МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Центральноукраїнський національний технічний університет

Кафедра кібербезпеки та програмного забезпечення

На правах рукопису

Дзюбинський Олексій Володимирович

Програмне забезпечення системи кібербезпеки проактивного захисту на основі використання API-функцій

Спеціальність: 123 «Комп'ютерна інженерія»

Освітній ступінь: бакалавр

Науковий керівник:

Савеленко Олена Костянтинівна

_____

(підпис) (дата)

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри

(_____) Смірнов О.А.

(підпис) ПІБ

_____2021 р.

Кропивницький – 2021

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

Факультет м*еханіко-технологічний*

Кафедра к*ібербезпеки та програмного забезпечення*

Освітньо-кваліфікаційний рівень *бакалавр*

Спеціальність *123 «Комп'ютерна інженерія»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ О.А. Смірнов
"__" _____2021 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Дзюбинському Олексію Володимировичу
(прізвище, ім'я, по батькові)

1. Тема проекту *Програмне забезпечення системи кібербезпеки проактивного захисту на основі використання API-функцій*

керівник проекту *Савеленко Олена Костянтинівна*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу № 37-02 від 14. 04.2021 року

2. Строк подання студентом проекту *22.05.2021 р.*

3. Вихідні дані до проекту *Пояснювальна записка, графічні матеріали, робоча програма*

4. 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.

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

Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схеми головної програми	*1 аркуш*
Блок-схема підпрограми	*3 аркуші*

6. Дата видачі завдання 14. 04.2021 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної бакалаврської роботи	Строк виконання етапів роботи	Примітка
1.	Аналіз існуючих систем	14.04. 2021 р.	
2.	Постановка задачі, оформлення ТЗ	18.04.2021 р.	
3.	Розробка моделі компонента	25.04.2021 р.	
4.	Розробка структур даних	28.04.2021 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.04.2021 р.	
6.	Програмування алгоритмів	05.05.2021 р.	
7.	Оформлення ПЗ	17.05.2021 р.	
8.	Попередній захист проекту	22.05.2021 р.	

Студент _____ О.В.Дзюбинський
(підпис) (прізвище та ініціали)

Керівник проекту _____ О.К.Савеленко
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дзюбинський О.А. Програмне забезпечення системи кібербезпеки проактивного захисту на основі використання API-функцій. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2021.

В даній кваліфікаційній бакалаврській роботі розроблене програмне забезпечення, яке призначене для проактивного захисту інформаційних та програмних ресурсів персональних комп'ютерів на основі використання API-функцій від вірусів.

Метою розробки є програмне забезпечення системи кібербезпеки проактивного захисту на основі використання API-функцій.

Результат роботи – програмна реалізація системи кібербезпеки проактивного захисту на основі використання.

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

В процесі побудови архітектури ПЗ системи в основу було покладено поєднання двох методик реалізації системи проактивного захисту: поведінковий аналізатор та обмеження виконання операцій. Це дозволило одержати покращені функціональні характеристики системи: забезпечити меншу кількість звернень до користувача у порівнянні з системами, побудованими на IPS-методах; низьке споживання системних ресурсів; невибагливість до апаратного забезпечення ПК.

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

Програма може використовуватися на ПК Intel Core™ з ОС Windows або сумісні з ним ПК.

Ключові слова: кібербезпека, проактивний захист, API-функції.

ANNOTATION

Dzyubynsky O.A. Proactive defense cybersecurity software based on the use of API functions.
123 Computer Engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2021

In this qualifying bachelor's thesis, software has been developed that is designed to proactively protect the information and software resources of personal computers based on the use of API functions against viruses.

The purpose of the development is the software of the cybersecurity system of proactive protection based on the use of API functions.

The result is a software implementation of a proactive protection cybersecurity system based on use.

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.

In the process of building the software architecture of the system, a combination of two methods of implementing a proactive protection system was used: a behavioral analyzer and operations restriction. This allowed to obtain improved functional characteristics of the system: to provide fewer requests to the user compared to systems based on IPS-methods; low consumption of system resources; unpretentiousness to PC hardware.

Developed user-friendly interface. Instructions for working with software are given.

The program can be used on Intel Core PCs with Windows or compatible PCs.

Keywords: cybersecurity, proactive protection, API functions.

ВСТУП

Інформаційна безпека з моменту появи можливості передачі даних за допомогою глобальної мережі Internet, завжди стояла на першому місці. Найбільший інтерес мережа Internet представляє саме в якості знаряддя для здійснення злочинів в сфері економіки й фінансів. За свідченням експертів, найпривабливішим сектором економіки для злочинців є комп'ютерна кредитно-фінансова система.

Найпоширеніші комп'ютерні злочини вчиняються шляхом несанкціонованого доступу до інформаційних та програмних ресурсів за допомогою телекомунікаційних мереж. Корпоративні організації та банки, через конкурентну боротьбу між собою, змушені для забезпечення зручності й швидкодії роботи із клієнтами, надавати їм можливість доступу з мереж загального користування до своїх локальних та обчислювальних систем з метою аудиту власних цінних паперів та рахунків. І, не дивлячись на складність їх систем безпеки та досить сильного захисту проти атак зловмисників, систем ефективного контролю на сьогодення явно недостатньо.

Фахівці в області інформаційної безпеки такі, як Конєєв Іскандер Рустамович і Касперський Євгеній Валентинович стверджують, що існуючі методи захисту інформації застарівають. Вони відзначили, що в другій половині 2020 були зафіксовані одиночні віруси нового покоління.

Боротьба між системами інформаційної безпеки й вірусами нескінченний процес. У результаті, протидія погрозам сприяє їхньому ускладненню, а ускладнення погроз веде до вдосконалювання засобів протидії. Ця спіраль призвела до того, що безупинно зростає швидкість появи вірусів. А існуючіі системи захисту сьогодні несуть вже не захисну функцію, а скоріше використовуються в якості засобу визначення оцінки одержаних збитків і видалення зарази.

Таким чином, вище зазначена ситуація без впровадження інноваційних технологій у системи інформаційної безпеки, незабаром може призвести до того, що злом комп'ютерних кредитно-фінансових систем стане звичною справою.

Традиційні антивірусні програми, розроблені на основі сигнатурного аналізу вже виявлених вірусів, не здатні виявити і нейтралізувати новий тип вірусу чи шпіонської програми. Адже для того, щоб згенерувати сигнатуру, антивірусній компанії необхідно одержати цей вірус, виділити специфічний для нього фрагмент, занести до бази даних. Це досить довгий проміжок часу, на протязі якого програми-антивіруси не здатні протистояти новому ворогу. Тобто, класичні сигнатурні віруси на сьогодення не здатні попередити глобальні епідемії комп'ютерних мереж. Тому розробка систем інформаційної безпеки з використанням інноваційних технологій та їх вбудова в автоматизовані інформаційні системи є актуальним завданням на даний час.

Проактивний захист – одна із сучасних технологій. Суть технології в тому, що аналізу піддаються не алгоритми атаки, розкрадання й видалення даних, а сукупність дій, чинених вірусами в автоматизованій системі. Головною перевагою проактивного захисту є здатність виявити й заблокувати зовсім новий вірус, сигнатури до якого ще немає у базах сигнатур наявної системи захисту. Тобто, використання проактивних методів захисту дозволить виявити погрозу появи вірусів на ПК, при цьому – кількість хибних спрацювань фактично відсутня.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

Згідно технічного завдання (ТЗ) метою виконання кваліфікаційної бакалаврської роботи (КБР) є розробка програмного забезпечення (ПЗ) системи кібербезпеки проактивного захисту інформаційних та програмних ресурсів персональних комп'ютерів на основі використання API-функцій від вірусів (в подальшому –системи).

ПЗ системи повинне мати високу ступінь мобільності та адаптивності, не бути жорстоко прив'язаним до конкретного типу обладнання та його технічних характеристик і буде чітко виконувати покладені на нього функції. Якщо всі означені вимоги будуть забезпечені в процесі виконання КБР, то розроблена система буде мати досить широку область застосування. Розглянемо це питання.

1.1 Призначення системи

Задачі антивірусного захисту – це самостійна і складна задача в області захисту інформації (ЗІ). Відзначимо, що домінуючим на сьогодення підходом до антивірусного захисту є розробка засобів виявлення вірусів по визначених ознаках (сигнатурний аналіз) з послідуючим їх видаленням з носія інформації. Такому підходу присутні наступні недоліки:

Щоб виявити вірус в системі, яка підлягає захисту, необхідно однозначно визначити його класифікаційні признаки, зокрема – визначити сигнатуру. З урахуванням необхідного часу на визначення формалізованих признаків вірусу, а також часу на оновлення БД вірусних сигнатур, протидія вірусу здійснюється уже після нанесення збитків вірусною атакою.

Інтенсивність появи нових вірусів невпинно зростає, а відповідно – зростають бази даних (БД) формальних признаків вірусів (сигнатур). Це, в свою чергу, призводить до росту навантаження антивірусних засобів захисту та зменшення швидкодії роботи ПК.

Таким чином, робимо цілком обґрунтований висновок, що розповсюджений на сьогодення підхід до побудови систем антивірусного захисту не можна вважати дійсно ефективним. Тому розробка систем проактивного захисту, які базуються на використанні новітніх технологій, на даний час є дійсно актуальним питанням. Але, перш ніж приступити до визначення структури побудови та означення функцій майбутньої системи, необхідно визначитись з існуючою термінологією по цьому питанню та означити ціль вірусних атак (а не їх способи, яких дуже багато).

Основною ціллю вірусних атак є віддалене, без безпосереднього контакту з ПК, який підлягає захисту, звершення будь-яких дій. Але ці дії напрямки пов'язані з запуском на ПК якої-небудь програми/процесу. Більшість таких атак може бути нейтралізовано за рахунок роботи інших механізмів захисту програмних продуктів та інформаційних масивів:

- використання механізму забезпечення замкнутості програмного середовища, який призваний протидіяти несанкціонованим діям (саме таким і є програми-віруси);

- визначення та моніторинг кола програм, запуск яких дозволений на конкретному ПК. Адже якщо навіть програма-вірус і попадає на ПК, вона буде ідентифікована в якості забороненої на запуск.

Але для реалізації цих задач потрібні складні системи, які коштують надто дорого, адже означені недоліки присутні: офісним додаткам операційних систем; віртуальним машинам JVM, JAVA; офісним додаткам Microsoft, які дозволяють створювати макроси із-за недосконалості захисту. Тому проблему антивірусної протидії будемо розглядати в загальному аспекті наявності віртуальних машин та вбудованих інтерпретаторів команд офісних додатків.

Означимо два класи: атаки, ціллю яких є звершення дій, які нам ще невідомі; атаки, які реалізовуються з ціллю розповсюдження вірусів, як всередині захищеного комп'ютера, та і поза ним – по мережі.

По першому класу атака представляє собою програму, виконану у вигляді макросу, який є складовою документу. Ця програма запускається при читанні даного документа інтерпретатором команд.

По другому класу захист даних користувача напрямки пов'язаний з протидією розповсюдження вірусу на комп'ютері, який підлягає захисту. І якщо навіть користувач має змогу використати дорогі системи захисту від несанкціонованого доступу (НСД), вони не забезпечать швидкої ідентифікації наявного вірусу та процесу його «лікування».

І тому питання розробки автономних систем саме антивірусного захисту на даний час є дійсно актуальним. Про це свідчать наступні факти. По даних компанії «Ернест енд Янг» - 43 % випадків порушення безпеки відносяться саме на вірусні атаки, як це показано на рисунку 1.1.

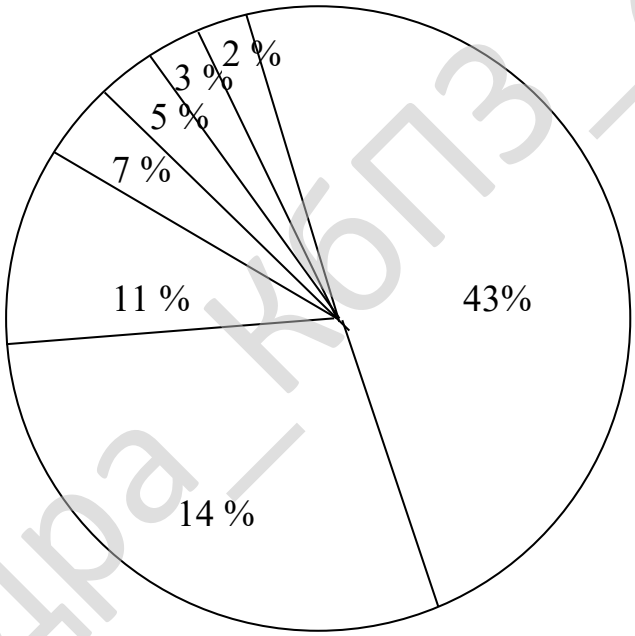

Рисунок 1.1 – Структура випадків порушення безпеки

Як видно з рисунку 1.1, на інші випадки порушення безпеки приходиться: 2 % - атака «відмова» на обслуговування; 3 % - несанкціонований доступ колишніх співробітників; 5 % - несанкціонований доступ співробітників; 7 % - несанкціонований доступ з мережі; 11 % - незаконне розсилання спаму; 14 % - саботаж, навмисне ушкодження, знищення: інформації, систем, мереж.

Ці факти підтверджують той аспект, що з різних причин основна боротьба з порушенням інформації направлена проти випадків НСД та несанкціонованого використання інформації і програмних продуктів. Навіть розробники комплексних систем інформаційної безпеки у переважній кількості випадків обмежуються тільки рекомендаціями щодо вибору антивірусного пакету.

Разом з тим, небезпека «зараження» обчислювальних мереж реальна для будь-якого суб'єкта господарської діяльності, і реальний розвиток вірусної епідемії можна одержати в локальних та глобальних мережах у будь-який час. Тоді антивірусний захист стає досить складною задачею вже не тільки з точки зору технічного захисту та наявності спеціального програмного забезпечення для захисту від вірусів, а й буде вимагати значних фінансових витрат (особливо – на відновлення знищених в результаті вірусної атаки даних).

Рішення цього питання можливе лише шляхом використання методики проактивного захисту, який за останні роки визнається провідними спеціалістами-антивірусологами комп'ютерних систем та мереж, як найефективніший засіб захисту від вірусів будь-якого типу.

Автоматизована система проактивного захисту розглядається окремо від іншого інформаційного простору користувача, але цілком зрозуміло, що ряд її специфічних механізмів і технологій не можуть не впливати на перелік наданих вимог користувача. Тому система проактивного захисту повинна працювати на рівні ядра системи, яке захищається таким чином, щоб жодна дія користувача або спроба запустити процес не відбувались поза увагою системи безпеки.

Таким чином, основне призначення системи проактивного захисту, яка підлягає розробці, полягає в тому, що аналізу будуть підлягати не дані на носіях інформації, а поводження самих даних – сукупність дій, чинених у системі. Якщо вважати якийсь клас даних шкідливим, то можна визначити й характерні риси поводження.

На даний час на ринку програмних продуктів країни такі пакети/системи фактично відсутні. Тому розробка системи проактивного захисту є дійсно своєчасною та буде мати високу ступінь новизни.

1.1 Область застосування

Фактично, організаційні заходи (це прерогатива замовника) не потребують, в разі впровадження такої антивірусної системи, суттєвої реорганізації існуючих виробничих відносин. Враховуючи вищеозначене та вимоги ТЗ, розроблена в процесі виконання КБР у система проактивного захисту дозволить забезпечити:

- своєчасне визначення API-функцій, які підлягають перевірці, при запуску процесу користувачем чи зловмисником;

- своєчасне виявлення спроби проникнення до ПК, який підлягає захисту, програми – вірусу шляхом перехвату та аналізу API-функцій, поведінка яких є підозрілою;

- блокування будь-яких несанкціонованих/підозрілих дій в ПК, який захищається.

Система планується для розробки як автономна, але для розширення області її застосування необхідно передбачити можливість її роботи в якості складової більш потужного програмного комплексу аналогічного класу та спрямування. Для цього слід передбачити забезпечення нею наступних вимог:- система повинна бути сумісною з існуючими операційними системами родини Windows; система проактивного захисту не повинна впливати на логіку інших додатків, що використовуються; ПЗ системи не слід жорстко прив’язувати до апаратної конфігурації ПК, оскільки його окремі компоненти можуть і повинні замінюватись новими компонентами сучасної модифікації та більш високого рівня.

Враховуючи що сучасний користувач потребує цілеспрямованого і надійного програмного забезпечення з високим рівнем антивірусного захисту, область застосування системи, яка буде розроблена в процесі виконання КБР, можна охарактеризувати одним містким словом – всюди: установи, організації, підприємства будь-якої форми власності, приватні користувачі, навчальні заклади тощо.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

Предметом розробки кваліфікаційної бакалаврської роботи (КБР) є програмне забезпечення проактивного захисту на основі використання API-функцій.

Для більш детального визначення функцій та структури майбутньої системи, визначення методики та принципів її побудови, необхідно розглянути існуючі системи – аналоги та провести аналіз їх структури та технічних властивостей.

2.1 Огляд існуючих систем, технологій, архітектур та програмних рішень по профілю теми дипломного проекту

Антивірусне програмне забезпечення так тісно увійшло до життя кожного користувача комп'ютерів, що рідко хто замислюється про наявність альтернативного рішення. Відношення користувачів до антивірусів варіюється від повної упевненості в захисті до усвідомлення того, що головне - витримати перший удар. Дійсно, антивіруси сьогодні несуть вже не стільки захисну функцію, а швидше використовуються в якості засобу оцінки нанесеного збитку і видалення зарази.

Антивірусні компанії, повністю усвідомлюючи те, що вірус – це якийсь алгоритм, що включає не тільки програму зі всіма командами, але і певні дії, що призводять до пошкодження даних, наполегливо продовжують йти найбільш простим шляхом. У результаті традиційні антивірусні рішення фактично не здатні поодинці протистояти якісній різноманітності існуючих погроз.

Існують програми проактивного захисту комп'ютерів, які на відміну від реактивних систем, використовуваних в даний час, здатні виявляти нові погрози. Розглянемо найбільш відомі з них.

WinPatrol (http://www.winpatrol.com), статус freeware, підтримується сімейством Windows.

WinPatrol робить знімок критичних системних ресурсів і попереджає у разі будь-яких змін. Після запуску в треї біля годинника з'явиться значок із зображенням собаки, названої Scotty the Windows Watch Dog. Ось тепер цей самий Scotty WWD безперервно і стежитиме за всім, що відбувається на довіреному йому комп'ютері, «рознюхає» всі наявні віруси: Adware, Spyware, трояни, що пробралися на ПК користувача. Scotty дозволяє підтверджувати установку будь-яких нових програм на комп'ютері. Викликавши програму, отримаємо вікно програми з декількома вкладками (рисунок 2.1).

