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 першим (бакалаврським) рівнем вищої освіти
на тему
“Програмне забезпечення системи моделювання взаємодії 64-х розрядних ОС з апаратними засобами ПК”

Виконав здобувач вищої освіти
IV курсу, групи КІ-19
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Бубела М.О.
«____» _____ 2023 р.

Керівник проекту
кандидат технічних наук, доцент
_____ Коваленко А.С.
«____» _____ 2023 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет
Факультет *Механіко-технологічний*
Кафедра *Кібербезпеки та програмного забезпечення*
Освітній ступінь *бакалавр*
Галузь знань ._12 _____*"*Інформаційні технології*"*_____
Спеціальність *123 "*Комп'ютерна інженерія*"*
Освітньо-професійна (освітньо-наукова) програма _*"*Комп'ютерна інженерія*"*_

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

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ПЕРШИМ (БАКАЛАВРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Бубелі Максиму Олеговичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Програмне забезпечення системи моделювання взаємодії 64-х розрядних ОС з апаратними засобами ПК*

2. Керівник роботи *Коваленко Анна Степанівна, канд. техн. наук, доцент*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 7-02 від 5.01.2023 року

3. Строк подання студентом роботи до захисту *23.05.2023 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою роботи є розробка програмного забезпечення системи моделювання взаємодії 64-х розрядних ОС з апаратними засобами ПК*

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

1. Призначення та область використання.
2.Перегляд аналогічних існуючих систем.
3. Опис і обґрунтування проектних рішень.
4. Етапи програмування системи.
5. Впровадження системи в промислову експлуатацію.
6. Висновки

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

Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*

7. Дата видачі завдання « 17 » січня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.03.2023 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2023 р.	
3.	Розробка моделі компонента	20.03.2023 р.	
4.	Розробка структур даних	25.03.2023 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.03.2023 р.	
6.	Програмування алгоритмів	10.04.2023 р.	
7.	Оформлення ПЗ	17.04.2023 р.	
8.	Попередній захист роботи	23.05.2023 р.	

Дата видачі завдання
« 17 » січня 2023 р.

Підпис керівника
_____ Коваленко А.С.
(прізвище та ініціали)

Завдання прийнято до виконання
« 17 » січня 2023 р.

Підпис здобувача
_____ Бубела М.О.
(прізвище та ініціали)

АНОТАЦІЯ

Бубела М.О. Програмне забезпечення системи моделювання взаємодії 64-х розрядних ОС з апаратними засобами ПК. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2023.

В даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи моделювання взаємодії 64-х розрядних ОС з апаратними засобами ПК.

Метою розробки є програмне забезпечення системи моделювання взаємодії 64-х розрядних ОС з апаратними засобами ПК.

Результат роботи – програмна реалізація системи моделювання взаємодії 64-х розрядних ОС з апаратними засобами ПК.

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 Delphi 10.4.

Ключові слова: комп'ютерна інженерія, моделювання, 64-х розрядні ОС

ABSTRACT

Bubela M.O. Software for modeling the interaction of 64-bit OS with PC hardware. 123 Computer engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2023.

In this final qualification work for the first (bachelor) level of higher education, software is developed, which is intended for the system of modeling the interaction of 64-bit OS with PC hardware.

The purpose of the development is the software of the system for modeling the interaction of 64-bit OS with PC hardware.

The result of the work is a software implementation of a system for modeling the interaction of 64-bit OSes with PC hardware.

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 Delphi 10.4 environment.

Keywords: computer engineering, modeling, 64-bit OS

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

DMA – Direct Memory Access;

DOS – дискова операційна система;

RS-232C – Recommended Standart 232 Version C, ревізія – EIA-232D;

USRT (Universal Synchronous Receiver/Transmitter) – універсальний синхронний прийомопередавач;

АЛП– арифметико–логічний пристрій;

АЦП – аналогово–цифровий перетворювач;

ВДТ – відеодісплейні термінали;

ЕЛТ – електронно–люмінісцентна трубка;

ЕОМ – електронна обчислювальна машина;

ЗПР– запит переривання;

ОС – операційна система;

ПДП – теж саме що і DMA;

ПЗП – постійний запам’ятовуючий пристрій;

ПЗП – постійний запом’ятовуючий пристрій;

ПК – персональний комп’ютер;

ППЗП – перепрограмувальний постійний запом’ятовуючий пристрій;

ПЧ – перетворювач частоти;

РА – регістр адреси;

РК – регістр команд;

РМП – регістр маски переривань;

РП – регістр пріоритетів;

ТД – технічна документація;

ТЗ – технічна задача;

ЦАП – цифро–аналоговий перетворювач;

ВСТУП

Актуальність теми. Останні декілька десятиріч бурхливо розвивається область інформаційних технологій. Але усі ці технології не можливі без використання персональних електронно-обчислювальних машин (ПЕОМ). Дослідження ринку ПЕОМ показало, що на даних час левову частку ринку займають ПЕОМ построєні на архітектурі 64-х розрядних мікропроцесорів [1-5]. Відповідно, на сучасному рівні існує потреба в кваліфікованих спеціалістах по розробці додатків під архітектуру 64-х розрядних мікропроцесорів. Але для того, щоб програмувати під цю архітектуру, потрібно добре знати її складові, зв'язки та взаємодію між ними.

В процесі навчання спеціалістів виникають потреби в навчальних засобах. Це різноманітні програми, мови програмування, навчальні стенди тощо. Однак, не всі з перерахованих вище потреб, може забезпечити сучасний комп'ютерний ринок. Отже, виникає потреба в створенні принципово нових навчальних засобів.

Таким чином, виходячи із усього перерахованого вище, актуальним завданням є розробка програмного забезпечення системи моделювання взаємодії 64-х розрядних ОС з апаратними засобами ПК з апаратними засобами персональних електронно-обчислюваних машин, чому й присвячений дана бакалаврська робота.

Створення програмної моделі має велике значення, як для навчального процесу, так і для створення складних систем, оскільки будь-яка потужна система базується саме на операціях низького рівня і розуміння цього процесу значно полегшує роботу зі складними системами, дії яких замасковані від користувача.

В процесі роботи над програмною моделлю необхідно виконати аналіз існуючих апаратних та програмних засобів. Також необхідно в повній мірі описати всі компоненти персонального комп'ютера, реалізувати основні зв'язуючи елементи та контролери пристроїв. Окрім цього, програмна модель

повинна включати в себе підрозділи глосарію, документації, посилання на виробників материнських плат та приклади організації портів вводу-виводу.

Мета й завдання дослідження. Метою роботи є програмне забезпечення системи моделювання взаємодії 64-х розрядних ОС з апаратними засобами ПК.

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

– Огляд існуючих систем моделювання взаємодії 64-х розрядних ОС з апаратними засобами ПК.

– Дослідження системи моделювання взаємодії 64-х розрядних ОС з апаратними засобами ПК.

– Програмна реалізація системи моделювання взаємодії 64-х розрядних ОС з апаратними засобами ПК.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі моделювання взаємодії 64-х розрядних ОС з апаратними засобами ПК.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи моделювання взаємодії 64-х розрядних ОС з апаратними засобами ПК, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Одним з важливих аспектів початкової стадії навчання студента правильно і цілком правильно зрозуміти будову персонального комп'ютера, без цих знань можливо неправильне тлумачення студентом роботи комп'ютера в цілому, що приводить до низького рівня знань майбутнього фахівця і неможливості написати повноцінні програмні продукти.

Існує трирівнева система навчання будови комп'ютера, а саме:

– користувальницький рівень. На даному рівні студент взнає загальні поняття і принципи роботи ЕОМ і термінологію застосовувану в даній спеціалізованій області.

– рівень операційної системи. Тут розглядаються питання заглибленого розгляду компонентів системи і їхня взаємодія між собою у певній операційній системі з використанням високорівневих мов програмування і усіляких віртуальних пристроїв.

– мікроархітектурний рівень. На останньому рівні студент познає, як відбувається взаємодія портів введення-виведення, також роботу електронної складової ЕОМ і елементи використовувані в операційних системах. Також студент учиться зневажати обмеженнями операційних систем.

Усі вище описані рівні в досить повному обсязі повинні бути описані у розробленому програмному продукту. Це розширює можливості застосування програми, не тільки для навчання студентів вузькоспеціалізованих спеціальностей, але і для навчання школярів, учнів професійно технічних училищ, а також інших навчальних закладів.

Сучасній людині важко представити своє життя без *електронно-обчислювальних машин* (ЕОМ). У цей час будь-якому бажаючому під силу зібрати

в себе на робочому столі повноцінний обчислювальний центр, ступінь функціональності якого може бути обмежена тільки фантазією й фінансовими можливостями його власника.

У мінімальній комплектації типовий персональний комп'ютер складається з наступних компонентів: системного блоку, монітора, клавіатури й інших периферійних пристроїв.

Розглянемо структурну схему типового сучасного персонального комп'ютера. Вона не претендує на безумовну точність і має на меті лише показати призначення, взаємозв'язок і типовий склад його елементів.

На рисунку 1.1 наведена функціональна схема системного блоку комп'ютера на базі процесорів сімейства Intel. На схемі представлені: центральний процесор, оперативна пам'ять, зовнішні пристрої. Усі компоненти з'єднані між собою через системну шину. Системна шина має додаткову шину – шину розширення. У комп'ютерах на базі Pentium як така шина використовується шина PCI (Peripheral Component Interface), до якої приєднуються зовнішні пристрої, а також шини більш ранніх стандартів, наприклад ISA (Industry Standard Architecture). На рисунку показана сама загальна схема серця комп'ютера – процесора. Основу процесора становлять блок мікропрограмного керування, виконавчий пристрій, позначений як «конвеєр», і регістри. Інші компоненти процесора виконують допоміжні функції. Більш докладний варіант цієї схеми ми розглянемо в третьому розділі.

У даному розділі буде розглянуте фундаментальне для розуміння логіки функціонування комп'ютера питання – архітектурні особливості процесора й основи його взаємодії з іншими компонентами комп'ютера. За основу узяті процесори фірми Intel. Необхідно відзначити, що ці процесори не є єдиними процесорами на ринку *апаратного забезпечення* (hardware), хоча й займають досить великий сегмент цього ринку. Історично склалося так, що архітектура процесорів Intel повністю або частково підтримується процесорами інших фірм.

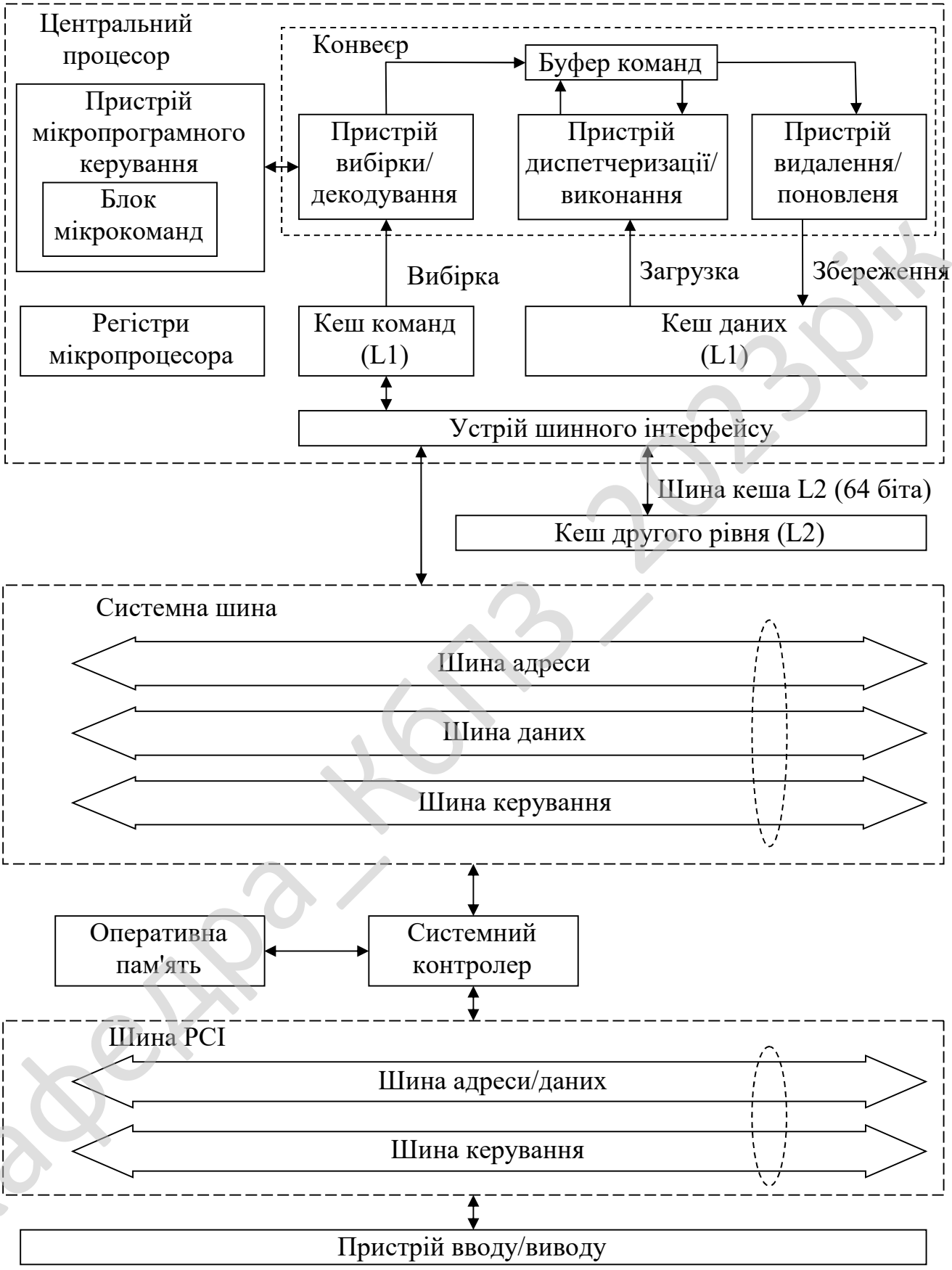


Рисунок 1.1 – Функціональна схема системного блоку комп'ютера на базі процесорів сімейства Intel

Тому процесорам фірми Intel доводиться ділити свій сегмент ринку із процесорами фірм AMD, VIA, Transmeta. В зв’язку з тим, що у своєму сегменті ринку (Intel-сумісних процесорів) процесори фірми Intel є стандартом де-факто, то в даній роботі мова буде йти винятково про ці процесори.

1.2 Область застосування

Види архітектури сучасних процесорів

Під архітектурою комп'ютера розуміється те, яким він представ перед програмістом або користувачем. У загальному випадку, архітектура включає опис форматів даних, системи команд, використовуваних методів адресації, організації пам'яті, системи введення, системи переривань, які визначають продуктивність комп'ютера.

Процесори є головним модулем будь-якого комп'ютера. Саме процесор виконує дії, наказані програмою по обчисленню результатів, управлінню зовнішніми пристроями, введенням і виведенням інформації та інші.

У свою чергу команда процесора визначає необхідні для виконання дії. Оскільки такими діями можуть бути виконання арифметичних і логічних операцій, операцій пересилок даних, управління потоком команд – команди умовних і безумовних переходів, введення і виведення інформації, виклику процедур і багато інших, то процесору потрібна система команд. Чим більше різноманітних команд в системі команд процесора, тим більші можливості він надає для програмування.

Оскільки основним призначенням процесора виконання обчислень, тому більшість команд містять поля кодами виконуваної операції і адресами операндів, над якими ці операції виконуються. Очевидно, що при виконанні операції процесор повинен вибрати за вказаними адресами один або два операнди, виконати вказану командою дію і помістити результат також за заданою

адресою. Кількість в коді команди операндів визначає адресність процесора. За цією ознакою розрізняють:

– безадресні процесори: адреси операндів і результату фіксованими.

– одноадресні процесори: адреса одного з операндів задається в коді команди, а другий операнд завжди знаходиться у фіксованому регістрі, названий «акумулятором». Результат операції зберігається в акумуляторі;

– двоадресні процесори: у коді команди адреси обох операндів. Результат виконання команди зберігається за адресою одного з операндів;

– триадресні процесори: у коді команди адреси операндів і адреса, по якій необхідно зберегти результат операції.

Очевидно, що триадресні процесори найбільш переважні з погляду надання більшої гнучкості по роботі з операндами і вимагають меншої кількості команд для виконання обчислень. У процесорах меншої адресності потрібні додаткові команди, щоб записати операнди у фіксовані регістри, заздалегідь зберегти один з операндів, якщо на його місце записується результат операції, а він буде потрібен в операціях. Проте збільшення адресності процесора до збільшення розрядності команди і ускладнення схем її декодування, збільшення устаткування процесора.

Операнди можуть знаходитися в регістрах процесора, в коді команди, в елементах оперативної пам'яті слідом за командою і в будь-яких інших довільних осередках. Способи місцезнаходження операндів називаються методами адресації.

Від 1 до 4 байт	1 або 2 байти	1 байт	1 байт	1 або 4 байти	1 або 4 байти
Префікс	Код операції	ModR/M	SIB	Зсув	Безпосередньо заданий операнд
		Режим адресації			

Рисунок 1.2 – Адресація операндів

Якщо операнд знаходиться в регістрі процесора, то в полі команди задається номер цього регістра. Якщо ж операнд знаходиться в елементі пам'яті, то можливі різноманітні способи завдання його адреси. Адреса може знаходитися в одному з регістрів процесора, в елементі оперативної пам'яті, обчислюватися процесором деякого базового значення або задаватися іншим способом. Чим більше способів адресації реалізовані в процесорі, тим більше ефективними є його команди і тим менша їх кількість потрібна для вирішення . Але, як і у випадку із збільшенням адресності, реалізація великої кількості методів адресації до ускладнення і збільшення устаткування процесора.

Відмічена раніше ефективність багатоадресних процесорів з великою кількістю команд і розвиненими методами адресації є достатньо умовною, частота використання окремих команд і методів адресації залежить від класу вирішуваних задач. Ускладнення ж апаратури до збільшення робочого такту процесора, тобто до зниження кількості команд, що виконуються за одиницю часу. Все це привело до появи двох різних процесорних архітектур:

– RISC (Reduced Instruction Set Computer) – процесори з скороченим набором команд або "регістрово-насичений процесори, орієнтований на операції типу load/store". Область їх застосування дуже широка – від мікроконтролерів до суперкомп'ютерів. Основними рисами RISC-архітектури є: однакова довжина команд, однаковий формат команд – код команди регістр-приймач два регістри-джерела, команди виконують лише прості дії, конвеєри, виконання команди не довше, ніж за один такт, проста адресація. Обмежений набір команд і прості методи адресації дозволяють скоротити кількість необхідного процесора, спростити структуру процесорів і підвищити робочу тактову частоту, що компенсує збільшення довжини їх програм. Архітектурно RISC процесори міститять дуже велика кількість регістрів від декількох сотень до 1000, ці процесори орієнтовані на операції типу "регістр-пам'ять" або "пам'ять-пам'ять": всі операнди, необхідні для виконання тієї або іншої команди, повинні бути обов'язково розміщені в робочих регістрах процесора. Недоліком RISC

архітектури є незручність програмування на низькому рівні і складність управління розподілом великою кількістю регістрів для розробників компіляторів.

– CISC (Complete Instruction Set Computer) – процесори з повним набором команд. Дані процесори містять розвинені системи команд з складними системами адресації. CISC-процесори є антиподом RISC і використовують "регистрово-ненасичену архітектуру", здатну рівноцінно використовувати регістри і оперативну пам'ять. Така архітектура не дозволяє створити "дуже швидку" систему команд, за один такт такі завдання не виконуються. Зате розробникам компіляторів легше -як не збільшуй кількість регістрів в процесорі, в ОЗП все одно більше.

Сучасні процесори Intel і AMD можна віднести більше до архітектури CISC але їх можна так само віднести і до RISC архітектури, дані процесори містять транслятор x86 команд в RISC набір інструкцій, що виконуються RISC-ядром процесора. Такий підхід виявився дуже перспективним. Дійсно, виконувати код краще на простому і високошвидкісному RISC-ядрі, а ось програмувати на високорівневій CISC-мові і плюс зберігається зворотна сумісність з попередніми поколіннями процесорів і програмним забезпеченням.

Крім того, існує й інші архітектури процесорів:

– VLIW (Very Long Instruction Word) – архітектура процесора з командними словами дуже великої довжини;

– EPIC (Explicitly Parallel Instruction Computing) – обчислення із заданим паралелізмом команд.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи моделювання взаємодії 64-х розрядних ОС з апаратними засобами ПК, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

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

На сьогоднішній день, існує ринок висококваліфікованих фахівців, але все одно кадрові агентства шукають по всій країні вузькоспеціалізованих комп'ютерних фахівців за замовленням приватних підприємців і комп'ютерних фірм. На даний час не існує навчальних систем на основі програмної моделі персонального комп'ютера, які могли б забезпечити здатність навчити висококваліфікованого фахівця. Хоча не виключене існування комерційних проектів, закордонних аналогів такої системи але з закритим вихідним кодом і з вузьконаправленим ухилом у певний компонент ПЕОМ. Існують програми орієнтовані на розробку програмної продукції і наступного тестування її, але навчання в таких системах неможливо.

В процесі навчання у вищих навчальних закладах виникає гостра потреба в добре організованих і детально розроблених навчальних системах.

У даній бакалаврській роботі, необхідно реалізувати навчальну систему на основі програмної моделі побудови персонального комп'ютера.

Для цього розглянемо архітектуру ПЕОМ. При виборі конкретного типу (або моделі) комп'ютера неминуче виникають питання: як оцінити його можливості, які його відмінності від комп'ютерів інших типів (моделей)? Розгляду однієї лише його функціональної схеми явно недостатньо, тому що вона навряд чи чимось принципово відрізняється від схем інших машин. У всіх комп'ютерів є оперативна пам'ять, процесор, зовнішні пристрої. Вся справа в процесорі. Саме він задає ті правила, по яких в остаточному підсумку всі пристрої комп'ютера функціонують як єдиний механізм. Для всього, що

характеризує комп'ютер з погляду його функціональних програмно-керованих властивостей, існує спеціальний широко розповсюджений термін – *архітектура* ЕОМ. Слід зазначити, що більшість складових цих термінів понять відносяться саме до процесора. Архітектура ЕОМ – це абстрактне подання ЕОМ, що відбиває її структурну, схемотехнічну й логічну організацію. Поняття архітектури ЕОМ є комплексним, у нього входять:

- структурна схема ЕОМ;
- засоби й способи доступу до елементів структурної схеми ЕОМ;
- організація й розрядність інтерфейсів ЕОМ;
- набір і доступність регістрів;
- організація й способи адресації пам'яті;
- способи подання й формати даних ЕОМ;
- набір машинних команд ЕОМ;
- формати машинних команд;
- правила обробки позаштатних ситуацій (переривань).

Таким чином, опис архітектури містить у собі практично всю необхідну для програміста інформацію про комп'ютер. Поняття архітектури ЕОМ – ієрархічне.

Всі сучасні комп'ютери мають деякі загальні й індивідуальні архітектурні властивості. Індивідуальні властивості властиві тільки конкретній моделі комп'ютера. Загальні архітектурні властивості, навпаки, властиві деякій, часто досить великій групі комп'ютерів. На сьогоднішній день загальні архітектурні властивості більшості сучасних комп'ютерів підпадають під поняття *фон-неймановської архітектури.* Фон Нейман запропонував схему ЕОМ із програмою в пам'яті й двійковою логікою замість десяткової. Логічно машину фон Неймана складали п'ять блоків: оперативна пам'ять, арифметико-логічний пристрій (АЛП) з акумулятором, блок керування, пристрої вводу й виводу. Особливо варто виділити роль акумулятора. Фізично він являє собою регістр АЛП. Для процесорів Intel, у яких більшість команд – двооперандні, його роль не настільки очевидна. Але існували й існують процесорні середовища з однооперандними

машинними командами. У них наявність акумулятора відіграє ключову роль, в зв'язку з тим, що більшість команд використовують його вміст у якості або другого, або єдиного операнда команди.

Нижче описані властивості й принципи роботи машини фон Неймана.

Лінійний простір пам'яті. Для оперативного зберігання інформації комп'ютер має сукупність осередків з послідовною нумерацією (адресами) 0, 1, 2,... Дана сукупність осередків називається *оперативною пам'яттю.*

Принцип збереженої програми. Відповідно до цього принципу, код програми і її даних перебувають у одному й тому ж самому адресному просторі оперативної пам'яті.

Принцип мікропрограмування. Суть цього принципу полягає в тому, що машинна мова ще не є тією кінцевою субстанцією, що фізично пускає в хід процеси в машині. До складу процесора входить *пристрій мікропрограмного керування, що* підтримує набір дій-сигналів, які потрібно згенерувати для фізичного виконання кожної машинної команди.

Послідовне виконання програм. Процесор вибирає з пам'яті команди строго послідовно. Для зміни прямолінійного ходу виконання програми або здійснення розгалуження необхідно використовувати спеціальні команди. Вони називаються командами *умовного* й *безумовного* переходів.

Відсутність різниці між даними й командами в пам'яті. З погляду процесора, немає принципової різниці між даними й командами. Дані й машинні команди перебувають в одному просторі пам'яті у вигляді послідовності нулів і одиниць. Ця властивість пов'язана з попередньою. Процесор, по черзі обробляючи деякі комірки пам'яті, завжди намагається трактувати вміст осередків як коди машинних команд, а якщо це не так, то відбувається аварійне завершення програми. Тому важливо завжди чітко розділяти в програмі простори даних і команд.

Байдужність до призначення даних. Машині однаково, яку логічне навантаження несуть оброблювані нею дані.

Як приклад *індивідуальних* архітектурних принципів комп'ютера, у силу ієрархичності цього поняття, розглянемо індивідуальні архітектурні принципи й властивості процесорів Intel. Їхній повний розгляд не є нашою метою, приділимо увагу лише тим, які найбільш характерні й знадобляться нам для подальшого викладу.

Відповідно до матеріалів фірми Intel, індивідуальні архітектурні властивості й принципи роботи всіх її процесорів, починаючи з i8086 і закінчуючи Pentium IV, вибудувані в рамках єдиної архітектури, що пізніше одержала назву IA (Intel Architecture). Ця архітектура не є «закостенілим» утворенням. У процесі еволюції процесорів Intel вона постійно змінюється й розвивається. Кожний із процесорів вносив в IA одне або кілька архітектурних нововведень.

Конвеєрне виконання команд

Для підвищення продуктивності процесорів використовуються різні структурні методи, найбільш поширеними яких нарощування тактової частоти (збільшення кількості виконаних операцій за одиницю часу) і збільшення ступеня паралелізму процесора (дозволяє виконувати більше операцій протягом кожного періоду тактової частоти).

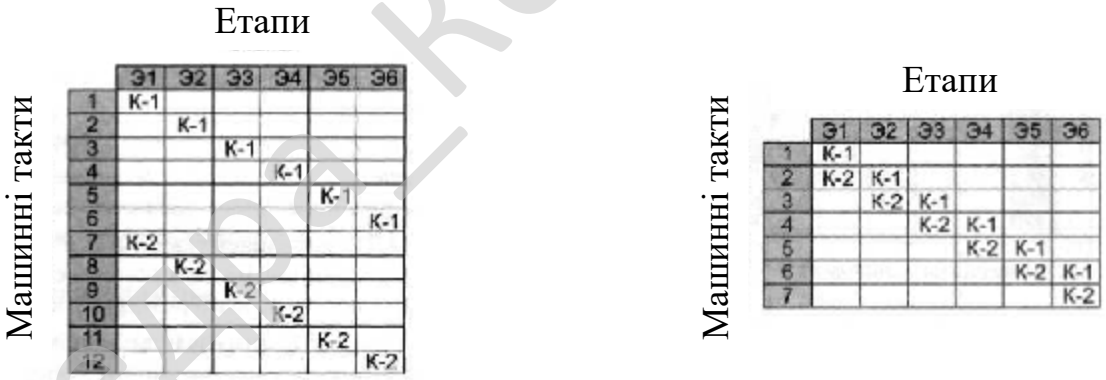
Під паралельною роботою розуміється виконання в одному робочому такті процесора декількох операцій, наприклад, робота АЛП і вибірка операнда з оперативної пам'яті, виконання операції введення або , обчислення адреси команди або операнда та інші.

Конвеєрне виконання команд, також зване виконання операцій, є окремим випадком паралельної роботи вузлів процесора. Воно засноване на тому, що, в більшості випадків, виконувані процесором команди знаходяться в послідовних елементах пам'яті. Виконання процесором простої команди можна розбити на п'ять етапів: вибірка команди з пам'яті, її декодування, вибірка операндів, виконання операції і збереження результату. Оскільки ці п'ять етапів виконання команди підтримуються різними вузлами процесора, то можливо виконання

послідовності п'яти команд. При цьому в одному такті процесора записується результат першої команди послідовності, виконується операція, задана другою командою, вибираються операнди для третьої команди, декодується четверта і вибирається з пам'яті п'ята команда.

По черговому тактовому імпульсу кожна команда в конвеєрі пересувається на наступний р; виконана команда покидає конвеєр, а нова вступає в нього. Таким чином, при виконанні послідовності коротких команд продуктивність процесора збільшується в п'ять разів.

