec;
  ByteCount: Integer;
begin
  ByteCount := Length(FArgs) * SizeOf(Integer) + $40;
{Створити буфер з нулями для функцій, які хочуть більше аргументів, ніж
вказано}

asm // використання asm вставок
    xor eax, eax
    mov ecx, $40 / 8
```

```
@@FillBuf:
    push eax
    push eax
    dec ecx
    jnz @@FillBuf
  end;

  for I := High(FArgs) downto 0 do // cdecl => right to left
  begin
    V := @FArgs[I];
    case V.VType of
      vtInteger:     // [const] Arg: Integer
        asm // використання asm вставок
          mov eax, V
          push [eax].TVarRec.VInteger
        end;

      vtBoolean,     // [const] Arg: Boolean
      vtChar:        // [const] Arg: AnsiChar
        asm // використання asm вставок
          mov eax, V
          xor edx, edx
          mov dl, [eax].TVarRec.VBoolean
          push edx
        end;

      vtWideChar:    // [const] Arg: WideChar
        asm // використання asm вставок
          mov eax, V
          xor edx, edx
          mov dx, [eax].TVarRec.VWideChar
          push edx
        end;

      vtExtended:    // [const] Arg: Extended
        asm // використання asm вставок
          add [ByteCount], 8 // two additional DWORDs
          mov eax, V
          mov edx, [eax].TVarRec.VExtended
          movzx eax, WORD PTR [edx + 8]
          push eax
          push DWORD PTR [edx + 4]
          push DWORD PTR [edx]
        end;

      vtCurrency,    // [const] Arg: Currency
      vtInt64:       // [const] Arg: Int64
        asm // використання asm вставок
          add [ByteCount], 4 // an additional DWORD
          mov eax, V
          mov edx, [eax].TVarRec.VCurrency
          push DWORD PTR [edx + 4]
          push DWORD PTR [edx]
        end;

      vtString,      // [const] Arg: ShortString
      vtPointer,     // [const] Arg: Pointer
      vtPChar,       // [const] Arg: PChar
      vtObject,      // [const] Arg: TObject
      vtClass,       // [const] Arg: TClass
      vtAnsiString,  // [const] Arg: AnsiString
      {$IFDEF UNICODE}

      vtUnicodeString, // [const] Arg: UnicodeString
      {$ENDIF UNICODE}
      vtPWideChar,   // [const] Arg: PWideChar
      vtVariant,     // const Arg: Variant
      vtInterface,   // [const]: IInterface
      vtWideString:  // [const] Arg: WideString
```

```
        asm // використання asm вставок
          mov eax, V
          push [eax].TVarRec.VPointer
        end;
    else
      UnknownVarRecType(V.VType);
    end;
  end;

  Result := FProc;

  asm // використання asm вставок // cdecl => we must clean up
    add esp, [ByteCount]
  end;
end;

{$ENDIF SUPPORT_LOCAL_FUNCTIONS}

{ ---------------------------------------------------------------------------- }
{ TAsyncCallArgRecord }

constructor TAsyncCallArgRecord.Create(AProc: TAsyncCallArgRecordProc; AArg:
Pointer);
begin
  inherited Create;
  FProc := AProc;
  FArg := AArg;
end;

function TAsyncCallArgRecord.ExecuteAsyncCall: Integer;
begin
  Result := FProc(FArg^);
end;

{ ---------------------------------------------------------------------------- }
{ TAsyncCallMethodArgRecord }

constructor TAsyncCallMethodArgRecord.Create(AProc: TAsyncCallArgRecordMethod;
AArg: Pointer);
begin
  inherited Create;
  FProc := AProc;
  FArg := AArg;
end;

function TAsyncCallMethodArgRecord.ExecuteAsyncCall: Integer;
begin
  Result := FProc(FArg^);
end;

{ ---------------------------------------------------------------------------- }
{ TAsyncCallArgObject }

constructor TAsyncCallArgObject.Create(AProc: TAsyncCallArgObjectProc; AArg:
TObject);
begin
  inherited Create;
  FProc := AProc;
  FArg := AArg;
end;



function TAsyncCallArgObject.ExecuteAsyncCall: Integer;
begin
  Result := FProc(FArg);
end;

{ ---------------------------------------------------------------------------- }
```

```
{ TAsyncCallMethodArgObject }

constructor TAsyncCallMethodArgObject.Create(AProc: TAsyncCallArgObjectMethod;
AArg: TObject);
begin
  inherited Create;
  FProc := AProc;
  FArg := AArg;
end;

function TAsyncCallMethodArgObject.ExecuteAsyncCall: Integer;
begin
  Result := FProc(FArg);
end;

{ ---------------------------------------------------------------------------- }
{ TAsyncCallArgString }

constructor TAsyncCallArgString.Create(AProc: TAsyncCallArgStringProc;
  const AArg: string);
begin
  inherited Create;
  FProc := AProc;
  FArg := AArg;
end;

function TAsyncCallArgString.ExecuteAsyncCall: Integer;
begin
  Result := FProc(FArg);
end;