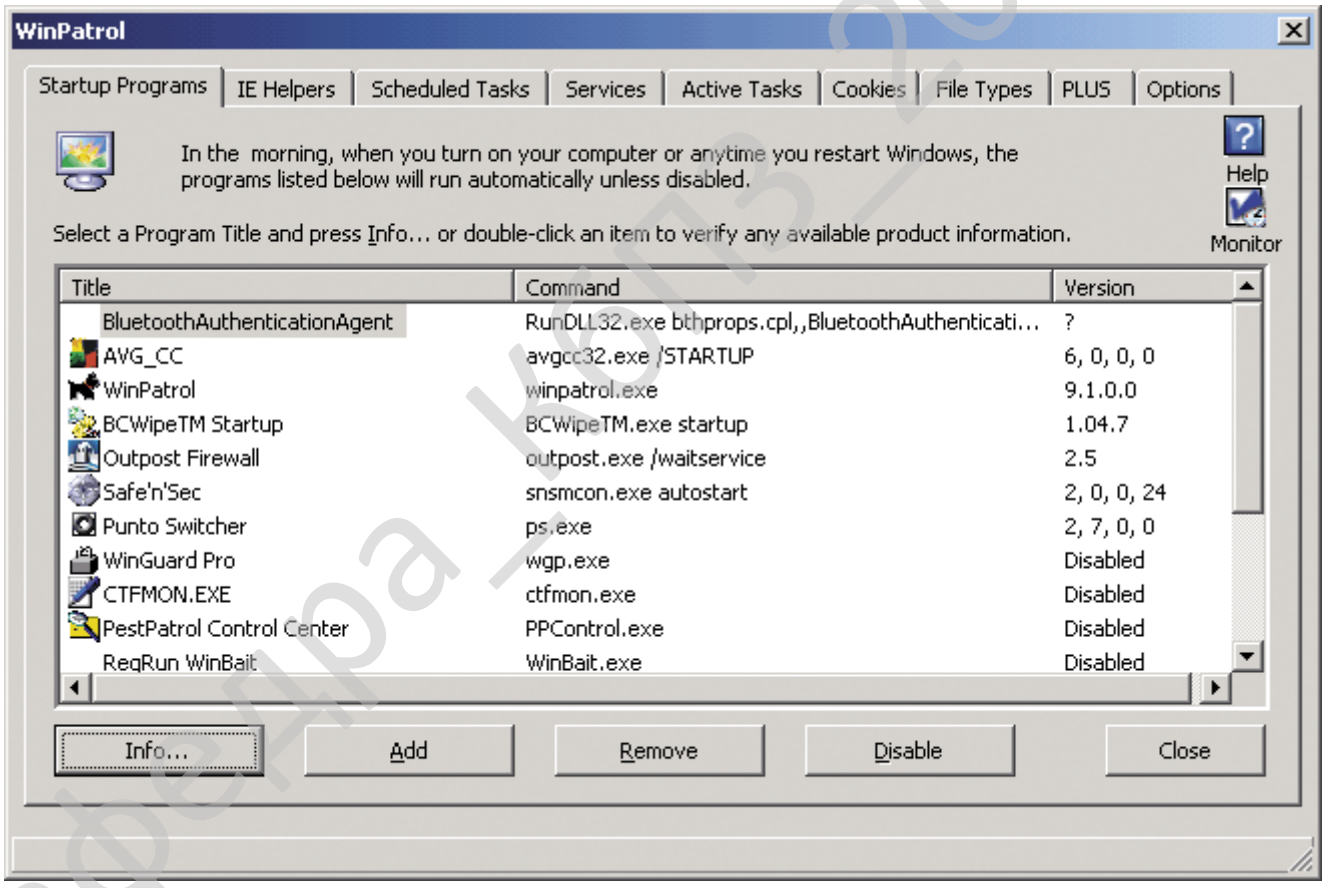

Рисунок 2.1 – Скриншот основного вікна WinPatrol

Менеджер процесів AnVir Task Manager. Перше, що приходить в голову після знайомства з менеджером процесів AnVir Task Manager (http://anvir.com/anvirrus.exe), що це не серйозно. Причиною такого сприйняття є

невеликий розмір програми, трохи менше 4311 Кб. Але не.дивлячись на такий маленький розмір, програма допомагає:

- отримати повну інформацію про запущені процеси (шлях до файлу, опис, час роботи, батьківський процес, командний рядок, використання процесора, пам'яті, список використовуваних dll, файлів, драйверів, потоків, вікон тощо);

- здійснити лікування зараженого комп'ютера від вірусів, spyware і резидентних вірусів-троянів;

- управляти додатками, що запускаються при старті Windows, з можливістю відкладеного (через 1 хвилину) запуску і блокування додавання нових програм (рисунок 2.2).

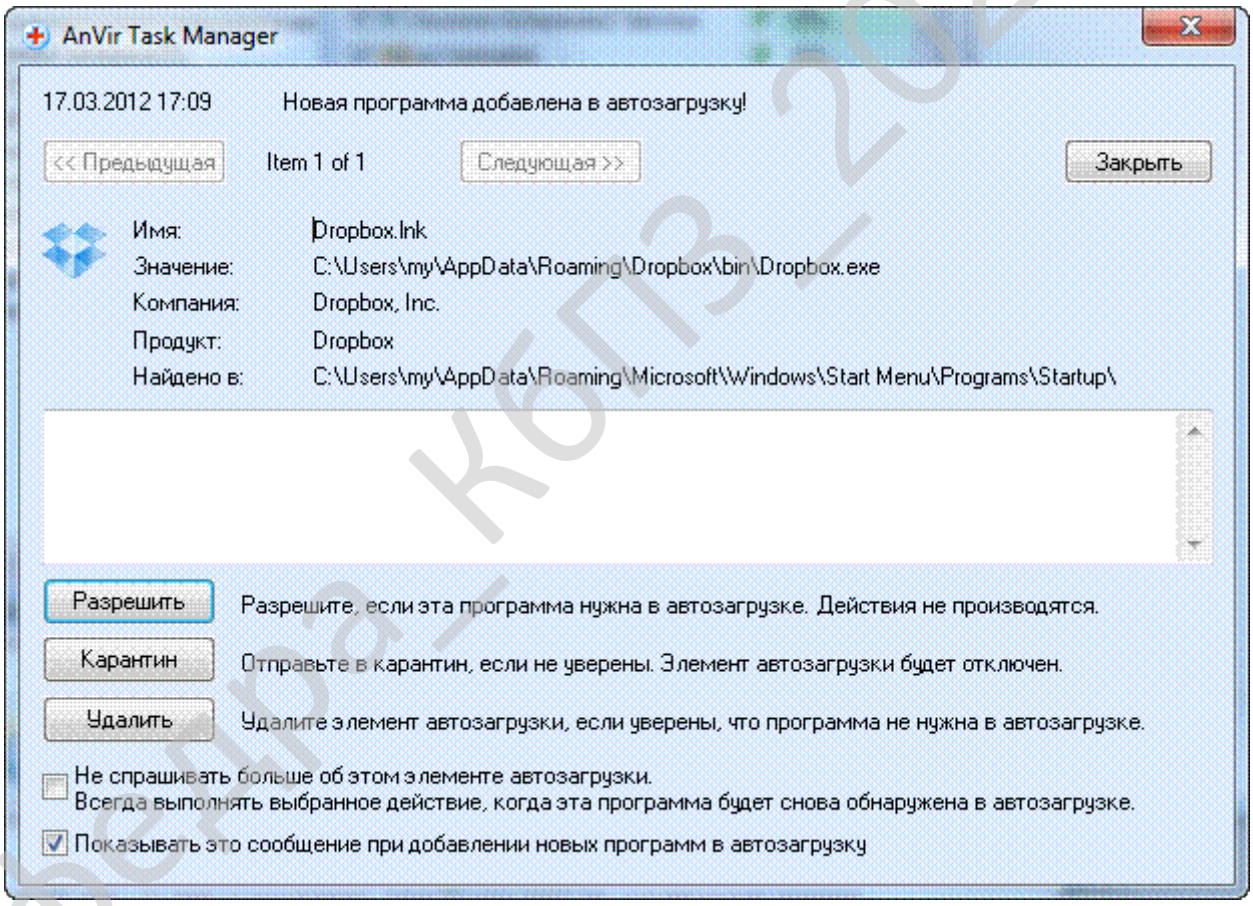

Рисунок 2.2 – Скриншот застережливого повідомлення AnVir Task Manager

Startup Programs – дозволяє проглянути всі запущені застосування, зупинити або взагалі видалити з автозапуску непотрібні, отримати інформацію

про них (ім'я, версія програми, ключ реєстру) або натисненням кнопки «Full Report» отримати звіт про всі програми за один раз. При першому запуску Scotty проглядає список програм, що автоматично запускаються, і при його зміні попереджає про це користувача.

IE Helpers – контроль за Browser Helper Objects: інформація про наявні об'єкти, запит на установку нових об'єктів, видалення підозрілих об'єктів. Треба сказати, що Browser Helper Objects запускається кожного разу разом з Internet Explorer (який стараннями Microsoft є неподільною частиною системи), у тому числі і при відкритті тек на локальному комп'ютері. Тому при її допомозі можна без проблем стежити за користувачем, що і застосовується в програмах типу SpyWare.

Scheduled Task Monitoring – відображення запланованих завдань, контроль над додаванням нових і отримання додаткової інформації про плановані роботи.

Services – відключення або тимчасова зупинка запущених сервісів і отримання додаткової інформації про них. Щоб не вишукувати підозрілі програми у великому списку, можна включити пункт «List non-Microsoft services only», прибравши таким чином системні ресурси.

View Active Tasks – отримання інформації про запущені на даний момент програми і завдання, знищення і припинення непотрібних і підозрілих. Також за допомогою цього пункту можна розібратися в роботі запущеної системи, тобто дізнатися, для чого призначена та або інша програма.

Cookies – інформування про появу нових Cookies, відхилення, управління і проглядання інформації, записаної в Cookies для ухвалення рішення. Можна скласти правило фільтрування для Cookies, які завжди повинні видалятися (Cookies with Nuts).

Options – установка опцій самої програми. Встановлено контроль за підміною домашньої сторінки у веб-сервері браузера, реакція на зміну файлу hosts або його повне блокування, виведення повного звіту по системі, ведення історії змін.

File Types – дозволяє переглядати, контролювати і відновлювати змінену асоціацію додатків з розширеннями файлів.

PLUS – реєстрація та доступ до мережевої бази даних програм, мета якої допомогти користувачам розібратися в списку незнайомих назв, адже напис servise.exe у таблиці процесів більшості нічого не скаже.

За допомогою AnVir Task Manager можливо зупинити будь-який процес, змінити пріоритет, додати в автозавантаження.

Антивірусна база містить тільки найбільш поширені віруси, всі програми, що запускаються, і записи реєстру автоматично перевіряються на наявність вірусів. Крім усього іншого, в треї відображаються іконки завантаження процесора і диска, а з консолі управління можна швидко дістати доступ до основних системних утиліт.

Перехоплення системних викликів з Safe'n'Sec. Компанія StarForce вже давно відома своїм однойменним механізмом захисту дисків від нелегального копіювання. Нова розробка Safe'n'Sec, представлена в листопаді минулого року, практично відразу отримала визнання і була названа журналом PC Magazine/RE антивірусом місяця. При цьому Safe'n'Sec не відноситься до антивірусів, а належить до класу систем проактивного захисту, які аналізують підозрілу поведінку користувача або програми.

Для малих і середніх підприємств, а також індивідуального використання призначена версія Personal. У корпоративній версії Safe'n'Sec Business є адміністративна консоль, яка дозволяє системному адміністраторові дистанційно інсталювати і налаштувати програму на комп'ютерах користувачів.

Версія 3.0.0 доступна в двох варіантах – усічена Safe'n'Sec ver. 3.0.0 і повна Safe'n'Sec ver. 3.0.0 + antivirus. Остання, як зрозуміло з назви, включає можливість антивірусної перевірки. Основу продукту складає модуль Intelligent activity control, що дозволяє виявляти комбіновані атаки, запобігти спробам внести зміни до системного реєстру або стану сервісів операційної системи, відкрити доступ до реєстраційних даних користувача тощо. При цьому механізм ухвалення рішення Safe'n'Sec діє на основі правил, що враховують всі можливі послідовності дій, що класифікуються як шкідливі.

Захист всіх даних на комп'ютері користувача здійснюється відповідно до політики контролю активності, що визначає, які дії і їх послідовність потрібно вважати шкідливими. На даний момент є три політики – жорстка, строга і довірча. Після виявлення підозрілого додатку Safe'n'Sec самостійно ухвалює рішення про його шкідливість і повідомляє користувача, який повинен визначити, що робити з таким застосуванням (дозволити або заблокувати) (рисунок 2.3).

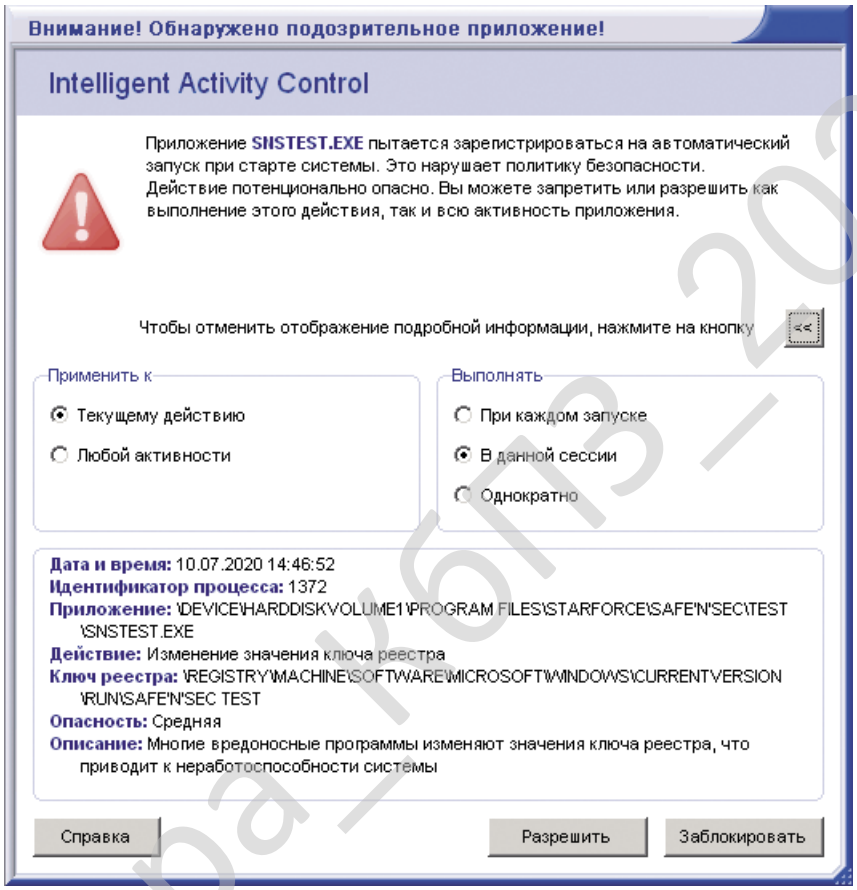

Рисунок 2.3 – Скриншот застережливого повідомлення Safe'n'Sec

Для спрощення ухвалення рішення доступна історія активності, по якій користувач може детально вивчити послідовність дій, виконаних додатком. Додатково з консолі можна дістати доступ до списку заборонених і дозволених застосувань, на які можна тут же відредагувати. Для тестування працездатності разом з програмою поставляється утиліта snstest.exe, яка імітує занесення даних в системний реєстр, спробу запису і видалення з системного каталога.

Всесторонній контроль з RegRun. RegRun Security Suite (http://www.greatis.com) працює зі всіма версіями Windows. Ще один комплект засобів для захисту комп'ютера проти вірусів або Трої, spyware і adware. Він доступний в трьох основних варіантах: Standart (для звичайних користувачів, забезпечує легке управління і безпеку), Professional (має додаткові можливості по роботі з системним реєстром, дозволить швидко оптимізувати системні параметри і забезпечити надійний захист), і найбільші можливості має Gold, призначений для професіоналів.

По посиланню http://www.greatis.com/security/rus.exe доступний русифікатор програми, до того сайт http://www.ruhelp.narod.ru надає неофіційний переклад файлу допомоги, що полегшує знайомство з програмою.

Після запуску RegRun активізується захист, визначений рівнем безпеки, – від низького до ультрависокого. В центрі Управління RegRun налаштовуються параметри роботи основних модулів. Наприклад, «WATCH DOG» забезпечує контроль за автозавантаженням протягом роботи, він може бути налаштований на перевірку автозавантаження при старті і виключенні Windows або через задані проміжки часу. Якщо при перевірці виявляються зміни, користувач буде сповіщений про це, і додатково запуститься утиліта Start Control, за допомогою якої можна отримати детальну інформацію про програми, що автоматично запускаються.

Однією з найкорисніших функцій є File Protection, що захищає комп'ютер від вірусів, троянів і програм, що працюють із збоями. Спочатку створюється список файлів, стан яких необхідно контролювати. Для подальшого відновлення вони копіюються в сховищі, яке знаходиться в «Мої документи/RegRun». Правда, саме сховище чомусь програма ніяк не захищає і на підміну файлів ніяк не реагує. Це є суттєвим недоліком.

При виявленні модифікації файлів, що захищаються, буде виведено попередження, з вказівкою розміру і дати створення обох файлів. Тепер цей файл можна копіювати, перейменувати, видалити, відкрити або просканувати антивірусною програмою. Для пошуку вірусів, не відомих антивірусним програмам, RegRun відкриває і контролює безліч програм-«приманок», якщо

виявляється зміна будь-якого з цих файлів, RegRun повідомить про присутність вірусу. Для Windows такою приманкою є файл winbait.exe, автозапуск якого заноситься в реєстр і порівнюється з winbait.org, крім того, передбачена можливість додавання свого підконтрольного файлу.

RegRun має базу даних застосувань, яка містить опис програм, що часто зустрічаються. Це допомагає користувачу визначити приналежність до однієї з чотирьох груп: необхідні, даремні, небезпечні і по вибору.

RegRun перевіряє присутність будь-яких потенційно небезпечних програм і інформує користувача про них, крім того, він сумісний зі всіма відомими антивірусами, і Антивірус Координатор може швидко перевірити файли, розміщені в автозавантаження. Додатково детектор підстановки порівнює реальний шлях до виконуваного файлу із завантаженим. Наприклад, якщо процес з ім'ям explorer.exe запускається не з теки c:\windows\system, а з іншого місця, то користувач буде оповіщений.

Трасувальник системного реєстру Registry Tracer, який є у версіях Professional або Gold, дозволяє вказати список ключів реєстру, при спробі зміни яких RegRun видасть застережливе повідомлення.

Менеджер процесів дозволяє не тільки проглянути і при необхідності видалити підозрілий процес, а також видає інформацію по завантажених DLL і мережевій активності додатків.

Утиліта RunGuard, доступна у версії Gold, перевіряє файли перед виконанням. Контролюються файли Microsoft Office (doc, dot, xls, xlt, ppt), html, Windows script (vbs, wsh, js, bat pif, cmd), hta і reg. При появі підозрілих запитів вони блокуються, і користувач може проаналізувати зміст файлу, перевірити його антивірусом, заблокувати або дозволити подальше виконання (рисунок 2.4).

Контроль цілісності з Xintegrity. Трохи іншим шляхом пішли розробники Xintegrity (http://www.xintegrity.com). Основне завдання якої – контроль цілісності і виявлення навіть самої незначної зміни файлів. Окрім контролю змісту файлів, утиліта виявляє зміни в структурі каталогу, включаючи альтернативні потоки даних (Alternate Data Streams), зміни параметрів доступу до файлів, реєстраційних даних і сервісів. Цікаво, що розробники врахували

можливість прихованого застосування утиліти, і можна задати будь-яку назву, розширення і місце розташування баз. Тому, назвавши файл, наприклад, kursovik.rtf або song.mp3 і поклавши в групу подібних файлів, можна його приховати.

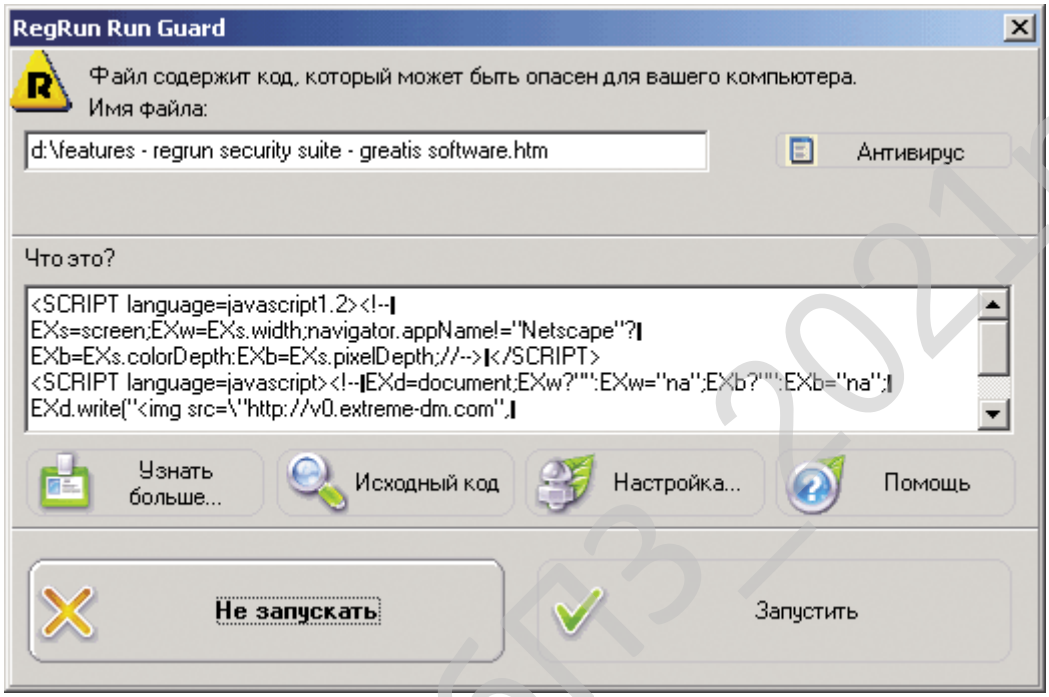

Рисунок 2.4 - Застережливе повідомлення RegRun Run Guard

При створенні бази можливі три варіанти: у неї заносяться тільки контрольні суми файлів, робиться резервна копія всіх даних, резервні копії додатково шифруються (256 біт AES). У останніх двох випадках, окрім контролю, дозволяють відновлювати змінені файли. В якості контрольної суми можна використовувати декілька варіантів – від 128-розрядної хеш-функції MD5 до AES з 256-розрядною довжиною ключа.