Насправді одержати такий ступінь підвищення продуктивності процесора не вдається через те, що в програмах існують команди умовних і безумовних переходів, викликів процедур, а в конвеєрі можливі конфліктні ситуації, що змінюють гладку послідовність виконання команд. Очевидно, що щоб збільшити м паралелізму, треба використовувати по можливості більше число , наскільки дозволить технологія, але із збільшенням рівнів операцій кількість чинників, що порушують роботу конвеєра команд, збільшується і чином ускладнюється схема управління процесором.



Шестиступінчастий цикл виконання команди в процесорі, що не підтримує режим конвеєрної обробки

Шестиступінчастий цикл виконання команди в процесорі, що підтримує режим конвеєрної обробки

Рисунок 2.1 – Шестиступінчастий цикл виконання команди в процесорі, що не підтримує режим конвеєрної обробки та підтримує його

Конвеєрна обробка команд, була вперше реалізована в процесорах I80486. У ранніх моделях застосовувався стандартний п'ятиступінчастий конвеєр. У сучасних процесорах Intel і AMD використовується суперконвеєрна архітектура (що містить 10 або 12 сходинок (Athlon XP(64)) і 20 або 31 сходинка (Pentium IV)). При суперконвейризації стандартного конвеєра діляться на дрібніші частини. Із збільшенням числа кожен окремий виконує меншу роботу і, отже, містить менше апаратної логіки. Зменшення складності логіки скорочує затримку розповсюдження (Propagation Delay) – часовий інтервал між надходженням набору вхідних дій на входи схеми і появою результуючих сигналів на її виходах. Завдяки коротким затримкам розповсюдження стає можливим підвищення частоти. Проте, суперконвеєрна архітектура має серйозний недолік: команди, які процесор очищати свої конвеєри, а до їх числа входять неправильно передбачені переходи і деякі інші операції можуть продуктивність.

У зв'язку з тим, що процесори Athlon 64 мають довжину конвеєра 12 сходинок (Pentium4 довжина конвеєра 31 сходинка) максимальна тактова частота (на даний момент) для Athlon 64 складає 2,8ГГц (Pentium4 – 4 ГГц) хоча продуктивність Athlon 64 вище ніж в Pentium IV в переважному числі додатків. Але оскільки, основна маса користувачів як «мірило» продуктивності використовує тактову частоту, то компанія AMD маркірувати свої процесори не реальною тактовою частотою а рейтингом продуктивності, на порівнянний по продуктивності Pentium IV. Наприклад Pentium IV 3200 Мгц (реальна тактова частота) і Athlon 64 3200+ (реальна тактова частота 2000 Мгц.)

Для пояснення відмінностей в продуктивності процесорів різних класів, але що працюють на однаковій частоті, AMD ввела поняття QuantiSpeed Architecture. Коротко, сенс цього терміну полягає в тому, що процесори Athlon XP або Athlon 64, що мають QuantiSpeed Architecture, виконують в середньому більше інструкцій за такт, ніж процесори сімейства Pentium IV. Втім, це недавно: NetBurst архітектура, що використовується в Pentium IV, оптимізована в першу

чергу на обробку потокових даних лінійними алгоритмами. Необхідно відзначити, що одним з ключових моментів NetBurst архітектури є наддовгий конвеєр обробки команд, завдяки якому процесори Pentium IV можуть працювати на відносно високих частотах. Проте, довгий конвеєр має і зворотну сторону. У разі неминучих помилок в переходів процесор вимушений скидати весь конвеєр повністю і декодувати іншу гілку програми . В результаті, хоч частоти Pentium IV і виглядають переконливо, продуктивність їх не так вже і велика.

Тобто, оскільки продуктивність процесора визначається як частотою, так і середнім числом інструкцій, що виконується їм за такт, а це число інструкцій у Athlon однозначне вище, гігагерци від AMD і від Intel набувають абсолютно різного значення.

Обгрунтування вибору принципу розробки і методики побудови системи

Коли студент починає вивчати внутрішню будову персонального комп'ютера, він стикається з такими темами:

– побудова шини PCI (Peripheral Component Interconnect). Шина з'єднання периферійних компонентів. Займає особливе місце в PC – архітектурі, і є мостом між локальною шиною процесора і шиною введення-виведення ISA/EISA;

– побудова шини ISA. Являє собою параллельну шину, створену на базі шини пам'яті і введення/виведення IBM PC/AT;

– робота контролера клавіатури. Взаємодія з клавіатурою в PC AT базується на двох мікропроцесорах Intel 8042. Один мікропроцесор знаходиться в системному блоці, другий – в клавіатурі. В залежності від фірми виробника мікроконтролер клавіатури може допрацьовуватися й удосконалитися з 100% сумісністю з базовими командами Intel 8042;

– робота контролера прямого доступу в пам'ять. Прямий доступ до пам'яті (DMA), це метод безпосереднього звертання до пам'яті, минаючи процесор. Процесор відповідає тільки за програмування DMA: настроювання на визначений тип передачі, завдання початкової адреси і розміру масиву обмінюваних даних.

Звичайно DMA використовується для обміну масивами даних між системною пам'яттю і пристроями введення-виведення;

– робота контролера переривань. В архітектурі PC AT підсистема апаратних переривань складається з двох контролерів 8259A (головного MASTER і підлеглого – SLAVE). Вони об'єднані таким чином, що можуть обслужити 15 запитів на переривання;

– сигнали шини PCI і ISA;

– послідовний інтерфейс. Послідовний інтерфейс RS-232 (вітчизняний аналог КР580ВВ51) обумовлений стандартом EIA і рекомендаціями V.24 CCITT для передачи даних в одному напрямку використовує одну сигнальну лінію, по якій інформаційні біти передаються один за другим – послідовно;

– параллельний інтерфейс. Параллельний порт використовується для підключення принтера до комп'ютера. Також принтер можливо підключити через асинхронний адаптер (старі моделі матричних принтерів) і USB порт (практично всі нові принтери);

– контролер USB (Universal Serial Bus – універсальна послідовна шина). Це швидкий, двонаправлений, ізохронний, дешевий, послідовний інтерфейс, що приєднується динамічно, що буде сумісний з вимогами платформи PC сьогодні і завтра;

– структура системного таймера. Архітектура PC AT використовує підсистему триканального 16-розрядного таймера 8254 як системний таймер. Програмувальний таймер призначений для одержання програмно-керованих тимчасових затримок і генерації время’задаючих функцій. Таймер дозволяє підвищити ефективність програмування процесів керування і синхронізації зовнішніх пристроїв, особливо в реальному масштабі часу;

– доступ до енергонезалежної пам'яті і годинника реального часу. Основне призначення енергонезалежної пам'яті – збереження найбільш важливих параметрів системи при відключенні живлення комп'ютера. Годинник RTC використовуються для встановлення значення поточного часу в момент

ініціалізації системного таймера персонального комп'ютера, а ячейки енергонезалежної пам'яті зберігають параметри поточної конфігурації системи;

– робота контролера гнучких дисків (Floppy Disk Controller, FDC). Керування гнучкими дисками в PC-подібних ПЕОМ здійснюється мікросхемою 8272A фірми Intel, що виконує функції контролера гнучких дисків;

– робота контролера твердих дисків з інтерфейсами IDE, EIDE і SCSI;

– робота з Plug and Play пристроями;

– робота AGP. Спеціалізована надбудова над шиною PCI, що дозволяє створити швидкісний канал обміну даними між графічним акселератором і системною логікою PC;

– розуміння роботи арбітра;

– робота з підсистемою пам'яті;

– складності побудови й організації взаємодії з процесором.

Усі вищенаведені теми розрізнені між собою. Зрозуміло, що буде дуже складно розібратися у всіх темах докладно, тим більше зрозуміти внутрішню організацію контролерів і пристроїв і їх взаємодію між один одним. Тому не виключене неправильне розуміння основних вузлів комп'ютера, що позначається на рівні знань майбутнього фахівця. Автор вважає, що найкраще подання навчального матеріалу студенту, повинно здійснюватись за допомогою строгої типирізації інформації і розбивку її на розділи та підрозділи.

Звідси випливає, що створення моделі комп'ютера в цілому з розбивкою досліджуваних тем на окремі блоки – найкращий варіант створення навчальної програми.

Реалізуючи таку систему, є можливість наочно показати взаємодію контролерів, пам'яті й інтерфейсів персонального комп'ютера, що так часто є головним питанням у розумінні роботи системи в цілому.

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 першим (бакалаврським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи моделювання взаємодії 64-х розрядних ОС з апаратними засобами ПК.

В процесі розробки випускної кваліфікаційної роботи за першим (бакалаврським) 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

Опис архітектурної будови 64-х розрядної ПЕОМ

Архітектура процесора Pentium IV

Архітектура 6-го покоління представлена в 1995 в процесорі Pentium Pro була основою для всіх процесорів Intel: такі як Pentium II, Celeron, Pentium III. У основі Pentium IV лежить нова архітектура 7 покоління, яка має назву Intel NetBurst architecture.

Перерахуємо основні відмінності архітектури NetBurst від архітектури 6 покоління:

– Hyper Pipelined Technology (гіперконвеєрна технологія). Конвеєр Pentium IV має дуже велику глибину – 20 (31) стадій. Для порівняння – довжина конвеєра Pentium III складає 10 стадій. При подовженні конвеєра з'являється можливість декомпозиції виконання кожної команди на дрібніші етапи, кожний з цих етапів тепер може виконуватися швидше, що дозволяє практично безперешкодно збільшувати частоту процесора. Наприклад при технологічному процесі 0.18 мкм гранична частота для Pentium III складає 1 ГГц а Pentium IV досяг частоти 2 ГГц. У надмірно довгого конвеєра є свої недоліки. Перший недолік – кожна команда, проходячи більше число стадій, виконується довше. Другий недолік виявляється при помилках в переходів. Як і будь-який сучасний процесор, Pentium IV може виконувати інструкції не тільки послідовно, але і паралельно, відповідно не завжди в тому порядку, як вони слідують в програмі і, не знаючи напряму умовних переходів. Для того, щоб вибирати в таких випадках гілки програми для подальшого виконання, процесор прогнозує результати виконання умовних переходів на підставі накопиченої статистики. О, блок переходів все ж таки помиляється, і в цьому випадку доводиться повністю

очищати конвеєр, зводячи нанівець всю заздалегідь виконану процесором роботу по виконанню не тієї гілки в програмі. Природно, при довшому конвеєрі, на нове заповнення конвеєра йде більше процесорних тактів, а отже і часу.

– Trace Cache. Для кешування декодованих інструкцій в Pentium IV використовується спеціальний кеш. Замість звичайного L1 кеша, який в Pentium III був роздільний на область інструкцій і область даних в Pentium IV застосований новий підхід. Інструкції в L1 кеші не зберігаються, він призначений тепер для даних. Для кешування інструкцій тепер використовується Trace Cache, в порівнянні із звичайним L1-кешем він має багато переваг, на мінімізацію простоїв процесора при виконанні неправильних переходів. Основне в Trace Cache те, що там зберігаються вже декодовані інструкції. Це означає, що в ньому зберігаються не класичні x86 інструкції, а так звані мікрокоманди, це прості операції, якими безпосередньо оперує процесорне ядро. Збереження в TraceCache мікрооперацій дозволяє уникнути повторного декодування x86 інструкцій при повторному виконанні тієї ж ділянки програми або при неправильному переходів. Intel не розкриває розмірів свого Trace Cache в кілобайтах, , відомо, що в ньому може бути збережено до 12000 мікрооперацій.

– Rapid Execute Engine. Найбільш проста частина сучасного процесора – це ALU (арифметико-логічний пристрій). Intel визнав можливим збільшити його тактову частоту усередині Pentium IVпо відношенню до самого процесора. Таким чином, наприклад, в 3.8 ГТц Pentium IV ALU працює на частоті 7.6 ГГц. У ALU виконуються прості цілочисельні інструкції, тому, продуктивність процесора при операціях цілими числами повинна бути дуже високою. Наприклад, на виконання однієї інструкції типу add Pentium IV 3.8 ГГц витрачає всього 0.25нс, тоді як виконання цієї командиPentium III 1 ГГц йде 1 нс.

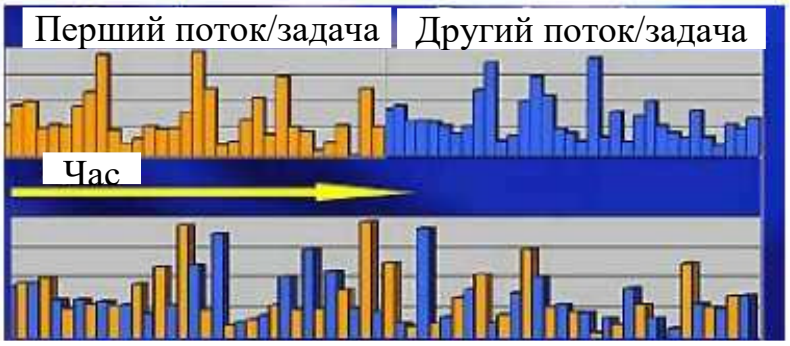
– SSE2, SSE3. Розширений набір інструкцій для обробки потокових і медіа даних 144+7? інструкцій.

– L2 кеш. Збільшено розмір кешу і пропускна спроможність (до 120 Гбайт/сек).

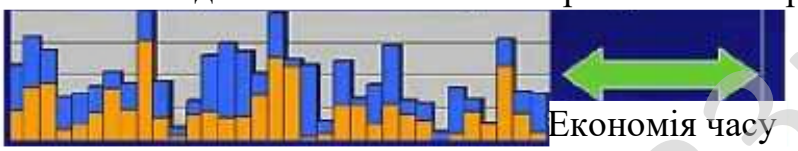
– Системна шина. Процесор Pentium IV використовує нову Quad Pumped процесорну шину, що працює з частотою 400 (533, 800, 1066) Мгц. Фізично, реалізується нова системна шина шляхом множення на 4 в контролерах процесорної шини чіпсета і процесора тактової частоти, яка дляPentium IV складає 100 (133, 200, 266) Мгц. Тобто, на збільшеній учетверо частоті працює ділянка між процесором і чіпсетом.

– Технологія Hyper-Threading (Гіперпотокова обробка). Архітектура Pentium IV дозволяє процесору виконувати максимум шість інструкцій за такт дякуючи тому що процесор містить чотири паралельні блоки для операцій з цілими числами, два блоки для роботи з дійсними числами і два блоки для роботи з пам'яттю, одночасно всі ці ресурси практично ніколи не використовуються більшість x86 коду не може виконуватися паралельно на рівні інструкцій. Для того, щоб ефективніше використовувати масив виконавчих блоків, була технологія Hyper-Threading. Технологія Hyper-Threading може працювати тільки в багатозадачних операційних системах, ідеологія яких побудована на одночасному існуванні декількох обчислювальних потоків (threads), що відносяться до одного або різних активних додатків, або до операційної системи. Якщо на багатопроцесорних системах ці потоки можуть виконуватися одночасно (поодинці на процесор), то в одно процесорних системах CPU вимушені безперервно між потоками, квантуючи процесорний час між виконанням їх різних частин.

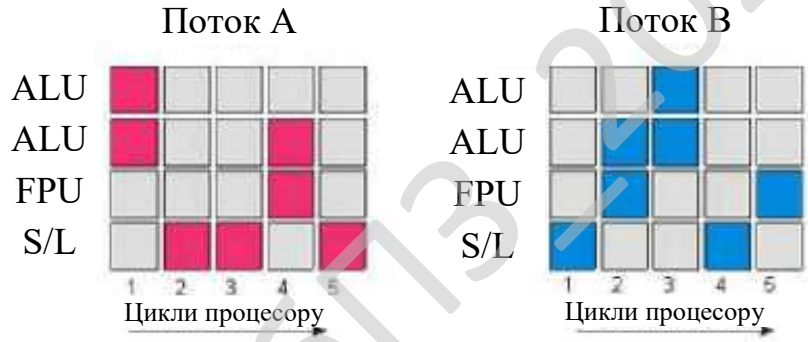
Таким чином, якщо дозволити процесору одночасне виконання більш ніж одного потоку, його потужності можна завантажити ефективніше. Саме в цьому і полягає основна ідея Hyper-Threading. Завдяки технології Hyper-Threading один фізичний процесор сприймається операційною системою і застосуваннями як два логічні процесори. Кожен логічний процесор має свій власний контролер переривань. Переривання посилаються окремому логічному процесору і обробляються л цим логічним процесором. Сам же процесор використовує для виконання другого потоку свої простоюючи ресурси.



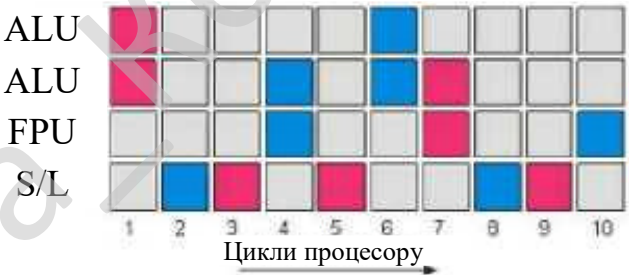
Обидва потоки/задачі без технології гіперпотокової обробки



Обидва потоки/задачі з технологією гіперпотокової обробки



Виконання на процесорі без гіперпотокової обробки



Виконання на процесорі з гіперпотоковою обробкою

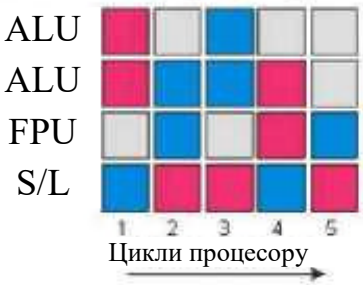


Рисунок 3.1 – Технологія Hyper-Threading (Гіперпотокова обробки)

Для реалізації технології Hyper-Threading були про дубльовані регістри загального призначення і регістри, управляючі регістри, вдосконалений програмований контролер переривань (Advanced Programmable Interrupt Controller -APIC) і деякі службові регістри, решта ресурсів, включаючи кеши, виконавчі блоки, логіку і т.п. логічні процесори розділяють один з одним.

З моменту появи першого Pentium IV (у листопаді 2000 г) встигло змінитися декілька ядер:

– Pentium IV (Willamette) 0,18 мкм, 42 млн. транзисторів, платформа Socket 423 або Socket 478,тактові частоти 1,3-2,0 ГГц, частота системної шини 400МГц (100 Мгц фіз.) L1 кеш 8 Кбайт(дані) + 12000 інструкцій L2 кеш 256 Кбайт, шина що зв'язує L1 і L2 256 біт, підтримка інструкцій SSE і SSE2.

– Pentium IV на ядрі Northwood (січень 2002 р.) 0,13 мкм, 55 млн. транзисторів, платформаSocket 478, тактові частоти 1,6-3,2 ГГц, частота системної шини 400, 533, 800 Мгц L1 кеш 8Кбайт (дані) + 12000 інструкцій L2 кеш 512 Кбайт, шина що зв'язує L1 і L2 256 біт, підтримка інструкцій SSE і SSE2. Пізніше в моделях з 800 Мгц системною шиною з'явилася підтримка технології Hyper-Threading.

– Pentium IV на ядрі Prescott (лютий 2004 р.) 0,09 мкм, 125 млн. транзисторів, платформа Socket478 і Socket 775, тактові частоти 2,53-3,8 ГГц, частота системної шини 533, 800 Мгц L1 кеш 16Кбайт (дані) + 12000 інструкцій L2 кеш 1024 Кбайт, шина що зв'язує L1 і L2 256 біт, підтримка інструкцій SSE SSE2 і SSE3.

– У лютому 2005 р. з'явилося ядро Prescott 2M, з L2 кешем 2 Мбайта і підтримкою 64 бітових інструкцій і технологій, на оптимізацію енергоспоживання, захисту від вірусів.

– Pentium D на ядрі Smithfield (травень 2005 г). Перший двох ядерний процесор компанії Intel. 0,09 мкм, 230 млн. транзисторів, платформа Socket 775, тактові частоти 2,8-3,4 ГГц, частота системної шини 800 Мгц L1 кеш 2x16 Кбайт (дані) + 2x12000 інструкцій L2 кеш 2x1024 Кбайт, шина що зв'язує L1 і L2 256

біт, підтримка інструкцій SSE SSE2 і SSE3 і всіх можливих технологій.

– У січні 2006 р. Pentium IV на ядрі Cedar Mill (одноядерний) і Pentium D на ядрі Presler (двоядерний). Виготовлені по нормах 0,065 мкм техпроцесу, і практично не чим не відрізняються від ядер Prescott 2M і Smithfield.

Паралельно з випуском процесорів Pentium IV на різних ядрах, проводився випуск процесорів Celeron, Pentium IV Extreme Edition і Pentium Extreme Edition. Що відрізняються від Pentium IV частотою системної шини і розміром Кеша L2 і наявністю Кеша L3 у деяких моделях Pentium IV Extreme Edition.

Структурна схема процесора Pentium IV показана на рисунку 3.2.

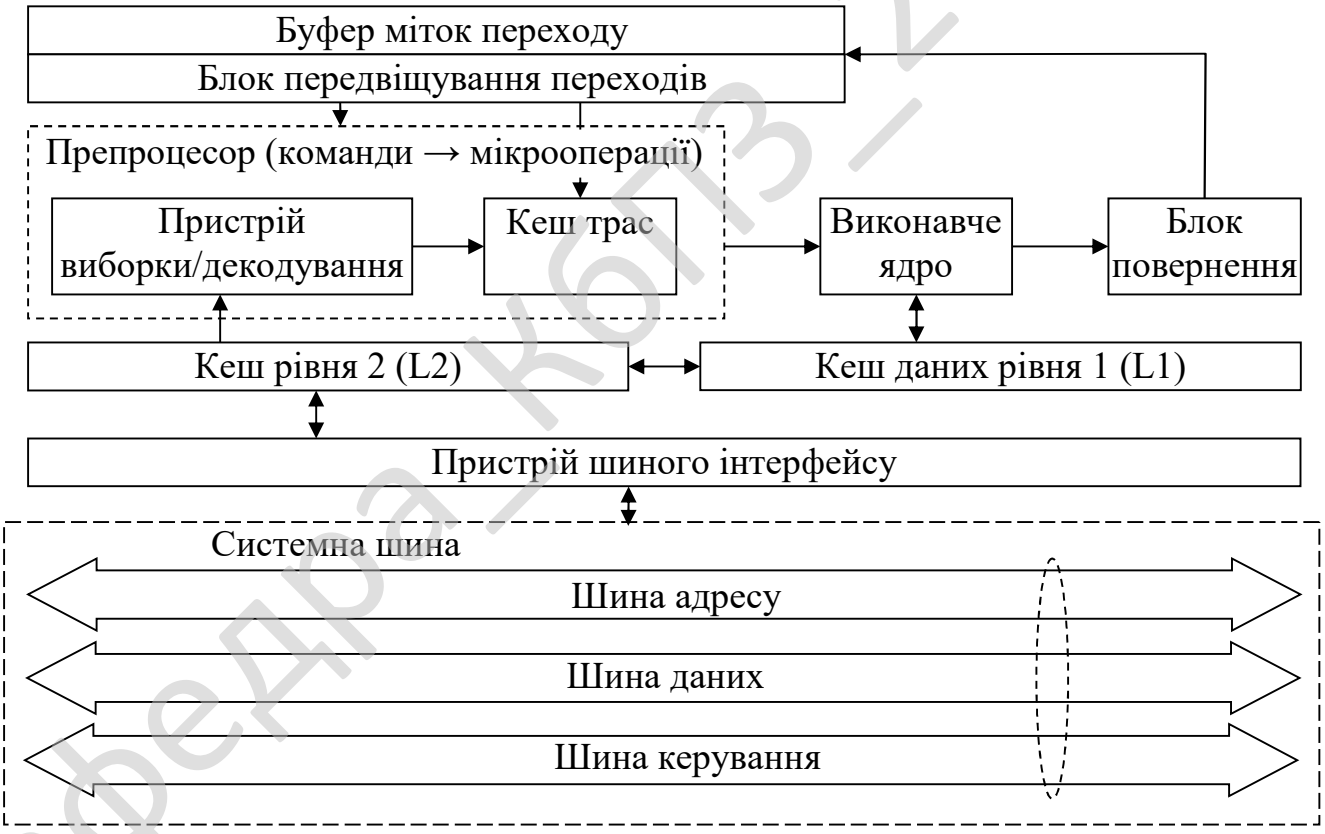


Рисунок 3.2 – Структурна схема процесора Pentium IV

Розглянувши структурну схему процесора перейдемо до розгляду структурної схеми ПЕОМ. При розробці структурної схеми відштовхнемося від внутрішньої архітектурної будови персонального комп'ютера.

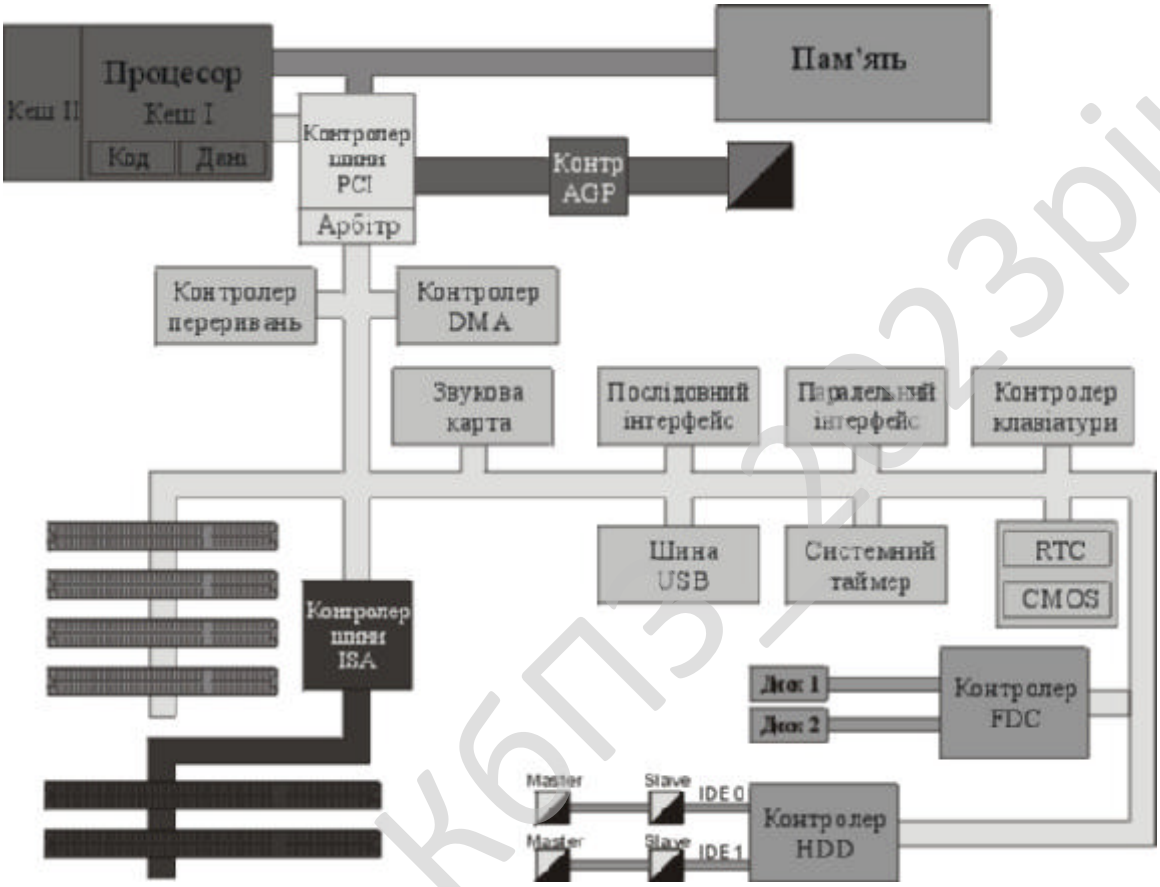


Рисунок 3.3 – Структурна схема персонального комп'ютера

На рисунку 3.3 відображена структурна схема персонального комп'ютера на якій відображені такі важливі елементи:

– Шина PCI (Peripheral Component Interconnect) – шина з'єднання периферійних компонентів, займає особливе місце в PC-архітектурі, є містком між локальною шиною процесора і шиною введення-виведення ISA/EISA чи MCA;

– Шина ISA – являє собою паралельну шину, створену на базі шини пам'яті і введення/виведення IBM PC/AT;

– контролер клавіатури – взаємодія з клавіатурою в PC AT базується на

двох мікропроцесорах Intel 8042. Один мікропроцесор знаходиться в системному блоці, другий – в клавіатурі.