{ ---------------------------------------------------------------------------- }
{ TAsyncCallMethodArgString }

constructor TAsyncCallMethodArgString.Create(AProc: TAsyncCallArgStringMethod;
const AArg: string);
begin
  inherited Create;
  FProc := AProc;
  FArg := AArg;
end;

function TAsyncCallMethodArgString.ExecuteAsyncCall: Integer;
begin
  Result := FProc(FArg);
end;

{ ---------------------------------------------------------------------------- }
{ TAsyncCallArgWideString }

constructor TAsyncCallArgWideString.Create(AProc: TAsyncCallArgWideStringProc;
const AArg: WideString);
begin
  inherited Create;
  FProc := AProc;
  FArg := AArg;
end;

function TAsyncCallArgWideString.ExecuteAsyncCall: Integer;
begin
  Result := FProc(FArg);
end;


{ ---------------------------------------------------------------------------- }
{ TAsyncCallMethodArgWideString }

constructor TAsyncCallMethodArgWideString.Create(AProc:
TAsyncCallArgWideStringMethod; const AArg: WideString);
```

```
begin
  inherited Create;
  FProc := AProc;
  FArg := AArg;
end;

function TAsyncCallMethodArgWideString.ExecuteAsyncCall: Integer;
begin
  Result := FProc(FArg);
end;

{ ---------------------------------------------------------------------------- }
{ TAsyncCallArgInterface }

constructor TAsyncCallArgInterface.Create(AProc: TAsyncCallArgInterfaceProc;
const AArg: IInterface);
begin
  inherited Create;
  FProc := AProc;
  FArg := AArg;
end;

function TAsyncCallArgInterface.ExecuteAsyncCall: Integer;
begin
  Result := FProc(FArg);
end;

{ ---------------------------------------------------------------------------- }
{ TAsyncCallMethodArgInterface }

constructor TAsyncCallMethodArgInterface.Create(AProc:
TAsyncCallArgInterfaceMethod; const AArg: IInterface);
begin

  inherited Create;
  FProc := AProc;
  FArg := AArg;
end;

function TAsyncCallMethodArgInterface.ExecuteAsyncCall: Integer;
begin
  Result := FProc(FArg);
end;

{ ---------------------------------------------------------------------------- }
{ TAsyncCallArgExtended }

constructor TAsyncCallArgExtended.Create(AProc: TAsyncCallArgExtendedProc; const
AArg: Extended);
begin
  inherited Create;
  FProc := AProc;
  FArg := AArg;
end;

function TAsyncCallArgExtended.ExecuteAsyncCall: Integer;
begin
  Result := FProc(FArg);
end;



{ ---------------------------------------------------------------------------- }
{ TAsyncCallMethodArgExtended }

constructor TAsyncCallMethodArgExtended.Create(AProc:
TAsyncCallArgExtendedMethod; const AArg: Extended);
begin
  inherited Create;
```

```
  FProc := AProc;
  FArg := AArg;
end;

function TAsyncCallMethodArgExtended.ExecuteAsyncCall: Integer;
begin
  Result := FProc(FArg);
end;

{ ------------------------------------------------------------------------- }
{ TAsyncCallArgVariant }

constructor TAsyncCallArgVariant.Create(AProc: TAsyncCallArgVariantProc; const
AArg: Variant);
begin
  inherited Create;
  FProc := AProc;
  FArg := AArg;
end;

function TAsyncCallArgVariant.ExecuteAsyncCall: Integer;
begin
  Result := FProc(FArg);
end;

{ ------------------------------------------------------------------------- }
{ TAsyncCallMethodArgVariant }

constructor TAsyncCallMethodArgVariant.Create(AProc: TAsyncCallArgVariantMethod;
const AArg: Variant);
begin
  inherited Create;
  FProc := AProc;
  FArg := AArg;
end;

function TAsyncCallMethodArgVariant.ExecuteAsyncCall: Integer;
begin
  Result := FProc(FArg);
end;

{ ------------------------------------------------------------------------- }

{ TAsyncCallLocalProc }

constructor TAsyncCallLocalProc.Create(AProc: TLocalAsyncProc; ABasePointer:
Pointer);
begin
  inherited Create;
  @FProc := @AProc;
  FBasePointer := ABasePointer;
end;

function TAsyncCallLocalProc.ExecuteAsyncCall: Integer;
asm // використання asm вставок
  mov edx, [eax].TAsyncCallLocalProc.FBasePointer
  mov ecx, [eax].TAsyncCallLocalProc.FProc
  xor eax, eax // paramater
  push edx
  call ecx
  pop ecx
end;

{ TAsyncCallLocalProcEx }

constructor TAsyncCallLocalProcEx.Create(AProc: TLocalAsyncProc; AParam:
INT_PTR; ABasePointer: Pointer);
begin
  inherited Create;
```

```
  @FProc := @AProc;
  FBasePointer := ABasePointer;
  FParam := AParam;
end;

function TAsyncCallLocalProcEx.ExecuteAsyncCall: Integer;
asm // використання asm вставок
  mov edx, [eax].TAsyncCallLocalProcEx.FBasePointer
  mov ecx, [eax].TAsyncCallLocalProcEx.FProc
  mov eax, [eax].TAsyncCallLocalProcEx.FParam
  push edx
  call ecx
  pop ecx
end;

end.
```