За допомогою Xintegrity можна перевіряти цілісність файлів на вимогу або запустити її у фоновому режимі, тоді при виявленні зміни користувач буде відразу ж повідомлений. Коли Xintegrity виявить такий змінений файл, користувачу буде видана докладна інформація про те, як і коли файл був

змінений, і при певних опціях створену базу буде запропоновано замінити резервною копією.

В цілях безпеки при відкритті бази Xintegrity автоматично перевіряє як себе, так і всі зареєстровані бази на предмет цілісності. При створенні бази даних, в неї можна занести всі файли з каталогу, за допомогою набору фільтрів відібрати файли, що задовольняють певним критеріям (розмір, час модифікації або створення, за типом, вмістом, атрибутами і функціональними можливостями). Підтримується Drag'and'Drop і вставка файлу з буфера обміну.

Файли, розташовані на дисках з файловою системою FAT32, NTFS, і мережеві теки можуть бути перераховані в одній базі даних. Тобто, можна задати практично будь-які умови. Тому можна створити декілька баз, в одну зібрати виконувані і системні файли, в іншу – мережеві ресурси, до яких є доступ, в третю – ресурси, здатні працювати в мережі. І для кожної задати свій режим перевірки.

SafenSoft SysWatch Personal є проактивним захистом комп'ютера, що відноситься до класу HIPS-систем і, на відміну від антивірусів, в його основі лежить контроль запуску і поведінки додатків, а не пошук у файлах ділянок відомого шкідливого коду на основі антивірусних баз або евристичного аналізу.

Проактивний захист комп'ютера класу HIPS, заснований на контролі додатків, виключає залежність від оновлення антивірусних баз, адже навіть останні версії антивірусних баз не гарантують наявності в них всіх зразків шкідливого ПЗ. Контролюючи додатки, SafenSoft SysWatch запобігає попаданню будь-якої шкідливого коду в систему, забезпечує безпеку даних і системи в цілому, ефективно захищаючи комп'ютер від новітнього шкідливого коду і вірусів, написаних на замовлення.

Здійснюється контроль додатків, прихований запуск додатків блокується, запуск нового застосування обробляється автоматично або припиняється до рішення користувача: виконати без обмежень, виконати в безпечному середовищі або блокувати. Оновлення встановлених застосувань розпізнаються автоматично.

Здійснюється контроль доступу додатків до файлової системи, ключів реєстру, USB-накопичувачів. Контролюється доступ додатків до важливих

областей ОС. Можливо створити індивідуальні правила доступу додатків до ресурсів для захисту особистих даних і цінної інформації:

- захист від відключення програми з боку шкідливого ПЗ;

- здійснюється контроль запуску і поведінки додатків, а не аналіз їх коду. Це дозволяє не використовувати антивірусні бази і блокувати шкідливу активність ще до випуску патчів виробниками додатків і оновлень сигнатурних баз виробниками антивірусів;

- запуск невідомих або потенційно уразливих додатків, наприклад, таких як браузери, SysWatch здійснює в безпечному ізольованому середовищі - "пісочниці". Це дозволяє нейтралізувати їх вплив на інші процеси або завдати шкоду системі, блокуючи шкідливу активність ще до випуску оновлень антивірусних баз;

- завдяки роботі на основі списку довірених застосувань і контролю додатків, для ефективного захисту не потрібні регулярні оновлення і підключення до Інтернет;

- в процесі роботи не потрібне сканування коду файлів, що мінімізує навантаження на процесор і кількість звернень до жорсткого диска комп'ютера, що захищається, дозволяє економити заряд батареї ноутбука;

- максимальна швидкодія системи при високому рівні захисту;

- в основі SysWatch лежить запатентована технологія контролю додатків V.I.P.O.®, яка об'єднує в собі 3 рівні захисту;

- захист всіх виконуваних додатків системи, завдяки виявленню спроб несанкціонованого запуску процесів і блокування їх запуску до того, як процес може завдати шкоди системі. Спеціальне середовище для запуску потенційно небезпечних застосувань забезпечує контроль системних привілеїв для блокування шкідливих дій. Забезпечує захист комп'ютера від шкідливих застосувань;

- контролює доступ додатків до файлової системи, ключів реєстру, а також доступ до зовнішніх пристроїв і мережевих ресурсів.

Таким чином, розглянути та виконавши аналіз зазначених вище систем, можемо зробити наступний висновок. Рішення, що існують на сьогоднішній день,

в області захисту інформації ефективні в боротьбі з вже відомими погрозами. Час реакції антивірусних компаній після виявлення невідомого вірусу можна назвати повільним і достатнім для поразки тисяч комп'ютерів. Тому використання систем захисту файлів, побудованих на основі сигнатурного аналізу вже відомих вірусів (реактивні системи) не надають користувачу дійсно надійного захисту інформаційних масивів та ПЗ, оскільки мають цілий ряд серйозних недоліків. На зміну їм приходять нові проактивні технології такі як HIPS, Sandbox, VIPS та інші, які ми розглянули.

Разом з тим, класичні проактивні системи виявлення шкідливих програм, не дивлячись на простоту реалізації і надійність, мають ряд істотних недоліків, а саме:

- слабка ефективність проти погроз типу 0-day, оскільки ефективність безпосередньо пов'язана з базою сигнатур шкідливого ПЗ, до якої внесені сигнатури тільки відомого, на даний момент, шкідливого ПЗ;

- необхідність постійного оновлення бази сигнатур вірусів для ефективного захисту від нового шкідливого ПЗ;

- для визначення шкідливого ПЗ необхідна процедура сканування, яка віднімає достатньо багато часу і системних ресурсів;

- значна вартість ліцензованого програмного пакету та необхідність щорічно доплачувати розробнику за оновлення придбаного пакету;

- низька мобільність тиражуємих версій, адже відомо багато конфліктних ситуацій при спробі вбудувати придбані пакети проактивного захисту в наявні системи захисту.

Тому доцільно розробити власну систему проактивного захисту, яка буде вільною від зазначених вище недоліків. В основу побудови системи, яка підлягає розробці, покладемо виявлені в процесі аналізу позитивні якості.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

При розробці ПЗ-системи планується використати програмування під ОС Windows з використанням подій Windows та елементів ActiveX. Це дозволить надати програмній моделі більшої наглядності та зрозумілості для сучасного користувача, який звик працювати в цій операційній системі.

При розробці ПЗ системи будуть використовуватись низькорівневі функції Windows (API). Windows API – це набір базових інтерфейсів програмування додатків ОС Windows, які є самим прямим способом взаємодії додатків з Windows, а також найбільш близьким до ОС способом взаємодії з нею із прикладних програм (більш низький рівень доступу необхідний для драйверів пристроїв). Ми використаємо версію Win 32.

Win 32 – це 32-розрядна API для сучасних версій Windows. Базові функції цього API реалізовані в DLL kernel32.dll та advapi32.dll; базові модулі GUI – buser32.dll. Використання Windows API дозволить значно скоротити терміни розробки ПЗ та покращити його якісні характеристики.

Для перехвату API-функцій використаємо дієву методику сплайсингу. Сплайсинг – це підміна коду функції. Звичайно, є й інші методи перехвату API-функцій. Це, перш за все, редагування таблиці імпорту додатку. Розглянемо сутність цього методу.

Програмна реалізація методу редагування таблиці імпорту додатку полягає в наступному. До складу ПЗ СПЗ вводимо функцію:

```
Function Funcl (param*:type*): restype*; stdcall; external`libname.dll;
```

Це дозволить імпортувати функцію статично.

Адресу функції пропишемо в таблицю імпорту застосування:

адреса	значення
…	…
$00405F56	7BC56010.

При виклику функції відбудеться наступне:

```
Push///push///call dword ptr$00405F56.
```

Отже, для перехвату функції нам треба тільки підмінити значення за адресом $00405F56 на своє, а для виклику оригінальної функції досить отримати адрес функції через Get ProcAddress. Але додаток може отримати адрес функції і через Get ProcAddress і викликати перехвачену функцію, оминувши перехоплювач. Тому даний метод для нашої системи не підходить і більш доцільно використати техніку сплайсингу.

Перш, ніж приступити до розробки архітектури системи, яка буде реалізована в процесі виконання КБР, необхідно визначитись з технологією, яка буде покладена в основу побудови системи.

Проактивні технології – сукупність технологій і методів, використовуваних в антивірусному програмному забезпеченні, основною метою яких, на відміну від реактивних (сигнатурних) технологій, є запобігання зараженню системи користувача, а не пошук вже відомого шкідливого програмного забезпечення в системі.

Проактивні технології почали розвиватися практичні одночасно з класичними (сигнатурними) технологіями. Проте, перші реалізації проактивних технологій антивірусного захисту вимагали високого рівня кваліфікації користувача, тобто не були розраховані на масове використання простими користувачами персональних комп'ютерів (ПК). Через десятиліття антивірусної індустрії стало очевидно, що сигнатурні методи виявлення вже не можуть забезпечити ефективний захист користувачів. Цей факт і підштовхнув до відродження проактивних технологій.

До технологій проактивного захисту віднесемо: евристичний аналіз, емуляцію коду, поведінковий аналіз, обмеження привілеїв виконання, віртуалізацію робочого оточення.

Технологія евристичного аналізу дозволяє на основі аналізу коду виконуваного застосування, скрипта або макросу виявити ділянки ПЗ, які відповідають за шкідливу активність. Ефективність даної технології не висока,

що обумовлене великою кількістю помилкових спрацьовувань при підвищенні чутливості аналізатора, а також великим набором техніки, використовуваної авторами шкідливого ПЗ для обходу евристичної компоненти антивірусного ПЗ.

Технологія емуляції коду дозволяє запускати додаток в середовищі емуляції, емулюючи поведінку ОС або центрального процесора. При виконанні додатку в режимі емуляції, додаток не зможе нанести шкоди системі користувача, а шкідлива дія буде детектована емулятором. Не дивлячись на ефективність даного підходу, він також не позбавлений недоліків – емуляція займає дуже багато часу і ресурсів комп'ютера користувача, що негативно позначається на швидкодії при виконанні повсякденних операцій. Разом з тим, сучасні шкідливі програми здатні виявляти роботу системи захисту в середовищі емулювання і припиняти своє виконання в ній.

Технологія аналізу поведінки ґрунтується на перехваті всіх важливих системних функцій або установці міні-фільтрів, що дозволяє відстежувати всю активність в системі користувача. Технологія поведінкового аналізу дозволяє оцінювати не тільки одиничну дію, але і ланцюжок дій. Це в багато разів підвищує ефективність протидії вірусним погрозам. Також, поведінковий аналіз є технологічною основою для цілого класу програм – поведінкових блокаторів (HIPS – Host-based Intrusion Systems).

Sandboxing (Пісочниця) – обмеження привілеїв виконання. Технологія Пісочниці працює за принципом обмеження активності потенційно шкідливих застосувань так, щоб вони не могли нанести шкоди системі користувача.

Обмеження активності досягається за рахунок виконання невідомих застосувань в обмеженому середовищі – власне пісочниці, звідки додаток немає прав доступу до критичних системних файлів, гілок реєстру і іншої важливої інформації. Технологія обмеження привілеїв виконання є ефективною технологією протидії сучасним погрозам, але користувач повинен володіти знаннями, необхідними для правильної оцінки невідомого застосування.

Технологія віртуалізації робочого оточення працює за допомогою системного драйвера, який перехвачує всі запити на запис на жорсткий диск і замість виконання запису на реальний жорсткий диск виконує запис в спеціальну

дискову область – буфер. Таким чином, навіть в тому випадку, коли користувач запустить шкідливе програмне забезпечення, воно проживе тільки до очищення буфера, яке по умовчанню виконується при виключенні комп'ютера.

Але технологія віртуалізації робочого оточення не зможе захистити від шкідливих програм, основною метою яких є крадіжка конфіденційної інформації, оскільки доступ на читання до жорсткого диска не заборонений.

Розглянувши вищезазначені технології, можемо цілком обґрунтовано зробити висновок, що в нашій розробці доцільно використати поєднання двох технологій: аналіз поведінки та обмеження привілеїв виконання (пісочниця). Це дозволить дійсно реалізувати закладені ТЗ функції та розробити дієспроможну систему.

В якості програмного середовища розробки оберемо Delphi та мову програмування Delphi. Цей вибір обумовлений наступними причинами:

- Мова програмування Delphi достатньо проста, швидка, ефективна та потужна. Їй використання дозволяє розробити ПЗ в стислі терміни та з високою якістю.

- Об’єктно-орієнтована мова програмування Delphi дозволяє зручно абстрагувати об’єкт проектування – систему роботи з ресурсами ПК та мереж, її елементи і організувати класи загального призначення із контролем доступу до даних за рахунок введення захищених та відкритих членів класу, що ефективно моделюють проектування складних систем керування і роботу реальних об’єктів з використанням пересилки інформації між ними.

- У мові Delphi представлені зручні засоби для роботи як на високому віні, так і на низькому в той час, як інші сучасні мови надають перевагу одному напрямку (або високий рівень із створенням зручного інтерфейсу, або низький рівень із програмуванням мережного зв’язку з недостатнім забезпеченням функцій інтерфейсу).

Таким чином, вибір мови програмування Delphi є доцільним, оскільки вона має засоби дуже зручного опису елементів об’єкта та зв’язків між ними та зовнішнім середовищем і забезпечує програміста ефективним інструментом для написання в мінімальні строки дійсно якісних програм.

При побудові ПЗ будуть використані стандартні модулі, які виконують наступні функції:

- Menus – модуль, що забезпечує (обслуговує) всі об’єкти та процедури для системи меню DELPHI, а також випадаючі меню та активні елементи рядка статусу, робота з меню;

- Messages, Dialogs – модулі, що забезпечують процедури і функції виклику діалогових вікон;

- SysUtils – модуль, що забезпечує процедури та функції нескладних математичних функцій, функції перетворення з одного типу в інші; деякі тригонометричні та гіперболічні функції; процедури та функції перетворення дат і часу, роботи з рядками та інше;

- Classes – процедури та функції для роботи з класами;

- Variants – процедури та функції перетворення типуVariant в заданий тип;

- Graphics – процедури та функції по роботі з графікою і графічними об’єктами;

- Forms – процедури та функції по роботі з формами, що підключають та забезпечують можливість використання стандартних бібліотечних функцій, а саме: підключення діалогових вікон, меню користувача, організації функції натиснення кнопок для отримання з клавіатури режиму роботи системи в поточний момент часу, виведення на екран файлу з інформацією про режим роботи системи, підготовки екрану до діалогу з користувачем, тощо.

Використання стандартних бібліотек дозволить значно полегшити побудову деяких частин програми та скоротити обсяги програмного коду.

Організація процесу побудови архітектури майбутньої системи повинна відповідати та проводитись згідно діючих державних стандартів, які пред’являють ряд загальних та специфічних вимог, а саме:

- повинні використовуватись сучасні методи організації та керування, включаючи системно-структурний аналіз та моделювання процесів, економічно-математичні методи;

- повинні використовуватись раціональні параметричні та типорозмірні ряди виробів засобів обчислювальної техніки (ЗОТ);

- повинна бути орієнтація на оптимальний для конкретних умов рівень автоматизації збору, підготовки, обробки, передачі та представлення інформації.

Для зручності розробки та впровадження, процес створення системи необхідно, згідно діючих стандартів, розподілити на три етапи: виділення функцій майбутньої системи; визначення задач кожної функції; визначення підзадач в кожній задачі.

При розробці програмного забезпечення планується організація системи згідно модульної структури: основна (керуюча) програма та ряд підпрограм загального та спеціального призначення.

Підпрограми загального призначення необхідні для реалізації в системі функцій, які вирішує ЕОМ: інформаційний пошук, кодування, контроль, перетворення інформації, формування та представлення вихідних даних для програм керування та контролю, оформлення визначеного пакету технічної та звітної документації.

Підпрограми спеціального призначення будуть виконувати функції контролю некоректних дій користувача та оптимізації нестандартних ситуацій в роботі системи. Структурними елементами підпрограм другого типу є модулі програми.

Побудова системи повинна відповідати шести основним принципам: композиції; єдності; відкритості; максимальної універсальності; інформаційної єдності; інваріантності.

Розглянемо більш детально кожен з них.

Принцип композиції складається з того, що система, яка підлягає розробці, створюється на базі вже існуючих систем та програм, які були розглянуті в цьому розділі, але з використанням нового типу алгоритмів, що будуть розроблені при виконанні ДП. Це дозволить значно зекономити час при розробці нашої системи при виконанні дипломної роботи.

Принцип єдності допускає, що на усіх стадіях розробки, функціонування та подальшого розвитку системи її цілісність повинна бути забезпечена зв’язками між окремими частинами та компонентами, а саме: методичним забезпеченням;

програмним забезпеченням; технічним забезпеченням; інформаційним забезпеченням; організаційним забезпеченням.

Принцип відкритості вимагає, щоб система створювалась та функціонувала як динамічна система, тобто розширяючи та удосконалюючи свої компоненти. Це обумовлено складністю та багатофункціональністю самої системи і наявністю систем-аналогів на ринку програмних продуктів, які постійно удосконалюються. З другої сторони, постійно ускладнюється та вдосконалюється технічне обладнання, що буде використовуватись системою, а це також буде вимагати відповідної доробки системи.

Принцип максимальної універсальності вимагає розробки багатофункціональної мобільної системи. Реалізація цієї задачі передбачає рішення наступних питань: технічного забезпечення системи; математичного забезпечення системи; програмного забезпечення системи.

Принцип інформаційної єдності передбачає уніфікацію інформаційних масивів, систем керування, а також вхідної та вихідної мов описання компонентів всередині системи.

Принцип інваріантності означає, що система, яка підлягає розробці, з точки зору її використання на інших ЗОТ, повинна бути універсальною або типовою. Вирішення цієї задачі планується введенням в ПЗ універсальних або спеціальних модулів. Наявність достатньої кількості такого типу модулів в структурі ПЗ дозволить забезпечити закінченість рішення задач системою.

2.3 Розгорнута постановка завдання

В процесі розробки КБР необхідно виконати наступний обсяг робіт:

а) визначити сутність розробки згідно теми КБР та обґрунтувати актуальність теми КБ;

б) визначити призначення системи та окреслити область її впровадження в промислову експлуатацію;

в) провести аналіз існуючих систем-аналогів програм та систем з ціллю виявлення їх позитивних та негативних якостей; результати аналізу врахувати в подальших розробках; провести обґрунтування та вибір засобів для розробки ПЗ системи; розробити ПЗ на реалізацію закладених технічним завданням функцій, які має виконувати майбутня система;

г) розробити і обґрунтувати методику побудови архітектури системи, означити принципи розробки, визначити та обґрунтувати обмеження, які необхідно врахувати при розробці ПЗ; розробити функціональну та структурну схеми системи, визначивши їх складові та обґрунтувати вибір;

д) розробити ПЗ системи, яке дозволить реалізувати поставлену ТЗ задачу. Побудувати блок-схеми алгоритмів основної програми та підпрограм;

е) визначити тип механізму захисту розроблено ПЗ, програмно реалізувати обраний механізм захисту ПЗ від несанкціонованого доступу (НСД);

є) розробити рекомендації користувачу щодо розробки методики та плану впровадження розробленої системи в промислову експлуатацію;

ж) для більшої оглядовості і зрозумілості ілюструвати виконану розробку графічним матеріалом в кількості 7 листів формату А4.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

В процесі виконання КБР розробці підлягає програмне забезпечення системи кібербезпеки проактивного захисту на основі використання API-технологій. Проактивний захист – одна із сучасних технологій, які втілюють якісний стрибок в області протидії вірусним атакам, адже аналізу піддається сукупність дій, чинених в системі, яка підлягає захисту, і в яку вбудовується розроблена система.