– контролера прямого доступу в пам'ять – прямий доступ до пам'яті (DMA) це метод безпосереднього звертання до пам'яті, минаючи процесор. Процесор відповідає тільки за програмування DMA: настроювання на визначений тип передачі, завдання початкової адреси і розміру масиву даних;

– контролер переривань – в архітектурі PC AT підсистема апаратних переривань складається з двох контролерів 8259A. Вони об'єднані таким чином, що можуть обслужити 15 запитів на переривання;

– послідовний інтерфейс – послідовний інтерфейс RS-232 (КР580ВВ51) обумовлений стандартом EIA для передачі даних в одному напрямку використовує одну сигнальну лінію, по якій інформаційні біти передаються один за одним – послідовно;

– паралельний інтерфейс – паралельний порт використовується для підключення принтера до комп'ютера. Також принтер можливо підключити через асинхронний адаптер і USB порт;

– контролер USB –швидкий, двонаправлений, ізохронний, дешевий, послідовний інтерфейс;

– системний таймер – архітектура PC AT використовує підсистему триканального 16-розрядного таймера 8254, як системний таймер. Програмувальний таймер призначений для одержання програмно-керованих тимчасових затримок і генерацій время’задаючих функцій. Таймер дозволяє підвищити ефективність програмування процесів керування і синхронізації зовнішніх пристроїв, особливо в реальному масштабі часу;

– енергонезалежна пам'ять і годинник реального часу – основне призначення енергонезалежної пам'яті – збереження найбільш важливих параметрів системи при відключенні живлення комп'ютера. Годинник RTC використовуються для установки значення поточного часу в момент ініціалізації системного таймера персонального комп'ютера;

– контролер гнучких дисків (Floppy Disk Controller, FDC) – Керування гнучкими дисками здійснюється мікросхемою (контролером) 8272A;

– контролер твердих дисків з інтерфейсами IDE, EIDE і SCSI;

– робота AGP – спеціалізована надбудова над шиною PCI, що дозволяє створити швидкісний канал обміну даними між графічним акселератором і системною логікою PC.

При розробці функціанальної схеми була розглянута внутрішня побудова компьютера, позначено основні функціональні блоки програми і внесені на схему, також були розглянуті взаємодії між функціональними блоками і зроблені відповідні зміни на схемі.

Після наповнення схеми основними функціональними блоками програми і нанесення взаємодій, був зроблений аналіз схеми на правильну організацію усіх взаємодій і внесення остаточних змін.

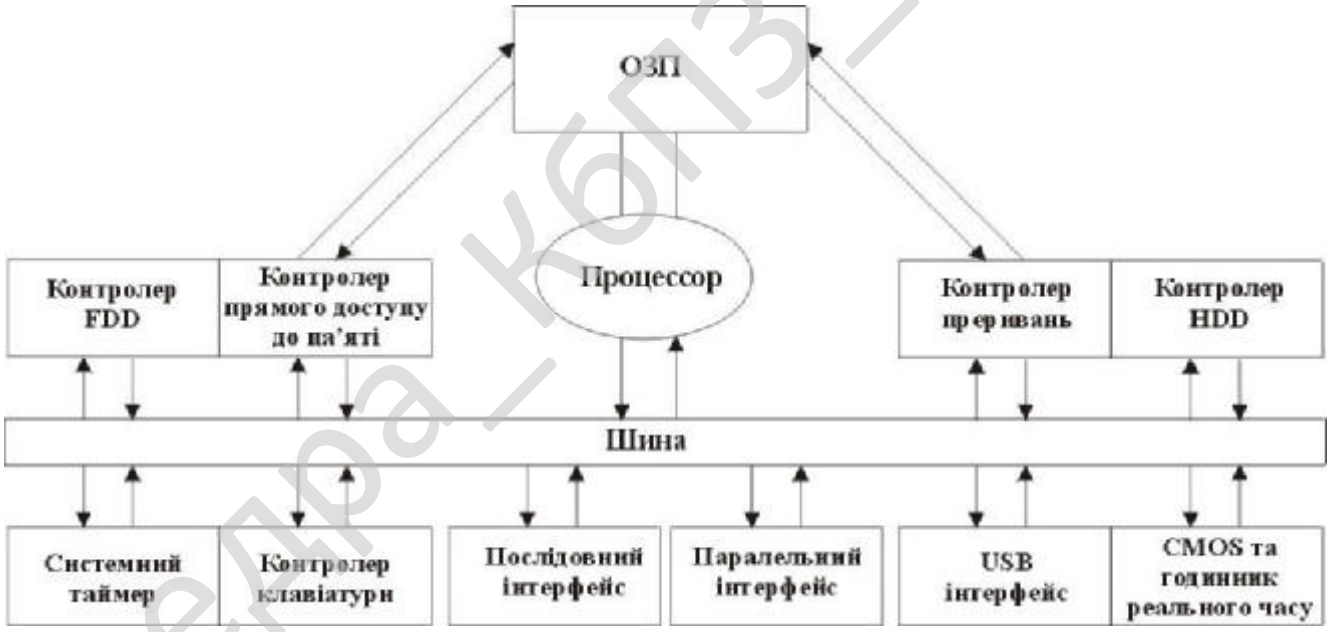


Рисунок 3.4 – Функціональна схема ПЕОМ

Архітектурно комп'ютер являє собою комплекс мікроконтролерів і мікросхем які взаємодіють між собою певним чином.

Фізично складові ПК можна розділити на компоненти системного блоку і компоненти блоку розширення. Всі основні плати, що входять до складу будь-якої моделі ПК, розміщаються у великому блоці, що одержав назву системного. Системний блок включає всі необхідні компоненти, що дозволяють комп'ютеру працювати без будь-яких доповнень.

Будь-які додаткові пристрої підключаються до ПК за допомогою одного з роз’ємів розширення, які залежать від відповідного типу материнської плати. Ці лінії дозволяють передавати всі сигнали, необхідні для керування будь-яким устаткуванням, що може бути підключене до ПК. Любий сигнал, що посилається одному з блоків розширення, передається і всім іншим, оскільки вони підключені до паралельних ліній. Тут має місце розширення ідеї загальної шини даних: усі блоки розширення використовують загальне з'єднання, назване каналом введення/виведення.

У персональному комп'ютері можна виділити такі основні компоненти:

– процесор;
– оперативно запам'ятовуючий пристрій;
– контролер переривань;
– контролер прямого доступу в пам'ять;
– паралельний інтерфейс;
– послідовний інтерфейс;
– контролер клавіатури;
– шина USB;
– системний таймер;
– годинник реального часу та енергонезалежна пам'ять;
– контролер твердих дисків;
– контролер гнучких дисків;
– шина ISA;
– шина PCI.

Самим головним компонентом у комп'ютері звичайно є процесор, але робота комп'ютера буде некоректна без інших не менш важливих компонентів, таких як:

– контролер переривань;

– контролер прямого доступу в пам'ять;

– системний таймер.

Розуміння роботи цих важливих вузлів персонального комп'ютера дуже важливо, тому що зрозуміти взаємодію всієї системи без них неможливо. Майже всі зв'язки в сучасному персональному комп'ютері чи проходять, чи тісно зв'язані з цими трьома компонентами.

Опис устрою 64-х розрядної операційної системи

Внутрішній устрій ядра ОС Windows 10/11

Цей розділ присвятимо нововведенням у ядрі ОС Windows 10/11. У цій роботі будуть розглянуті зміни в області процесів, потоків і вводу-виводу, керуванню пам'яттю, запуску й завершенню роботи. Даний розділ охоплює зміни тільки ядра ОС Windows 10/11™, особливо файлу Ntoskrnl.exe і пов'язаних з ним компонентів. Варто пам'ятати, що багато істотних змін в ОС Windows 10/11 не зачіпають властиво ядро системи, і отже, тут розглянуті не будуть. Це стосується вдосконалень оболонки (наприклад інтегрований пошук на робочому столі), роботи в мережі (наприклад новий стек IPv6 і двосторонній брандмауер) і графічної моделі нового покоління (Aero™ Glass, платформа Windows® Presentation Foundation, диспетчер вікон робочого стола й нова модель графічних драйверів). Також не буде розглянута нова інфраструктура драйверів користувальницького режиму (UMDF) і режиму ядра (KMDF) ОС Windows, оскільки її можна встановлювати й на більше ранні версії цієї ОС.

Лічильник циклів центрального процесора. ОС Windows 10/11 містить безліч удосконалень в області процесів і потоків, включаючи використання лічильника циклів центрального процесора для більше рівномірного виділення ресурсів, а також нову службу Multimedia Class Scheduler Service (MMCSS), що

сприяє безпомилковому відтворенню вмісту мультимедійними додатками. Всі версії ОС Windows NT® до Windows 10/11 включно виконують підпрограми переривання інтервального таймера приблизно кожні 10 або 15 мс, залежно від апаратної платформи. Дана підпрограма обновляє статистику завантаження центрального процесора (ЦП) потоком, що нею перерваний, так як якби цей потік виконувався весь інтервал, хоча в дійсності виконання потоку могло початися перед самим завершенням даного інтервалу. Далі потік міг формально одержати ресурси ЦП, але не одержати можливості виконуватися, оскільки замість цього могли бути виконані підпрограми апаратного або програмного переривання.

Служба Multimedia Class Scheduler Service. Користувачі очікують від мультимедійних додатків, таких як програвачі аудіо й відео, рівного й безпомилкового відтворення. Однак через зайнятість ЦП іншими додатками, що виконуються паралельно, наприклад антивірусними програмами, індексуванням вмісту або навіть поштовим клієнтом можуть виникати неприємні затримки й переривання. Для забезпечення більше якісного відтворення в ОС Windows 10/11 вводиться служба MMCSS, що управляє пріоритетами мультимедійних потоків для виконання процесором. Мультимедійний додаток, реєструється в службі MMCSS за допомогою нових функцій прикладного інтерфейсу, що відображають його мультимедійні характеристики, які повинні збігатися з характеристиками, перерахованими по іменах у відповідному розділі реєстру. Розділи різних завдань указують, яке перевагу одержують потоки, пов'язані з різними типами мультимедіа, при виділенні ресурсів ЦП і графічного процесора (хоча керування ресурсами графічного процесора в ОС Windows 10/11 не реалізовано).

Файлові символічні посилання. Зміни, що відносяться до операцій вводу-виводу в ОС Windows 10/11, містять у собі файлові символічні посилання, більше ефективну обробку завершення операцій вводу-виводу, повну підтримку скасування операцій вводу-виводу й підтримку пріоритетів операцій вводу-виводу. Однією з функцій файлової системи, який, на думку багатьох, бракувало в системі NTFS, є підтримка символічних посилань на файли (або, як вони

називаються в UNIX-системах, гнучких посилань), що з'явилася в ОС Windows 10/11. У версії NTFS для Windows 2000 були представлені символічні посилання на каталоги, які називалися з'єднаннями каталогів, що дозволяло створювати каталоги, що вказують на інші каталоги, але до виходу ОС Windows 10/11 файлова система NTFS підтримувала тільки тверді посилання на файли. Основною відмінністю в тім, як в ОС Windows дозволяються символічні посилання й з'єднання каталогів, є спосіб їхньої обробки. ОС Windows обробляє символічні посилання в локальній системі, навіть якщо вони посилаються на місце на віддаленому файловому сервері. ОС Windows обробляє з'єднання каталогів, що посилаються на віддалений файловий сервер, на самому сервері. Символічні посилання на сервері можуть, таким чином, посилатися на місця розташування, доступні тільки із клієнтських комп'ютерів, наприклад тому на інших клієнтських комп'ютерах, що неможливо для з'єднань каталогів. Для рішення цієї проблеми ОС Windows 10/11 підтримує новий тип символічних посилань, як для файлів, так і для каталогів.

Завершення й скасування операцій вводу-виводу. У систему вводу-виводу введено безліч внутрішніх змін, які можуть поліпшити продуктивність серверних додатків. Як правило, такі додатки використовують об'єкт синхронізації, називаний портом завершення, для очікування завершення асинхронних запитів вводу-виводу. До виходу ОС Windows 10/11 при завершенні операції вводу-виводу потік, що ініціював дану операцію, виконував завдання завершення вводу-виводу, що приводило до перемикання в процес, до якого належав даний потік, і перериванню інших дій. Потім система вводу-виводу обновляла стан порту завершення, що приводило до поновлення потоку, що очікувало зміни стану порту. В ОС Windows 10/11 обробка завершення вводу-виводу виконується не потоком, що ініціював операцію вводу-виводу, а потоком, що очікує зміни стану порту завершення. Ця відносно невелика зміна дозволяє уникнути непотрібного планування виконання потоків і перемикання контексту, що приводило до зниження загальної продуктивності системи й додатків. Для ще

більшого підвищення продуктивності сервер може одержувати результати різних операцій вводу-виводу від завершення одним запитом, уникаючи переходів у режим ядра.

Пріоритет операцій вводу-виводу. Незважаючи на те, що операційна система Windows завжди підтримувала пріоритети використання ЦП, у ній було відсутнє поняття пріоритетів операцій вводу-виводу. ОС Windows 10/11 забезпечує два нових способи розподілу пріоритетів для надання переваги операціям вводу-виводу переднього плану: пріоритети на окремі операції вводу-виводу й резервування смуги пропущення вводу-виводу.

Динамічний адресний простір ядра. ОС Windows і додатки, виконувані під її керуванням, зіштовхуються з обмеженнями адресного простору 32-розрядних процесорів. Розмір адресного простору ядра ОС Windows за замовчуванням обмежений значенням 2 Гбайт (половина 32-розрядного віртуального адресного простору). Друга половина зарезервована для використання процесом, що виконується в сучасний момент на ЦП. На свою половину пам'яті ядро повинне відобразити себе, драйвери пристроїв, кеш файлової системи, стеки ядра, структури даних коду для кожного сеансу, а також невивантажуємі (заблоковані у фізичній пам'яті) і що вивантажуються буфери, виділені драйверами пристроїв. В операційних системах, що передували ОС Windows 10/11, диспетчер пам'яті розподіляв адресні простори для перерахованих потреб у момент завантаження системи. Цей негнучкий механізм приводив іноді до ситуацій, коли деякі області адресного простору виявлялися вичерпаними, у той час як в інші ще залишалося багато вільного місця. Недостача адресного простору в одній з областей може привести до збоїв у роботі додатків і порушити операції вводу-виводу, виконувані драйверами пристроїв. В 32-розрядній версії ОС Windows 10/11 диспетчер пам'яті динамічно розподіляє адресний простір ядра, виділяючи й звільняючи адресний простір з урахуванням потреб поточного робочого навантаження. Таким чином, кількість віртуальної пам'яті, використовувана для зберігання буферів, що вивантажуються, може

збільшуватися, коли цього вимагають драйвери пристроїв, і зменшуватися, коли драйвери пристроїв звільняють пам'ять. Звичайно, в 64-розрядній версії ОС Windows 10/11 межі адресних просторів не представляють практичних обмежень, тому вони не вимагають яких-небудь особливих мір і встановлені на максимальні значення.

Пріоритети пам'яті. Крім пріоритетів вводу-виводу, в ОС Windows 10/11 з'явилися також і пріоритети пам'яті. Щоб зрозуміти, як система використовує пріоритети пам'яті, потрібно розібратися, як у диспетчері пам'яті реалізований кеш пам'яті, називаний списком очікування (Standby List). У всіх версіях ОС Windows, що передували Windows 10/11, фізична сторінка, викликана системою в додатку (розмір сторінки звичайно становить 4 Кбайт), містилася диспетчером пам'яті в кінець списку очікування. Якщо процесу знову був потрібний доступ до цієї сторінки, диспетчер пам'яті витягав неї зі списку очікування й знову призначав процесу. Якщо процесу була потрібна нова сторінка фізичної пам'яті, але вільної пам'яті не було, диспетчер пам'яті віддавав процесу сторінку з початку списку очікування. У такій схемі всі сторінки в списку очікування були рівнозначні, і порядок сторінок визначався тільки часом їхнього приміщення в список.В ОС Windows 10/11 кожна сторінка пам'яті має пріоритет від 0 до 7, і диспетчер пам'яті розділяє список очікування на 8 списків, у кожному з яких зберігаються сторінки з відповідним пріоритетом. Якщо диспетчерові пам'яті потрібно забрати сторінку зі списку очікування, першими будуть витягати сторінки зі списків з більше низьким пріоритетом. Пріоритет сторінки звичайно збігається із пріоритетом потоку, що першим викликав виділення цієї сторінки. (Пріоритет сторінки зі спільним доступом відповідає найвищому пріоритету пам'яті серед потоків, що спільно використовують цю сторінку). Потік успадковує значення пріоритету пам'яті від процесу, якому він належить. Диспетчер пам'яті призначає низький пріоритет для сторінок, які він зчитує з диска, коли передбачається обіг процесу до пам'яті. За замовчуванням значення пріоритету пам'яті процесів рівняється 5, але додатки й система можуть міняти

значення пріоритетів пам'яті для процесів і потоків. Вигода від використання пріоритетів пам'яті стає зрозумілою, якщо розглядати систему відносних пріоритетів сторінок на більше високому рівні, а саме на рівні функції SuperFetch.

Функція SuperFetch. Серед істотних змін у роботі диспетчера пам'яті – зміна способу керування фізичною пам'яттю. У способу керування на основі списку очікування, що використовувався в попередніх версіях ОС Windows, є два обмеження. По-перше, пріоритет сторінок визначається тільки на підставі попереднього поводження процесів без пророкування можливих майбутніх вимог процесів до пам'яті. По-друге, визначення пріоритету здійснюється тільки на підставі списку сторінок пам'яті, якими процес володіє в конкретний момент часу. В ОС Windows XP була реалізована підтримка читання, що попереджає, завдяки якій відбувалося більш швидке завантаження системи й запуск додатків. До пам’яті заздалегідь завантажувалися дані, до яких приблизно буде вимагатися доступ. Таке рішення приймалося на підставі історії попередніх завантажень системи й запусків додатків. Функція SuperFetch в ОС Windows 10/11 є істотним розвитком схеми керування пам'яттю, оскільки, крім статистики недавнього доступу, вона враховує історію звертань до пам'яті за тривалий період і проактивне керування пам'яттю.

Функція ReadyBoost. Швидкість роботи ЦП і пам'яті набагато вище швидкості роботи жорсткого диска, через що жорсткі диски часто є вузьким місцем для продуктивності системи. Операції довільного дискового вводу-виводу особливо сильно впливають на продуктивність, тому що час позиціонування головок жорстких дисків (порядку 10 мс) – це довго для сучасних процесорів з тактовою частотою 3 Ггц. Хоча оперативна пам'ять ідеально підходить для кешування дискових даних, її вартість порівняно висока. Флеш-пам'ять звичайно дешевше оперативної пам'яті, при цьому час довільного доступу у флеш-пам'яті може бути на порядок менше, ніж у жорсткого диска. Тому в ОС Windows 10/11 додана функція за назвою ReadyBoost, що дозволяє скористатися перевагами запам'ятовувальних пристроїв на основі флеш-пам'яті, створюючи на них

проміжний рівень кешування, логічно розташований між оперативною пам'яттю й жорсткими дисками.

Функція ReadyBoot. Якщо в системі встановлено менш 512 Мб оперативної пам'яті, механізм читання, що попереджає, при завантаженні ОС Windows 10/11 не відрізняється від механізму, що використовувався при завантаженні ОС Windows XP. Якщо ж розмір оперативної пам'яті перевищує 700 Мб, то для оптимізації процесу завантаження використовується кеш в ОЗП. Розмір цього кешу залежить від загального обсягу доступної пам'яті; він досить великий, щоб забезпечити ефективне кешування, але залишає при цьому досить вільній пам'яті для нормального виконання процедури завантаження системи. Після кожного завантаження системи служба ReadyBoost (у моменти простою ЦП планує кешування для наступного завантаження системи. Вона аналізує інформацію про звертання до файлів за п'ять попередніх завантажень і визначає, до яких файлів вироблялися звертання, і де ці файли розташовані на диску.

Функція ReadyDrive. ReadyDrive – це функція ОС Windows 10/11, що використовує можливості нових гібридних жорстких дисків (дисків H-HDD). Гібридний жорсткий диск являє собою жорсткий диск із вбудованою енергонезалежної флеш-пам'яттю (відомі також за назвою NVRAM). Як правило, гібридні жорсткі диски містять від 50 Мб до 512 Мб кеш-пам'яті, але ОС Windows 10/11 підтримує кеші розміром до 2 ТБ. Щоб визначити, які дані варто помістити у флеш-пам'ять, ОС Windows 10/11 використовує команди ATA-8.

База даних конфігурації завантаження. В ОС Windows 10/11 удосконалено кілька аспектів процедур завантаження й завершення роботи. Поліпшення процедури завантаження досягається завдяки наступним нововведенням: використання бази дані конфігурації завантаження (BCD), у якій зберігається конфігурація запуску системи й ОС; нова послідовність і організація процесу завантаження системи; нова архітектура входу в систему, а також підтримка служб із відкладеним автозапуском. Удосконалення процесу завершення роботи в ОС Windows 10/11 включають повідомлення служб

Windows перед завершенням роботи, упорядкування завершення роботи служб і істотно перероблене керування переходами між режимами енергоспоживання в операційній системі. Одним із самих помітних змін процесу завантаження системи є відсутність файлу Boot.ini у кореневій папці системного тому. Це пояснюється тим, що конфігурація завантаження, що раніше зберігалася в цьому файлі, тепер зберігається в базі даних BCD. Однієї із причин використання бази даних BCD в ОС Windows 10/11 є об'єднання двох підтримуваних в ОС Windows архитектур завантаження: основного завантажувального запису (MBR) і розширюваного інтерфейсу мікропрограм (EFI).

Процес завантаження системи. В ОС Windows 10/11 було реструктуровано кілька системних процесів. Диспетчер сеансів (Smss.exe) – це перший процес користувальницького режиму, що запускається під час завантаження, як і в попередніх версіях ОС Windows. Однак в ОС Windows 10/11 диспетчер сеансів запускає ще одну свою копію для настроювання сеансу 0, що виділений винятково для системних процесів. Процес диспетчера сеансів для сеансу 0 запускає додаток ініціалізації Windows (Wininit.exe), процес підсистеми Windows (Csrss.exe) для сеансу 0, а потім завершує свою роботу. Далі додаток ініціалізації Windows запускає диспетчер керування службами, підсистему локального адміністратора безпеки й новий процес – диспетчер локальних сеансів (Lsm.exe), що управляє сеансами термінального сервера на цьому комп'ютері.

Постачальники облікових даних. Архітектура процесу входу в систему також помінялася в ОС Windows 10/11. У попередніх версіях ОС Windows процес Winlogon завантажував зазначену в реєстрі динамічну бібліотеку GINA, що відображала користувальницький інтерфейс для входу в систему, що запитує в користувачів уведення облікових даних. На жаль, у моделі GINA є ряд обмежень: може бути використана тільки одна бібліотека GINA; розробка повнофункціональної бібліотеки GINA складна для сторонніх розроблювачів; сторонні бібліотеки GINA з нестандартним користувальницьким інтерфейсом

міняють звичну взаємодію з користувачем. Замість моделі GINA в ОС Windows 10/11 використовується архітектура постачальників облікових даних. Процес Winlogon запускає окремий процес – сервер користувальницького інтерфейсу входу в систему (Logonui.exe), що виконує завантаження постачальників облікових даних.

Служби з відкладеним автозапуском. В ОС Windows 10/11 введений новий тип запуску служб – відкладений автоматичний запуск. Цей режим можуть використовувати служби, які не зобов'язані бути активними відразу після завантаження системи. Диспетчер керування службами запускає служби, настроєні на відкладений автоматичний запуск, тільки після завершення запуску всіх служб, настроєних на автоматичний запуск. Диспетчер установлює первісному потоку таких служб пріоритет THREAD_PRIORITY_LOWEST (щонайнижчий пріоритет потоку). Установка такого пріоритету приводить до того, що всі операції вводу-виводу, виконувані потоком, виконуються із пріоритетом «Very Low» (дуже низький). Коли служба завершує ініціалізацію, диспетчер керування службами встановлює їй пріоритет «Normal» (звичайний). Сполучення відкладеного запуску, низького пріоритету використання ЦП і пам'яті, а також фонового пріоритету дискових операцій приводить до істотного зниження впливу запуску таких служб на процес входу користувача в систему.

Завершення роботи. Одна із проблем, з якими зіштовхувалися розроблювачі служб Windows, полягає в тому, що за замовчуванням у служби є не більше 20 секунд на завершення роботи. Коли ОС Windows 10/11 завершує роботу, диспетчер керування службами спершу повідомляє про завершення роботи всі служби, що запросили таке повідомлення. Диспетчер буде чекати завершення роботи таких служб нескінченно довго, але якщо в службі є помилка й вона не відповідає на запити протягом три мінут, диспетчер перестає чекати завершення роботи цієї служби. Після того, як всі служби, що запросили повідомлення, завершили роботу або минув час очікування, диспетчер керування

службами виконує завершення роботи інших служб так, як це було реалізовано в попередніх версіях.

Керування електроживленням. В ОС Windows 10/11 диспетчер керування електроживленням, що входить у ядро системи, по колишньому повідомляє драйвери й додатки про зміну режиму енергоспоживання, дозволяючи їм підготуватися до переходу в інший режим. Однак тепер ядро не запитує в них дозволу. Більше того, диспетчер керування електроживленням чекає відповіді додатків не більше 20 секунд, а не 2 мінути, як це було в попередніх версіях ОС Windows. У результаті користувачі ОС Windows 10/11 можуть бути впевнені в тім, що система дійсно перейде в режим, що чекає, або режим гібернації.

3.2 Розробка структурної схеми

Розглянемо розроблену структурну схему системи зображену на рисунку 3.5. В процесі практичної реалізації теоретичних принципів розробки системи, додатково розглянутих в розділі 3, була розроблена структурна схема системи, завдяки структурній схемі можна чітко побачити основні структурні блоки системи та взаємозв'язки між ними. Аналіз структурної схеми дозволяє чітко прослідити як працює програм. Розглянемо схему зверху вниз, з початку запуску ПЗ до завершення роботи. При розробці структурної схеми були використанні наступні джерела інформації [1-4].

Структурна схема складається з наступних блоків:

– програмного забезпечення;

– підсистеми ОС (Windows 10/11);

– емуляції доступу до портів введення/виведення;

– драйвера пристроїв;

– ядра 64х розрядної ОС (Windows 10/11);

– драйвер доступу;

– системної шини 64х та портів введення / виведення;

– апаратних засобів ПК.

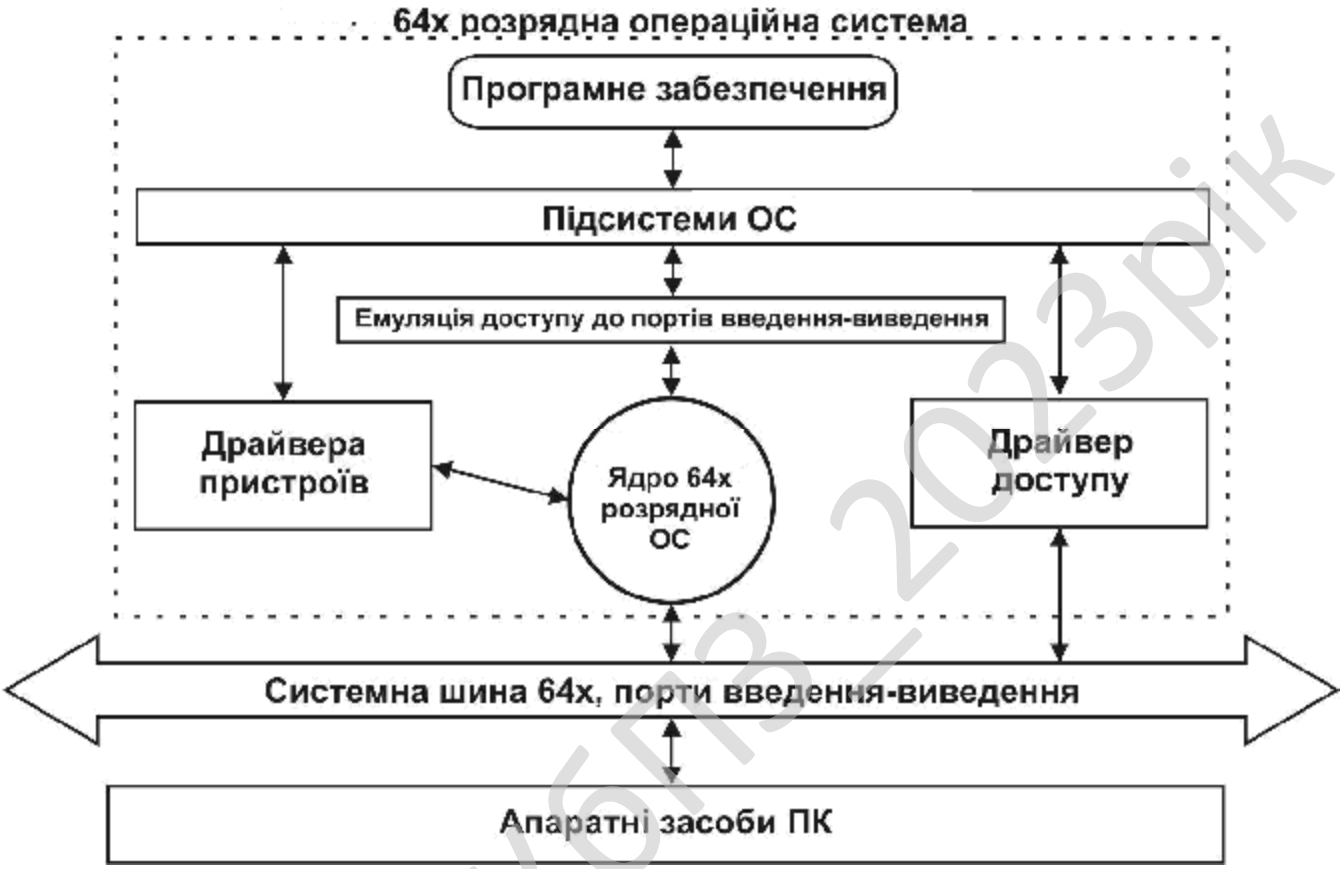


Рисунок 3.5 – Структурна схема роботи системи

Коли програма живлеться одержати доступ до апаратної частини ПК у нових 64 розрядних системах відбувається емуляція доступу до апаратного рівня й портів уведення/виводу, а саме взаємодія через ядро операційної системи. Вона перевіряє запущений процес і вирішує допускати програму до апаратного забезпечення. ОС x64 повністю перехоплює керування апаратною частиною й дає прямий доступ тільки до портів 70h і 71h. Для організації прямого доступу до ресурсів ПК необхідний спеціально розроблений драйвер доступу котрий буде обходити ядро операційної системи.