Для визначення складових системи, їх взаємозв’язку, побудови структурної схеми, функціональної схеми, визначення напрямку розробки алгоритмів необхідно детально розглянути і означити вимоги та обмеження щодо системи.

3.1 Опис функціонування системи

В попередніх розділах пояснювальної записки ми вже частково розглянули задачу проактивного захисту та дійшли до висновку, що це є украй важлива задача в різноманітних ситуаціях, коли виникає реальна загроза вірусної активності, адже наявні системи захисту нездатні своєчасно захистити програмні та інформаційні продукти користувача.

В процесі побудови архітектури ПЗ системи в основу було покладено поєднання двох методик реалізації системи проактивного захисту: поведінковий аналізатор та обмеження виконання операцій. Це дозволить одержати покращений функціонал нашої системи, а саме:

– забезпечить меншу кількість звернень до користувача у порівнянні з системами, побудованими на IPS-методах;

– низьке споживання системних ресурсів;

– невибагливість до апаратного забезпечення ПК (можливість працювати на різних платформах).

Разом з тим, в процесі розробки КБР необхідно вирішити наступні основні задачі, які саме і визначають технологію проактивного захисту:

– створення політик для обмеження доступу до директорій або окремих файлів;

– моніторинг списку запущених процесів в системі, яка підлягає захисту;

– блокування підозрілих процесів, вибраних при необхідності або заборона запуску нових файлів;

– моніторинг породження/розмноження однакових файлів на ПК і візуалізація одержаних результатів або обмеження тиражування вірусів.

Таким чином, в процесі розробки ПЗ системи необхідно забезпечити реалізацію наступних основник функціональних можливостей:

– моніторинг усіх процесів з моменту запуску і до їх завершення. Це дозволить повною мірою оцінити дії того або іншого процесу;

– моніторинг створення/знищення файлів в системній директорії;

– стеження за вмістом вказаних каталогів (на наявність спроби створення копій файлів);

– спеціально розроблений модуль повинен забезпечити задання в налаштуваннях параметрів, які дозволять стежити тільки за потрібними застосуваннями (це необхідно для того, щоб не загубитись у величезному потоці інформації про всі процеси, запущені на ПК);

– забезпечення можливості відстеження ієрархії запуску того або іншого процесу;

– виведення користувачу подробиць про той або інший процес на основі одержаної в процесі моніторингу детальної інформації про нього (уміст процесу).

Процесом називають екземпляр виконуваної програми. При цьому програма є статичним набором команд, а процес - це набір ресурсів і даних, що використовуються при виконанні програми.

Процеси інертні. Відповідають же за виконання коди, що міститься в адресному просторі процесу, потоці. Потік (thread) – якась суть усередині процесу, одержуюча процесорний час для виконання. У кожному процесі є мінімум один потік. Цей первинний потік створюється системою автоматично при створенні процесу. Далі ця нитка може породити інші нитки, ті у свою чергу нові і так далі. Таким чином, один процес може володіти декількома потоками, і тоді вони одночасно виконують код в адресному просторі процесу. Кожен потік має: унікальний ідентифікатор; уміст набору регістрів процесора, що відображають стан процесора; два стеки, один з яких використовується потоком при виконанні в режимі ядра, а інший – в призначеному для користувача режимі; закриту область пам'яті, звану локальною пам'яттю потока (thread local storage, TLS) і використовувану підсистемами, run-time бібліотеками і DLL.

Щоб всі потоки працювали, операційна система відводить кожній з них певний процесорний час. Тим самим створюється ілюзія одночасного виконання потоків. У Windows реалізована система витісняючого планування на основі пріоритетів, в якій завжди виконується потік з найбільшим пріоритетом, готовий до виконання.

У будь-якому випадку операційна система повинна визначити, який потік виконувати наступним. Вибравши новий потік, операційна система перемикає контекст. Ця операція полягає в збереженні параметрів виконуваного потоку і завантаженні аналогічних параметрів для іншої нитки, після чого починається виконання нової нитки.

Для того, щоб отримати перелік запущених процесів в системі, необхідно, перш за все, їх перерахувати. ОС Windows надає декілька способів перерахування запущених процесів. На жаль, немає єдиного способу, який би працював на всіх Win-платформах. Тому необхідно скомбінувати декілька методів в одній програмі, щоб вона працювала на всіх версіях Windows.

Розглянемо наступні методи:

– за допомогою бібліотеки Process Status Helper (PSAPI);

– за допомогою ToolHelp32 API;

– за допомогою недокументованої функції ZwQuerySystemInformation;

– через лічильники продуктивності;

– з використанням інтерфейсів Windows Management Instrumentation, жоден з яких не є універсальним.

Визначимо їх застосування в родині ОС Windows, на яку орієнтована КБР (таблиця 3.1).

Таблиця 3.1 – Застосування методів для отримання запущених процесів

	Windows NT 4.0	**Windows Vista/7/8/10**	**Примітка**
Спосіб 1	Так*	Так	* Вимагає установки додаткових компонентів.
Спосіб 2	Немає	Так	
Спосіб 3	Так	Так	
Спосіб 4	Так *	Так	* Вимагає установки додаткових компонентів.
Спосіб 5	Так *	Так	* Вимагає установки додаткових компонентів.

Використовуючи цю таблицю, можемо зробити цілком обґрунтований висновок, що для того, щоб забезпечити розробку функції перерахування процесів, яка працюватиме на всіх версіях ОС Windows, необхідно використати спосіб 1 та спосіб 2.

Функція перерахування викликає для кожного процесу вказану призначену для користувача функцію, передаючи їй інформацію про черговий процес. Користувацька функція розпоряджається цими даними відповідно до логіки програми, наприклад, вона додає черговий процес - елемент в список.

Розглянемо використання бібліотеки Process Status Helper при побудові системи. Бібліотека Process Status Helper, відома також під назвою PSAPI, надає набір функцій, що дозволяють отримати інформацію про процеси і драйвери пристроїв. Бібліотека поставляється у складі ОС Windows. Для реалізації функції перерахування процесів, бібліотека надає функцію EnumProcesses, яка повертає масив ідентифікаторів запущених процесів.

Наступним кроком необхідно зв'язатись з PSAPI.DLL, динамічно завантажуючи бібліотеку за допомогою LoadLibrary і отримуючи адреси необхідних функцій за допомогою GetProcAddress. Це дозволить надалі включити цю функцію в програму, яка повинна виконуватися.

Функція EnumProcesses не дозволяє дізнатися, скільки місця потрібно для того, щоб прийняти весь масив ідентифікаторів. Тому викличемо її в циклі, збільшуючи розмір буфера до тих пір, поки розмір поверненого масиву не буде менше розміру буфера.

Оскільки, крім ідентифікаторів процесів, необхідно визначити і імена процесів, виконаємо додаткову роботу. Для кожного процесу спочатку отримаємо його описувач (handle) за допомогою функції OpenProcess і потім використаємо функцію Enumprocessmodules, яка повертає список модулів, завантажених в адресний простір процесу. Першим модулем в списку завжди є модуль, який відповідає EXE-файлу програми. Нарешті, викликаємо функцію GetModuleFileNameEx (яка також є частиною PSAPI), щоб отримати шлях до EXE-файлу по описувачу модуля. Для цього використовуємо ім'я EXE-файла без шляху, як ім'я процесу.

При розробці структури алгоритму функції перерахування, слід передбачити спеціальну обробку для двох процесів. Ми вимушені обробляти окремо процес бездіяльності системи (Idle) з ідентифікатором 0, і системний процес (System), який має ідентифікатор 2, тому що OpenProcess не дозволяє відкрити описувач для цих процесів, повертаючи код помилки ERROR_ACCESS_DENIED.

Пошук процесів в системі почнемо з отримання знімка пам'яті. Для цього використаємо наступну функцію:

```
function CreateToolhelp32Snapshot(dwFlags, th32ProcessID: DWORD): THandle;
```

При цьому повертається дескриптор цього знімка (або 0, якщо зняти знімок не вдалося), який, після проведення операцій із знімком, необхідно закрити стандартною функцією:

```
function CloseHandle(hObject: THandle): BOOL;
```

Параметр dwFlags визначає вміст знімка (таблиця 3.2).

Таблиця 3.2 – Вміст змімка процесів системи

Значення	**Опис**
TH32CS_INHERIT	Указує, що дескриптор знімка буде успадкованим
TH32CS_SNAPALL	Є комбінацією прапорців: TH32CS_SNAPHEAPLIST, TH32CS_SNAPMODULE, TH32CS_SNAPPROCESS, and TH32CS_SNAPTHREAD.
TH32CS_SNAPHEAPLIST	Список куп, створених процесом за умовчанням або під час його виконання. У купі містяться блоки пам'яті.
TH32CS_SNAPMODULE	Список модулів, завантажених процесом.
TH32CS_SNAPPROCESS	Список всіх процесів.
TH32CS_SNAPTHREAD	Список потоків, створених процесом.

Виходячи з означення задач, які має реалізувати система, і в яких головним все-таки є процес, а вже інше так або інакше, належить йому, спочатку необхідно знайти всі процеси, що виконуються в даний момент. Слідуючи логіці – отримуємо знімок всіх процесів:

```
hSnapshot := CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS,0);
```

В якості другого параметру указуємо 0, це дозволить зберегти дескриптор знімка для подальшого його використання.

Тепер залишилося прочитати знімок. Для цього застосовуємо функції:

```
function Process32First(hSnapshot: THandle; var lppe: TProcessEntry32): BOOL;
function Process32Next(hSnapshot: THandle; var lppe: TProcessEntry32): BOOL;
```

Спочатку викликаємо функцію Process32First, яка визначає перший процес в отриманому знімку, по поверненому значенню можемо дізнатися, чи є хоч а б один процес. Для отримання інформації про наступний процес, треба викликати функцію Process32Next (інформація про процес знаходитиметься в структурі lppe типу TProcessEntry32). Означимо в таблиці 3.3 уміст інформації про процес.

Таблиця 3.3 – Уміст інформації про процес

Параметр	Опис
dwSize: DWORD	Розмір структури.
cntUsage: DWORD	Число посилань на процес.
Th32ProcessID: DWORD	ID-процесу. Або PID.
Th32DefaultHeapID: DWORD	ID «купи за умовчанням» процесу.
Th32ModuleID: DWORD	Модульний ID процесу.
cntThreads: DWORD	Кількість потоків.
Th32ParentProcessID: DWORD	ID батьківського процесу.
pcPriClassBase: Longint	Базовий пріоритет для потоків процесу. Але це не пріоритет процесу.
dwFlags: DWORD	Не використовується.
szExeFile: array[0..MAX_PATH - 1] of Char	Ім'я файлу процесу.

Наступним кроком необхідно заповнити поле dwSize до виклику Process32First. Це робиться наступним чином:

```
lppe.dwSize := sizeof(TPROCESSENTRY32).
```

Отриманням інформації про процеси – це лише перший крок. Згідно визначених ТЗ функцій, які має виконувати система, необхідно маніпулювати процесом: завершувати його і міняти пріоритет.

Для цього скористаємося функціями не з ToolHelp32, а OpenProctss (таблиця 3.4).

Таблиця 3.4 – Параметри функції OpenProcess

Значення	**Опис**
PROCESS_ALL_ACCESS	Повний доступ до процесу.
PROCESS_CREATE_PROCESS	Безпосереднє використання.
PROCESS_CREATE_THREAD	Створення віддаленого потоку функцією CreateRemoteThread.
PROCESS_DUP_HANDLE	Для спадкоємства дескриптора функцією DuplicateHandle.
PROCESS_QUERY_INFORMATION	Отримання інформації про процес функціями GetExitCodeProcess і GetPriorityClass.
PROCESS_SET_QUOTA	Встановлення обмеження пам'яті функціями AssignProcessToJobObject і SetProcessWorkingSetSize.
PROCESS_SET_QUOTA	Встановлення обмеження пам'яті функціями AssignProcessToJobObject і SetProcessWorkingSetSize.
PROCESS_SET_INFORMATION	Установка пріоритету функцією SetPriorityClass.
PROCESS_VM_OPERATION	Модифікація віртуальної пам'яті процесу функціями VirtualProtectEx і WriteProcessMemory.
PROCESS_TERMINATE	Завершення процесу функцією TerminateProcess.
PROCESS_VM_READ	Читання віртуальної пам'яті процесу функцією ReadProcessMemory.
PROCESS_VM_WRITE	Запис в віртуальну пам'ять процесу функцією WriteProcessMemory.
SYNCHRONIZE	Тільки Windows NT/2000/XP: використання процесу для викликів wait-функцій.

Тепер визначимось із функціоналом алгоритму програми завершення чужого застосування.

Користувачі Win9x можуть видаляти будь-які процеси, а ось NT ОС не дасть завершити критичні системні процеси. Для користувача/професіонала дуже

не зручно уручну вводити довгий номер, щоб завершити процес. Тому було прийняте рішення автоматизувати цю процедуру.

Для цього необхідно знайти PID процесу.

Після того, як була визначена реалізація функції завершення чужих процесів, визначимо шляхи реалізації функції пріоритету процесу.

Для початку необхідного розібратися, які пріоритети бувають і яким процесам властиві (таблиця 3.5).

Визначимо функцію для отримання пріоритету процесу і повертає вона пріоритет вищого списку:

```
function GetPriorityClass(hProcess: THandle): DWORD;
```

Встановлення пріоритету вимагає в якості другого параметру використати потрібну константу таблиці 3.5:

```
function SetPriorityClass(hProcess: THandle; dwPriorityClass: DWORD): BOOL.
```

Таблиця 3.5 – Опис пріоритетів процесів

Пріоритет	**Опис**
NORMAL_PRIORITY_CLASS	Нормальний пріоритет для звичайних застосувань. Процес за умовчанням запускається з цим пріоритетом.
IDLE_PRIORITY_CLASS	Для додатків, що працюють під час простою операційної системи. Будь-який процес може віднімати у цього процесу кванти процесорного часу.
HIGH_PRIORITY_CLASS	Для додатків, що вимагають виконання негайно. Підходить якраз для нашої програми, тобто отримання списку завдань.
REALTIME_PRIORITY_CLASS	Найвищий пріоритет. Може віднімати процесорний час у критичних системних застосувань, унаслідок чого можуть бути відкладені завдання кешування диска і не рухатися курсор. Тому краще не використовувати.

Тепер розглянемо вміст процесу. Почнемо з найпростішого - модулів. Аналогія з пошуком процесів повна, окрім того, що при виклику функції замість 0 необхідно вказати PID процесу і назви функцій, тип структури буде іншим. Перераховано відмінностей багато, але вони не такі істотні.

Для модуля використаємо структуру TMODULEENTRY32, яка має опис, наведений в таблиці 3.6.

Таблиця 3.6 – Опис структури TMODULEENTRY32

Поле	Опис
dwSize: DWORD	Розмір структури, його потрібно вказати до того, як вона використовуватиметься.
th32ModuleID: DWORD	ID модуль.
th32ProcessID: DWORD	ID процесу, що завантажив модуль.
GlblcntUsage: DWORD	Глобальний лічильник використання модуля.
ProccntUsage: DWORD	Локальний лічильник використання модуля.
modBaseAddr: PBYTE	Базова адреса в контексті процесу th32ProcessID.
modBaseSize: DWORD	Розмір модуля в пам'яті.
hModule: HMODULE	Дескриптор модуля, в контексті даного процесу th32ProcessID.
szModule: array[0..MAX_MODULE_NAME32] of Char	Ім'я модуля.
szExePath: array[0..MAX_PATH - 1] of Char	Повний шлях до модуля, включаючи його ім'я.

При реалізації цієї функції необхідно вираховувати наступне обмеження:

Завантаження модуля в пам'ять в Win9x і NT відбувається по-різному. Якщо в NT процес завантажує в свій адресний простір кожен модуль, який йому необхідний, не залежно від того, чи використовує його який-небудь інший процес, то в Win9x модулі знаходяться в одному адресному просторі. Тобто виходить, що в NT існують декілька копій одного і того ж модуля, а в Win9x тільки одна. Тому в Win9x значення полів GlblcntUsage і ProccntUsage буде різним, а в NT співпадатиме.

Для роботи з потоком використаємо структуру TThreadEntry32.

Її опис наводимо в таблиці 3.7.

Таблиця 3.7 – Опис структури TThreadEntry32

Поле	Опис
dwSize: DWORD	Розмір структури, його потрібно вказати до того, як вона використовуватиметься.
cntUsage: DWORD	Число посилань на потік.
th32ThreadID: DWORD	ID потоку.
th32OwnerProcessID: DWORD	ID процесу, що володіє потоком.
tpBasePri: Longint	Базовий пріоритет.
tpDeltaPri: Longint	Додатковий пріоритет.
dwFlags: DWORD	Не використовується.

Ось тут є розбіжності з процедурою пошуку модулів. Пошук потоків аналогічний пошуку процесів, тобто - отримаємо всі потоки, що виконуються в даний момент в системі. Але нам треба потоки, що належать якомусь процесу. Для цього просто порівнюватимемо PID вибраного процесу з PID процесу – батька потоку.

Потоком можна теж маніпулювати, як і процесом. У Delphi присутня функція TerminateThread, проте немає функції OpenThread. Але ця функція описана в SDK, значить, її можна викликати. До розділу implementation додаємо наступну функцію:

```
function OpenThread(dwDesiredAccess: DWORD; bInheritHandle: BOOL; dwThreadId: DWORD): THandle; stdcall; external 'kernel32.dll'.
```

Ця функція аналогічна функції OpenProcess, і тепер її можна використовувати спільно з функцією TerminateThread.

Якщо при завершенні процесу ми як би «виходимо з програми», то при завершенні одного з декількох потоків процесу, можна отримати несподівані результати, адже є ймовірність, що один з потоків програми може «випадково» завершитися. Тому в системі, що підлягає розробці, необхідно передбачити збереження даних усіх проведених налаштувань, в спе ціально організовані INI-файли.

Слід, також, врахувати і наступний аспект: першим в списку буде головний потік, який і відповідає за процес. Тому, завершивши цей потік, тим самим знищуємо і процес.

Якщо процес - це інертний об'єкт, то потік - динамічний, тобто потік виконується. Виконання потоків можна припиняти і відновлювати. Для нього використаємо наступні функції:

```
function SuspendThread(hThread: THandle): DWORD;
function ResumeThread(hThread: THandle): DWORD;
```

Перша функція припиняє виконання потоку, а друга - відновлює його виконання. Функції повертають (-1) у разі провалу.

Всі функції, що працюють з процесами і потоками, приймають тільки дескриптор, а не PID. Тому і викликають функцію «відкриття» процесу або потоку. Відкрити процес або потік можна для різних дій з ними, тому слід забезпечити уважне стеження за прапорами, що передаються в якості параметрів функції «відкриття».

Механізм моніторингу ОС Windows буде базуватись на принципі розширення технології аудиту. Зареєстровані події будуть аналізуватись системою моніторингу автоматично в реальному масштабі часу. В залежності від результатів аналізу будуть прийматись ті або інші рішення, що впливають на функціонування системи (тобто, буде вироблятись реакція системи захисту).

Як ми вже визначились, події, що можуть відбуватись в системі, будуть об’єднані в групи (списки) по функціональній ознаці. Перевірка списків буде здійснюватись послідовно. При цьому буде введено ще одну ступінь захисту. Адже перевірка наступного рівня буде залежати від результатів перевірки на попередньому рівні. Якщо на будь-якому рівні буде виявлено розходження між реальною подією та еталоном – буде генеруватись стан порушення ОС системи та відповідна реакція системи моніторингу на це порушення. Але слід врахувати наступне:

– умовою вибору і складання списків подій в системі повинна бути непряма залежність її безпеки від події, що включена до списку, але слід враховувати і пряму залежність;

– порушення події по одному конкретному списку не завжди є прямим доказом порушення безпеки системи але завжди вказує на підвищення ймовірності такої події.

Враховуючи вищеозначене, можемо відзначити наступне: використаний рівневий моніторинг списків санкціонованих подій представляє собою механізм захисту інформації, оснований на визначені подій, що відбуваються при доступі до ресурсу, виявлення списків санкціонованих подій і безперервний контроль виникаючих в процесі функціонування ОС Windows подій.

При розробці системи плануємо використання API-функцій:

– для роботи з бібліотекою ОС Windows;

– для вирішення основної задачі - реалізації в системі функцій проактивного захисту.

Дійсно для того, щоб своєчасно виявити та блокувати в мережі команди на запуск процесів, а не запуск процесів (як це використовують антивірусні системи сигнатурного аналізу типів виявлених вірусів), доцільно здійснювати перехват саме API-функцій, поведінка яких є підозрілою з точки зору нашої системи.