3.3 Розробка функціональної схеми

На функціональній схемі, зображеній на рисунку 3.6, визначена схема роботи системи. Розглянемо принцип будови ПК з коректівами новітніх 64 розрядних операційних систем.

Архітектурно комп'ютер являє собою комплекс мікроконтролерів і мікросхем які взаємодіють між собою певним чином. Всі ЕОМ, сумісні з IBM PC, мають ту саму архітектуру, оскільки їх основою є мікросхеми фірми Intel.

Фізично складові ПК можна розділити на компоненти системного блоку і компоненти блоку розширення. Всі основні плати, що входять до складу будь-якої моделі ПК, розміщаються у великому блоці, що одержав назву системного.

Системний блок включає всі необхідні компоненти, що дозволяють комп'ютеру працювати без будь-яких доповнень та новітні ОС вносять свою корективу. Будь-які додаткові пристрої підключаються до ПК за допомогою одного з роз'ємів розширення, які залежать від відповідного типу материнської плати. Ці лінії дозволяють передавати всі сигнали, необхідні для керування будь-яким устаткуванням, що може бути підключене до ПК. Любий сигнал, що посилається одному з блоків розширення, передається і всім іншим, оскільки вони підключені до паралельних ліній. Тут має місце розширення ідеї загальної шини даних: усі блоки розширення використовують загальне з'єднання, назване каналом введення/виведення.

Обмін даними в комп'ютері, при класичній його побудові, здійснює процесор. Однак таку задачу, як обмін даними з периферійними пристроями (тобто при здійсненні зв'язку з зовнішнім світом), намагаються по можливості виконати за допомогою спеціалізованих пристроїв обміну інформацією.

Самим головним компонентом у комп'ютері звичайно є процесор, але робота комп'ютера буде некоректна без інших не менш важливих компонентів:

– контролер переривань;

– контролер прямого доступу в пам'ять;

– системний таймер.

Розуміння роботи цих важливих вузлів персонального комп'ютера дуже важливо, тому що зрозуміти взаємодію всієї системи без них неможливо. Майже всі зв'язки в сучасному персональному комп'ютері чи проходять, чи тісно зв'язані з цими трьома компонентами.

На функціональній схемі, є можливість прослідкувати за шляхами проходження сигналів від одного функціонального блоку до іншого.

При розробці функціональної схеми було розглянуто внутрішня побудова комп’ютера та новітньої операційної системи Windows 10/11, позначено основні функціональні блоки програми і внесені на схему, також були розглянуті взаємодії між функціональними блоками і зроблені відповідні зміни на схемі.

Після наповнення схеми основними функціональними блоками програми і нанесення взаємодій, був зроблений аналіз схеми на правильну організацію усіх взаємодій і внесення остаточних змін.

ОС 64х розрядні при спробі доступу до системної шини чи до портів введення/виведення проводить наступні дії.

1. Через інтерфейс обміну операційної системи проводить аналіз запиту.

2. Проводить обробку запиту і перевірку можливості доступу до портів. Якщо такий доступ є критичним для системи проводиться емуляція доступу до портів введення-виведення тобто ОС проводить запит самостійно та вертає результат до програми за допомогою блоку обробки запиту.

3. Якщо запит не е критичним, а це лише 3 відсотки від можливих запитів (порти 70h, 71h) проходе взаємодія з апаратною частиною.

Рисунок 3.6 – Функціональна схема роботи системи

3.4 Розробка діаграми процесів

Одним з важливих критеріїв при розробці будь-якого програмного забезпечення це грамотна розробка структури роботи системи потоків і процесів. Розглянемо діаграму процесів (рисунок 3.7).

На діаграмі процесів можна точно зрозуміти як працює і взаємодіє ПЗ в цілому. Починається і закінчується програма в першому блоці є основною крапкою відрахунку діаграми. При перемішені по стрілках можна побачити загальну схему взаємодії блоків і їх входження один в одного.

Чітко простежується те, що без введення даних користувача та проходження обробника помилок здійснення емуляції взаємодії системи з встановленням початкових параметрів ПЗ, інтерфейсом ініціалізації та роботи програмної моделі та обробкою введених даних неможливе.

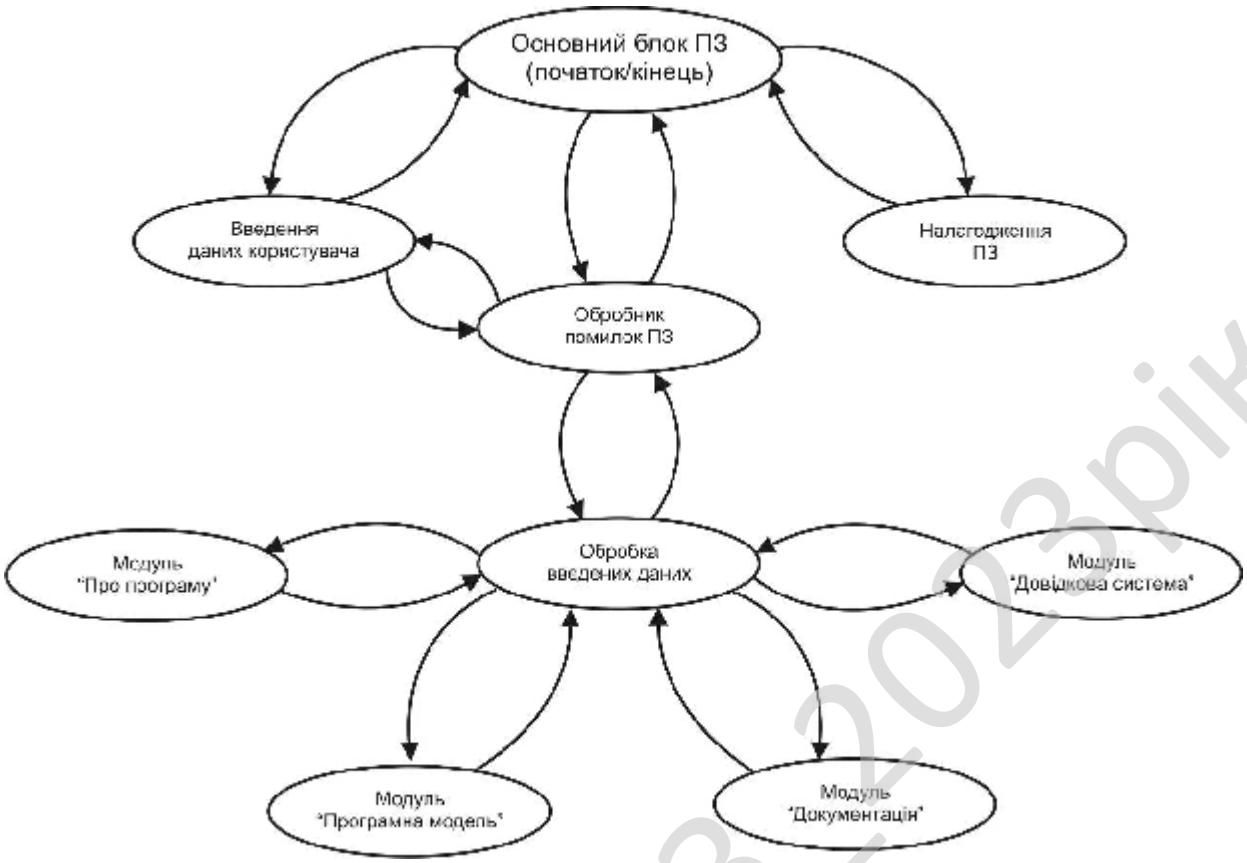


Рисунок 3.7 – Схема взаємодії процесів

Таким чином, розглянувши опис системи, структурну, функціональну схеми системи, та діаграму взаємодії процесів перейдемо до опису блок-схем основної програми, та підпрограм, які використовуються, для реалізації системи.

4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ ТА ПРОГРАМНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

При написанні програми використовувався модульно-реверсивний алгоритм, суть якого полягає в тому, що взаємодія всіх модулів проходить через блок керування модулями, цей алгоритм використовується при написанні програмних моделей та справ очних систем. Спрощена схема взаємодії приводе до оптимізації ПЗ.

Опишемо послідовність дій, що відбуваються при переході до форми. При запуску програми і підключенні модулів. При виборі підсистеми яка його цікавить, він переходить у цей блок, де відбувається заплановані дії. Після блоку ініціалізації на екран виводиться вікно модуля програми. Після з'ясування цікавлячих питань чи повного вивчення необхідного матеріалу програма входить у блок завершення роботи модуля після чого в залежності від дій користувача програма виводить головну форму чи виходить із програми.

Необхідне створення повноцінного коду програмного продукту з розробленою ціновою політикою і системою знижок покупцям програмної продукції. Блок-схеми є першоджерелами стратегії розвитку ПЗ. Первинною стадією без якої не відбувається розробка програмного забезпечення це звичайно розробка блок-схем. Тому від точності і детальної блок-схеми залежить результат всієї програми.

На рисунку 4.1 та 4.2 зображені основна блок-схема програми та блок-схема підпрограм.

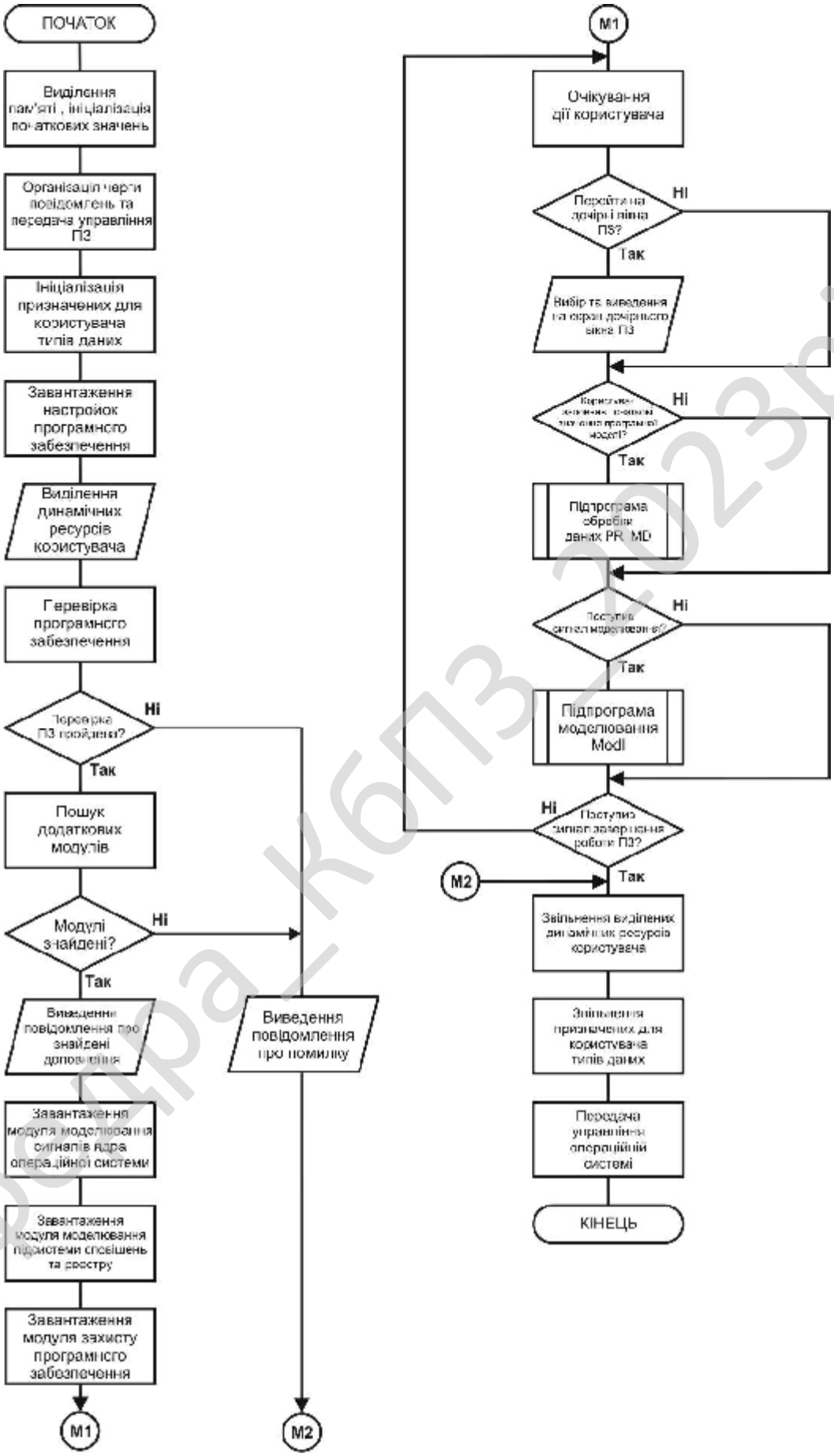


Рисунок 4.1 – Блок схема основної частини ПЗ

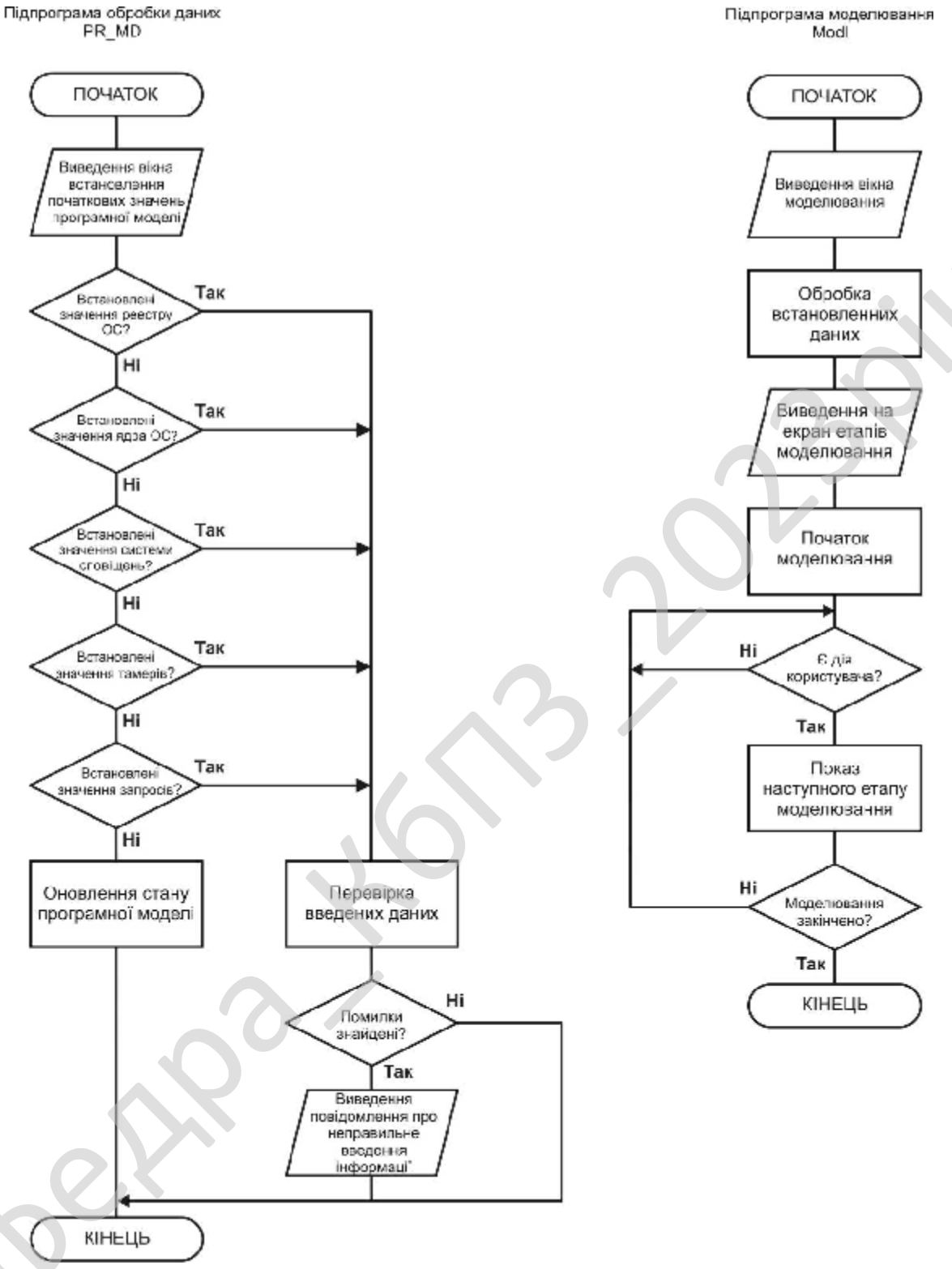


Рисунок 4.2 – Блок схема підпрограм

При детальному розгляді програма розбита на декілька важливих блоків таких як:

– блок ініціалізації початкових значень типів даних, констант, змінних, класів, масивів і початкових значень та блоку організації черги повідомлень та передача управління ПЗ;

– блоків завантаження настройок програмного забезпечення та блоку виділення динамічних ресурсів користувача;

– блоку перевірки програмного забезпечення;

– блоку пошуку додаткових модулів;

– блоків завантаження модуля моделювання сигналів ядра операційної системи, моделювання підсистеми сповіщень та реєстру, захисту програмного забезпечення;

– блоку очікування дії користувача;

– блоку підпрограми обробки даних PR_MD (рисунок 4.2);

– блоку підпрограми моделювання Modl (рисунок 4.2);

– блоку аналізу даних;

– блоку виконання додаткових модулів;

– блок завершення роботи програми.

Обґрунтування методу шифрування внутрішніх модифікованих даних

Однією з найактуальніших на сьогодні тем в сфері розробки програмних продуктів є організація якісного методу шифрування внутрішніх модифікованих даних. Річ у тім, що створення якісного програмного продукту – дуже складана річ, яка потребує великих затрат людського та машинного часу. Тому і оплачуватися ця праця повинна відповідно високо. Зазвичай кошти на оплату праці програмістів надходять від реалізації розробленої програми. Але так вже склалося на сьогоднішній день, що 90% використовуваних програм не покупаються у розробників чи їхніх представників, а копіюються у тих

користувачів, хто вже придбав їх раніше. Саме тому розроблюваний програмний продукт повинен бути захищений:

– від несанкціонованого доступу;

– від копіювання.

Методи захисту даних на персональних комп'ютерах дуже різноманітні як по кінцевій цілі, так і по технічному втіленню.

Нижче наведен розроблений вихідний код, методу шифрування внутрішніх модифікованих даних.

```
(*************************************)
(* Модуль шифрування внутрішніх даних *)
(* використовується 128 бітний ключ *)
(*************************************)

Схема дії алгоритму
 1. У функцію передається масив даних.
2. Генерується унікальний 128 бітний ключ для кодування на основі guid.
3. Обчислюється розмір даних, вони повинні бути кратні парній кіл-ті 32 бітних
блоків, об'єм, що бракує заповнюється випадковою інформацією.
4. Перші три байти займає заголовок eta
5. У четверту позицію даних додається значення яке указує обсяг сміття.
6. Виробляється кодування раніше згенерованим ключем
Сам ключ кодується внутрішнім ключем, ділиться на чотири частини.
7. 128 бітний ключ розділяється на чотири 32 бітних блоки
8. Ключ додається в дані зі зрушенням у будь-яке місце
в наступному порядку 4-ий, 3-ий, 1-ий, 2-ий блоки.
9. На 14 у позицію даних вставляється зі зрушенням покажчик на початок ключа.
10. Разом у закодованому буфері перебуває крім основної ще й 21 байт службової
інформації.
Для раскодування всі дії виробляється у зворотному порядку

unit Program_Model;

interface

uses
windows,
sysutils,
classes;

const
delta=$9e3779b9;
// Зсув контрольної суми ~ 32 біт
_k0: int64 = $982c98c1;
// Головний 128 бітний ключ (4 частини по 32 біта)
_k1: int64 = $7f8f4d4b;
_k2: int64 = $bcae3151;
_k3: int64 = $971bc789;
header = ' Program_Model '; // Заголовок

 type
phash = ^thash;
thash = array of byte;

function decript(var s: thash): boolean;
```

```
procedure encript(var s: thash);

implementation

//////////////////////////////////////////////////////
 // ЧАСТИНА ПЕРША * * * КОДУВАННЯ * * *

procedure encript(var s: thash);
var
inbuf,
outbuf,
resultbuf: thash;
// Вхідний, вихідний і результуючий буфери
y, z, sum: longword;
// Тимчасові змінні для кодуємих блоків даних
k0, k1, k2 , k3: longword;
// Поточний ключ для шифрування
i, a, len: integer;
// Змінні для циклів
c: byte;
// Лічильник кіл-ті сміття
guid, key: string;
g: tguid;
begin
// Перевірка розміру даних
if length(s) = 0 then exit;

createguid(g); // Генеруємо ключ на основі guid
guid := guidtostring(g);
for i := 1 to length(guid) do
if guid[i] in ['0'..'9', 'a'..'f'] then
key := key + guid[i];

k0 := strtoint64('$' + copy(key, 1, 8));
k1 := strtoint64('$' + copy(key, 9, 8));
k2 := strtoint64('$' + copy(key, 17, 8));
k3 := strtoint64('$' + copy(key, 25, 8));

c := 0; // Ініціалізуємо лічильник дозаповнень
// Дозаповняємо дані щоб останній блок даних був рівний 64 бітам
while (length(s) div 8) * 8 <> length(s) do
begin
len := length(s);
inc(len);
setlength(s, len);
s[len - 1] := random(255); // Заповнюємо випадковими даними
inc(c);
end;

len := length(s); // Обчислюємо розмір кодуємого блоку
setlength(inbuf, len); // Установлюємо розмір буферів

// Розмір вихідного буфера збільшений на 21 байт через
// 3 байти - заголовок eta
// 1 байт - лічильник кіл-ті сміття наприкінці буфера
// 16 байт - кодований ключ - 4 cardinal
// 1 байт - мітка розташування кодованого ключа
inc(len, 21);
setlength(outbuf, len);
setlength(resultbuf, len);
//Збільшимо кіл-у сміття для видалення самого поля лічильника
inc(c);
 outbuf[0] := ord(header[1]); // додаємо ідентифікатор
```

```
outbuf[1] := ord(header[2]);
outbuf[2] := ord(header[3]);
outbuf[3] := c; // Додаємо лічильник сміття

move(s[0], inbuf[0], length(s));
// Заповнюємо вхідний буфер даними

i := 0;
while i < len - 21 do // Безпосередньо кодування
begin
move(inbuf[i], y, 4); // Беремо перші 32 біта
move(inbuf[i + 4], z, 4); // Беремо другі 32 біта
// Кодуємо
sum := 0;
// 64 бітний кодуємий блок (2 частини по 32 біта)
for a := 0 to 31 do
begin
inc(sum, delta);
inc(y,((z shl 4)+k0)xor(z+sum)xor((z shr 5) + k1));
inc(z,((y shl 4)+k2) xor (y+sum) xor ((y shr 5)+k3));
end;
move (y, outbuf[i + 4], 4);
// Поміщаємо кодовані блоки у вихідному буфер
move (z, outbuf[i + 8], 4);
inc(i, 8);
// Пропускаємо оброблений блок, переходимо до наступного
end;

sum := 0;
for a := 0 to 31 do
// Кодуємо перші 2 частини ключа внутрішнім ключем
begin
inc(sum,delta);
inc(k0,((k1 shl 4)+_k0)xor(k1+sum)xor((k1 shr 5)+_k1));
inc(k1,((k0 shl 4)+_k2)xor(k0+sum)xor((k0 shr 5)+_k3));
end;

sum := 0;
for a := 0 to 31 do
// Кодуємо другі 2 частини ключа внутрішнім ключем
begin
inc(sum, delta);
inc(k2,((k3 shl 4)+_k0) xor (k3+sum)xor((k3 shr 5)+_k1));
inc(k3,((k2 shl 4)+_k2) xor (k2+sum)xor((k2 shr 5)+_k3));
 end;
// Визначаємо позицію розміщення ключа в блоці даних
randomize;
if len < 255 then
i := len
else
i := 255;
repeat
i := random(i);
if i < 4 then i := 4;
until i <= len - 16;

// Зміщаємо дані звільняючи місце для чотирьох
// частин ключа
move(outbuf[0], resultbuf[0], i);
move(outbuf[i], resultbuf[i + 16], len - i - 17);

// Розбиваємо четверту чверть ключа на 4 восьмибітних //частини
resultbuf[i] := byte(k3 shr 24);
```

```
resultbuf[i + 1] := byte(k3 shr 16);
resultbuf[i + 2] := byte(k3 shr 8);
resultbuf[i + 3] := byte(k3);

// Розбиваємо третю чверть ключа на 4 восьмибітних частини
resultbuf[i + 4] := byte(k2 shr 24);
resultbuf[i + 5] := byte(k2 shr 16);
resultbuf[i + 6] := byte(k2 shr 8);
resultbuf[i + 7] := byte(k2);

// Розбиваємо першу чверть ключа на 4 восьмибітних частини
resultbuf[i + 8] := byte(k0 shr 24);
resultbuf[i + 9] := byte(k0 shr 16);
resultbuf[i + 10] := byte(k0 shr 8);
resultbuf[i + 11] := byte(k0);

// Розбиваємо другу чверть ключа на 4 восьмибітних частини
resultbuf[i + 12] := byte(k1 shr 24);
resultbuf[i + 13] := byte(k1 shr 16);
resultbuf[i + 14] := byte(k1 shr 8);
resultbuf[i + 15] := byte(k1);

// Зрушуємо дані з 14 позиції на одну вправо для мітки
// (буфер починається з нуля)
for a := len - 1 downto 14 do
resultbuf[a] := resultbuf[a - 1];

// Поміщаємо мітку початку ключа ( 14-й байт)
resultbuf[13] := i;

s := resultbuf;
end;


////////////////////////////////////////////////////
//ЧАСТИНА ДРУГА * * * ДЕКОДУВАННЯ * * *

function decript (var s: thash): boolean;
var
inbuf,
outbuf,
resultbuf: thash;
//Вхідний, вихідний і результуючий буфери
y , z, sum: longword;
// Тимчасові змінні для кодуємих блоків даних
k0, k1, k2, k3: longword; // Поточний ключ для шифрування
i, a, len: integer; // Змінні для циклів
aheader: string;
begin
result := false;

len := length(s); // Обчислюємо розмір декодуємого блоку

// Перевірка розміру
if len < 27 then exit;
if len <> (((len - 21) div 8) * 8) + 21 then exit;

// Перевірка заголовка
aheader := char(s[0]) + char(s[1]) + char(s[2]);
if aheader <> header then exit;

// Перевірка позиції ключа
if not(s[13] in [4..255]) then exit;
```

```
if s[13] + 16 > len then exit;

// Перевірка лічильника сміття
if s[3] > 8 then exit;

setlength(inbuf, len);
// Установлюємо розмір буферів

move(s[0], inbuf[0], len);
// Заповнюємо вхідний буфер даними

i := inbuf[13]; // Довідаємося початкову позицію ключа

// Видаляємо мітку на початок ключа
for a := 13 to len - 2 do
begin
inbuf[a] := inbuf[a + 1];
inbuf[a + 1] := 0;
end;
dec(len);

// Витягаємо ключ
k3:=(inbuf[i+3]or(inbuf[i+2] shl 8)or(inbuf[i+1] shl 16) or (inbuf[i] shl 24));
k2:=(inbuf[i+7]or(inbuf[i+6]shl 8)or(inbuf[i+5]shl 16)or (inbuf[i+4] shl 24));
k0:=(inbuf[i+11]or(inbuf[i+10]shl 8)or(inbuf[i+9]shl 16) or (inbuf[i+8] shl 24));
k1:=(inbuf[i+15]or(inbuf[i+14]shl 8)or(inbuf[i+13]shl 16) or (inbuf[i+12] shl
24));

// Видаляємо ключ із блоку даних
for a := i + 16 to len do
begin
inbuf[a - 16] := inbuf[a];
inbuf[a] := 0;
end;
setlength(outbuf, len);
zeromemory(outbuf, len);
dec(len, 16); // Видаляємо розмір ключа

// Декодуєм перші дві частини ключа
sum := delta shl 5;
for a := 0 to 31 do
begin
dec(k1,((k0 shl 4)+_k2)xor(k0+sum) xor ((k0 shr 5)+_k3));
dec(k0,((k1 shl 4)+_k0) xor (k1+sum)xor((k1 shr 5)+_k1));
dec(sum, delta);
end;

// Декодуєм другі дві частини ключа
sum := delta shl 5;
for a := 0 to 31 do
begin
dec(k3,((k2 shl 4)+_k2)xor(k2+sum)xor((k2 shr 5) + _k3));
dec(k2,((k3 shl 4)+_k0)xor(k3+sum) xor ((k3 shr 5)+_k1));
dec(sum, delta);
end;

i := 0;
dec(len); // Видаляємо з розміру місце лічильника сміття
dec(len, 3); // Видаляємо з розміру заголовок eta
while i < len do // Безпосередньо декодировка
begin
move(inbuf[i + 4], y, 4); // Беремо перші 32 біта
move(inbuf[i + 8], z, 4); // Беремо другі 32 біта
```

```
// Декодуєм
sum := delta shl 5;
for a := 0 to 31 do
// 64 бітний кодуємий блок (2 частини по 32 біта)
begin
dec(z,((y shl 4)+k2)xor(y+sum)xor((y shr 5)+k3));
dec(y,((z shl 4)+k0)xor(z+sum)xor((z shr 5)+ k1));
dec(sum, delta);
end;
move(y, outbuf[i], 4);
// Запам'ятовуємо кодовані блоки у вихідному буфері
move(z, outbuf[i + 4], 4);
inc(i, 8);
// Пропускаємо оброблений блок, переходимо до наступного
end;

//Відрізаємо -1 тому що місце для лічильника вже вилучене з //len
 len := len - (inbuf[3] - 1);
setlength(resultbuf, len);
move(outbuf[0], resultbuf[0], len);

// Виводимо текст із вихідного буфера
s := resultbuf;
result := true;
end;

end.
```

4.2 Захист розробленого програмного забезпечення

Дані які використовуються у даній роботі захищаються алгоритмом ДСТУ 9041:2020. Його повна назва: ДСТУ 9041:2020. Інформаційні технології. Криптографічний захист інформації. Алгоритм шифрування коротких повідомлень, що ґрунтується на скручених еліптичних кривих Едвардса.

Цей алгоритм призначений для шифрування коротких (до 616 біт) повідомлень для будь-яких алгоритмів шифрування, в тому числі визначених національними стандартами України.

Як і стандарт цифрового підпису ДСТУ 4145:2002, новий алгоритм використовує криптографічні перетворення у групі точок еліптичних кривих, використовуючи замість кривих у формі Вейєрштрасса найновітніші розробки у галузі еліптичної криптографії – криві у формі Едвардса. Це дає суттєві переваги у швидкодії більш ніж у 3 рази. Новий стандарт розроблений з урахуванням усіх найсучасніших вимог до стійкості криптографічних алгоритмів. Так, нижня межа стійкості криптосистем у цьому стандарті дорівнює 2127 (≈1042) (це більш ніж у

півтора рази вище, ніж у ДСТУ 4145) і можуть бути обрані інші рівні, такі як 2255 (≈1085), 2383 (≈10127) та 2767 (≈10255); крім того, строго обґрунтована його стійкість як до атак на відновлення відкритого тексту, так і до розрізнюючих атак.

Проект алгоритму шифрування, що ліг в основу цього стандарту, пройшов апробацію як в Україні, так і за її межами (Центрально-Європейська конференція з криптографії (червень 2020 року) – форум ведучих криптологів з усього світу).

Стандарт ДСТУ-9041 узгоджений з усіма діючими в Україні національними стандартами. Новиною стандарту є його сфера застосування – інкапсуляція ключів, найсучасніший математичний апарат, а також новий алгоритм генерації псевдовипадкових послідовностей, який, на відміну від аналогічного алгоритму генерації з ДСТУ 4145, використовує виключно національні криптографічні алгоритми національних стандартів та не містить посилань на відповідні пост-радянські стандарти, термін дії яких вже практично вичерпався.

Новий стандарт не належить до так званих постквантових стандартів. Але його стійкість буде під загрозою лише тоді, коли з’являться квантові комп’ютери з 700 і більше кубітами (на даний час кількість "робочих" кубітів, які вдалося створити, – близько 50). Його перевагою перед постквантовими алгоритмами є відносно невелика довжина ключа (у десятки або навіть у сотні разів менша, ніж у постквантових алгоритмах).

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Розроблене програмне забезпечення системи моделювання взаємодії 64-х розрядних ОС з апаратними засобами ПК рекомендується для впровадження в учбових закладах для підготовки фахівців з системного програмування та в будь-яких організаціях, що займаються розробкою програмного забезпечення для надання можливостей, які забезпечує система шляхом підключення створених бібліотек функцій введення/виведення графічної та текстової інформації на екран.

Після написання програмного продукту його потрібно впровадити в промислову експлуатацію. При цьому виправляються помилки, які були помічені, система настроюється на відповідний режим роботи. Для налаштування системи на оптимальні умови роботи необхідно виконати наступні кроки:

– Встановити додаткові бібліотеки налагодження ОС;

– Налагодити параметри запуску розробленої програми.

Після запуску EXE файлу програми починається початковий етап роботи ПЗ. Ініціалізуються початкові значення та змінні, завантажуються модулі ПЗ. Після чого модуль захисту ПЗ на стадії початкового завантаження проводить перевірку програми та виводить на екран вікно завантаження програми, далі здійснюється приховання вікна завантаження та виведення головного вікна програми (рисунок 5.1).

За допомогою меню навігації головного вікна програми здійснюються наступні дії:

– запустити та зупинити програмну модель на виконання;

– налагодити ПЗ;

– встановити використовуваємі підсистеми;

– встановити особливості перегляду програмної моделі (які підсистеми будуть задіяні) – повна програмна модель, часткова, задіяння дискової підсистеми, пам`яті, системи;

– переглядати додаткову інформацію;
– задіяти механізм завершення роботи програми;
– вивести на екран форму авторських даних.

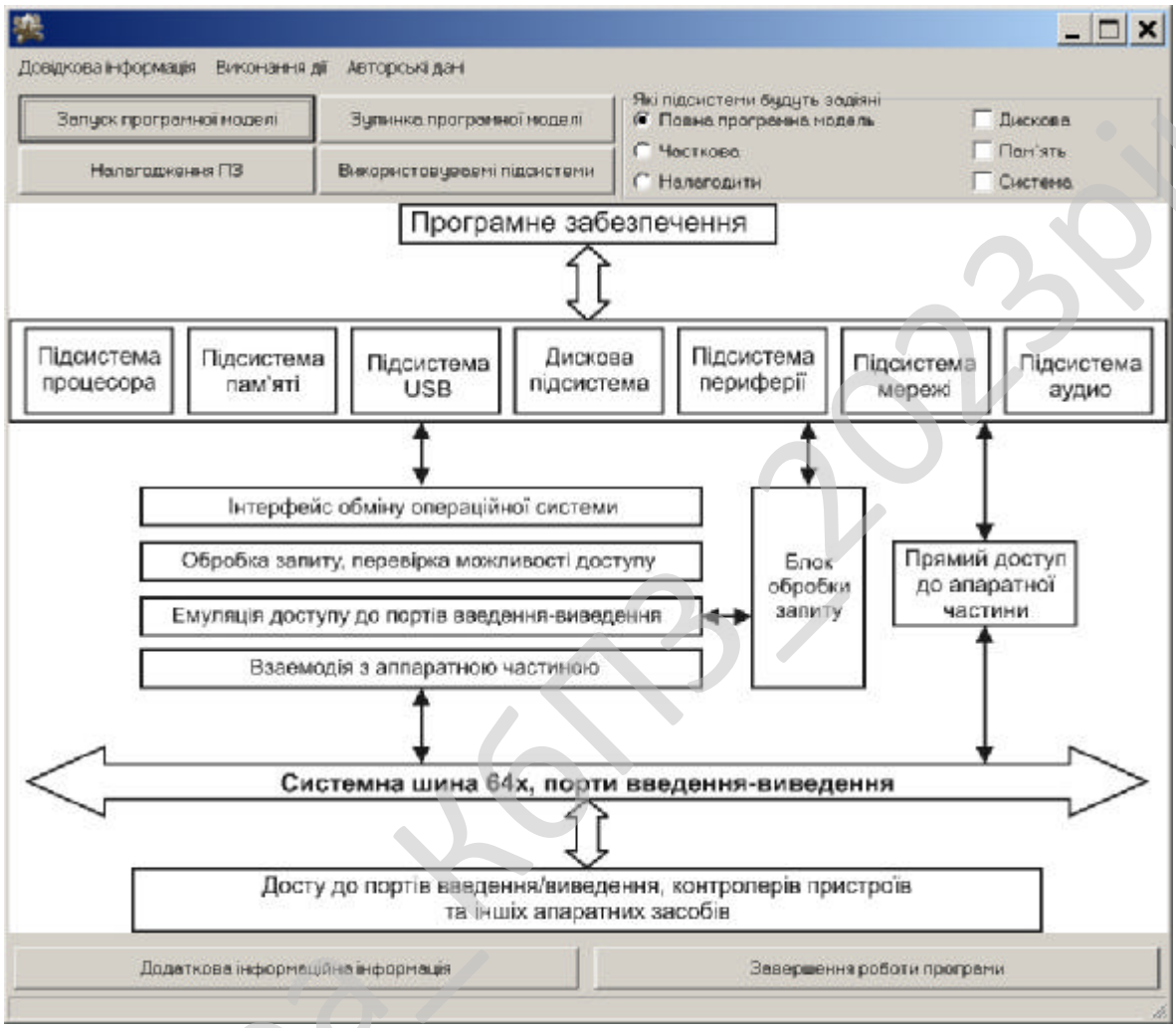


Рисунок 5.1 – Головне вікно програми

З головного вікна програми можливо здійснити перехід на вікна схеми взаємодії, моделювання, довідкової системи, авторського права. Після проведення процесу моделювання та натискання кнопки завершення роботи ПЗ здійснюється згортання програми в трєй та завершення її роботи з перевіркою цілісності програми у модулі захисту, звільненням ресурсів програми.

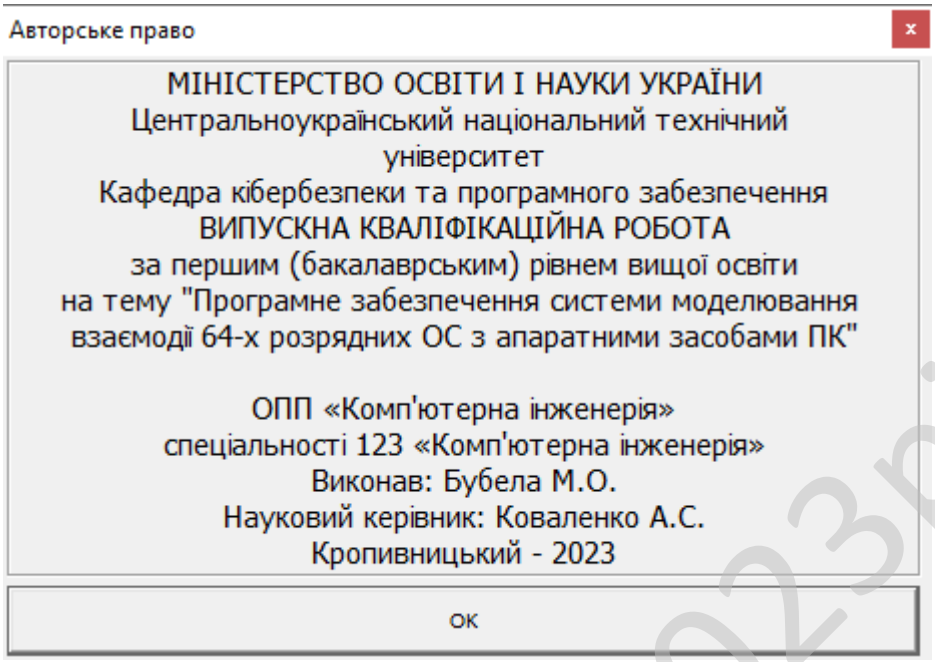


Рисунок 5.2 – Авторська форма програми

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти, призначено для системи моделювання взаємодії 64-х розрядних ОС з апаратними засобами ПК.

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

Рішення завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем моделювання взаємодії 64-х розрядних ОС з апаратними засобами ПК.

– Досліджена система моделювання взаємодії 64-х розрядних ОС з апаратними засобами ПК.

– На основі отриманих результатів досліджень створена програмна реалізація системи моделювання взаємодії 64-х розрядних ОС з апаратними засобами ПК.

Розроблені під час виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання моделювання взаємодії 64-х розрядних ОС з апаратними засобами ПК.

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 Delphi 10.4. Дана мова програмування дозволяє найбільш ефективно обробляти дані призначені для системи моделювання взаємодії 64-х розрядних ОС з апаратними засобами ПК.

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 ДСТУ 9041:2020.

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.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Kovalenko O., Popereshnyak S., Grinenko S., Grinenko O., Radivilova T. «Methods for Assessing the Maturity Levels of Software Ecosystems». *CEUR Workshop Proceedings* Volume 2654, 2019, Pages 251-261. Режим доступу: https://www.scopus.com/record/display.uri?eid=2-s2.0-85083214331&origin=AuthorNamesList&txGid=5633fba897776a6e0f3d5633fbcd3fbe **(Scopus)**.

2. Kovalenko O., Drieieva H., Simakhin V., Bondar S., Drieiev O., Zhumadilova M. «Multifractal Properties of Traffic Generator Based on Markov Chains ». *CEUR Workshop Proceedings* Volume 2588, 2019, Pages 567-579. Режим доступу: https://www.scopus.com/record/display.uri?eid=2-s2.0-85083214331&origin=AuthorNamesList&txGid=176e2cada8976a6e0f3d5633fbcd3fbe **(Scopus)**.

3. Kovalenko Oleksandr Qualitative risk analysis of software development / Oleksandr Kovalenko, Jamil Al-Azzeh, Oleksii Smirnov, Anna Kovalenko, Serhii Smirnov // Asian Journal of Information Technology. – Volume 17 Issue 3. – Medwell Journals. – 2018. – P. 218-230. ISSN: 1682-3915. URL: http://medwelljournals.com/abstract/?doi=ajit.2018.218.230 Doi: ajit.2018.218.230

4. Kovalenko Oleksandr, The mathematical model of the testing technology for DOM XSS vulnerabilities / O. Kovalenko, O. Smirnov, A.Kovalenko, S. Smirnov, V. Vialkova // Scientific & practical cyber security journal (SPCSJ) Volume 2 Issue 1, P. 22-28. Georgia. Tbilisi. Scientific Cyber Security Association (SCSA), 2018 ISSN: 2587-4667. URL: https://journal.scsa.ge/wp-content/uploads/2018/12/04-3-o.kovalenko-a.kovalenko-o.smirnov-s.smirnov-v.vialkova.pdf

5. Коваленко А.В. Технология тестирования DOM XSS уязвимости / А.В. Коваленко, А.С. Коваленко, А.А. Смирнов, С.А. Смирнов // Scientific & practical cyber security journal (SPCSJ) Volume 1. Issue 1. P. 38-45 Georgia. Tbilisi. Scientific Cyber Security Association (SCSA), 2017 ISSN: 2587-

4667. URL: https://journal.scsa.ge/wp-content/uploads/2018/12/8-dom-xss-testing-technology-vulnerabilities.pdf

6. Коваленко О.В. Моделі та методи розроблення програмного забезпечення комп'ютерних систем для підвищення безпеки даних: монографія / О.В. Коваленко // К.: Вид. «КОД» – 2019. – 305 с.

7. Коваленко А.В. Методы качественного анализа и количественной оценки рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Информационные технологии в управлении, образовании, науке и промышленности: монография / Под редакцией профессора В.С. Пономаренко. – Х.: Видавець Рожко С.Г., 2016. – 566 с.

8. Коваленко А.В. Разработка метода управления рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Інформаційні технології: проблеми та перспективи: монографія / За загальною редакцією В.С. Пономаренка. – Х.: Видавець Рожко С.Г., 2017. – 447 с.

9. Коваленко А.В. Комплекс математических моделей технологии тестирования web-приложений / А.В. Коваленко, А.А. Смирнов // Інформаційні технології: сучасний стан та перспективи: монографія / За загальною редакцією В.С. Пономаренка. – Х.: ТОВ «ДІСА ПЛЮС», 2018. – 461 с.

10. Коваленко А.В. Задачи распознавания ситуаций в ERP системах / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко // Збірник наукових праць "Системи обробки інформації". – Випуск 4(120). – Х.: ХУПС – 2014. – С. 161-164.

11. Коваленко А.В. Методы качественного анализа и количественной оценки рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник наукових праць "Системи обробки інформації". – Випуск 5(142). – Х.: ХУПС – 2016. – С. 153-157.

12. Коваленко А.В. Проблемы анализа и оценки рисков информационной деятельности / А.В. Коваленко, А.А. Смирнов, Н.Н. Якименко, А.П. Доренский // Збірник наукових праць "Системи обробки інформації". –

Випуск 3(140). – Х.: ХУПС – 2016. – С. 40-42.

13. Коваленко А.В. Метод качественного анализа рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, Н.Н. Якименко, А.П. Доренский // Наука і техніка Повітряних Сил Збройних Сил України. – Випуск 2(23). – Харків: ХУПС. – 2016. – С. 150-158.

14. Коваленко А.В. Метод количественной оценки рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, Н.Н. Якименко, А.П. Доренский // Збірник наукових праць Харківського університету Повітряних Сил. Випуск 2 (47). – Харків: ХУПС. – 2016. – С. 128-133.

15. Коваленко А.В. Использование псевдобулевых методов бивалентного программирования для управления рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Системи управління, навігації та зв'язку. – Випуск 1 (37). – Полтава: ПолтНТУ. – 2016. – С. 98-103.

16. Коваленко А.В. Метод управления рисками разработки программного обеспечения / А.В. Коваленко // Системи управління, навігації та зв'язку. – Випуск 2 (38). – Полтава: ПолтНТУ. – 2016. – С. 93-100.

17. Коваленко А.В. Технология тестирования уязвимости к SQL инъекциям / А.В. Коваленко // Системи управління, навігації та зв'язку. – Випуск 5 (45). – Полтава: ПолтНТУ. – 2017. – С. 66-71.

18. Коваленко А.В. Масштабирование имитационной модели технологии тестирования безопасности / А.В. Коваленко // Системи управління, навігації та зв'язку. – Випуск 6 (46). – Полтава: ПолтНТУ. – 2017. – С. 181-184.

19. Коваленко А.В. Имитационная модель технологии тестирования безопасности Web-приложений / А.В. Коваленко // Системи управління, навігації та зв'язку. – Випуск 1 (47). – Полтава: ПолтНТУ. – 2018. – С. 114-123.

20. Коваленко О.В. Методи якісного аналізу та кількісної оцінки ризиків розроблення програмного забезпечення/ О.В. Коваленко // Системи управління, навігації та зв'язку. – Випуск 3 (49). – Полтава: ПолтНТУ. – 2018. – С. 116-125.

21. Коваленко О.В. Управління ризиками розроблення програмного

забезпечення за умови обмеженості коштів виділених на усунення помилок безпеки/ О.В. Коваленко // Техніка в сільськогосподарському виробництві, галузеве машинобудування, автоматизація. – Випуск 31. – Кропивницький: ЦНТУ. – 2018. – С. 128-140.

22. Коваленко О.В. GERT-мережевий синтез технології тестування на вразливість WEB-додатків/ О.В. Коваленко // Захист інформації. – Випуск 20(2) – К.: НАУ. – 2018. – С. 89-94.

23. Коваленко О.В. Імітаційна модель технології тестування безпеки на основі положень теорії масштабування / О.В. Коваленко // Безпека інформації. – Випуск 24 (2). – К.: НАУ. – 2018. – С. 110-117.

24. Коваленко О.В. Оцінка ефективності технології тестування безпеки / О.В. Коваленко // Вчені записки Таврійського національного університету імені В.І. Вернадського. Серія: Технічні науки. Том 29 (68) № 2, 2018. – С. 137-141

25. Коваленко О.В. Методи та засоби управління безпекою додатків / О.В. Коваленко // Інформаційно-керуючі системи на залізничному транспорті. №4, 2018. – С. 41-44.

26. Коваленко О.В. Розробка інформаційної технології передтестової компіляції та розподілу доступу / О.В. Коваленко // Системи управління, навігації та зв'язку. – Випуск 4 (50). – Полтава: ПолтНТУ. – 2018. – С. 115-119.

27. Коваленко О.В. Аналіз та дослідження інформаційних технологій розробки програмного забезпечення/ О.В. Коваленко // Вчені записки Таврійського національного університету імені В.І. Вернадського. Серія: Технічні науки. Том 29 (68) № 5, 2018. – С. 131-137.

28. Коваленко О.В. Удосконалений метод управління ризиками розроблення програмного забезпечення на основі напівмарковської моделі прийняття рішень/ О.В. Коваленко // Сучасні інформаційні системи. – Випуск 2(3). – Харків. – 2018. – С. 41-48.

29. Коваленко О.В. Математичні моделі технології тестування DOM XSS

вразливості та вразливості до SQL ін'єкцій / О.В. Коваленко // Вісник Черкаського державного технологічного університету. Серія : Технічні науки №4, 2018. – С. 29-36.

30. Коваленко О.В. Математична модель технології тестування вразливості до SQL ін’єкцій / О.В. Коваленко // Системи управління, навігації та зв’язку. – Випуск 6 (58). – Полтава: ПолтНТУ. – 2019. – С. 43-47.

31. Коваленко О.В. Математична модель технології тестування комплексу DOM XSS вразливостей для аналітичної оцінки часових витрат / О.В. Коваленко // Центральноукраїнський науковий вісник. Технічні науки. № 2(33). с. 173-180, 2019.

32. Коваленко А.В. Проблемы анализа и оценки рисков информационной деятельности / А.В. Коваленко, А.А. Смирнов // Збірник наукових праць II міжнародної науково-практичної конференції «Актуальні питання забезпечення кібернетичної безпеки та захисту інформації». м. Київ. 24-27 лютого 2016 р. – Київ: Європейський університет. – 2016. – С. 138-139.

33. Коваленко А.В. Анализ и оценка рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез «Securitea informationala 2015-2016». Conferenta internationala (editia a XII-a). Chisinau. Moldova. 3 martie 2016. – Chisinau: ADSEM. – 2016. – P. 96-102.

34. Коваленко А.В. Исследование источников и причин риска разработки программного обеспечения, этапов и работ, при выполнении которых возникает риск / А.В. Коваленко, А.А. Смирнов // Збірник тез VII всеукраїнської науково-практичної конференції "Інформатика та системні науки (ІСН-2016)". м. Полтава. 10-12 березня 2016 р. – Полтава.: ПУЕТ – 2016. – С. 264-266.

35. Коваленко А.В. Оценка показателя чистой приведенной стоимости для количественной оценки рисков проекта разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез науково-практичної конференції “Проблеми кібербезпеки інформаційно-телекомунікаційних

систем”. м. Київ. 10-11 березня 2016 р. – Київ: КНУ ім. Тараса Шевченко – 2016. – С. 81-82.

36. Коваленко А.В. Методика структурной идентификации рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез Міжнародної науково-практичної конференції «Інформаційна безпека та комп'ютерні технології» (IS&CT). м. Кіровоград. 24-25 березня 2016 р. – Кіровоград: КНТУ. – 2016. – С. 71-72.

37. Коваленко А.В. Методы качественного анализа рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез першої міжнародної науково-практичної конференції «Проблеми науково-технічного та правового забезпечення кібербезпеки у сучасному світі» (ПНПЗК-2016). м. Харків. 30 березня – 1 квітня 2016 р. – Харків: НТУ «ХПІ». – 2016. – С. 6-7.

38. Коваленко А.В. Структурная идентификация рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез XVIII міжнародного науково-практичного семінару «Комбінаторні конфігурації та їх застосування». м. Кіровоград. 15-16 квітня 2016 р. – Кіровоград: КНТУ. – 2016. – С. 175-182.

39. Коваленко А.В. Исследование разработанной методики структурной идентификации рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез VIII міжнародної науково-практичної конференції “Проблеми і перспективи розвитку ІТ-індустрії”. м. Харків. 28-29 квітня 2016 р. – Харків: ХНЕУ. – 2016. – С. 49.

40. Коваленко А.В. Исследование дерева рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез III міжнародної науково-практичної конференції «Інформаційна та економічна безпека» (INFECO-2016)». м. Харків. 28-30 квітня 2016 р. – Харків: ХННІ ДВНЗ «УБС». – 2016. – С. 174-178.

41. Коваленко А.В. Методы качественного анализа и количественной

оценки рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко // Сборник тезисов XII международной конференции "Стратегия качества в промышленности и образовании". г. Варна. Болгария. 30 мая – 02 июня 2016 г – Варна. ТУВ. – 2016. – С. 585-589.

42. Коваленко А.В. Разработка метода управления рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко // Матеріали Всеукраїнської науково-практичної конференції «Кібербезпека в Україні: правові та організаційні питання». м. Одеса, 21 жовтня 2016 р. – Одеса : ОДУВС, 2016. – С.146-148.

43. Коваленко А.В. Метод управления рисками разработки программного обеспечения с использованием псевдобулевых методов бивалентного программирования / А.В. Коваленко, А.А. Смирнов // Матеріали Всеукраїнської науково-практичної конференції «Актуальні задачі та досягнення у галузі кібербезпеки». м. Кропивницький, 23-25 листопада 2016 року – Кропивницький: ЦНТУ, 2016. – С. 162.

44. Коваленко А.В. Псевдобулевые методы бивалентного программирования для управления рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко, С.А. Смирнов // Збірник наукових праць III міжнародної науково-практичної конференції «Актуальні питання забезпечення кібернетичної безпеки та захисту інформації». м. Київ. 22-25 лютого 2017 р. – Київ: Європейський університет. – 2017. – С. 158-162.

45. Коваленко А.В. Метод управления рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез II науково-практичної конференції “Проблеми кібербезпеки інформаційно-телекомунікаційних систем”. м. Київ. 23-24 березня 2017 р. – Київ: КНУ ім. Тараса Шевченко – 2017. – С. 203-205.

46. Коваленко А.В. Алгоритмы анализа уязвимостей при управлении рисками разработки программного обеспечения / А.В. Коваленко,

А.А. Смирнов, А.С. Коваленко // Conferenta internationala (editia a XIII-a). «Securitea informationala 2017». Chisinau. Republic of Moldova. 4-5 aprilie 2017. – Chisinau: ADSEM. – 2017. – P. 19-22.

47. Коваленко А.В. Алгоритм анализа DOM XSS уязвимости при управлении рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко // Збірник тез дев'ятнадцятого міжнародного науково-практичного семінару «Комбінаторні конфігурації та їх застосування». м. Кропивницький 7-8 квітня 2017 р. – Кропивницький: ГЛА НАУ. – 2017. – С. 125-127.

48. Коваленко А.В. Алгоритм анализа уязвимости SQL Injection для управления рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко // Збірник тез другої міжнародної науково-технічної конференції «Проблеми науково-технічного та правового забезпечення кібербезпеки у сучасному світі» (ПНПЗК-2017). м. Харків.10-12 квітня 2017 р. – Харків: НТУ «ХПІ». – 2017. – С. 27.

49. Коваленко А.В. Метод управления рисками разработки программного обеспечения на основе алгоритмов анализа уязвимостей / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко // Збірник тез Міжнародної науково-практичної конференції «Інформаційна безпека та комп'ютерні технології» (IS&CT). м. Кіровоград. 20-22 квітня 2017 р. Кіровоград: КНТУ. – 2017. – С. 92.

50. Коваленко А.В. Алгоритмы анализа DOM XSS уязвимости и уязвимости SQL Injection при управлении рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко // Збірник тез IX міжнародної науково-практичної конференції "Проблеми і перспективи розвитку IT-індустрії". м. Харків. 20-21 квітня 2017 р. – Харків: ХНЕУ. – 2017. – С. 61.

51. Kovalenko O.V. Method of testing the dom xss vulnerability / Kovalenko Oleksandr, Kovalenko Anna, Smirnov Oleksii, Smirnov Serhii // International

Conference «information technologies, systems and networks ITSN-2017». Chisinau, Republic of Moldova. 17 – 18 October 2017. – Chisinau: Academy of Sciences of Moldova, Military Academy of Armed Forces “Alexandru cel Bun”. – 2017. – P. 7.

52. Коваленко О.В. Метод тестування DOM XSS уразливості / О.В. Коваленко, О.А. Смірнов, А.С. Коваленко, С.А. Смірнов // Збірник тез всеукраїнської науково-практичної інтернет-конференції «Автоматика та комп’ютерно-інтегровані технології у промисловості, телекомунікаціях, енергетиці та транспорті». м. Кропивницький. 16-17 листопада 2017 р. – Кропивницький: ЦНТУ. – 2017. – С. 198-199.

53. Коваленко О.В. GERT-модель технології тестування DOM XSS уразливості / О.В. Коваленко, А.С. Коваленко, О.А. Смірнов, С.А. Смірнов // Збірник наукових праць IV міжнародної науково-практичної конференції «Актуальні питання забезпечення кібернетичної безпеки та захисту інформації». м. Київ. 21-24 лютого 2018 р. – Київ: Європейський університет. – 2018. – С. 65-70.

54. Коваленко О.В. Технології тестування уразливостей Web-застосунків з використанням GERT-моделі / О.В. Коваленко, А.С. Коваленко, О.А. Смірнов, С.А. Смірнов // Збірник тез всеукраїнської науково-практичної конференції "Комп'ютерні інтелектуальні системи та мережі (КІСМ-2018)". м. Кривий Ріг. 21-23 березня 2018 р. – Кривий Ріг.: ДВНЗ КНУ – 2018. – С. 227-230.