Визначимо перелік підозрілих команд, які можуть використовувати зловмисники:

– копіювання файлів в системний директорію;

– видалення файлів з системної директорії;

– створення копій віруса вірусом (наприклад, вірус P2P.Worm.* проводить власне тиражування, використовуючи пірінгові мережі);

– запуск програмних додатків, які не санкціоновані переліком дозволених на запуск процесів.

Для перехвату API-функцій використаємо дієву методику сплайсингу. Сплайсинг – це підміна коду функції. Обґрунтування цього вибору надається в розділі 2 цієї пояснювальної записки.

Для розробки ПЗ системи нами обрано мову програмування DELPHI. Обґрунтування цього вибору надається в розділі 2 пояснювальної записки.

Таким чином, нами визначені та обґрунтовані:

– вимоги та обмеження, врахування яких повинна забезпечити майбутня система;

– методи, які будуть використані при розробці структурної та функціональної схем системи та її програмного забезпечення;

– структура організації та збереження даних;

– функції, виконання яких повинне забезпечити ПЗ, що підлягає розробці.

Маємо всі необхідні дані для розробки структурної, функціональної схем системи та діаграми процесів.

3.2 Розробка структурної схеми

Основною задачею, що повинна бути виконана згідно технічного завдання та постановки задачі на розробку КБР, є розробка програмного забезпечення системи кібербезпеки проактивного захисту.

До складу системи доцільно ввести 2 змістовних та 2 допоміжних модулів.

До змістовних модулів віднесемо:

– модуль проактивного захисту 2 (моніторинг програмних додатків) забезпечить в системі виконання наступних функцій: обробку отриманих від користувацької бібліотеки proControl повідомлень; вибір опцій налаштування захисту; збереження/читання даних з файлів налаштувань; завантаження dll-бібліотек;

– модуль проактивного захисту (моніторинг процесів) дозволить отримати перелік активних/неактивних процесів в системі.

До допоміжних модулів віднесемо:

– модуль відображення процесів: в системі забезпечить візуалізацію результатів моніторингу та дозволить в будь-який відлік часу одержати відображення стану контролюємих об’єктів;

– модуль відображення моніторингу захисту дозволить забезпечити відображення стану контролюємих об’єктів.

Окрім цього, до складу структури системи необхідно ввести файл налаштувань, який буде утримувати декілька INI-файлів. Це необхідно для збереження в розробленій системі налаштувань графічного інтерфейса та налаштувань функціоналів: списку дозволених («білий») до запуску процесів/додатків, списку заборонених («чорний») до запуску процесів/додатків тощо.

Кожен з означених модулів буде виконувати тільки притаманні йому функції, а одержані результати передавати до log-файла та поля статистики графічного інтерфейсу.

Під час розробки структури ПЗ згідно теми КБР, автором було зроблено особливий акцент на можливості вирішення програмою найбільшого кола задач, врахувавши при цьому позитивні якості систем-аналогів, розглянутих в розділі 2 пояснювальної записки, та здійснено спробу уникнення виявлених недоліків.

Таким чином, цільова функція системи вже чітко означена, тобто, задача на розробку системи поставлена. Для одержання текстів вихідних програм необхідно виконати ряд послідовних дій:

– докладно описати задачу;

– виконати аналіз задачі;

– провести інженерну інтерпретацію задачі (з залученням апарату формалізації);

– розробити загальну блок-схему алгоритму БСА роботи системи в цілому та БСА кожного модуля;

– розподілити робочі регістри і пам’ять МП ЕОМ;

– розробити тексти робочої програми.

При розробці БСА та текстів програм слід використати метод декомпозиції, при якому вся задача послідовно розподіляється на менші функціональні модулі, кожний з яких можна аналізувати, розробляти та

налагоджувати окремо від інших. При виконанні кожної окремої програми в МП керування передається від одного функціонального модуля до іншого.

Схема зв'язаності цих функціональних модулів, кожен з яких реалізує окрему процедуру, створює загальну БСА системи.

Це розподілення задачі на модулі і підмодулі виконується послідовно до такого рівня, коли розробка БСА стає простою і зрозумілою справою. Окрім цього, метод послідовної декомпозиції є достатньо гнучким, що дозволить провести ступінь деталізації БСА у відповідності із складністю процедури.

Така ієрархічна багаторівнева організація комплексу програм є на практиці основним засобом боротьби зі складністю розв'язуваних проблем. Окрім цього, модульна побудова ПЗ системи забезпечить високу ступінь мобільності та адаптивності, як це вимагає ТЗ та вимоги до сучасних систем даного класу.

Кожна програма та підпрограма будуть мати своє власне ім'я, по якому проводиться звертання до них, а в текстах програм будуть використовуватись імена – мітки для ідентифікації окремих програмних фрагментів та переходів на українській мові. В свою чергу, підпрограми та програми повинні забезпечити роботу головних програм, призначення яких – виконання визначених функцій системи.

Згідно з ТЗ до складу ПЗ необхідно ввести програму захисту ПЗ від несанкціонованого доступу та використання.

Мінімальна конфігурація комплексу технічних засобів КТЗ, що забезпечить в повному обсязі функціонування розробленого ПЗ, буде мати наступну конфігурацію:

– персональний комп'ютер ПК Intel® Celeron1,7Hz/1 Gb/200 Gb або сумісні з ним;

– операційна система Windows.

Таким чином, нами визначені складові програмного забезпечення, компоненти системи, тому можемо розробити структурну схему системи, яка наведена у відповідності з рисунком 3.1.

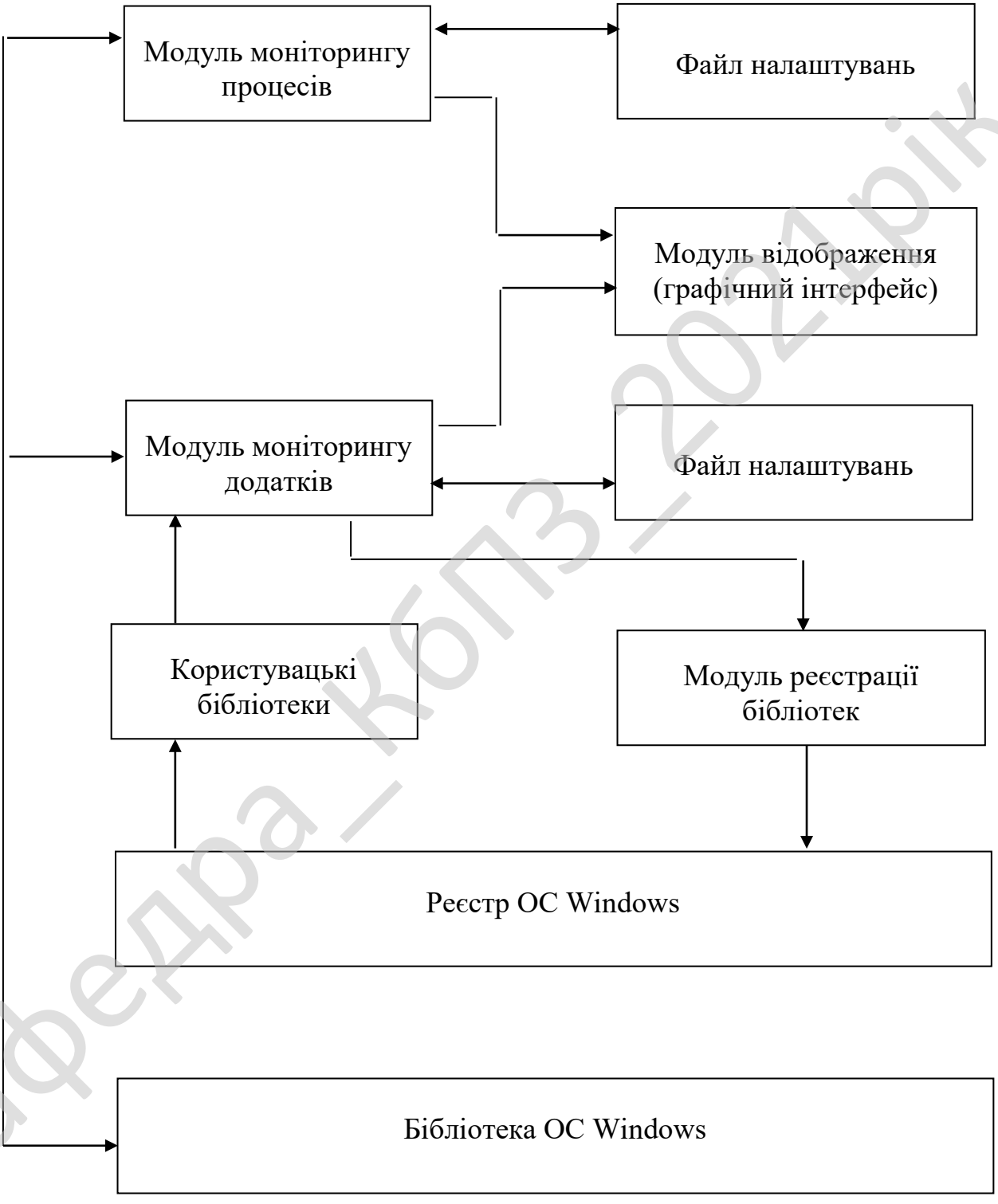

Рисунок 3.1 – Структурна схема системи

3.3 Розробка функціональної схеми

В розділі 3.2 пояснювальної записки визначені компоненти системи, що підлягають розробці в процесі виконання КБР, та розроблена її структурна схема.

Для побудови функціональної схеми системи необхідно конкретизувати функції кожного компоненту системи та визначити взаємозв'язок між ними. До складу системи введена головна програма, яка по суті є адміністратором системи і забезпечить в період експлуатації виконання наступних функцій:

– керування модулями системи: запуск на виконання, контроль виконання, закриття (після виконання модулем заданого завдання);

– робота графічного інтерфейсу;

– інтерактивний режим роботи користувача;

– обробка помилок,

та чотири модуля: два головних та два допоміжних.

До головних модулів системи віднесемо: модуль проактивного захисту 1 (моніторинг процесів), модуль проактивного захисту 2 (моніторинг програмних додатків). Їх функції в повному обсязі визначені в розділі 3.2 пояснювальної записки і тому деталізації та уточнень не потребують.

До допоміжних модулів віднесемо: модуль відображення процесів в системі та модуль відображення моніторингу захисту. Функції, які повинні виконувати ці модулі в процесі роботи системи, також в повному обсязі визначені в розділі 3.2 пояснювальної записки.

Слід зазначити, що модуль проактивного захисту 2 буде доповнювати функціонал модуля проактивного захисту 1 в частині роботи з дозволеними/забороненими в системі списками процесів/додатків. Вищезазначені модулі в своїй роботі будуть використовувати чотири бібліотеки ОС Windows: kernel 32. dll, ntdll.dll, PSAPI.dll, tehelp 32.dll та дві користувацькі бібліотеки: proControl.dll, proDialog.dll. Користувацькі бібліотеки підлягають розробці в процесі виконання КБР. Визначимо призначення кожної з цих бібліотек.

Бібліотека kernel32.dll підключається динамічно, присутня у всіх 32 - і 64-розрядних версіях Microsoft Windows. Вона надає додаткам багато базових API Win32, таких, як управління пам'яттю, операції введення/виводу, створення процесів, потоків і функцій синхронізації.

Бібліотека Ntdll.dll служить сполучною ланкою між API і ядром системи. Ntdll.dll дозволяє працювати з внутрішніми функціями ОС Windows без зміни функцій в інтерфейсних бібліотеках.

Бібліотеки PSAPI.dll та TlHelp32.dll містять функції, які надають доступ до отримання списку процесів в системі, їх модулів та іншої інформації про процеси. Наприклад функції:EnumProcesses, EnumProcessModules, GetModuleBaseName тощо.

Бібліотека proDialog.dll зберігає наступні графічні форми та формує білий список за допомогою їх інтерфейсу. Дані білого списку записуються у відповідний інформаційний файл, як це показано на рисунку 3.2.

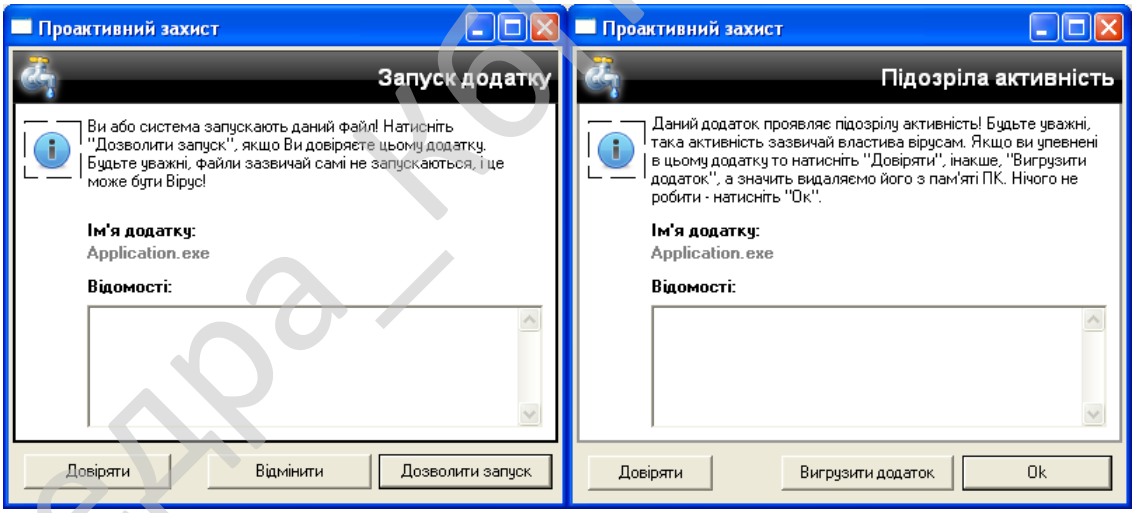

Рисунок 3.2 –Скриншот графічного інтерфейсу

Бібліотека proControl.dll є основним функціональним блоком проактивного захисту. Бібліотека стежить за запуском додатків, копіюванням/видаленням файлів з системної бібліотеки, створенням копій файлів. По суті - бібліотека є пасткою системних процесів по вищеописаних

подіях з послідуючою їх обробкою і подальшою передачею системі для виконання.

Наводимо перелік подій, які можна відстежити за допомогою бібліотеки proControl.dll:

– створення додатку, що може бути завантажником об’єкта через автозапуск;

– створення більше 7 копій програмного додатку;

– створення копії в системній директорії;

– створення підозрілого файлу в системній директорії;

– створення більше 5 файлів, схожих на вірус;

– спроба видалити файл з системної директорії;

– спроба створення файлу, схожого на вірус.

Ini-файл модуля відображення процесів використовується для зберігання переліку запущених процесів в ОС з указанням їх статусу: дозволений, заборонений, а також зберігає налаштування відповідного модуля: розмір вікна, режим роботи.

Модуль проактивного запуску має два файли налаштувань. Один зберігає список програм, які не підлягають контролю, оскільки внесені в довірену «білу» зону. Другий - зберігає налаштування модуля, виконані користувачем при попередньому запуску, або налаштовані розробником по замовченню.

Користувацька бібліотека prodialog.dll реєструється в реєстрі при інсталяції продукта, і завантажується в пам'ять ПК, знаходячись там постійно. Вона виконує функції проактивного захисту.

Графічний інтерфейс програмного додатку спрощує налаштування програми і активність користувацьких бібліотек, обмежуючи їх функції захисту.

Таким чином, маємо всі необхідні дані для побудови функціональної схеми системи, яка надається на рисунку 3.3.

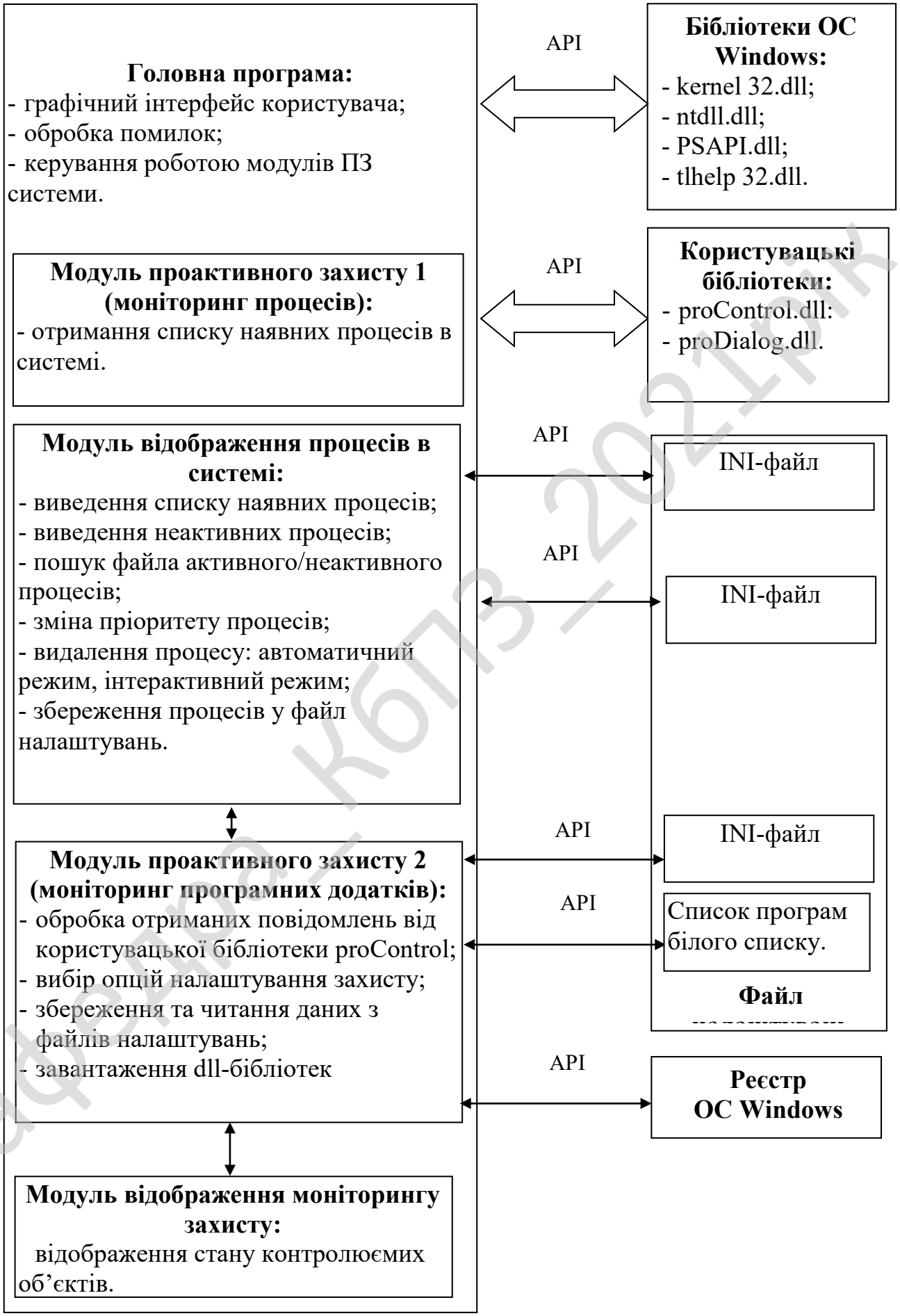

Рисунок 3.3 - Функціональна схема системи

Основні модулі ПЗ системи виконані у вигляді скомпільованих окремих програм і можуть функціонувати окремо одна від одної або разом, доповнюючи функціонал одна одній.

Для керування запуском всіх модулів проактивної системи захисту, розроблена головна програма, яка за своїм призначенням є диспетчером запуску.

Активація (запуск) модулів може відбуватись в будь-якій послідовності. Модуль моніторингу процесів не потребує додаткових інсталяцій бібліотек і модулів в систему (на відміну від модуля моніторингу програмних додатків.

Для функціонування модуля моніторингу програмних додатків, необхідно встановити користувацьку бібліотеку proControl.dll.

Після запуску бібліотека proControl.dll прописує себе в реєстрі ОС Windows на автозапуск. Дана бібліотека встановлює програмні пастки на виконання перехоплення API-функцій, описаних в системній бібліотеці kernel 32.dll для функцій: створення процесу, створення або відкриття файлу чи каталогу, копіювання файлу з одного місця в інше, видалення файлу.