55. Коваленко А.В. Тестирование уязвимости Web-приложений к атаке вида межсайтовый скриптинг / А.В. Коваленко, А.С. Коваленко, А.А. Смирнов, С.А. Смирнов // Збірник тез «Securitea informationala 2018». Conferenta internationala (editia a XIV-a). Chisinau. Moldova. 20-21 martie 2018. – Chisinau: ADSEM. – 2018. – P. 54-56.

56. Коваленко А.В. Комплекс математических моделей технологии тестирования web-приложений / А.В. Коваленко, А.С. Коваленко, А.А. Смирнов, С.А. Смирнов // Збірник тез X міжнародної науково-практичної

конференції “Проблеми і перспективи розвитку ІТ-індустрії”. м. Харків. 19-20 квітня 2018 р. – Харків: ХНЕУ. – 2018. – С. 38.

57. Коваленко О.В. Розробка методу передтестової компіляції й розподілу доступу / О.В. Коваленко, А.С. Коваленко, О.А. Смірнов, С.А. Смірнов // Збірник наукових праць III міжнародної науково-практичної конференції “Інформаційна безпека та комп’ютерні технології”, м. Кропивницькй. 19-20 квітня 2018 р. – Кропивницький: ЦНТУ. – 2018. – С. 214-215.

58. Коваленко О.В. Оцінка ефективності технологій тестування безпеки уразливостей DOM XSS й SQL-ін'єкцій / О.В. Коваленко, А.С. Коваленко, О.А. Смірнов, С.А. Смірнов // Сборник тезисов XIV международной конференции "Стратегия качества в промышленности и образовании", Варна, Болгария. 04-07 июня 2018 г – Варна. ТУВ. – 2018. – С. 271-274.

59. Коваленко О.В. Аналіз основних підходів математичного моделювання та методологій для забезпечення максимальних показників безпеки програмного забезпечення / О.В. Коваленко, А.С. Коваленко // Збірник наукових праць всеукр. наук.-практ. конф. здобувачів вищої освіти й молодих учених «Комп’ютерна інженерія і кібербезпека : досягнення та інновації, м. Кропивницькй. 27-29 листопада

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку системи моделювання взаємодії 64-х розрядних ОС з апаратними засобами ПК.

2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 7-02 від 5.01.2023 року).

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є розробка програмного забезпечення системи моделювання взаємодії 64-х розрядних ОС з апаратними засобами ПК.

4 Джерела розробки

Джерелом цієї випускної кваліфікаційної роботи за першим (бакалаврським) 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;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– системи моделювання взаємодії 64-х розрядних ОС з апаратними засобами ПК;

– 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 Delphi 10.4.

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 Перелік документів, що розробляються

– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 2 аркуша.
– Пояснювальна записка – 78 аркушів.

8 Етапи розробки

8.1 Збір і обробка інформації по темі випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти. Постановка задачі на виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти (складання ТЗ).

8.2 Проведення досліджень або експериментальних робіт для уточнення основних положень випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти.

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

8.4 Побудова схем взаємодії даних.

8.5 Створення прототипу ПЗ.

8.6 Віднаходження ПЗ, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

11 Порядок контролю та приймання

11.1 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на попередній захист 23.05.2023 р.

11.2 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на захист 5.06.2023 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи за

першим (бакалаврським) рівнем вищої освіти

_____ Коваленко А.С.

Програмне забезпечення системи моделювання взаємодії 64-х розрядних ОС з апаратними засобами ПК

Лістинг програми

Код документу 12

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

Загальна кількість аркушів: 38

Літера: РП

Кропивницький – 2023 року

Основний файл проекту розробленого програмного забезпечення (Program_Model_Vista.dpr)

```
program ProgramModelVista;

uses
  Forms,
  Messages,Graphics,
  Windows,Unit1 in 'Unit1.pas' {Form1},
  Unit2 in 'BOLUKReport.pas' {Form2},
  Unit3 in 'fset.pas' {Form3},
  Unit4 in 'data.pas' {Form4},
  Unit5 in 'VistaEmulation.pas' {Form5},
  Unit6 in 'U6.pas' {Form6},
  U_splash in 'U7.pas' {U_Form_Splash}.

const
WM_CONST1=WM_USER+322; WM_CONST2=WM_USER+105; WM_CONST3=WM_USER+99;

{$R *.res}

///Розробив Бубела Максим Олегович ЦНТУ 2023

begin
  Application.HintPause:=400; Application.HintHidePause:=1000;
  Application.Title:= ProgramModelVista';
  Application.Initialize;
try
    U_Form_Splash:=TU_Form_Splash.Create(Application);
    U_Form_Splash.Show;
    U_Form_Splash.Update;
  Sendmessage(U_Form_Splash.Handle,WM_MY,0,'Start');
  Application.CreateForm(TForm1, Form1);
  Application.CreateForm(TForm2, Form2);
  Application.CreateForm(TForm3, Form3);
  Application.CreateForm(TForm4, Form4);
  Sendmessage(U_Form_Splash.Handle,WM_MY,0,'End');
  Application.CreateForm(TForm5, Form5);
Finally U_Form_Splash.free; end;
  Application.Run;
end.
```

Файл форми взаємодії та відображення звіту роботи
(BOLUKReport.pas)

```
unit BOLUKReport;

interface

uses
Windows, glReport, glCapt, glBevel, glPage, Printers,
glLabel, glRuler, Mask, glLBox, dsgnintf, Spin, geRPEdit,
Menus, ExtCtrls, StdCtrls, Buttons, ComCtrls, Controls, Dialogs,
Forms, Classes,Sysutils;
///Розробив Бубела Максим Олегович ЦНТУ 2023

type

tmyRep_Property = class( TPropertyEditor )
function GetAttributes: TPropertyAttributes; override;
function GetValue: string; override;
procedure Edit; override;
end;

tmyRep_Editor = class(TComponentEditor)
procedure ExecuteVerb(Index: Integer); override;
function GetVerb(Index: Integer): string; override;
function GetVerbCount: Integer; override;
end;

tmyBOLUKReportEditor = class(TComponent)
FBOLUKReport: tmyBOLUKReport;
protected
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
public
procedure Preview;
procedure Edit;
published
property BOLUKReport: tmyBOLUKReport read FBOLUKReport write FBOLUKReport;
end;

tmyRepEditor = class(TForm) //основний классвзаємодії
OpenDialog1: TOpenDialog; SaveDialog1: TSaveDialog;
PM_Control: TPopupMenu; N_Linktofile: TMenuItem;
N1: TMenuItem; N2: TMenuItem;
PC: tmyPageControl; Panel1: TPanel;
Bevel4: TBevel; P_Sides: TPanel;
Panel2: TPanel; glBevel1: tmyBevel;
Bevel2: TBevel; Bevel1: TBevel;
B_Label: TSpeedButton; sb_Open: TSpeedButton;
sb_Save: TSpeedButton; sb_Preview: TSpeedButton;
sb_Book: TSpeedButton; sb_Album: TSpeedButton;
Bevel3: TBevel; sb_OLE: TSpeedButton;
sb_SnapToGrid: TSpeedButton; b_Bevel: TSpeedButton;
sb_Print: TSpeedButton; TabSheet1: TTabSheet;
TabSheet2: TTabSheet; Memo1: TMemo;
P_Font: TPanel; ColorDialog1: TColorDialog;
N3: TMenuItem; N_DeleteObject: TMenuItem;
P_HRuler: TPanel; P_Main: TPanel;
ScrollBox_: TScrollBox; ShapeSize: TShape;
P_VRuler: TPanel; TabSheet3: TTabSheet;
ImageList1: TImageList; HRuler: tmyRuler;
VRuler: tmyRuler; glBevel4: tmyBevel;
sb_FixAllMoving: TSpeedButton; sb_FixMoving: TSpeedButton;
glLabel3: TLabel; N_OLESize: TMenuItem;
N_Clip: TMenuItem; N_Center: TMenuItem;
N_Scale: TMenuItem; N_Stretch: TMenuItem;
N_AutoSize: TMenuItem; P_SBar: TPanel;
Panel5: TPanel; glLabel4: TLabel;
glLabel5: TLabel; glLabel7: TLabel;
```

```
se_Width: TSpinEdit; se_Top: TSpinEdit;
se_Left: TSpinEdit; glLabel6: TLabel;
se_Height: TSpinEdit; cb_Components: TComboBox;
glLabel8: TLabel; N4: TMenuItem;
Bevel5: TBevel; SB_Left: TSpeedButton;
SB_Bottom: TSpeedButton; SB_Right: TSpeedButton;
SB_Top: TSpeedButton; sb_AlignL: TSpeedButton;
sb_AlignC: TSpeedButton; sb_AlignR: TSpeedButton;
sb_AlignW: TSpeedButton; RxSpeedButton8: TSpeedButton;
RxSpeedButton9: TSpeedButton; sb_BevelBold: TSpeedButton;
glBevel2: tmyBevel; Panel3: TPanel;
RxSpinEdit1: TSpinEdit; Panel6: TPanel;
sbFontColor: TSpeedButton; glBevel3: tmyBevel;
Panel7: TPanel; Edit1: TMemo;
FE_OLE: TEdit; SpeedButton1: TSpeedButton;
OpenOLEFile: TOpenDialog; sb_FontUnderline: TSpeedButton;
sb_FontItalic: TSpeedButton; sb_FontBold: TSpeedButton;
TabSheet4: TTabSheet; glLabel1: TLabel;
lb_Params: tmyListBox; Panel4: TPanel;
SpeedButton2: TSpeedButton; sbBackColor: TSpeedButton;
SpeedButton3: TSpeedButton; CheckBox1: TCheckBox;
FontComboBox1: TComboBox; ColorComboBox1: TComboBox;
procedure OpenClick(Sender: TObject);
procedure Save1Click(Sender: TObject);
procedure ScrollBox_MouseDown(Sender: TObject; Button: TMouseButton; Shift:
TShiftState; X, Y: Integer);
procedure FormCreate(Sender: TObject);
procedure Edit1Change(Sender: TObject);
procedure SB_LeftClick(Sender: TObject);
procedure FontComboBox1Change(Sender: TObject);
procedure RxSpinEdit1Change(Sender: TObject);
procedure ColorComboBox1Change(Sender: TObject);
procedure FormShow(Sender: TObject);
procedure sb_BookClick(Sender: TObject);
procedure N1Click(Sender: TObject);
procedure sb_FontBoldClick(Sender: TObject);
procedure sb_AlignLClick(Sender: TObject);
procedure Memo1Change(Sender: TObject);
procedure sbFontColorClick(Sender: TObject);
procedure N_DeleteObjectClick(Sender: TObject);
procedure ScrollBox_MouseMove(Sender: TObject; Shift: TShiftState; X,
Y: Integer);
procedure ScrollBox_MouseUp(Sender: TObject; Button: TMouseButton;
Shift: TShiftState; X, Y: Integer);
procedure FormDestroy(Sender: TObject);
procedure FormKeyDown(Sender: TObject; var Key: Word;
Shift: TShiftState);
procedure sb_FixMovingClick(Sender: TObject);
procedure N_AutoSizeClick(Sender: TObject);
procedure sb_SnapToGridClick(Sender: TObject);
procedure se_SizeChange(Sender: TObject);
procedure cb_ComponentsChange(Sender: TObject);
procedure N4Click(Sender: TObject);
procedure sb_BevelBoldClick(Sender: TObject);
procedure se_TopChange(Sender: TObject);
procedure se_WidthChange(Sender: TObject);
procedure se_HeightChange(Sender: TObject);
procedure FormClose(Sender: TObject; var Action: TCloseAction);
procedure FE_OLEChange(Sender: TObject);
procedure SpeedButton1Click(Sender: TObject);
procedure sb_PrintClick(Sender: TObject);
procedure sb_PreviewClick(Sender: TObject);
procedure SpeedButton2Click(Sender: TObject);
procedure cb_ComponentsKeyDown(Sender: TObject; var Key: Word;
Shift: TShiftState);
procedure CheckBox1Click(Sender: TObject);
private
ScrollBox: TlaScrollBox;
SelectedControl: tmyBOLUKReportItem;
```

```
fSelection: boolean;
SelectionRect: TRect;
procedure RemakeComponentsList;
procedure read( FileName: string; ParentWnd: TWinControl );
procedure Save( FileName: string );
procedure OnMouseDown_(Sender: TObject; Button: TMouseButton; Shift:
TShiftState; X, Y: Integer);
procedure OnMouseUp_(Sender: TObject; Button: TMouseButton; Shift: TShiftState;
X, Y: Integer);
procedure OnMouseMove_(Sender: TObject; Shift: TShiftState; X, Y: Integer);
procedure OnResize_(Sender: TObject);
procedure ShowComponentPos(Control: TControl);
procedure AssignEventsToAllComponents;
procedure UpdateToolBar( Control: tmyBOLUKReportItem);
public
Component: tmyBOLUKReport;
procedure Preview(glBOLUKReport: tmyBOLUKReport);
procedure Edit(glBOLUKReport: tmyBOLUKReport);
end;

TPublicControl = class(TControl)
public
property Caption;
end;

TPublicControlClass = class of TPublicControl;

const
IGNORE_VALUE = 65536;
var
glRepEditor: tmyRepEditor;
Form2: TComponent;

implementation
uses glTypes, glUtils;
{$R *.DFM}

procedure tmyRepEditor.OnMouseMove_(Sender: TObject; Shift: TShiftState; X, Y:
Integer);
var
DC: HDC;
i: integer;
begin
if fSelection then ScrollBox_MouseMove(Sender, Shift, X, Y);
if sb_FixAllMoving.Down then exit;
if TControl(Sender).Tag = 0 then exit;
if not fMouseDown then exit;
with ScrollBox do
for i:=0 to ControlCount-1 do
if (Controls[i] is tmyBOLUKReportItem) then with tmyBOLUKReportItem(Controls[i])
do
if Selected and not bool(Fixed) then
begin
Left:= ((Left + X - ControlPos.x)div Step.X)*Step.X;
Top:= ((Top + Y - ControlPos.y)div Step.Y)*Step.Y;
end;
fSkipSizeUpdate:= true;
ShowComponentPos(SelectedControl);
TControl(Sender).Left:= TControl(Sender).Left + X - ControlPos.x;
TControl(Sender).Top:= TControl(Sender).Top + Y - ControlPos.y;
{
TControl(Sender).Tag:= 2;//...on moving
DC:= GetDC( TControl(Sender).Parent.Handle );
DrawFocusRect( DC, FocusRect );
FocusRect:= Bounds( TControl(Sender).Left+X-ControlPos.x,
TControl(Sender).Top+Y-ControlPos.y,  SelectedControl.Width,
SelectedControl.Height );
DrawFocusRect( DC, FocusRect );
ReleaseDC( TControl(Sender).Parent.Handle, DC );
```

```
}
end;

procedure tmyRepEditor.OnResize_(Sender: TObject);
begin
fSkipSizeUpdate:= true;
if Sender = SelectedControl then ShowComponentPos(TControl(Sender));
end;

procedure tmyRepEditor.read( FileName: string; ParentWnd: TWinControl );
begin
ScrollBox.HorzScrollBar.Position:= 0;
ScrollBox.VertScrollBar.Position:= 0;
SelectedControl:= nil;
UpdateToolBar( nil );
Component.LoadFromFile( FileName );
Component.CreateBOLUKReport( ParentWnd, true );
AssignEventsToAllComponents;
RemakeComponentsList;
end;

procedure tmyRepEditor.Save( FileName: string );
begin
ScrollBox.HorzScrollBar.Position:= 0;
ScrollBox.VertScrollBar.Position:= 0;
Component.SaveToFile( FileName );
end;

procedure tmyRepEditor.OpenClick(Sender: TObject);
begin
OpenDialog1.InitialDir:= ExtractFilePath(ParamStr(0));
if OpenDialog1.Execute then
Read( OpenDialog1.FileName, ScrollBox );
end;

procedure tmyRepEditor.Save1Click(Sender: TObject);
begin
SaveDialog1.InitialDir:= ExtractFilePath(ParamStr(0));
if SaveDialog1.Execute then Save( SaveDialog1.FileName );
end;

procedure tmyRepEditor.ScrollBox_MouseDown(Sender: TObject; Button:
TMouseButton;
Shift: TShiftState; X, Y: Integer);
var
l, Compon: tmyBOLUKReportItem;
R: TRect;
pt: TPoint;
begin
if ssCtrl in Shift then
begin
SelectionRect:=  Rect( 0,0,0,0 );
SelPt.X:= X - ScrollBox.HorzScrollBar.Position;
SelPt.Y:= Y - ScrollBox.VertScrollBar.Position;
SelPt:= ScrollBox.ClientToScreen(SelPt);
fSelection:= true;
end;
if (B_Label.Down)or(B_Bevel.Down)or(sb_OLE.Down) then
begin
Compon:= Component.AddComponent;
with Compon do
begin
Left:= X - ScrollBox.HorzScrollBar.Position;
Top:= Y - ScrollBox.VertScrollBar.Position;
if B_Label.Down then
begin
SideLeft:= 0;
SideTop := 0;
SideRight:= 0;
```

```
SideBottom:= 0;
end;
OnMouseDown:= OnMouseDown_;
OnMouseUp:= OnMouseUp_;
OnMouseMove:= OnMouseMove_;
OnResize:= OnResize_;
ContainOLE:= sb_OLE.Down;
B_Label.Down:= false;
B_Bevel.Down:= false;
sb_OLE.Down:= false;
RemakeComponentsList;
end;
end else
begin
R:= ScrollBox.ClientRect;
pt.x:= 0; pt.y:= 0; pt:= ScrollBox.ClientToScreen(pt);
OffsetRect( R, pt.x, pt.y );
ClipCursor( @R );
end;
end;

procedure tmyRepEditor.ScrollBox_MouseMove(Sender: TObject; Shift: TShiftState;
X, Y: Integer);
var
DC: HDC;
pt: TPoint;
begin

if not fSelection then exit;
DC:= GetDC( 0 );
DrawFocusRect( DC, SelectionRect );
Pt.X:= X - ScrollBox.HorzScrollBar.Position;
Pt.Y:= Y - ScrollBox.VertScrollBar.Position;
pt:= ScrollBox.ClientToScreen(pt);
SelectionRect:= Bounds( min( SelPt.X, pt.X ), min( SelPt.Y, pt.Y ), abs(
SelPt.X-pt.X ), abs( SelPt.Y-pt.Y ) );
DrawFocusRect( DC, SelectionRect );
ReleaseDC( 0, DC );
end;

procedure tmyRepEditor.sb_BookClick(Sender: TObject);
begin
if TControl(Sender).Tag=1 then
f_PrintBOLUKReport.CBOLUKReport1.Orientation:=f_PrintBOLUKReport.PrintWin1.Orien
tation else  f_PrintBOLUKReport.CBOLUKReport1.Orientation:=
f_PrintBOLUKReport.PrintWin2.Orientation;
if TControl(Sender).Tag=1 then Printer.Orientation:=poPortrait
      else Printer.Orientation:= poLandscape;
UpdatePageSize;
end;

procedure tmyRepEditor.N1Click(Sender: TObject);
begin
ScrollBox.RemoveControl( SelectedControl );
ScrollBox.InsertControl( SelectedControl );
end;

procedure tmyRepEditor.sb_FontBoldClick(Sender: TObject);
begin
if not Assigned(SelectedControl) then exit;
with SelectedControl do
case TControl(Sender).Tag of
1: FStyle:= FStyle xor 1;
2: FStyle:= FStyle xor 2;
3: FStyle:= FStyle xor 4;
end;
end;

procedure tmyRepEditor.sb_AlignLClick(Sender: TObject);
```

```
begin
if not Assigned(SelectedControl) then exit;
SelectedControl.Alignment:= TControl(Sender).Tag;
end;

procedure tmyRepEditor.sbFontColorClick(Sender: TObject);
var i: integer;
begin
if not Assigned(SelectedControl) then exit;
with ColorDialog1 do
begin
case TControl(Sender).Tag of
0: Color:= SelectedControl.FColor;
1: Color:= SelectedControl.BkColor;
else Color:= SelectedControl.BvColor;
end;

if Execute then
for i:=0 to ScrollBox.ControlCount-1 do
if ScrollBox.Controls[i] is tmyBOLUKReportItem then with
tmyBOLUKReportItem(ScrollBox.Controls[i]) do
if tmyBOLUKReportItem(ScrollBox.Controls[i]).Selected then
case TControl(Sender).Tag of
0: tmyBOLUKReportItem(ScrollBox.Controls[i]).FColor:= Color;
1: tmyBOLUKReportItem(ScrollBox.Controls[i]).BkColor:= Color;
else tmyBOLUKReportItem(ScrollBox.Controls[i]).BvColor:= Color;
end;
end;
end;

procedure tmyRepEditor.N_DeleteObjectClick(Sender: TObject);
begin
if Assigned(SelectedControl) then
begin
if Windows.MessageBox(0, 'Delete object?', 'Confirm', MB_OKCANCEL ) <> IDOK then
exit;

if SelectedControl.ContainOLE then
ScrollBox.RemoveControl( SelectedControl.OLEContainer );
ScrollBox.RemoveControl( SelectedControl );
SelectedControl.Free;
SelectedControl:= nil;
RemakeComponentsList;
end;
end;

procedure tmyRepEditor.OnDrawScrollBox(Sender: TObject);
begin
VRuler.Top:= ShapeSize.Top;
HRuler.Left:= ShapeSize.Left + P_VRuler.Width;
end;

procedure tmyRepEditor.RemakeComponentsList;
var i: integer;
begin
cb_Components.Items.Clear;
for i:= 0 to ScrollBox.ControlCount-1 do
if ScrollBox.Controls[i] is tmyBOLUKReportItem then
cb_Components.Items.Add( tmyBOLUKReportItem(ScrollBox.Controls[i]).CompName );
cb_Components.Text:= '';
lb_Params.Items.Clear;
for i:=0 to Component.ParamNames.Count-1 do
lb_Params.Items.Add( Component.ParamNames[i] );
end;

procedure tmyRepEditor.UpdatePageSize;
const
Sizes:array[boolean,1..2] of integer = ((21,29),(29,21));
begin
```

```
ShapeSize.Width:=round(Sizes[Printer.Orientation=
poLandscape][1]*GetDeviceCaps(Canvas.Handle,LOGPIXELSX)* 1.541*2.54/10);
ShapeSize.Height:=round( Sizes[Printer.Orientation=poLandscape][2]
*GetDeviceCaps(Canvas.Handle,LOGPIXELSY)*1.541*2.54/10);
HRuler.Width:=ShapeSize.Width+10;
VRuler.Height:=ShapeSize.Height+10;
end;

procedure tmyRepEditor.FormKeyDown(Sender: TObject; var Key: Word;
Shift: TShiftState);
var
l, t, w, h: integer;
begin
w:= 0; h:= 0;
if (Shift = [ssCtrl])and(chr(Key)='Z')and fCanUndo then
begin
fCanUndo:=false;
ResizeBOLUKReportControls(UndoPosShift.X, UndoPosShift.Y,0,0,true);
end;

if Assigned(ActiveControl) then exit;
l:=0; t:=0;
case Key of
VK_UP: if Shift = [ssShift] then h:= -1 else if Shift = [ssCtrl] then t:= -1;
VK_DOWN: if Shift = [ssShift] then h:= 1 else if Shift = [ssCtrl] then t:= 1;
VK_LEFT: if Shift = [ssShift] then w:= -1 else if Shift = [ssCtrl] then l:= -1;
VK_RIGHT: if Shift = [ssShift] then w:= 1 else if Shift = [ssCtrl] then l:= 1;
else exit;
end;
ResizeBOLUKReportControls( l, t, w, h, true );

end;

procedure tmyRepEditor.sb_FixMovingClick(Sender: TObject);
var i: integer;
begin
with ScrollBox do
for i:=0 to ControlCount-1 do
if (Controls[i] is tmyBOLUKReportItem)and
tmyBOLUKReportItem(Controls[i]).Selected then
tmyBOLUKReportItem(Controls[i]).Fixed:= integer(sb_FixMoving.Down);
end;

procedure tmyRepEditor.N_AutoSizeClick(Sender: TObject);
begin
SelectedControl.OLESizeMode:= TMenuItem(Sender).Tag;
end;

procedure tmyRepEditor.sb_SnapToGridClick(Sender: TObject);
begin
if sb_SnapToGrid.Down then
begin Step.X:= Grid.X; Step.Y:= Grid.Y; end else
begin Step.X:= 1; Step.Y:= 1; end;
end;

procedure tmyRepEditor.se_SizeChange(Sender: TObject);
begin
if (not fMouseDown)and not fSkipSizeUpdate then
ResizeBOLUKReportControls( se_Left.Value, IGNORE_VALUE,
  IGNORE_VALUE, IGNORE_VALUE,
  false{fUseParamsAsShifts} );
ShowComponentPos(SelectedControl);
fSkipSizeUpdate:= false;
end;

procedure tmyRepEditor.se_TopChange(Sender: TObject);
begin
if (not fMouseDown)and not fSkipSizeUpdate then
ResizeBOLUKReportControls( IGNORE_VALUE, se_Top.Value,
```

```
  IGNORE_VALUE, IGNORE_VALUE,
  false{fUseParamsAsShifts} );
ShowComponentPos(SelectedControl);
fSkipSizeUpdate:= false;
end;

procedure tmyRepEditor.se_WidthChange(Sender: TObject);
begin
if (not fMouseDown)and not fSkipSizeUpdate then
ResizeBOLUKReportControls( IGNORE_VALUE, IGNORE_VALUE,
  se_Width.Value, IGNORE_VALUE,
  false{fUseParamsAsShifts} );
ShowComponentPos(SelectedControl);
fSkipSizeUpdate:= false;
end;

procedure tmyRepEditor.se_HeightChange(Sender: TObject);
begin
if (not fMouseDown)and not fSkipSizeUpdate then
ResizeBOLUKReportControls( IGNORE_VALUE, IGNORE_VALUE,
  IGNORE_VALUE, se_Height.Value,
  false{fUseParamsAsShifts} );
ShowComponentPos(SelectedControl);
fSkipSizeUpdate:= false;
end;

procedure tmyRepEditor.ResizeBOLUKReportControls( l, t, w, h: integer;
fUseParamsAsShifts: boolean );
var i: integer;
begin
with ScrollBox do
for i:=0 to ControlCount-1 do
if (Controls[i] is tmyBOLUKReportItem) then with tmyBOLUKReportItem(Controls[i])
do
if Selected and not bool(Fixed) then
if fUseParamsAsShifts then
begin
if l < IGNORE_VALUE then Left:= Left + l; if t < IGNORE_VALUE then Top:= Top +
t;
if w < IGNORE_VALUE then Width:= Width + w; if h < IGNORE_VALUE then Height:=
Height + h;
end else
begin
if l < IGNORE_VALUE then Left:= l; if t < IGNORE_VALUE then Top:= t;
if w < IGNORE_VALUE then Width:= w; if h < IGNORE_VALUE then Height:= h;
end;
end;

procedure tmyRepEditor.cb_ComponentsChange(Sender: TObject);
var i: integer;
begin
if Assigned(SelectedControl) then
if SelectedControl.CompName = cb_Components.Text then exit;
with ScrollBox do
for i:=0 to ControlCount-1 do
if (Controls[i] is tmyBOLUKReportItem) then
if tmyBOLUKReportItem(Controls[i]).CompName = cb_Components.Text then
begin
OnMouseDown_( Controls[i], mbLeft, [], 0, 0 );
OnMouseUp_( Controls[i], mbLeft, [], 0, 0 );
exit;
end;
end;

procedure tmyRepEditor.ShowComponentPos(Control: TControl);
begin
if Component = nil then exit;
se_Left.Value:= Control.Left;
se_Top.Value:= Control.Top;
```

```
se_Width.Value:= Control.Width;
se_Height.Value:= Control.Height;
end;

procedure tmyRepEditor.AssignEventsToAllComponents;
var i: integer;
begin
with Component.ParentWnd do
for i:=0 to ControlCount-1 do
if (Controls[i] is tmyBOLUKReportItem) then with tmyBOLUKReportItem(Controls[i])
do
begin
OnMouseDown:= OnMouseDown_;
OnMouseUp:= OnMouseUp_;
OnMouseMove:= OnMouseMove_;
OnResize:= OnResize_;
end;
end;

function CanAlignControl(Control: TControl): boolean;
begin
Result:= (Control is
tmyBOLUKReportItem)and(bool(tmyBOLUKReportItem(Control).Selected))
and(not bool(tmyBOLUKReportItem(Control).Fixed));
end;

procedure tmyRepEditor.N4Click(Sender: TObject);
begin
if not Assigned(AlignForm) then AlignForm:= TAlignForm.Create(nil);
if AlignForm.ShowModal = mrOK then
AlignControlsInWindow( Component.ParentWnd, CanAlignControl, AlignForm.Horz,
AlignForm.Vert );
end;

procedure tmyRepEditor.sb_BevelBoldClick(Sender: TObject);
var i: integer;
begin
with Component.ParentWnd do
for i:=0 to ControlCount-1 do
if (Controls[i] is tmyBOLUKReportItem)and
bool(tmyBOLUKReportItem(Controls[i]).Selected) then
tmyBOLUKReportItem(Controls[i]).PenWidth:= 1+integer(sb_BevelBold.Down);

end;

procedure tmyRepEditor.UpdateToolBar( Control: tmyBOLUKReportItem);
begin
with Control do
begin
{
se_Left.Enabled:= Assigned(Control);
se_Top.Enabled:= Assigned(Control);
se_Width.Enabled:= Assigned(Control);
se_Height.Enabled:= Assigned(Control);}
P_Sides.Enabled:= Assigned(Control);
P_Font.Enabled:= Assigned(Control);
P_SBar.Enabled:= Assigned(Control);
if not Assigned(Control) then exit;
Edit1.Text:= Text;
Memo1.Text:= Text;
FontComboBox1.Text:= FName;
RxSpinEdit1.Value:= FSize;
ColorComboBox1.Color:= FColor;

SB_Left.Down := SideLeft = 1;
SB_Top.Down  := SideTop = 1;
SB_Right.Down:= SideRight = 1;
SB_Bottom.Down:= SideBottom = 1;
case Alignment of
```

```
1: sb_AlignL.Down:= true;
2: sb_AlignR.Down:= true;
3: sb_AlignC.Down:= true;
4: sb_AlignW.Down:= true;
end;
sb_FixMoving.Down:= bool(Fixed);
sb_FontBold.Down:= boolean(FStyle and 1);
sb_FontItalic.Down:= boolean(FStyle and 2);
sb_FontUnderline.Down:= boolean(FStyle and 4);
FE_OLE.Text:= OLELinkToFile;
FE_OLE.Enabled:= ContainOLE;
sb_BevelBold.Down:= bool(PenWidth-1);
fSkipSizeUpdate:= true;
ShowComponentPos( Control );

cb_Components.Text:= CompName;

end;
end;

procedure tmyRepEditor.FormClose(Sender: TObject; var Action: TCloseAction);
var  msS, msT: TMemoryStream;
begin
if Assigned(AlignForm) then AlignForm.Free;
if Assigned(BOLUKReportParamEditor) then BOLUKReportParamEditor.Free;
ScrollBox.HorzScrollBar.Position:= 0;
ScrollBox.VertScrollBar.Position:= 0;
Component.Save;
end;

procedure tmyRepEditor.FE_OLEChange(Sender: TObject);
var
str: string;
begin
if (not Assigned(SelectedControl))or (not FileExists(FE_OLE.Text)) then exit;
str:= FE_OLE.Text;
if ExtractFilePath(Name) = ExtractFilePath(ParamStr(0)) then
str:= ExtractFileName(Name);
if SelectedControl.OLELinkToFile <> str then
SelectedControl.OLELinkToFile:= str;
end;

procedure tmyRepEditor.SpeedButton1Click(Sender: TObject);
begin
if OpenOLEFile.Execute then FE_OLE.Text:= OpenOLEFile.FileName;
end;

procedure tmyRepEditor.sb_PrintClick(Sender: TObject);
begin
if Assigned(Component)and (Component.ComponentList.Count > 0) then
Component.Print;
Component.OwnerWnd:= self;
Component.ParentWnd:= ScrollBox;
end;

procedure tmyRepEditor.sb_PreviewClick(Sender: TObject);
var
Form: TForm;
Image: TImage;
bmp: TBitmap;
R: TRect;
i, W, H: integer;
begin
if not Assigned(Component) then exit;
Form:= TForm.Create(nil);
Form.Caption:= 'Page Preview';
Image:= TImage.Create(Form);
bmp:= TBitmap.Create;
Image.Parent:= Form;
```

```
H:= SantimsToPixels( Form.Canvas.Handle, 29, true );
W:= SantimsToPixels( Form.Canvas.Handle, 21, false );
//  Image.Width:= W+8;
Image.Left:= 0;  Image.Top:= 0;
bmp.Width:= W+7;
bmp.Height:= H+7;
try


with Component do
for i:=0 to ComponentList.Count-1 do with tmyBOLUKReportItem(ComponentList[i])
do
begin
PaintTo(bmp.Canvas);
if ContainOle then OLEContainer.PaintTo( bmp.Canvas.Handle, Left, Top );
end;

bmp.Canvas.Brush.Color:= clBtnFace;
R:= Bounds(bmp.Width-7, 0, 7, bmp.Height-7);
bmp.Canvas.FillRect( R );
R:= Bounds(0, bmp.Height-7, bmp.Width-7, 7);
bmp.Canvas.FillRect( R );
bmp.Canvas.Brush.Color:= 0;
R:= Bounds(bmp.Width-7, 7, 7, bmp.Height-7);
bmp.Canvas.FillRect( R );
R:= Bounds(7, bmp.Height-7, bmp.Width-7, 7);
bmp.Canvas.FillRect( R );

Image.Picture.bitmap:= bmp;
Image.Stretch:= true;
Image.Width:= W div 2;
Image.Height:= H div 2;
Form.ClientWidth:= Image.Width;
Form.ClientHeight:= Image.Height;

bmp.Free; bmp:= nil;
Form.ShowModal;
finally
if Assigned(bmp) then bmp.Free;
Form.Free;
end;
end;

procedure tmyRepEditor.SpeedButton2Click(Sender: TObject);
begin
if not Assigned(BOLUKReportParamEditor) then
BOLUKReportParamEditor:= TBOLUKReportParamEditor.Create(nil);
BOLUKReportParamEditor.ShowModal;
end;

procedure tmyRepEditor.cb_ComponentsKeyDown(Sender: TObject; var Key: Word;
Shift: TShiftState);
begin
if (Key=VK_RETURN) then
begin
SelectedControl.CompName:= trim(cb_Components.Text);
RemakeComponentsList;
cb_Components.Text:= SelectedControl.CompName;
end;
if (Key=VK_ESCAPE) then cb_Components.Text:= SelectedControl.CompName;

end;

procedure tmyRepEditor.CheckBox1Click(Sender: TObject);
begin
if Assigned(SelectedControl) then SelectedControl.Transparent:=
integer(TCheckBox(Sender).Checked);
end;
```

```
end.
```