Обробка кожної з контролюємих функцій виконується наступним чином: спочатку контроль проводиться проактивною системою захисту, а потім передається на виконання її системою, яка підлягає захисту. Тобто, API-функціїї програмних додатків, що їх викликають, будуть виконуватись, але під контролем нашої проактивної системи захисту.

Якщо дії програмного додатку будуть класифікуватись, як шкідливі, то вони будуть обмежуватись на виконання як в інтерактивному, так і в автоматичному режимах.

Даний процес відбувається з повним протоколюванням підозрілих подій, тому можна визначитись з поведінкою додатка і пізніше, переглядаючи його дії з системою.

3.4. Розробка діаграми процесів

Під час розробки системи були визначені головні та підлеглі їм процеси, означена їх взаємодія та розроблена діаграма процесів, яка наведена на рисунку 3.4.

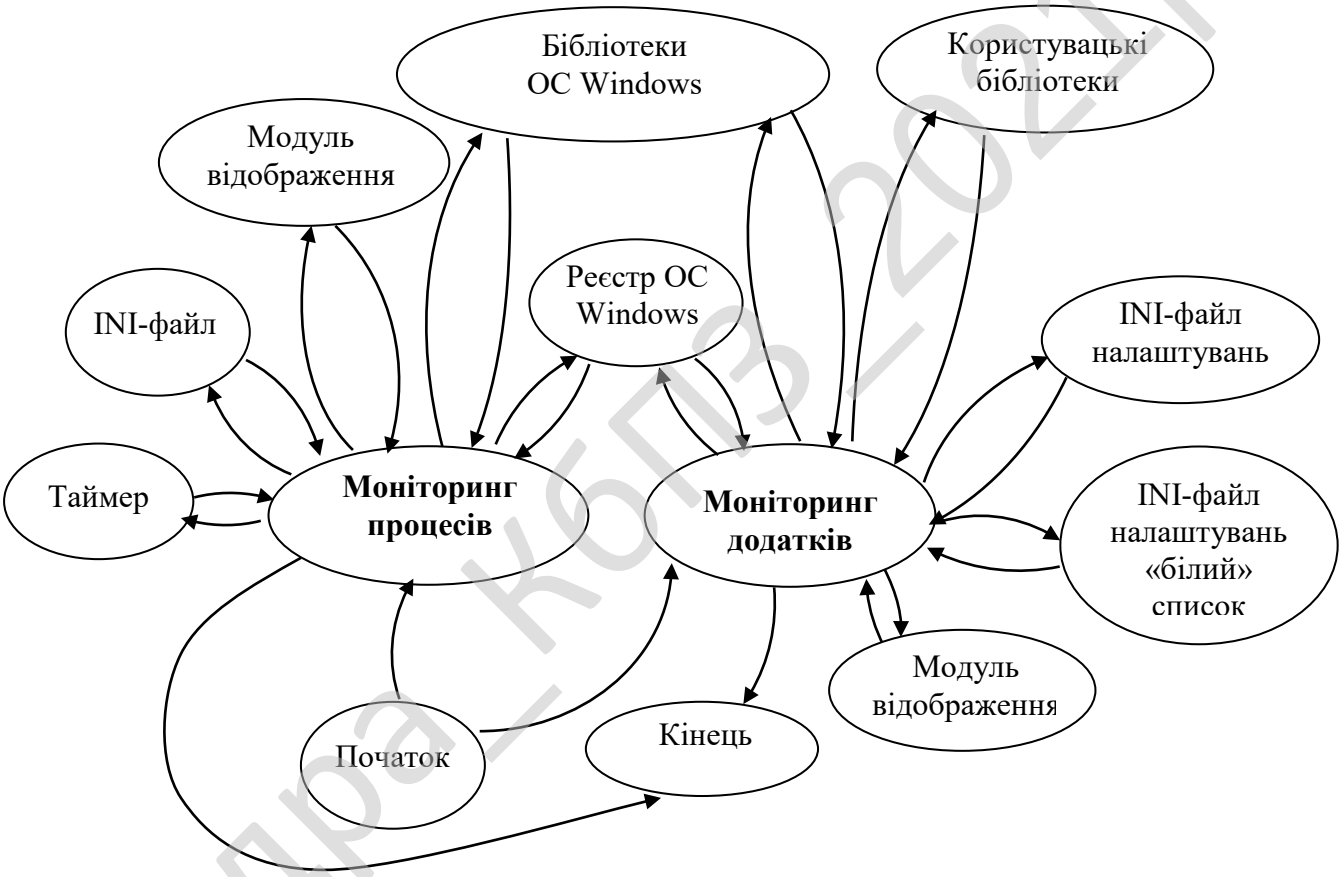

Рисунок 3.4 – Діаграма процесів

Таким чином, в процесі опису і обґрунтування проектних рішень був виконаний наступний обсяг роботи:

– проведений опису функціонування системи, визначені стандарті елементи системи та елементи, які підлягають програмній реалізації в процесі розробки КБР;

– проведена розробка структурної схеми системи, яка надається на рисунку 3.1;

– визначені функції складових системи, розроблена та надається на рисунку 3.3 функціональна схема системи;

– визначені основні та підлеглі їм процеси в системі, їх взаємодія; побудована та наводиться на рисунку 3.4 діаграма взаємодії процесів в системі.

Отже, маємо достатньо інформації для побудови блок-схем алгоритмів та їх опису програмним кодом.

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

Програмне забезпечення системи кібербезпеки проактивного захисту розроблене з урахуванням вимог розробленої концепції побудови систем аналогічного класу та спрямування (розділ 3.1 пояснювальної записки) і є, по сутті, багаторівневою системою реального часу, оскільки обробка даних здійснюється за позначками реального часу. Разом з тим, вона не повинна впливати на роботу інших систем захисту, не повинно виникати колізій і при роботі системи по безпосередньому призначенню з системним та програмним забезпеченням ПК. Тому ми притримувались системного підходу, в рамках якого окремі механізми системи, означені ТЗ, розглядались та проектувались в сукупності та з урахуванням їх можливого впливу на роботу інших модулів та додатків, що можуть знаходитись на ПК, де встановлена спроектоване ПЗ.

Таким чином, при проектуванні системи були враховані наступні аспекти:

- комплексування різнорідних механізмів в єдиній системі;
- взаємний вплив окремих механізмів, які можуть одночасно працювати з нашою системою;
- орієнтація на статистику погроз при визначені ключових механізмів моніторингу та проактивного захисту;
- ідентифікація та перевірка достовірності суб’єктів доступу при вході в систему по ідентифікатору/паролю/ключу.

4.1 Розробка блок-схем та опис алгоритмів функціонування системи

Дійсно, при використанні системи на ПК можуть суттєво змінитись вимоги до реалізації іншого механізмів захисту. Тому система повинна забезпечити функціонування механізму забезпечення замкнутості програмного

середовища, який дозволить ліквідувати будь-яку приховану погрозу. При цьому, користувач буде запускати лише санкціоновані для нього процеси, а значить – не зможе запустити програму, яка дозволить одержати несанкціоновані ПЗ/інформацію.

Виходячи з вищеозначеного, формуємо принципи системного підходу до проектування системи:

- система повинна проектуватись з урахуванням її впливу на безпеку інших систем в цілому та з урахуванням робочих функцій, що реалізуються іншими програмними продуктами;

- розробка системи – це багатокритеріальна задача, тому слід враховувати її можливий вплив на продуктивність роботи ПК вцілому;

- робота модуля моніторингу повинна бути направленою не тільки на виявлення погроз, а і на можливі канали їх виникнення.

В розробці системи використане середовище Borland та програмування під ОС WINDOWS з використанням подій WINDOWS та елементів ActiveX, що надає програмній моделі більшої наочності та зрозумілості для сучасного користувача, який звик працювати в цій операційній системі. Під час роботи системи працює досить потужний інтерфейс користувача, який забезпечує своєчасне виправлення помилок та надає, при необхідності, допоміжну інформацію.

Розроблене в процесі виконання КБР програмне забезпечення системи забезпечує виконання означених в ТЗ та більш детально визначених в розділі 3 пояснювальної записки функцій, направлених на виконання основної задачі – забезпечення проактивного захисту ПК.

Автоматизована система проактивного захисту, яка була розроблена в процесі виконання КБР, є складною багатофункціональною системою. В попередніх розділах пояснювальної записки детально описана структура системи та визначенні функції, виконання яких вона має забезпечити при роботі. Тому маємо всі необхідні дані для теоретичної побудови БСА та проведення необхідних теоретичних розрахунків алгоритмів.

Відзначимо, що задача проактивного захисту базується на використанні двох технологій: моніторингу та сплайсингу. Моніторинг має забезпечити контроль процесів додатків, які встановлені на ПК, який буде захищати розроблена система від вірусних атак.

При реалізації механізму рівневого контролю списків для попередження необхідно врахувати велику кількість часових параметрів, адже кожний з контролюємих списків має власний час перевірки/відновлення. Проте на практиці більшість контролюємих параметрів системи мають фактично однакові часові параметри, що дозволяє відокремити групи зі схожими параметрами. Тому було виділено три типи подій, а відповідно – і три типи механізмів контролю. При цьому дані механізми відбирались виходячи з того, що їх часові параметри відображають поведінку решти груп механізмів контролю. Вибрані такі механізми:

- механізми контролю процесів. Здійснює контроль заборонених/дозволених/обов'язкових до запуску процесів. Окрім прикладних і системних ресурсів до цієї групи також віднесемо драйвери, які заборонені/дозволені/обов'язкові до запуску;

- механізм контролю списків дозволених користувачів в системі.

Визначимо для обраних груп механізмів контролю часові параметри.

Для механізму контролю процесів ними будуть:

- $T_{ЗП}$ – час одержання управління процесом від моменту запуску до початку роботи;

- $T_{КП}$ – час контролю списків процесів, який є часом складання поточного списку запущених процесів;

- $T_{ОП}$ – час завершення процесу, який порушив цілісність списку заборонених/дозволених процесів.

Для того, щоб гарантовано виявити та захистити ОС від впливу програм, які порушують цілісність списку заборонених/дозволених процесів (завершити їх до того, як вони одержать управління), необхідно виконати наступну умову:

$$T_{МП} < T_{КП} + T_{ЗП} < T, \quad (4.1)$$

де: - $T_{МП}$ - максимальний інтервал часу, з яким повинна запускатися процедура контролю списку заборонених/дозволених процесів.

Для механізму контролю списків дозволених користувачів параметрами є час реєстрації нового користувача ($T_{РК}$); час контролю списку користувачів ($T_{КК}$), тобто – складення списку користувачів, які працюють на ПК/в мережі; час завершення сеансу користувачів ($T_{ЗС}$).

Для того, щоб гарантовано виявити та захистити операційну систему від дій незареєстрованого користувача, якого немає в списку заборонених/дозволених користувачів, необхідно виконання наступної умови:

$$T_{МІ} < T_{КК} + T_{ЗС} < T_{РК}\ , \qquad (4.2)$$

де: - $T_{МІ}$ - максимальний інтервал часу, з яким повинна запускатися процедура контролю користувачів.

Механізм контролю цілісності файлової системи використовує не списки прав доступу до файлових об’єктів/ключів реєстру пристроїв, а самі файлові об’єкти. Тобто, для даної групи контролюється не процес доступу, а безпосередньо стан файлового об’єкту. Тому параметрами для нього є: час доступу до об’єкту файлової системи ($T_{ДФ}$) на запис; контроль цілісності об’єктів файлової системи ($T_{КФ}$).

Для того, щоб гарантовано захистити систему від дій, пов’язаних з порушенням цілісності об’єктів файлової системи, необхідно виконати умову:

$$T_{МФ} < T_{КФ} < T_{ДФ}, \qquad (4.3)$$

де - $T_{МФ}$ - максимальний інтервал часу, з яким повинна запускатися процедура контролю цілісності об’єктів файлової системи.

Умова (4.3) виконуватись фактично ніколи не буде, адже, щоб проконтролювати цілісність файлового об’єкту, його необхідно прочитати. Але ж за тривалістю читання можна співставити з записом об’єкту, отже, умову (4.3) доцільно не брати при розробці алгоритмів системи до уваги.

Виконання умов (4.1) та (4.2) дозволяють використати рівневий контроль подій в позначках реального масштабу часу.

Контроль заборонених/дозволених/обов'язкових до запуску процесів та драйверів буде здійснюватись через рівні проміжки часу, тобто – періодично. Періодичність визначає конкретний замовник системи, виходячи з наявного типу ЕОМ та обсягу оперативної пам'яті. По замовченню в системі встановлена періодичність контролю 100 мілісекунд.

Для визначення на ПК наявності запущених процесів використаємо технологію сплайсингу, яка дозволяє організувати перехват API-функцій процесів і додатків. Розробимо блок-схему алгоритму роботи модуля проактивного захисту (рисунок 4.1) та розглянемо його програмну реалізацію.

Для перехвату API-функції потрібно переписати початковий код функції так, щоб він передавав управління обробнику. Для передачі управління обробнику достатньо лише однієї інструкції jmp на абсолютну адресу. Ця інструкція займе всього 5 байт: сам опкод цієї інструкції () і значення для стрибка. Значення інструкції обчислюється наступним чином:

```
v=0-(s-d)
s - Зсув наступної команди
d - Необхідна адреса для jmp, тобто-адреса обробника.
```

Якщо трохи переробити цю формулу, то вона прийме вигляд.

```
v=d-FunctionAddress-5.
```

Тепер при кожному виклику цільової функції, управління завжди передаватиметься обробнику.

Наступним кроком визначимо шляхи реалізації виклику оригінальної функції. При установці перехвату треба зберігати перші 5 байт функції. Для виклику оригіналу необхідно відновлювати функцію і викликати її, потім знову встановлювати перехват. Оголосимо структуру, в якій зберігатимемо перші 5 байт функції:

```
PFunctionRestoreData = ^ TFunctionRestoreData;
```

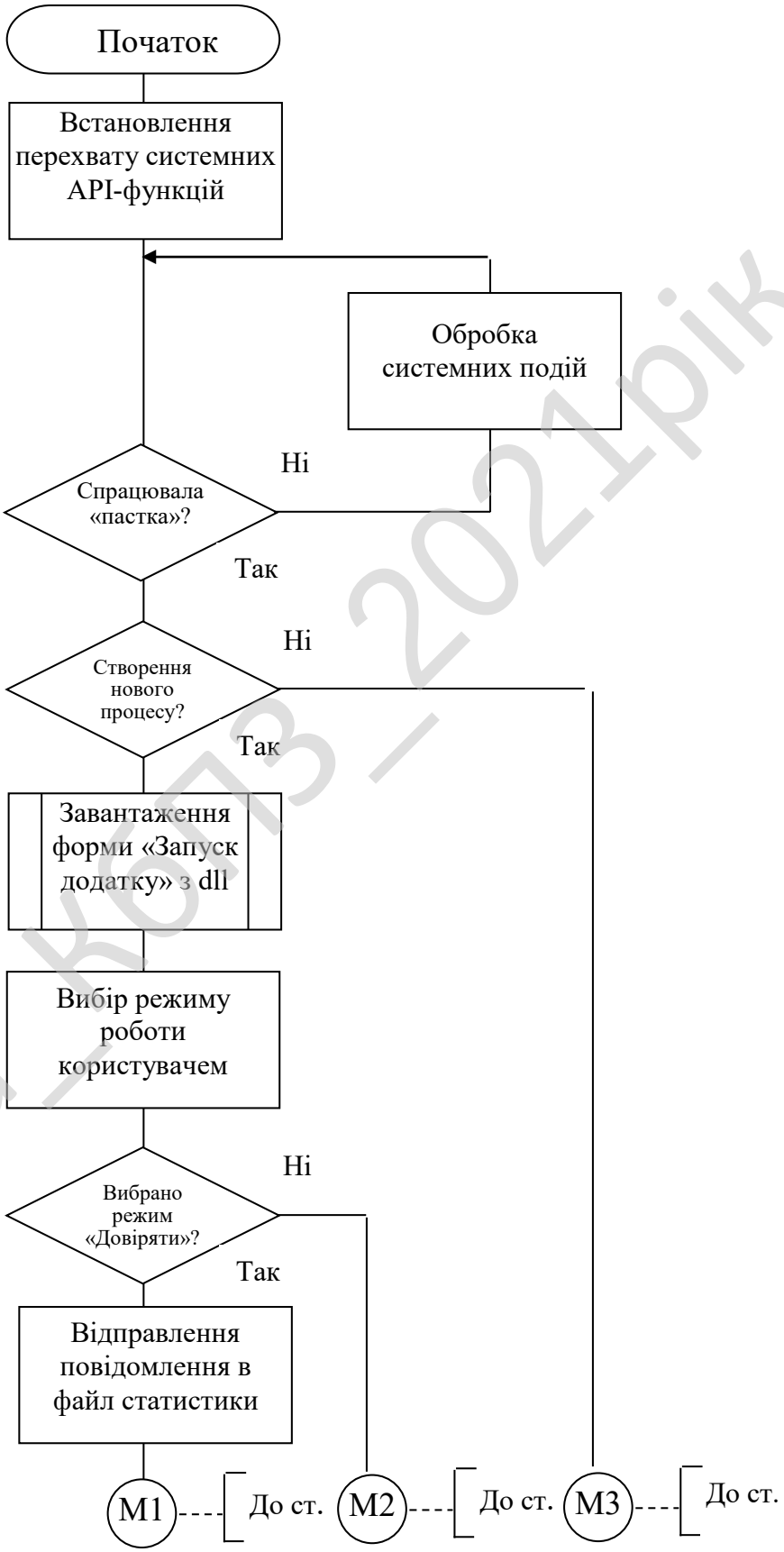

Рисунок 4.1 – Блок-схема алгоритму роботи модуля проактивного захисту 2

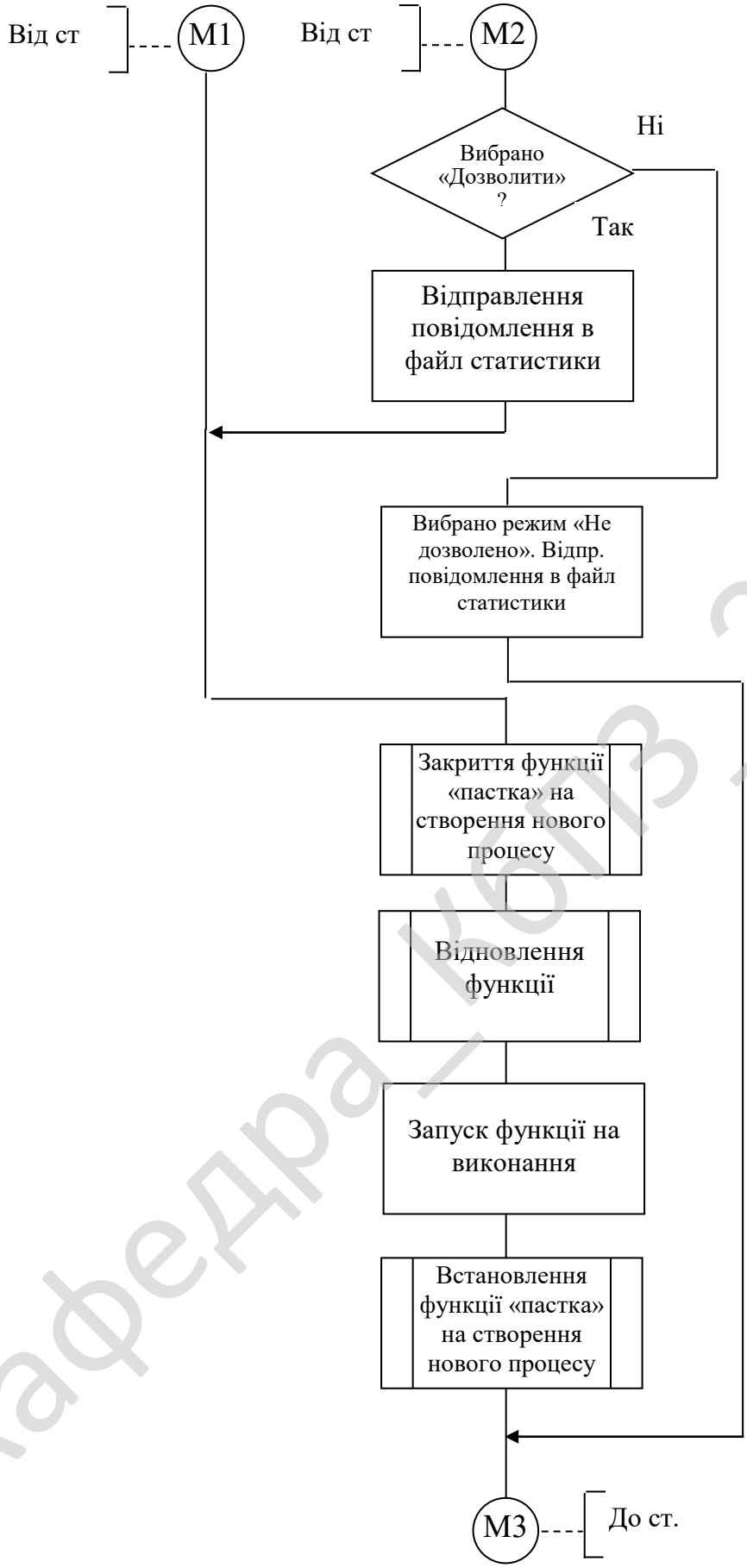

Рисунок 4.1, аркуш 2

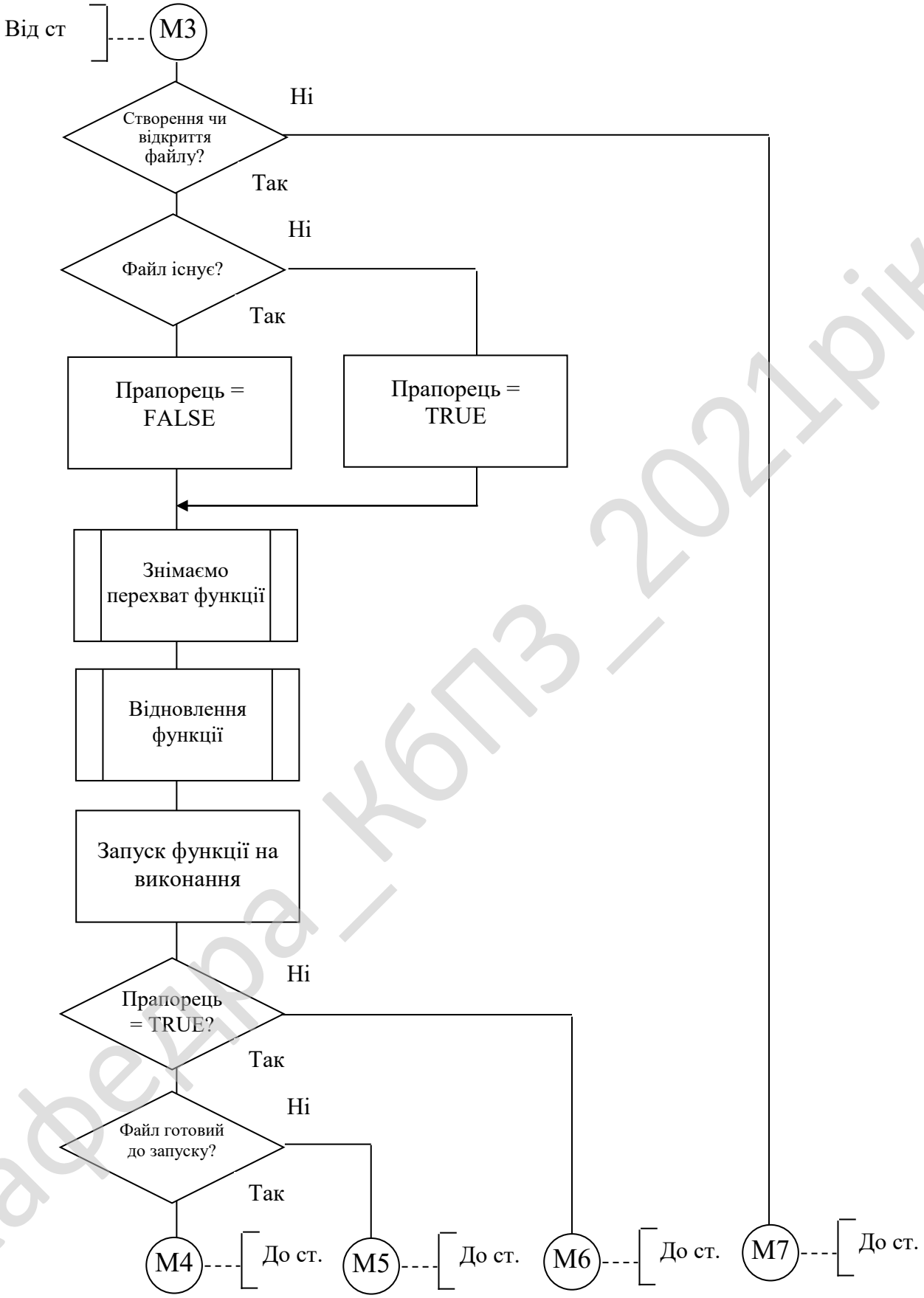

Рисунок 4.1, аркуш 3

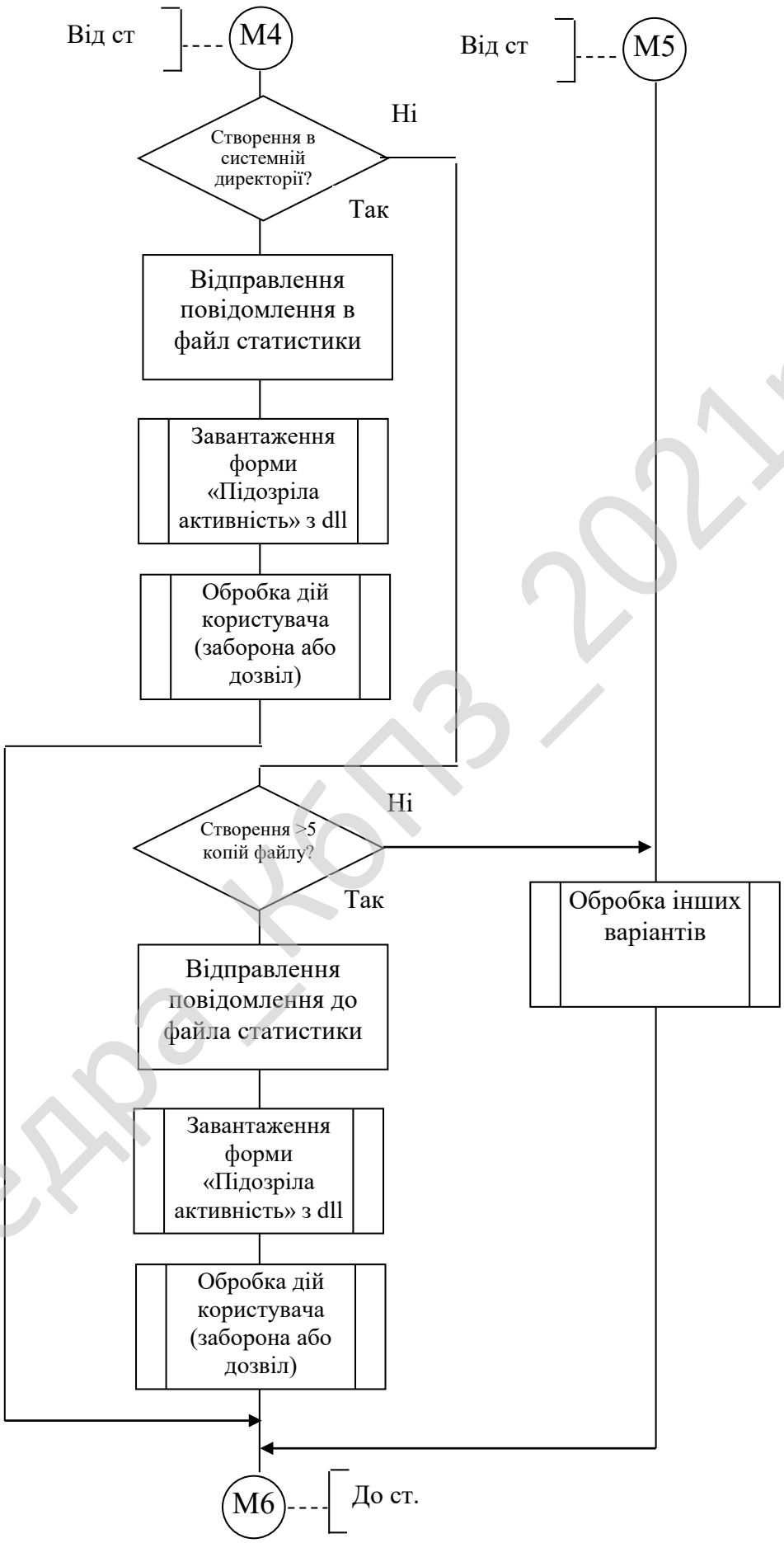

Рисунок 4.1, аркуш 4

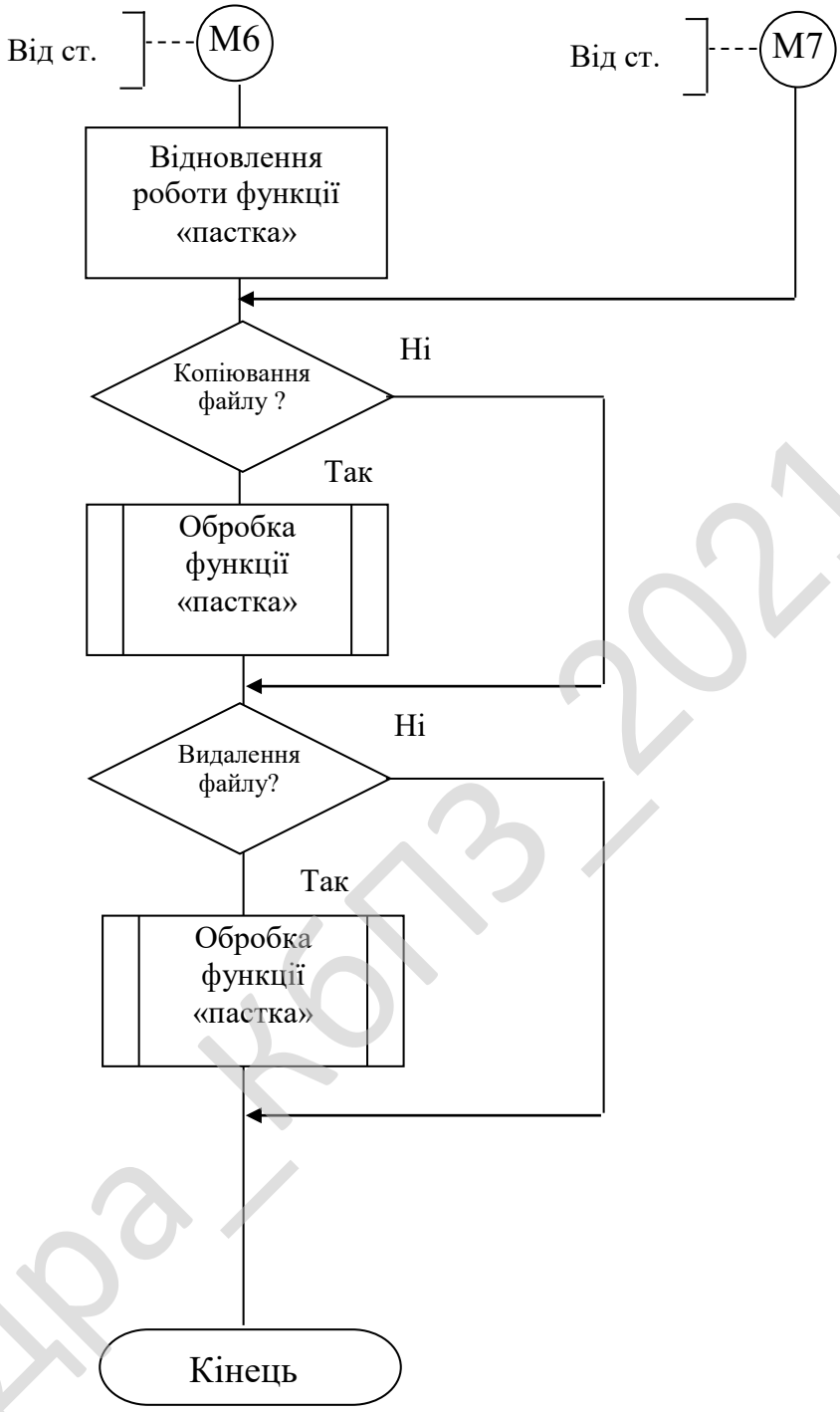

Рисунок 4.1, аркуш 5

```
TFunctionRestoreData = packed record
   Address:Pointer;
    val1:Byte;
    val2:DWORD;
 end;
```

Поле Address фактично в цій структурі не потрібне, але дозволяє більш зручніше знімати перехват. Назвемо цю структуру "містком" до старої функції.

Тепер напишемо функцію, яка встановлюватиме перехват:

```
{
  Procedure: UnHookCodeHook

  RestoreDATA:PFunctionRestoreData ????? ????? ? ?????? ???????
}

  function UnHookCodeHook(RestoreDATA:PFunctionRestoreData):Boolean;
var
  ProcAddress:Pointer;
  OldProtect,JMPValue:DWORD;
begin
  Result:=False;
  ProcAddress:=RestoreDATA^.Address;
  if  not  VirtualProtect(ProcAddress,5,PAGE_EXECUTE_READWRITE,OldProtect)
then exit;
  Byte(ProcAddress^):=RestoreDATA^.val1;
  DWORD(Pointer(DWORD(ProcAddress)+1)^):=RestoreDATA^.val2;
  Result:=VirtualProtect(ProcAddress,5,OldProtect,OldProtect);
end;
```

Спочатку встановимо атрибути доступу до коду функції так, щоб можна було його переписувати. Потім обчислимо значення для стрибка: спочатку зберігаємо початок функції в запис, потім переписуємо початок функції, в кінці встановимо старі атрибути доступу. Тепер напишемо функцію яка зніматиме перехват по імені функції:

```
{
  Procedure: UnHookCodeHook
     RestoreDATA:PFunctionRestoreData
}
  function UnHookCodeHook(RestoreDATA:PFunctionRestoreData):Boolean;
var
  ProcAddress:Pointer;
  OldProtect,JMPValue:DWORD;
begin
  Result:=False;
  ProcAddress:=RestoreDATA^.Address;
  if  not  VirtualProtect(ProcAddress,5,PAGE_EXECUTE_READWRITE,OldProtect)
then exit;
  Byte(ProcAddress^):=RestoreDATA^.val1;
  DWORD(Pointer(DWORD(ProcAddress)+1)^):=RestoreDATA^.val2;
  Result:=VirtualProtect(ProcAddress,5,OldProtect,OldProtect);
end;
{
  ProcAddress
NewProcAddress:pointer;
  RestoreDATA:PFunctionRestoreData
     Result: boolean
}
function SetCodeHook(ProcAddress, NewProcAddress: pointer;
     RestoreDATA:PFunctionRestoreData):boolean;
```

```
var
  OldProtect, JMPValue:DWORD;
begin
  Result:=False;
  if not VirtualProtect(ProcAddress,5,PAGE_EXECUTE_READWRITE,OldProtect)
    then exit;
  JMPValue := DWORD (NewProcAddress) - DWORD (ProcAddress) - 5;
  RestoreDATA^.val1:= Byte(ProcAddress^);
  RestoreDATA^.val2:= DWORD(Pointer(DWORD(ProcAddress)+1)^);
  RestoreDATA^.Address:=ProcAddress;
  byte(ProcAddress^):=;
  DWORD(Pointer(DWORD(ProcAddress)+1)^):=JMPValue;
  Result:=VirtualProtect(ProcAddress,5,OldProtect,OldProtect);
end;
```

Для цього відновимо початок функції, яка підлягає перехвату. Адресу її візьмемо із структури «Покажчик», на яку передається функція в якості єдиного параметру наступним чином:

```
{
  ModuleHandle:HMODULE; //Хендл модуля, в якому знаходиться цільова функція
  ProcedureName:PChar; //Ім'я процедури
  NewProcedureAddress:Pointer; //Покажчик на процедуру-перехватчик
  RestoreDATA:PFunctionRestoreData Покажчик на місток до старої функції

  Result: Boolean //Успішність установки перехвату

function SetProcedureHook(ModuleHandle:HMODULE;ProcedureName:PChar;
  NewProcedureAddress:Pointer;
  RestoreDATA:PFunctionRestoreData):Boolean;
var
   ProcAddress:Pointer;
begin
   ProcAddress:=GetProcAddress(ModuleHandle,ProcedureName);
   Result:=SetCodeHook(ProcAddress,NewProcedureAddress,RestoreDATA);
end;
```

Описані вище функції можуть перехвачувати функції тільки в поточному процесі. Тому приступимо до розробки алгоритму перехвату функцій в інших процесах.

Найбільш простий метод - це ввести перехватчик функції в DLL і в коді бібліотечної функції встановити перехватчик (якщо DLL завантажується в процес) і знімати перехват (якщо DLL вивантажується). Але потрібно вирішити ще одну проблему: як змусити інший процес завантажити нашу DLL. Найбільш просте рішення - це створення віддалених потоків. Віддалений потік створюється функцією CreateRemoteThread.

```
HANDLE CreateRemoteThread(
HANDLE hProcess, // хендл потоку, в якому створюється потік
```

```
LPSECURITY_ATTRIBUTES lpThreadAttributes,//атрибути безпеки
DWORD dwStackSize, //розмір стека
//адреса функції потоку, тобто - адреса першої інструкції потоку
LPTHREAD_START_ROUTINE lpStartAddress
LPVOID lpParameter, // покажчик на параметр для функції
DWORD dwCreationFlags, // прапорці створення
// покажчик на змінну, в якій буде збережений ID-потоку
LPDWORD lpThreadId
);
```

Функція потоку буде мати наступні атрибути:

```
DWORD WINAPI ThreadFunc(PVOID pvPararn)
```

WINAPI позначає stdcall, але це не важливо, що вона позначає, все одно вона буде оголошена, як stdcall. У функції тільки один параметр - це звичайний покажчик.

Тепер проаналізуємо ситуацію: функція LoadLibraryA має такі ж атрибути. Вона приймає покажчик на перший символ імені файлу DLL (рядок повинен кінчатися символом #0). Отже, функція LoadLibraryA повністю підходить для того, що б вона могла виступати, як функція потоку. Оскільки вона приймає покажчик на рядок в своєму процесі, треба записати в пам'ять чужого процесу наш рядок і ім'я файлу DLL. Зробимо це функцією WriteProcessMemory наступним чином:

```
BOOL WriteProcessMemory(
HANDLE hProcess, // хендл процесу
LPVOID lpBaseAddress,// адреса, по якій треба записати
LPVOID lpBuffer, // покажчик на буфер
DWORD nSize, // кількість байт для запису
LPDWORD lpNumberOfBytesWritten //кількість реально записаних байт
);
```

Адресу функції LoadLibrary A отримаємо за допомогою функції GetProcAddress так: бібліотеки kernel32.dll і ntdll.dll вантажаться у всі процеси завжди по одних і тих же адресах, отже, та адрес, який отримано в адресному просторі процесу, буде дійсним і в адресному просторі будь-якого іншого процесу.

Тепер напишемо функцію, яка завантажує DLL в адресний простір чужого процесу:

```
function LoadLibrary_Ex(ProcessID:DWORD;LibName:PChar):boolean;
```

```
var
  pLL,pDLLPath:Pointer;
  hProcess,hThr:THandle;
  LibPathLen,_WR,ThrID:DWORD;
begin
  Result:=False;
  LibPathLen:=Length(string(LibName));
  hProcess:=OpenProcess(PROCESS_ALL_ACCESS,false,ProcessID);
  if hProcess=0 then exit;
pDLLPath:=VirtualAllocEx(hProcess,0,LibPathLen+1,MEM_COMMIT,PAGE_READWRITE)
  if DWORD(pDLLPath)=0 then exit;
  pLL:=GetProcAddress(GetModuleHandle'(kernel32),'LoadLibraryA);
  WriteProcessMemory(hProcess,pDLLPath,LibName,LibPathLen+1,_WR);
  hThr:=CreateRemoteThread(hProcess,0,0,pLL,pDLLPath,0,ThrID);
  if hThr=0 then exit;
  Result:=CloseHandle(hProcess);
end;
```

Якщо процесу не дозволяється зміна пам'яті системних процесів, таких як winlogon.exe, lsass.exe, smss.exe, csrss.exe і ін., використаємо SeDebugPrivilege. В програмний код модуля для перехвату API-функцій введемо функцію EnableDebugPrivilege, яка включає цей привілей для поточного процесу.