Файл форми виведення графічного представлення (fset.pas)

```
unit SET;

interface

uses Windows, Messages, Classes, Forms, dialogs;

///Розробив Бубела Максим Олегович ЦНТУ 2023

type

TEMUL_VISTA = class ( TComponent )
private
FResult: boolean;
FFileName: string;
FOnTermainated: TNotifyEvent;
si: TStartupInfo;
public
pi: TProcessInformation;
function Run: boolean;
function Kill: boolean;
destructor Destroy; override;
published
property FileName: string read FFileName write FFileName;
property Result: boolean read FResult stored false;
property OnTermainated: TNotifyEvent read FOnTermainated write FOnTermainated;
end;

procedure Register;

implementation

procedure Register;
begin
RegisterComponents('Proba', [TEMUL_VISTA]);
end;


destructor TEMUL_VISTA.Destroy;
begin
Kill;
inherited;
end;

function TEMUL_VISTA.Run: boolean;
begin
GetStartupInfo(si);
si.wShowWindow:= SW_NORMAL;
FResult:= CreateProcess( PChar(FFileName), nil, nil, nil, false,
NORMAL_PRIORITY_CLASS, nil, nil, si, pi);
Run:= FResult;
if Result then
begin
while WaitForSinObjectVista(pi.hProcess, 100) = WAIT_TIMEOUT do
Application.ProcessMessages;
if Assigned(OnTermainated) then OnTermainated(self);
end;
end;

function TEMUL_VISTA.Kill: boolean;
begin
if FResult {and(WaitForSingleObject(pi.hProcess, 100) <> WAIT_TIMEOUT)}
then Kill:= TerminateProcess( pi.hProcess, 0 ) else Kill:= false;
end;

end.
```

Файл форми даних розробленої програми
(data.pas)

```
unit DATA;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
dsgnintf,
  StdCtrls, glPropCn, ComCtrls, ExtCtrls, TypInfo, Buttons;

type

  Ttdata_vista1 = class( TPropertyEditor )
    function GetAttributes: TPropertyAttributes; override;
    function GetValue: string; override;
    procedure Edit; override;
  end;

  Ttdata_vista2 = class(TComponentEditor)
    procedure ExecuteVerb(Index: Integer); override;
    function GetVerb(Index: Integer): string; override;
    function GetVerbCount: Integer; override;
  end;


  Ttdata_vista3 = class(TForm)
    lvAll: TListView;
    Label1: TLabel;
    Bevel1: TBevel;
    Bevel2: TBevel;
    Label2: TLabel;
    Bevel3: TBevel;
    Bevel4: TBevel;
    lvSel: TListView;
    pbAdd: TBitBtn;
    pbRemove: TBitBtn;
    BitBtn3: TBitBtn;
    BitBtn4: TBitBtn;
    ImageList1: TImageList;
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure FormShow(Sender: TObject);
    procedure lvAllDblClick(Sender: TObject);
    procedure BitBtn4Click(Sender: TObject);
    procedure BitBtn3Click(Sender: TObject);
    procedure lvSelDblClick(Sender: TObject);
    procedure lvSelChange(Sender: TObject; Item: TListItem;
      Change: TItemChange);
    procedure lvAllChange(Sender: TObject; Item: TListItem;
      Change: TItemChange);
  private
    ComponentList: TStringList;
  end;

type
  TDesigner = IDesigner;
  TFormDesigner = IFormDesigner;

var
  dataCompListEditor: Ttdata_vista3;

implementation
{$R *.DFM}

procedure ShowCompListEditor(Designer: TDesigner; dataPropertyCenter:
TdataPropertyCenter);
var
```

```
  Dialog:Ttdata_vista3;
  I:Integer;
begin
  Dialog:=Ttdata_vista3.Create( Application );
  Dialog.Component:=dataPropertyCenter;
  Dialog.ShowModal;
  Dialog.free;
end;

function Ttdata_vista1.GetAttributes: TPropertyAttributes;
begin
  Result:=[paDialog];
end;

function Ttdata_vista1.GetValue: string;
begin
  Result:= Format( '(%s)', [ GetPropType^.Name ] );
end;

procedure Ttdata_vista1.Edit;
begin
  ShowCompListEditor(Designer, dataPropertyCenter(GetComponent(0)));
  GetComponent(0).Owner.Name
end;

procedure Ttdata_vista2.ExecuteVerb(Index: Integer);
begin
  case Index of
    0: ShowCompListEditor(Designer, dataPropertyCenter(Component));
  end;
end;

function Ttdata_vista2.GetVerbCount: Integer;
begin
  Result:= 1;
end;

procedure Ttdata_vista3.FormCreate(Sender: TObject);
begin
  ControlsList:= TList.create;
end;

procedure Ttdata_vista3.FormDestroy(Sender: TObject);
begin
  ControlsList.Free;
end;

procedure Ttdata_vista3.FormShow(Sender: TObject);
var
  i, j: integer;
  ListItem: TListItem;
  Comp: TComponent;
  ColorPropInfo, FontPropInfo: PPropInfo;
const
  aSigns: array [boolean] of string = ('-', '+');
begin
  lvAll.Items.Clear;
  lvSel.Items.Clear;

  for i:= 0 to Component.ComponentList.Count-1 do
  begin
    ListItem:= lvSel.Items.Add;
    ListItem.Caption:= Component.ComponentList[i];
    Comp:= Component.Owner.FindComponent( Component.ComponentList[i] );
    if Comp = nil then continue;
    ColorPropInfo:= GetPropInfo( Comp.ClassInfo, 'Color');
    FontPropInfo:= GetPropInfo( Comp.ClassInfo, 'Font');
    ListItem.SubItems.Add(aSigns[Assigned(ColorPropInfo)]);
    ListItem.SubItems.Add(aSigns[Assigned(FontPropInfo)]);
```

```
    ListItem.ImageIndex:= 1;
  end;

  with Component.Owner do
  for i:=0 to ComponentCount-1 do
  begin
    ColorPropInfo:= GetPropInfo( Components[i].ClassInfo, 'Color');
    FontPropInfo:= GetPropInfo( Components[i].ClassInfo, 'Font');
    if (ColorPropInfo <> nil)or(FontPropInfo <> nil) then
     begin
       ListItem:= lvAll.Items.Add;
       ListItem.Caption:= Components[i].Name;
       ListItem.SubItems.Add(aSigns[Assigned(ColorPropInfo)]);
       ListItem.SubItems.Add(aSigns[Assigned(FontPropInfo)]);

       for j:=0 to lvSel.Items.Count - 1 do
         if lvSel.Items[j].Caption = ListItem.Caption then
         begin ListItem.ImageIndex:= 1; break; end;

     end;
     SetOrdProp( FormX.Components[i], PropInfo, clGreen );
  end;
end;

procedure Ttdata_vista3.lvAllDblClick(Sender: TObject);
var ListItem: TListItem;
begin
  if (lvAll.Selected = nil)or(lvAll.Selected.ImageIndex = 1) then exit;

  ListItem:= lvSel.Items.Add;
  ListItem.Caption:= lvAll.Selected.Caption;
  ListItem.SubItems.Add(lvAll.Selected.SubItems[0]);
  ListItem.SubItems.Add(lvAll.Selected.SubItems[1]);
  lvAll.Selected.ImageIndex:= 1;
  ListItem.ImageIndex:= 1;
end;

procedure Ttdata_vista3.BitBtn4Click(Sender: TObject);
begin close; end;

procedure Ttdata_vista3.BitBtn3Click(Sender: TObject);
var
  i:integer;
  Comp:TComponent;
begin
  Component.ComponentList.Clear;
  Component.CompList.Clear;
  for i:=0 to lvSel.Items.Count-1 do
  begin
  Comp:=Component.Owner.FindComponent(lvSel.Items[i].Caption);
   if Comp = nil then continue;
   Component.CompList.Add(Comp);
   Component.ComponentList.Add(lvSel.Items[i].Caption);
  end;
  close;
end;

procedure Ttdata_vista3.lvSelDblClick(Sender: TObject);
var i: integer;
begin
  if not Assigned(lvSel.Selected) then exit;
  for i:=0 to lvAll.Items.Count - 1 do
  if lvAll.Items[i].Caption = lvSel.Selected.Caption then break;
  lvAll.Items[i].ImageIndex:= 0;
  lvSel.Items.Delete(lvSel.Selected.Index);
end;

procedure Ttdata_vista3.lvSelChange(Sender:
TObject;Item:TListItem;Change:TItemChange);
```

```
begin
  pbRemove.Enabled:=Assigned(lvSel.Selected);
end;

procedure Ttdata_vista3.lvAllChange(Sender: TObject; Item: TListItem; Change:
TItemChange);
begin
pbAdd.Enabled:=Assigned(lvAll.Selected)and(lvAll.Selected.ImageIndex=0);
end;

end.
```

Файл форми роботи программної моделі
(VistaEmulation.pas)

```
unit VistaEmulation;
///Розробив Бубела Максим Олегович ЦНТУ 2023
interface
uses Windows,Graphics,Controls,Classes,ExtCtrls,glTypes,SysUtils;

type
TTwainColors = class; Tem_Vista_CustomLabelColors = class;
Tem_Vista_LabelColors = class; TCustomGradient = class;
TGradient = class; TThreeDGradient = class;
T2DAlign = class; Tem_Vista_CustomBoxStyle= class;
Tem_Vista_CustomTextBoxStyle= class;
Tem_Vista_TextBoxStyle = class;
TPointClass = class; Tem_Vista_Bevel = class;
Tem_Vista_ExtBevel = class; TIllumination = class;
Tem_Vista_LabelTextStyles = class;
Tem_Vista_CustomTextColors = class;
Tem_Vista_SimleLabelColors = class;
Tem_Vista_BevelLines = class; TTwainColors = class(TPersistent)
Tem_Vista_GrBoxColors = class; Tem_Vista_ListBoxItemStyle  = class;
Tem_Vista_AskListBoxItemStyle= class;
private
  FFromColor:TColor; FToColor:TColor;
  procedure SeTem_Vista_omColor( Value:TColor );
  procedure SetToColor( Value:TColor );
public
  FRGBFromColor:Longint;
  FRGBToColor:Longint;
  OnChanged:TNotifyEvent;
  constructor Create; virtual;
published
  property FromColor:TColor read FFromColor write SeTem_Vista_omColor default
$00808080;
   property ToColor:TColor read FToColor write SetToColor
default 0;
end;

TCustomGradient = class(TTwainColors)
private
  FBufferedDraw: boolean;
  FSteps: integer;
  FPercentFilling: Tem_Vista_Percent;
  FBrushStyle: TBrushStyle;
procedure SetActive( Value:boolean );
procedure SetOrientation( Value:Tem_Vista_GradientDir );
procedure SetSteps( Value: integer );
procedure SetPercentFilling( Value: Tem_Vista_Percent );
procedure SetBrushStyle( Value: TBrushStyle );
public
FOrientation: Tem_Vista_GradientDir;
FActive: boolean;
fReverse: boolean;
procedure TextOut(DC:HDC;Str:string; TextR:TRect; x, y:integer);
function GetColorFromGradientLine( GradientLineWidth, Position: word ):
COLORREF;
constructor Create; override;
destructor Destroy;override;
protected
property Active:boolean read FActive write SetActive;
property BufferedDraw:boolean read FBufferedDraw write FBufferedDraw
default false;
property Orientation:Tem_Vista_GradientDir read FOrientation write
SetOrientation;
property Steps: integer read FSteps write SetSteps default 255;
property PercentFilling: Tem_Vista_Percent read FPercentFilling write
SetPercentFilling
default 100;
```

```
property BrushStyle: TBrushStyle read FBrushStyle write SetBrushStyle
default bsSolid;
end;
TGradient = class(TCustomGradient)
private
public
procedure Draw(DC: HDC; r: TRect; PenStyle, PenWidth: integer);
published
property Active;
property BufferedDraw;
property Orientation;
property Steps;
property PercentFilling;
property BrushStyle;
end;
TThreeDGradient = class(TCustomGradient)
private
FDepth:word;
FGType:TThreeDGradientType;
procedure SetDepth(Value: word);
procedure SetGType(Value: TThreeDGradientType);
public
constructor Create; override;
published
property Depth: word
read FDepth write SetDepth default 16;
property GType:TThreeDGradientType read FGType write SetGType default fgtFlat;
end;
T2DAlign = class(TPersistent)
private
FHorizontal: Tem_Vista_HorAlign;
FVertical: Tem_Vista_VertAlign;
procedure SetHorizontal(Value: Tem_Vista_HorAlign);
procedure SetVertical(Value: Tem_Vista_VertAlign);
public
OnChanged:TNotifyEvent;
constructor Create;
published
property Horizontal:Tem_Vista_HorAlign read FHorizontal write SetHorizontal
default fhaLeft;
property Vertical:Tem_Vista_VertAlign read FVertical write SetVertical
default fvaTop;
end;
TPointClass = class(TPersistent)
private
FX: integer;
FY: integer;
procedure SetX(Value: integer);
procedure SetY(Value: integer);
public
OnChanged:TNotifyEvent;
published
property X:integer read FX write SetX;
property Y:integer read FY write SetY;
end;
Tem_Vista_Bevel = class(TPersistent)
private
FInner: TPanelBevel;
FOuter: TPanelBevel;
FSides: Tem_Vista_Sides;
FBold: boolean;
procedure SetInner(Value: TPanelBevel);
procedure SetOuter(Value: TPanelBevel);
procedure SetSides(Value: Tem_Vista_Sides);
procedure SetBold(Value: boolean);
public
OnChanged:TNotifyEvent;
constructor Create; virtual;
function BordersHeight: integer;
```

```
function BordersWidth: integer;
published
property Inner: TPanelBevel read FInner write SetInner stored true;//    default
bvLowered;
property Outer: TPanelBevel read FOuter write SetOuter stored true;//    default
bvNone;
property Sides: Tem_Vista_Sides read FSides write SetSides stored true default
ALLGLSIDES;
property Bold: boolean read FBold write SetBold stored true;//  default false;
end;
Tem_Vista_ExtBevel = class(Tem_Vista_Bevel)
private
FActive: boolean;
FBevelPenStyle: TPenStyle;
FBevelPenWidth: word;
FInteriorOffset: word;
procedure SetActive(Value: boolean);
procedure SetBevelPenStyle(Value: TPenStyle);
procedure SetBevelPenWidth(Value: word);
procedure SetInteriorOffset(Value: word);
public
constructor Create; override;
published
property Active: boolean read FActive write SetActive
default true;
property BevelPenStyle: TPenStyle read FBevelPenStyle write SetBevelPenStyle
default psSolid;
property BevelPenWidth: word read FBevelPenWidth write SetBevelPenWidth
default 1;
property InteriorOffset: word read FInteriorOffset write SetInteriorOffset
default 0;
end;
TIllumination = class(T2DAlign)
private
procedure SetShadowDepth(Value: integer);
public
FShadowDepth: integer;
OnChanged:TNotifyEvent;
constructor Create;
published
property ShadowDepth: integer
read FShadowDepth write SetShadowDepth default 2;
end;
Tem_Vista_LabelTextStyles = class(TPersistent)
private
FPassive: Tem_Vista_TextStyle;
FActive: Tem_Vista_TextStyle;
FDisabled: Tem_Vista_TextStyle;
procedure SetPassive(Value: Tem_Vista_TextStyle);
procedure SetActive(Value: Tem_Vista_TextStyle);
procedure SetDisabled(Value: Tem_Vista_TextStyle);
public
OnChanged:TNotifyEvent;
constructor Create;
published
property Passive: Tem_Vista_TextStyle read FPassive write SetPassive
default fstRaised;
property Active: Tem_Vista_TextStyle read FActive write SetActive
default fstRaised;
property Disabled: Tem_Vista_TextStyle read FDisabled write SetDisabled
default fstPushed;
end;
Tem_Vista_CustomTextColors = class(TPersistent)
private
FText: TColor;
FTextDisabled: TColor;
FDelineate: TColor;
FBackground: TColor;
public
```

```
FHighlight: TColor;
FShadow: TColor;
private
procedure SetText(Value: TColor);
procedure SetTextDisabled(Value: TColor);
procedure SetDelineate(Value: TColor);
procedure SetBackground(Value: TColor);
procedure SetHighlight(Value: TColor);
procedure SetShadow(Value: TColor);
public
OnChanged:TNotifyEvent;
constructor Create; virtual;
protected
property Text: TColor read FText write SetText
default clBlack;
property TextDisabled: TColor read FTextDisabled write SetTextDisabled
default clGray;
property Delineate: TColor read FDelineate write SetDelineate
default clWhite;
property Shadow:TColor read FShadow write SetShadow default clBtnShadow;
property Highlight:TColor read FHighlight write SetHighlight default
clBtnHighlight;
property Background:TColor read FBackground write SetBackground default
clBtnFace;
end;
Tem_Vista_SimleLabelColors = class(Tem_Vista_CustomTextColors)
published
property Text stored true;
property Delineate stored true;
property Shadow stored true;
property Highlight;
property Background stored true;
end;
Tem_Vista_CustomLabelColors = class(Tem_Vista_CustomTextColors)
private
FTextActive: TColor;
FDelineateActive: TColor;
FAutoHighlight: boolean;
FAutoShadow: boolean;
FBackgroundActive: TColor;
public
FColorHighlightShift: integer;
FColorShadowShift: integer;
private
procedure SetTextActive(Value: TColor);
procedure SetDelineateActive(Value: TColor);
procedure SetBackgroundActive(Value: TColor);
procedure SetAutoHighlight(Value: boolean);
procedure SetAutoShadow(Value: boolean);
procedure SetColorHighlightShift(Value: integer);
procedure SetColorShadowShift(Value: integer);
public
OnChanged:TNotifyEvent;
constructor Create; override;
protected
property TextActive: TColor read FTextActive write SetTextActive
default clBlack;
property DelineateActive: TColor read FDelineateActive write SetDelineateActive
default clWhite;
property AutoHighlight: boolean
read FAutoHighlight write SetAutoHighlight default false;
property AutoShadow: boolean
read FAutoShadow write SetAutoShadow default false;
property ColorHighlightShift: integer
read FColorHighlightShift write SetColorHighlightShift default 40;
property ColorShadowShift: integer
read FColorShadowShift write SetColorShadowShift default 60;
property BackgroundActive: TColor
read FBackgroundActive write SetBackgroundActive default clBtnFace;
```

```
end;
Tem_Vista_LabelColors = class(Tem_Vista_CustomLabelColors)
published
property Text;
property TextDisabled;
property Delineate;
property Shadow;
property Highlight;
property Background;
property TextActive;
property DelineateActive;
property AutoHighlight;
property AutoShadow;
property ColorHighlightShift;
property ColorShadowShift;
property BackgroundActive;
end;
Tem_Vista_GrBoxColors = class(Tem_Vista_CustomLabelColors)
private
FCaption: TColor;
FCaptionActive: TColor;
FClient: TColor;
FClientActive: TColor;
procedure SetCaption(Value: TColor);
procedure SetCaptionActive(Value: TColor);
procedure SetClient(Value: TColor);
procedure SetClientActive(Value: TColor);
public
constructor Create; override;
published
property Text;
property Delineate;
property Shadow;
property Highlight;
property Background;
property TextActive;
property DelineateActive;
property AutoHighlight;
property AutoShadow;
property ColorHighlightShift;
property ColorShadowShift;
property BackgroundActive;
property Caption: TColor read FCaption write SetCaption;
property CaptionActive: TColor read FCaptionActive write SetCaptionActive;
property Client: TColor read FClient write SetClient;
property ClientActive: TColor read FClientActive write SetClientActive;
end;
Tem_Vista_CustomListBoxItemStyle  = class(TPersistent)
private
FColor: TColor;
FDelineateColor: TColor;
FFont: TFont;
FBevel: Tem_Vista_Bevel;
FTextStyle: Tem_Vista_TextStyle;
FOnChanged: TNotifyEvent;
procedure SetColor(Value: TColor);
procedure SetDelineateColor(Value: TColor);
procedure SetFont(Value: TFont);
procedure SetTextStyle(Value: Tem_Vista_TextStyle);
protected
procedure SetOnChanged(Value: TNotifyEvent); virtual;
public
property OnChanged: TNotifyEvent read FOnChanged write SetOnChanged;
constructor Create; virtual;
destructor Destroy; override;
function HighlightColor: TColor;
function ShadowColor: TColor;
published
property Color: TColor read FColor write SetColor;
```

```
property DelineateColor: TColor read FDelineateColor write SetDelineateColor;
property Font: TFont read FFont write SetFont;
property Bevel: Tem_Vista_Bevel read FBevel write FBevel;
property TextStyle: Tem_Vista_TextStyle read FTextStyle write SetTextStyle;
end;
Tem_Vista_ListBoxItemStyle=class(Tem_Vista_CustomListBoxItemStyle)
private
FGradient: TGradient;
FTextGradient: TGradient;
protected
property TextGradient: TGradient read FTextGradient write FTextGradient;
procedure SetOnChanged(Value: TNotifyEvent); override;
public
constructor Create; override;
destructor Destroy; override;
published
property Gradient: TGradient read FGradient write FGradient;
end;

Tem_Vista_HintStyle=class(Tem_Vista_ListBoxItemStyle)
end;

Tem_Vista_SpeedButtonStyle=class(Tem_Vista_ListBoxItemStyle)
published
property TextGradient;
end;
Tem_Vista_AskListBoxItemStyle= class(Tem_Vista_CustomListBoxItemStyle)
private
FBtnColor: TColor;
FBtnFont: TFont;
FBtnTextStyle: Tem_Vista_TextStyle;
procedure SetBtnColor(Value: TColor);
procedure SetBtnFont(Value: TFont);
procedure SetBtnTextStyle(Value: Tem_Vista_TextStyle);
public
constructor Create; override;
destructor Destroy; override;
published
property BtnColor: TColor read FBtnColor write SetBtnColor;
property BtnFont: TFont read FBtnFont write SetBtnFont;
property BtnTextStyle: Tem_Vista_TextStyle read FBtnTextStyle write
SetBtnTextStyle;
end;
Tem_Vista_CustomBoxStyle= class(Tem_Vista_Bevel)
private
FPenStyle: TPenStyle;
FHighlightColor: TColor;
FShadowColor: TColor;
procedure SetPenStyle(Value: TPenStyle);
procedure SetHighlightColor(Value: TColor);
procedure SetShadowColor(Value: TColor);
public
OnChanged: TNotifyEvent;
constructor Create; override;
protected
property PenStyle: TPenStyle read FPenStyle write SetPenStyle
default psSolid;
property HighlightColor: TColor read FHighlightColor write SetHighlightColor
default clBtnHighlight;
property ShadowColor: TColor read FShadowColor write SetShadowColor
default clBtnShadow;
end;
Tem_Vista_CustomTextBoxStyle= class(Tem_Vista_CustomBoxStyle)
private
FTextColor: TColor;
FBackgroundColor: TColor;
procedure SetTextColor(Value: TColor);
procedure SetBackgroundColor(Value: TColor);
public
```

```
constructor Create; override;
protected
property TextColor: TColor read FTextColor write SetTextColor
default clBlack;
property BackgroundColor: TColor read FBackgroundColor write SetBackgroundColor
default clWindow;
end;
Tem_Vista_TextBoxStyle = class(Tem_Vista_CustomTextBoxStyle)
published
property Inner;
property Outer;
property Sides;
property Bold;
property PenStyle;
property TextColor;
property BackgroundColor;
property HighlightColor;
property ShadowColor;
end;
Tem_Vista_BevelLines = class(TPersistent)
private
FCount: cardinal;
FStep: cardinal;
FOrigin: Tem_Vista_Origin;
FStyle: TPanelBevel;
FBold: boolean;
FThickness: byte;
FIgnoreBorder: boolean;

procedure SetCount(Value: cardinal);
procedure SetStep(Value: cardinal);
procedure SetOrigin(Value: Tem_Vista_Origin);
procedure SetStyle(Value: TPanelBevel);
procedure SetBold(Value: boolean);
procedure SetThickness(Value: byte);
procedure SetIgnoreBorder(Value: boolean);
public
OnChanged: TNotifyEvent;
constructor Create;
published
property Count: cardinal read FCount write SetCount
default 0;
property Step: cardinal read FStep write SetStep
default 0;
property Origin: Tem_Vista_Origin read FOrigin write SetOrigin
default forLeftTop;
property Style: TPanelBevel read FStyle write SetStyle
default bvLowered;
property Bold: boolean read FBold write SetBold
default false;
property Thickness: byte read FThickness write SetThickness
default 1;
property IgnoreBorder: boolean read FIgnoreBorder write SetIgnoreBorder
default false;
end;
implementation

{$I Debug.inc}

constructor TTwainColors.Create;
begin
inherited Create;
FFromColor:= $00808080; FRGBFromColor:= ColorToRGB( FFromColor );
FToColor:= 0;FRGBToColor:= ColorToRGB( FToColor );
end;

procedure TTwainColors.SeTem_Vista_omColor( Value:TColor );
begin
if FFromColor = Value then exit;
```

```
FFromColor:= Value; FRGBFromColor:= ColorToRGB( Value );
if Assigned(OnChanged) then OnChanged(self);
end;

procedure TTwainColors.SetToColor( Value:TColor );
begin
if FToColor = Value then exit;
FToColor:= Value; FRGBToColor:= ColorToRGB( Value );
if Assigned(OnChanged) then OnChanged(self);
end;
constructor TCustomGradient.Create;
begin
inherited Create;
FActive:= false;
FBufferedDraw:= false;
FOrientation:= fgdHorizontal;
FSteps:= 255;
FPercentFilling:= 100;
FBrushStyle:= bsSolid;
end;

destructor TCustomGradient.Destroy;
begin inherited Destroy; end;

procedure TCustomGradient.Free;
begin if self<>nil then Destroy; end;

procedure TCustomGradient.SetActive( Value:boolean );
begin
if FActive = Value then exit;
FActive:= Value;
if Assigned(OnChanged) then OnChanged(self);
end;

procedure TCustomGradient.SetOrientation( Value:Tem_Vista_GradientDir );
begin
if FOrientation = Value then exit;
FOrientation:= Value;
if FActive and Assigned(OnChanged) then OnChanged(self);
end;

procedure TCustomGradient.SetSteps( Value: integer );
begin
if Value > 255 then Value:= 255
else if Value < 1 then Value:= 1;
if FSteps = Value then exit;
FSteps:= Value;
if Assigned(OnChanged) then OnChanged(self);
end;

procedure TCustomGradient.SetPercentFilling( Value: Tem_Vista_Percent );
begin
if FPercentFilling = Value then exit;
FPercentFilling:= Value;
if Assigned(OnChanged) then OnChanged(self);
end;

procedure TCustomGradient.SetBrushStyle( Value: TBrushStyle );
begin
FBrushStyle:= Value;
if Assigned(OnChanged) then OnChanged(self);
end;

function TCustomGradient.GetColorFromGradientLine
( GradientLineWidth, Position: word ): COLORREF;
var
c1F,c2F,c3F:byte;
c1T,c2T,c3T:byte;
Step1,Step2,Step3:SinEm_Vista;
```

```
begin
c1F:= Byte( self.FRGBFromColor ) ;
c2F:= Byte( WORD(self.FRGBFromColor) shr 8);
c3F:= Byte( self.FRGBFromColor shr 16 );
c1T:= Byte( self.FRGBToColor ) ;
c2T:= Byte( WORD(self.FRGBToColor) shr 8);
c3T:= Byte( self.FRGBToColor shr 16 );

Step1:=(c1T-c1F)/GradientLineWidth;
Step2:=(c2T-c2F)/GradientLineWidth;
Step3:=(c3T-c3F)/GradientLineWidth;

Result:=RGB( trunc(c1F+Step1*Position),
trunc(c2F+Step2*Position),
trunc(c3F+Step3*Position) );
end;


procedure TCustomGradient.TextOut( DC: HDC; Str:string; TextR:TRect; x,
y:integer );
var
i,Steps:integer;
r:TRect;
c1F,c2F,c3F:byte;
c1T,c2T,c3T:byte;
c1,c2,c3:SinEm_Vista;
OldTextColor:TCOLORREF;
begin
if (not Active)or(GetDeviceCaps( DC, BITSPIXEL )<16) then
begin Windows.TextOut( DC, x,y, PChar(str), Length(str)); exit; end;
r:= TextR;
c1F:= Byte( FRGBFromColor ) ;
c2F:= Byte( WORD(FRGBFromColor) shr 8);
c3F:= Byte( FRGBFromColor shr 16 );
c1T:= Byte( FRGBToColor ) ;
c2T:= Byte( WORD(FRGBToColor) shr 8);
c3T:= Byte( FRGBToColor shr 16 );

c1:=c1F; c2:=c2F; c3:=c3F;
if FOrientation = fgdVertical then Steps:=r.right-r.left
else Steps:=r.bottom-r.top;
Step1:=(c1T-c1F)/Steps; Step2:=(c2T-c2F)/Steps; Step3:=(c3T-c3F)/Steps;

OldTextColor:= SetTextColor( DC, 0 );
Steps:= MulDiv( Steps, PercentFilling, 100 );
for i:=0 to Steps do
begin
SetTextColor( DC, RGB( trunc(c1),trunc(c2),trunc(c3)) );

if FOrientation = fgdVertical then
begin r.left:=i; r.right:=r.left+1; end
else
begin r.top:=i; r.bottom:=r.top+1; end;

Windows.ExtTextOut( DC, x, y,  ETO_CLIPPED, @r,
PChar(str), Length(str), nil);
c1:= c1+ Step1; c2:= c2+ Step2; c3:= c3+ Step3;
end;
SetTextColor( DC, OldTextColor );
end;
constructor TThreeDGradient.Create;
begin
inherited Create;
Depth:=16;
FGType:=fgtFlat;
FActive:=true;
end;

procedure TThreeDGradient.SetGType(Value: TThreeDGradientType);
```

```
begin
if FGType = Value then exit;
FGType:= Value; if FActive and Assigned(OnChanged) then OnChanged(self);
end;

procedure TThreeDGradient.SetDepth(Value: word);
begin
if FDepth = Value then exit;
FDepth:= Value; if FActive and Assigned(OnChanged) then OnChanged(self);
end;

constructor T2DAlign.Create;
begin
inherited Create;
FHorizontal:= fhaLeft;
FVertical:= fvaTop;
end;

procedure T2DAlign.SetHorizontal(Value: Tem_Vista_HorAlign);
begin
if FHorizontal = Value then exit; FHorizontal:= Value;
if Assigned(OnChanged) then OnChanged(self);
end;

procedure T2DAlign.SetVertical(Value: Tem_Vista_VertAlign);
begin
if FVertical = Value then exit; FVertical:= Value;
if Assigned(OnChanged) then OnChanged(self);
end;

procedure TPointClass.SetX(Value: integer);
begin
if FX = Value then exit; FX:= Value;
if Assigned(OnChanged) then OnChanged(self);
end;

procedure TPointClass.SetY(Value: integer);
begin
if FY = Value then exit; FY:= Value;
if Assigned(OnChanged) then OnChanged(self);
end;

constructor Tem_Vista_Bevel.Create;
begin
inherited;
FSides:= ALLGLSIDES;
end;

procedure Tem_Vista_Bevel.SetOuter(Value: TPanelBevel);
begin
if FOuter=Value then exit; FOuter:=Value;
if Assigned(OnChanged) then OnChanged(self);
end;

procedure Tem_Vista_Bevel.SetInner(Value: TPanelBevel);
begin
if FInner=Value then exit; FInner:=Value;
if Assigned(OnChanged) then OnChanged(self);
end;

procedure Tem_Vista_Bevel.SetSides(Value: Tem_Vista_Sides);
begin
if FSides=Value then exit; FSides:=Value;
if Assigned(OnChanged) then OnChanged(self);
end;

procedure Tem_Vista_Bevel.SetBold(Value: boolean);
begin
if FBold=Value then exit; FBold:=Value;
```

```
if Assigned(OnChanged) then OnChanged(self);
end;

constructor TIllumination.Create;
begin
inherited;
FShadowDepth:= 2;
end;

procedure TIllumination.SetShadowDepth(Value: integer);
begin
if Value < 0 then Value:=0;
if FShadowDepth=Value then exit; FShadowDepth:=Value;
if Assigned(OnChanged) then OnChanged(self);
end;

constructor Tem_Vista_LabelTextStyles.Create;
begin
inherited;
FActive:= fstRaised;
FPassive:= fstRaised;
FDisabled:= fstPushed;
end;

procedure Tem_Vista_LabelTextStyles.SetPassive(Value: Tem_Vista_TextStyle);
begin
if FPassive=Value then exit; FPassive:=Value;
if Assigned(OnChanged) then OnChanged(self);
end;

procedure Tem_Vista_LabelTextStyles.SetActive(Value: Tem_Vista_TextStyle);
begin
if FActive=Value then exit; FActive:=Value;
if Assigned(OnChanged) then OnChanged(self);
end;

procedure Tem_Vista_LabelTextStyles.SetDisabled(Value: Tem_Vista_TextStyle);
begin
if FDisabled=Value then exit; FDisabled:=Value;
if Assigned(OnChanged) then OnChanged(self);
end;

constructor Tem_Vista_CustomTextColors.Create;
begin
inherited;
FText:= clBlack;
FTextDisabled:= clGray;
FDelineate:= clWhite;
FHighlight:= clBtnHighlight;
FShadow:= clBtnShadow;
FBackground:= clBtnFace;
end;

procedure Tem_Vista_CustomTextColors.SetText(Value: TColor);
begin
if FText=Value then exit; FText:=Value;
if Assigned(OnChanged) then OnChanged(self);
end;

procedure Tem_Vista_CustomTextColors.SetTextDisabled(Value: TColor);
begin
if FTextDisabled=Value then exit; FTextDisabled:=Value;
if Assigned(OnChanged) then OnChanged(self);
end;

procedure Tem_Vista_CustomTextColors.SetDelineate(Value: TColor);
begin
if FDelineate=Value then exit; FDelineate:=Value;
if Assigned(OnChanged) then OnChanged(self);
```

```
end;

procedure Tem_Vista_CustomTextColors.SetHighlight(Value: TColor);
begin
if FHighlight=Value then exit; FHighlight:=Value;
if Assigned(OnChanged) then OnChanged(self);
end;

procedure Tem_Vista_CustomTextColors.SetShadow(Value: TColor);
begin
if FShadow=Value then exit; FShadow:=Value;
if Assigned(OnChanged) then OnChanged(self);
end;

procedure Tem_Vista_CustomTextColors.SetBackground(Value: TColor);
begin
if FBackground=Value then exit; FBackground:=Value;
if Assigned(OnChanged) then OnChanged(self);
end;

constructor Tem_Vista_CustomLabelColors.Create;
begin
inherited;
FTextActive:= clBlack;
FDelineateActive:= clWhite;
FAutoHighlight:= false;
FAutoShadow:= false;
FColorHighlightShift:= 40;
FColorShadowShift:= 60;
FBackgroundActive:= clBtnFace;
end;

procedure Tem_Vista_CustomLabelColors.SetTextActive(Value: TColor);
begin
if FTextActive=Value then exit; FTextActive:=Value;
if Assigned(OnChanged) then OnChanged(self);
end;

procedure Tem_Vista_CustomLabelColors.SetDelineateActive(Value: TColor);
begin
if FDelineateActive=Value then exit; FDelineateActive:=Value;
if Assigned(OnChanged) then OnChanged(self);
end;

procedure Tem_Vista_CustomLabelColors.SetAutoHighlight(Value: boolean);
begin
if FAutoHighlight=Value then exit; FAutoHighlight:=Value;
if Assigned(OnChanged) then OnChanged(self);
end;

procedure Tem_Vista_CustomLabelColors.SetAutoShadow(Value: boolean);
begin
if FAutoShadow=Value then exit; FAutoShadow:=Value;
if Assigned(OnChanged) then OnChanged(self);
end;

procedure Tem_Vista_CustomLabelColors.SetColorHighlightShift(Value: integer);
begin
if FColorHighlightShift=Value then exit; FColorHighlightShift:=Value;
if Assigned(OnChanged) then OnChanged(self);
end;

procedure Tem_Vista_CustomLabelColors.SetColorShadowShift(Value: integer);
begin
if FColorShadowShift=Value then exit; FColorShadowShift:=Value;
if Assigned(OnChanged) then OnChanged(self);
end;

procedure Tem_Vista_CustomLabelColors.SetBackgroundActive(Value: TColor);
```

```
begin
if FBackgroundActive=Value then exit; FBackgroundActive:=Value;
if Assigned(OnChanged) then OnChanged(self);
end;

constructor Tem_Vista_GrBoxColors.Create;
begin
inherited;
FCaption:= clBtnFace;
FCaptionActive:= clBtnFace;
FClient:= clBtnFace;
FClientActive:= clBtnFace;
end;

procedure Tem_Vista_GrBoxColors.SetCaption(Value: TColor);
begin FCaption:= Value; if Assigned(OnChanged) then OnChanged(self); end;

procedure Tem_Vista_GrBoxColors.SetCaptionActive(Value: TColor);
begin FCaptionActive:= Value; if Assigned(OnChanged) then OnChanged(self); end;

procedure Tem_Vista_GrBoxColors.SetClient(Value: TColor);
begin FClient:= Value; if Assigned(OnChanged) then OnChanged(self); end;

procedure Tem_Vista_GrBoxColors.SetClientActive(Value: TColor);
begin FClientActive:= Value; if Assigned(OnChanged) then OnChanged(self); end;

constructor Tem_Vista_ExtBevel.Create;
begin
inherited;
FActive:= true;
FBevelPenStyle:= psSolid;
FBevelPenWidth:= 1;
end;

procedure Tem_Vista_ExtBevel.SetActive(Value: boolean);
begin
if FActive = Value then exit;
FActive:= Value; if Assigned(OnChanged) then OnChanged(self);
end;

procedure Tem_Vista_ExtBevel.SetBevelPenStyle(Value: TPenStyle);
begin
if FBevelPenStyle = Value then exit;
FBevelPenStyle:= Value; if Assigned(OnChanged) then OnChanged(self);
end;

procedure Tem_Vista_ExtBevel.SetBevelPenWidth(Value: word);
begin
if FBevelPenWidth = Value then exit;
FBevelPenWidth:= Value; if Assigned(OnChanged) then OnChanged(self);
end;

procedure Tem_Vista_ExtBevel.SetInteriorOffset(Value: word);
begin
if FInteriorOffset = Value then exit;
FInteriorOffset:= Value; if Assigned(OnChanged) then OnChanged(self);
end;

constructor Tem_Vista_CustomListBoxItemStyle.Create;
begin
inherited Create;
FBevel:= Tem_Vista_Bevel.Create;
FFont:= TFont.Create;
end;

destructor Tem_Vista_CustomListBoxItemStyle.Destroy;
begin
FFont.Free; FBevel.Free; inherited;
end;
```

```
procedure Tem_Vista_CustomListBoxItemStyle.SetOnChanged(Value: TNotifyEvent);
begin
FOnChanged:= Value; FBevel.OnChanged:= Value;
end;

procedure Tem_Vista_CustomListBoxItemStyle.SetColor(Value: TColor);
begin
if FColor = Value then exit;
FColor:= Value; if Assigned(OnChanged) then OnChanged(self);
end;

procedure Tem_Vista_CustomListBoxItemStyle.SetDelineateColor(Value: TColor);
begin
if FDelineateColor = Value then exit;
FDelineateColor:= Value; if Assigned(OnChanged) then OnChanged(self);
end;

procedure Tem_Vista_CustomListBoxItemStyle.SetFont(Value: TFont);
begin
FFont.Assign(Value); if Assigned(OnChanged) then OnChanged(self);
end;

procedure Tem_Vista_CustomListBoxItemStyle.SetTextStyle(Value:
Tem_Vista_TextStyle);
begin
FTextStyle:= Value; if Assigned(OnChanged) then OnChanged(self);
end;

constructor Tem_Vista_ListBoxItemStyle.Create;
begin
inherited Create;
FGradient:= TGradient.Create;
FTextGradient:= TGradient.Create;
end;

destructor Tem_Vista_ListBoxItemStyle.Destroy;
begin
FGradient.Free;
FTextGradient.Free;
inherited;
end;

procedure Tem_Vista_ListBoxItemStyle.SetOnChanged(Value: TNotifyEvent);
begin
inherited SetOnChanged(Value);
FGradient.OnChanged:= Value;
end;

constructor Tem_Vista_AskListBoxItemStyle.Create;
begin
inherited Create;
FBtnFont:= TFont.Create;
end;

destructor Tem_Vista_AskListBoxItemStyle.Destroy;
begin
FBtnFont.Free; inherited;
end;

procedure Tem_Vista_AskListBoxItemStyle.SetBtnColor(Value: TColor);
begin
if FBtnColor = Value then exit;
FBtnColor:= Value; if Assigned(OnChanged) then OnChanged(self);
end;

procedure Tem_Vista_AskListBoxItemStyle.SetBtnFont(Value: TFont);
begin
FBtnFont.Assign(Value); if Assigned(OnChanged) then OnChanged(self);
```

```
end;

procedure Tem_Vista_AskListBoxItemStyle.SetBtnTextStyle(Value:
Tem_Vista_TextStyle);
begin
FBtnTextStyle:= Value; if Assigned(OnChanged) then OnChanged(self);
end;

constructor Tem_Vista_CustomBoxStyle.Create;
begin
inherited;
FPenStyle:= psSolid;
FHighlightColor:= clBtnHighlight;
FShadowColor:= clBtnShadow;
end;

procedure Tem_Vista_CustomBoxStyle.SetPenStyle(Value: TPenStyle);
begin  FPenStyle:= Value; if Assigned(OnChanged) then OnChanged(self); end;

procedure Tem_Vista_CustomBoxStyle.SetHighlightColor(Value: TColor);
begin  FHighlightColor:= Value; if Assigned(OnChanged) then OnChanged(self);
end;

procedure Tem_Vista_CustomBoxStyle.SetShadowColor(Value: TColor);
begin  FShadowColor:= Value; if Assigned(OnChanged) then OnChanged(self); end;

constructor Tem_Vista_CustomTextBoxStyle.Create;
begin
inherited;
FTextColor:= clBlack;
FBackgroundColor:= clWindow;
end;

procedure Tem_Vista_CustomTextBoxStyle.SetTextColor(Value: TColor);
begin  FTextColor:= Value; if Assigned(OnChanged) then OnChanged(self); end;

procedure Tem_Vista_CustomTextBoxStyle.SetBackgroundColor(Value: TColor);
begin  FBackgroundColor:= Value; if Assigned(OnChanged) then OnChanged(self);
end;

constructor Tem_Vista_BevelLines.Create;
begin
inherited;
FStyle:= bvLowered;
FThickness:= 1;
end;

procedure Tem_Vista_BevelLines.SetCount(Value: cardinal);
begin FCount:= Value; if Assigned(OnChanged) then OnChanged(self); end;
procedure Tem_Vista_BevelLines.SetStep(Value: cardinal);
begin FStep:= Value; if Assigned(OnChanged) then OnChanged(self); end;
procedure Tem_Vista_BevelLines.SetOrigin(Value: Tem_Vista_Origin);
begin FOrigin:= Value; if Assigned(OnChanged) then OnChanged(self); end;
procedure Tem_Vista_BevelLines.SetStyle(Value: TPanelBevel);
begin FStyle:= Value; if Assigned(OnChanged) then OnChanged(self); end;
procedure Tem_Vista_BevelLines.SetBold(Value: boolean);
begin FBold:= Value; if Assigned(OnChanged) then OnChanged(self); end;
procedure Tem_Vista_BevelLines.SetThickness(Value: byte);
begin FThickness:= Value; if Assigned(OnChanged) then OnChanged(self); end;
procedure Tem_Vista_BevelLines.SetIgnoreBorder(Value: boolean);
begin FIgnoreBorder:= Value; if Assigned(OnChanged) then OnChanged(self); end;

{ TGradient }

procedure TGradient.Draw(DC: HDC; r: TRect; PenStyle, PenWidth: integer);
var
i,j,x,y,x2,y2,h,w,NumberOfColors:integer;
c1F,c2F,c3F:byte;
c1T,c2T,c3T:byte;
```

```
c1D,c2D,c3D:integer;
_R,_G,_B:byte;
Pen,OldPen:HPen;
FillBrush:HBrush;
BufferBmp, OldBMP:HBITMAP;
BufferDC,TargetDC:HDC;
ColorR:TRect;
LOGBRUSH:TLOGBRUSH;

procedure SwapColors;
var TempColor: Longint;
begin
TempColor:= FRGBFromColor; FRGBFromColor:= FRGBToColor; FRGBToColor:= TempColor;
end;
begin
if (Steps = 1)or(GetDeviceCaps( DC, BITSPIXEL )<16) then
begin exit;
FillBrush:= CreateSolidBrush( ColorToRGB( FromColor ) );
FillRect( DC, r, FillBrush );
DeleteObject( FillBrush );
exit;
end;
x:=r.left; y:=r.top; h:=r.bottom-r.top; w:=r.right-r.left;
x2:= 0; y2:= 0; pen:= 0; oldpen:= 0; BufferDC:= 0;

if Orientation = fgdHorzConvergent then
begin
FOrientation:= fgdHorizontal;
Draw(DC, Rect(R.Left, R.Top, R.Right, R.Bottom- h div 2), PenStyle, PenWidth);
SwapColors;
Draw(DC, Rect(R.Left, R.Top+ h div 2, R.Right, R.Bottom), PenStyle, PenWidth);
SwapColors;
FOrientation:= fgdHorzConvergent;
exit;
end;
if Orientation = fgdVertConvergent then
begin
FOrientation:= fgdVertical;
Draw(DC, Rect(R.Left, R.Top, R.Right- w div 2, R.Bottom), PenStyle, PenWidth);
SwapColors;
Draw(DC, Rect(R.Left+ w div 2, R.Top, R.Right, R.Bottom), PenStyle, PenWidth);
SwapColors;
FOrientation:= fgdVertConvergent;
exit;
end;

c1F:=Byte(FRGBFromColor );
c2F:=Byte(WORD(FRGBFromColor) shr 8);
c3F:=Byte(FRGBFromColor shr 16 );
c1T:=Byte(FRGBToColor );
c2T:=Byte(WORD(FRGBToColor) shr 8);
c3T:=Byte(FRGBToColor shr 16 );
c1D:=c1T- c1F;
c2D:=c2T- c2F;
c3D:=c3T- c3F;

if BufferedDraw then
begin
BufferDC:= CreateCompatibleDC(DC);
BufferBmp:= CreateBitmap( w, h, GetDeviceCaps( DC, PLANES ), GetDeviceCaps( DC,
BITSPIXEL ), nil );
OldBMP:= SelectObject( BufferDC, BufferBmp );
SetMapMode( BufferDC, GetMapMode(DC) );
TargetDC:= BufferDC;
end else TargetDC:= DC;

case Orientation of
fgdHorizontal:
begin
```

```
NumberOfColors:= min( Steps, h );
ColorR.Left:= r.left; ColorR.Right:= r.right;
end;
fgdVertical:
begin
NumberOfColors:= min( Steps, w );
ColorR.Top:= r.top; ColorR.Bottom:= r.bottom;
end;
fgdLeftBias, fgdRightBias:
begin
NumberOfColors:= min( Steps, w+h );
if PenStyle=0 then PenStyle:=PS_SOLID; if PenWidth=0 then PenWidth:=1;
y2:= y;
if Orientation = fgdLeftBias then x2:= x else
begin x:= r.right; x2:= r.right; end;
end;
else{fgdRectanEm_Vista}
begin
h:=h div 2; w:=w div 2;
NumberOfColors:= min( Steps, min(w,h) );
end;
end;
LOGBRUSH.lbStyle:= BS_HATCHED;
LOGBRUSH.lbHatch:= Ord(BrushStyle)- Ord(bsHorizontal);
for i:=0 to NumberOfColors-1 do
begin
_R:= c1F+ MulDiv(i, c1D, NumberOfColors- 1);
_G:= c2F+ MulDiv(i, c2D, NumberOfColors- 1);
_B:= c3F+ MulDiv(i, c3D, NumberOfColors- 1);

case Orientation of
fgdHorizontal, fgdVertical, fgdRectanEm_Vista:
begin
if BrushStyle = bsSOLID then
FillBrush:= CreateSolidBrush( RGB( _R, _G, _B ) )
else
begin
LOGBRUSH.lbColor:= RGB( _R, _G, _B );
FillBrush:= CreateBrushIndirect(LOGBRUSH);
end;

case Orientation of
fgdHorizontal:
begin
if fReverse then begin
ColorR.Top:= r.bottom- MulDiv( i, h, NumberOfColors);
ColorR.Bottom:= r.bottom- MulDiv( i+ 1, h, NumberOfColors);
end else begin
ColorR.Top:= r.top+ MulDiv( i , h, NumberOfColors);
ColorR.Bottom:= r.top+ MulDiv( i+ 1, h, NumberOfColors);
end;
end;
fgdVertical:
begin
if fReverse then begin
ColorR.Left:= r.right- MulDiv( i,w, NumberOfColors);
ColorR.Right:= r.right- MulDiv( i+ 1, w, NumberOfColors);
end else begin
ColorR.Left:= r.left+ MulDiv( i,w, NumberOfColors);
ColorR.Right:= r.left+ MulDiv( i+ 1, w, NumberOfColors);
end;
end;
fgdRectanEm_Vista:
begin
ColorR.Top:= r.top+ MulDiv( i, h, NumberOfColors);
ColorR.Bottom:= r.bottom- MulDiv( i, h, NumberOfColors);
ColorR.Left:= r.left+ MulDiv( i, w, NumberOfColors);
ColorR.Right:= r.right - MulDiv( i, w, NumberOfColors);
end;
```

```
end;
FillRect( TargetDC, ColorR, FillBrush );
DeleteObject( FillBrush );
end;
else {fgdLeftBias, fgdRightBias:}
begin
if Pen <> 0 then
DeleteObject(SelectObject( TargetDC, OldPen ));
Pen:= CreatePen( PenStyle, PenWidth, RGB( _R, _G, _B ) );
OldPen:= SelectObject( TargetDC, Pen );
for j:= 1 to MulDiv( i+ 1, h+w, NumberOfColors)- MulDiv( i, h+w, NumberOfColors)
do
begin
case Orientation of
fgdLeftBias:
begin
if y >= r.bottom then inc( x, PenWidth ) else y:= y+ PenWidth;
if x2 >= r.right then inc( y2, PenWidth ) else x2:= x2+ PenWidth;
MoveToEx( TargetDC, x, y, nil );
LineTo( TargetDC, x2, y2 );
end;
else{fgdRightBias:}
begin
if x <= r.left then inc( y, PenWidth ) else x:= x- PenWidth;
if y2 >= r.bottom then dec( x2, PenWidth ) else y2:= y2+ PenWidth;
MoveToEx( TargetDC, x, y, nil );
LineTo( TargetDC, x2, y2 );
end;
end;
end;
DeleteObject( SelectObject( TargetDC, OldPen ) );
end;
end;
if NumberOfColors=0 then exit;
if i/NumberOfColors*100 > PercentFilling then break;
end;

if BufferedDraw then
begin
BitBlt( DC, 0, 0, r.right-r.left, r.bottom-r.top, BufferDC, 0, 0, SRCCOPY );
DeleteObject( SelectObject( BufferDC, OldBMP ) );
DeleteDC( BufferDC );
end;

end;

function Tem_Vista_Bevel.BordersHeight: integer;
begin
Result:= 0;
if Inner <> bvNone then
begin
if fsdTop in Sides then inc(Result);
if fsdBottom in Sides then
if Bold then inc(Result, 1) else inc(Result);
end;
if Outer <> bvNone then
begin
if fsdTop in Sides then inc(Result);
if fsdBottom in Sides then
if Bold then inc(Result, 1) else inc(Result);
end;
end;

function Tem_Vista_Bevel.BordersWidth: integer;
begin
Result:= 0;
if Inner <> bvNone then
begin
if fsdLeft in Sides then inc(Result);
```

```
if fsdRight in Sides then
if Bold then inc(Result, 1) else inc(Result);
end;
if Outer <> bvNone then
begin
if fsdLeft in Sides then inc(Result);
if fsdRight in Sides then
if Bold then inc(Result, 1) else inc(Result);
end;
end;

function Tem_Vista_CustomListBoxItemStyle.HighlightColor: TColor;
begin
Result:= incColor(Color, 60);
end;

function Tem_Vista_CustomListBoxItemStyle.ShadowColor: TColor;
begin
Result:= decColor(Color, 60);
end;

end.
```