Задача завантажування своєї DLL в чужий процес вирішена, але для належного ефекту роботи системи треба перехвачувати DLL у всіх процесах системи. Для цього скористаємося технікою так званих хуків.

Коли ми ставимо глобальний хук на повідомлення вікон за допомогою функції SetWindowsHookEx, цією функцією фактично передаємо адресу функції обробника і хендл DLL, в якій знаходиться функція - обробник. Після того, як який-небудь додаток буде адресований на відправку якому-небудь вікну, то перевіряється, чи встановлена пастка на повідомлення, яке він хоче послати вікну. Якщо так, то завантажується DLL, в якій знаходиться обробник повідомлень, викликається обробник і так далі

Отже, після установки хука на повідомлення, DLL знаходитиметься у всіх процесах, які хоча б один раз відправили повідомлення вікнам, тобто - майже у всіх GUI процесах. Каркас DLL з перехватом функцій виглядатиме наступним чином:

```
library HIDEDLL;

const
   MutexName='__API_HOOK';
...
    //обробник повідомлень
```

```
function MsgProc(code:DWORD;wParam,lparam:DWORD):DWORD;stdcall;
begin
   CallNextHookEx(SH,code,wParam,lparam);
end;

     //впровадження DLL у всі GUI процеси
procedure SetWindowsHook(e:Boolean); stdcall;
var
   M:THandle;
begin
   if e then
     begin
      m:=CreateMutex(0,false,MutexName);
      if m=0 then exit;
      SH:=SetWindowsHookEx(WH_GETMESSAGE,@MsgProc,HInstance,0);
    end
        else
    UnhookWindowsHookEx(sh);
end;

    //головний бібліотечний обробник
procedure DLLEntryPoint(dwReason:DWord);
begin
  case dwReason of
    DLL_PROCESS_ATTACH:
    begin
    // StopProcess(GetCurrentProcessId);
       SetWindowsHook(true);

SetProcedureHook(GetModuleHandle'('ntdll.dll'),'ZwQuerySystemInformation
            @NewSystemFunction,@SystemFunctionBridge);
    // ResumeProcess(GetCurrentProcessId);
    end;
    DLL_PROCESS_DETACH:
    begin
    // StopProcess(GetCurrentProcessId);
        SetWindowsHook(false);
        UnHookCodeHook(@SystemFunctionBridge);
    //ResumeProcess(GetCurrentProcessId);
    end;
  end;
end;

begin
  DllProc:= @DLLEntryPoint; //встановлюємо бібліотечний обробник
  DLLEntryPoint(DLL_PROCESS_ATTACH);
end.
```

Тоді щоб встановити перехват на API - функції у всіх процесах (у всіх GUI - процесах) достатньо просто завантажити нашу DLL. Для цього напишемо наступний код.

```
LoadLibrary(pchar(ExtractFileDir(Application.ExeName)+'\'+'HideDLL.dll'));
```

Аналогічно проведена побудова алгоритму проактивного захисту 1 та його реалізація програмним кодом, тому більш детально це питання визначати не будемо.

Модуль проактивного захисту 1 та модуль проактивного захисту 2 завжди повинні бути в робочому стані зі всіма збереженими налаштуваннями та даними, але бажано, щоб їх не було в списку задач, доступних при перелистуванні через <Alt> + <Tab> і щоб вони не займали робочого простору на панелі задач.

Але разом з тим, ці модулі весь час в фоновому режимі виконують корисні для користувача дії, а не просто чекають, коли до них звернуться. Тобто, по суті, розроблена система є резидентною програмою. З метою забезпечення максимальної зручності роботи користувача з системою, було прийняте рішення використати замість її закриття - згортання в System Tray.

Система має графічний інтуїтивно зрозумілий інтерфейс та забезпечує високий ступінь візуалізації. Для цього в процесі виконання КБР розроблено модуль відображення моніторингу захисту, який відображає стан контролюємих об’єктів. Окрім даного, візуального модулю, в системі необхідно забезпечити (і в першу чергу – для роботи модулів проактивного захисту 1 та 2) і роботу невидимих модулів.

Одним з невидимих модулів в системі є робота dll-бібліотек. Dll-бібліотека при запуску системи завантажується в пам’ять ПК і забезпечує на протязі всього робочого циклу системи виконання функції перехвату API-функцій системних подій зміни стану вікон програмних додатків та процесів. Програмні додатки запускаються системою по мірі необхідності їх використання з послідуючим записом до Log-файлу.

ПЗ системи має модульну побудову. До його складу ввійшли як стандартні модулі, так і модулі, що були розроблені в процесі виконання КБР, які, в свою чергу, описані процедурами, програмами та підпрограмами. Керування модулями системи забезпечує розроблена в процесі виконання КБР головна програма, блок-схема алгоритму роботи якої наведена на рисунку 4.2.

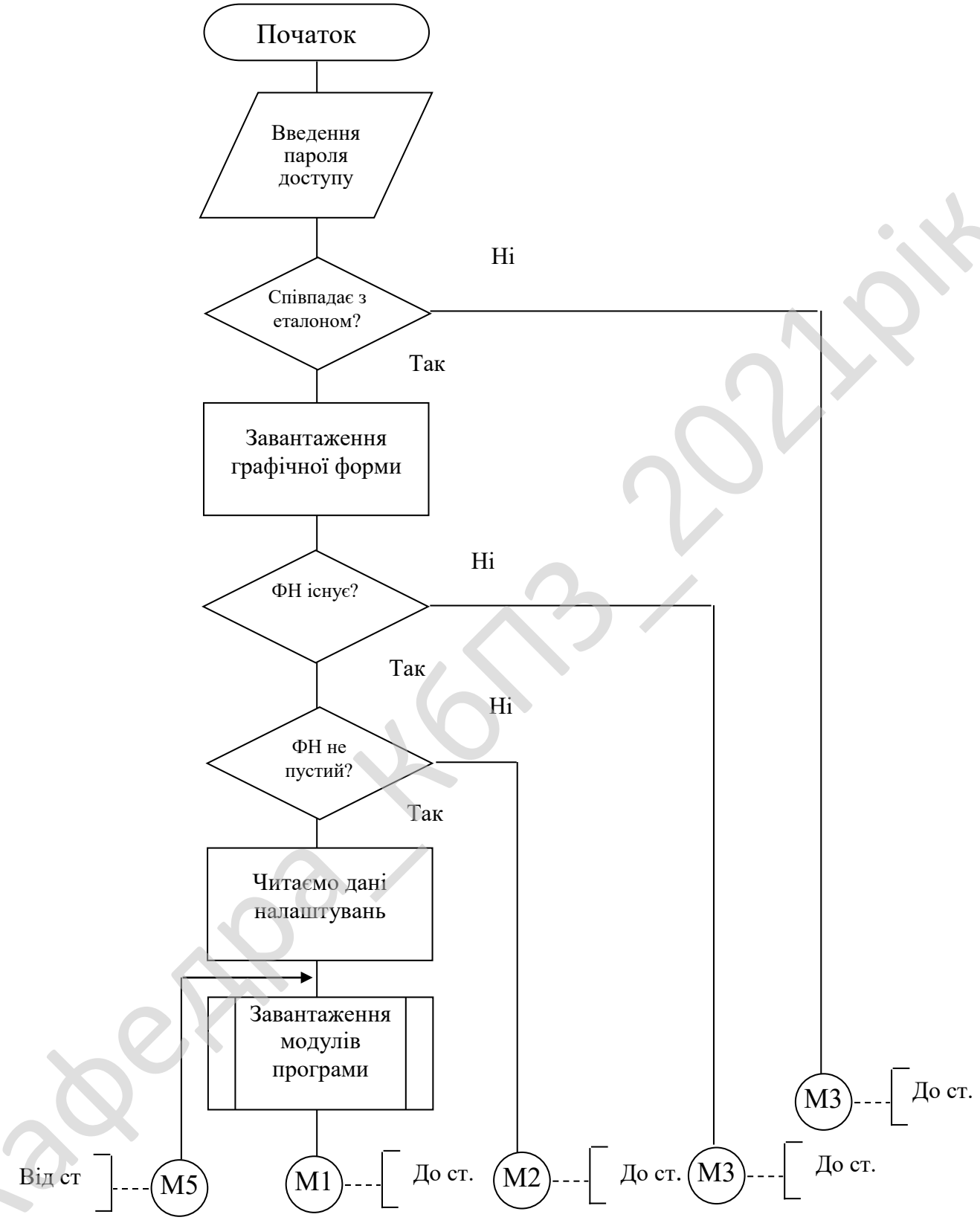

Рисунок 4.2 – Блок-схема роботи алгоритму головної програми

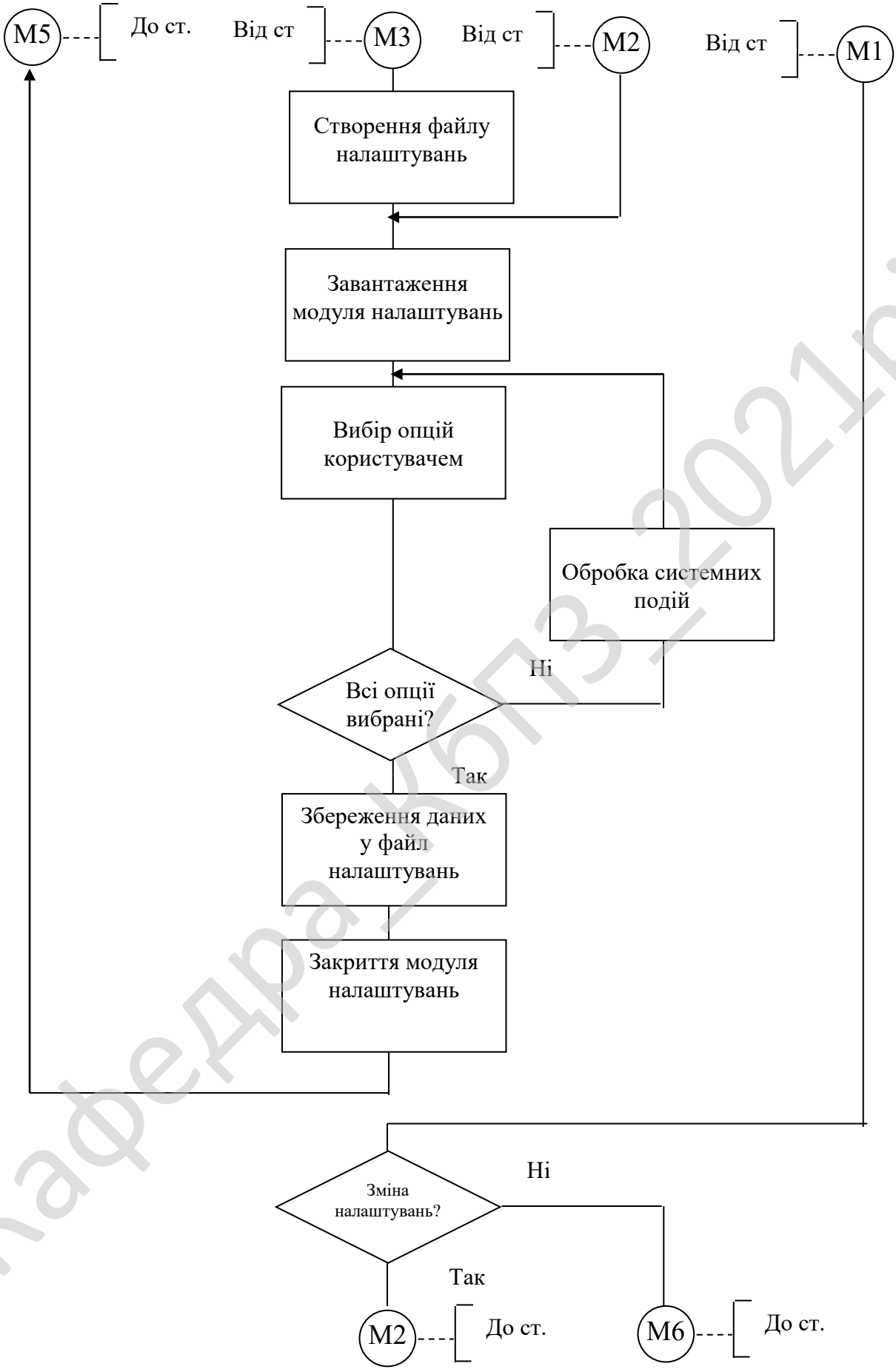

Рисунок 4.2, аркуш 2

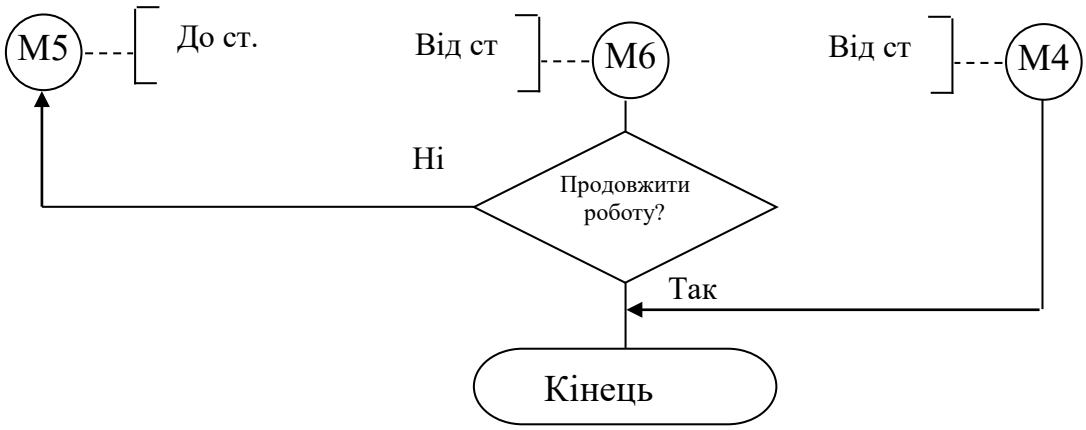

Рисунок 4.2, аркуш 3

Переваги системи, розробленої в процесі виконання КБР, визначаються наступним:

- низьке споживання системних ресурсів;

- невимогливі до апаратного забезпечення ПК (можуть працювати на різних платформах);

- мала кількість звернень до користувача ПК.

До недоліків системи віднесемо вимогу до наявності у користувача певного рівня знань про принципи функціонування ОС.

В наведених основних модулях та процедурах програмного забезпечення підтримується принцип інформативності, а саме: надання користувачу інформації про виконання встановлених перевірок та результати роботи окремих програм та підпрограм, надання в разі необхідності теоретичних відомостей щодо принципів роботи системи, необхідних для роботи технічних параметрів, тощо. Інформаційні повідомлення в процесі роботи системи виводяться на екран монітора чи роздруковуються (по бажанню користувача).

Таким чином, завдання визначене ТЗ та постановкою задачі, виконане в повному обсязі. Основні етапи розробки ПЗ системи включають:

- розробку структурної і функціональної схем системи для визначення та обґрунтування вибору структурних частин програмної моделі та їх функцій;

- розробку системи відповідно концепції, принципам та методиці реалізації систем аналогічного класу та спрямування;

- теоретичне обґрунтування розробки основних рішень побудови програмної моделі та виконання необхідних розрахунків;

- розробка алгоритмів та ПЗ системи.

Розроблене ПЗ має модульну структуру, що забезпечує виконання системою принципів мобільності та адаптивності і цілком дієспроможне. Про це свідчать результати дослідної експлуатації. Розроблено досить потужну систему з додержанням усіх вимог до систем даного класу, яка в подальшому може вдосконалюватись шляхом введення нових функцій – сервісних та робочих. Подальший рекомендований напрямок вдосконалення: розширення сервісних функцій по безпосередньому спрямуванню системи шляхом введення до її складу елементів експертних систем та елементів штучного інтелекту

Практична значимість роботи. Запропоновані методи й розроблені додатки можуть бути впроваджені в будь-якій організації/установі/навчальному закладі/підприємстві, які працюють з персональними комп’ютерами та комп’ютерними мережами.

4.2 Захист розробленого програмного забезпечення

Основною проблемою, яка пов’язана зі специфікою комп’ютерної програми як об’єкту авторського права, на даний час є нелегальне розповсюдження та використання “піратських копій” ПЗ, що порушує права власників на ПЗ. Саме тому необхідно створити повноцінний захист інтелектуальної власності, особливо, зважаючи на катастрофічне положення в Україні, пов’язане з нелегальним розповсюдженням копій ПЗ.

На даний час існує багато способів захисту ПЗ від незаконного розповсюдження та використання. Вони поділяються на програмні, організаційні, технічні та юридичні. На самому ефективному етапі боротьби з порушеннями –

етапі профілактики – реальний захист виключних прав на комп'ютерні програми можна забезпечити лише встановленням засобів програмного захисту. Задача розробника ПЗ полягає в тому, щоб користувач, встановивши задачу на ПК, не зміг би використати її внаслідок того, що спрацюють засоби програмного захисту від нелегального копіювання. Це – завдання програмістів, що має бути реалізоване на етапі розробки ПЗ, або на початку промислової експлуатації системи на підприємстві (в організації).

З цілком зрозумілих причин, в нашому випадку автору не доступні засоби технічного захисту ПЗ, аспекти використання організаційного та юридичного захисту. Тому, дбаючи про свої права, автор виконав захист розробленого ПЗ на програмному рівні, використавши для цього парольний захисту ПЗ системи від несанкціованого доступу користувачів в систему.

Розроблений парольний захист полягає в наступному: користувач вводить пароль, який порівнюється з еталоном в БД. Якщо пароль вв едено правильно – доступ до системи відкрито, якщо ні – кінець роботи системи. Від інших погроз чи дій зловмисників система здатна захистити себе сама.

5 ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Впровадження розробленої кваліфікаційної бакалаврської роботи в експлуатацію спряжене з виконанням певного обсягу роботи, яка потребує сумісної участі спеціалістів організації-замовника і розробника.

Впровадження здійснюється в відповідності з планом-графіком, в якому передбачені всі необхідні заходи для реалізації системи: підготовка системи до впровадження: розробка, узгодження та затвердження плана-графіка впровадження; наказ про готовність системи до впровадження; комплексація системи з наявним у користувача ПЗ та ОС в установленому порядку; навчання персоналу, який буде експлуатувати систему; налагоджувальні роботи: автономне налагодження функцій системи з складанням відповідного акту; проведення випробування системи на працездатність перед її передачею в промислову експлуатацію; дослідна експлуатація: включення системи в дослідну експлуатацію, оформлення по її результатах відповідного акту, прийняття рішення про передачу системи в промислову експлуатацію чи на доробку розробнику.

Розроблена система може бути впроваджена в промислову експлуатацію: на об’єкті будь-якої сфери діяльності для використання по прямому призначенню; в якості підсистеми ввійти до складу більш потужної системи аналогічного класу та спрямування.

Система, що була розроблена в процесі виконання кваліфікаційної бакалаврської роботи, являє собою комплекс програмних засобів, структура та зв’язок яких детально описані в розділах 3 та 4 пояснювальної записки і забезпечує вирішення актуальної в наш час задачі – захист інформаційних масивів від вірусних атак з глобальної мережі. Тому впровадження системи може бути ефективним при умові наявності методичного та організаційного

забезпечень системи, які є сукупністю документів і матеріалів, що визначають правила впровадження та експлуатації засобів забезпечення ефективної роботи системи.

Методичне забезпечення на стадії впровадження – це комплекс документів, що утримують правила та інструкції по використанню ПЗ, а саме: інструкції по використанню програмного забезпечення; інструкції користувача, який буде працювати з системою; штатний розклад обслуговуючого персоналу; посадові інструкції обслуговуючого персоналу; інструкції по супроводженню промислової експлуатації програмного забезпечення.

Організаційне забезпечення – це перелік документів, що регламентують взаємодію і функції підрозділів підприємства та матеріально-технічного забезпечення системи, а саме: накази та розпорядження, які визначають функції підрозділів по експлуатації та обслуговуванню програмного забезпечення та технічного обладнання системи; затверджений графік постачання матеріалів для проведення регламентних робіт ЗОТ та запасних частин для їх ремонту; графіки регламентних робіт.

Методичне та організаційне забезпечення забезпечують виконання плану впровадження.

План впровадження системи передбачає організацію підготовки кадрів, експлуатаційного персоналу та аналіз функціонування системи після виконання всіх робіт із її впровадження. Крім конкретних термінів виконання робіт, план впровадження системи установлює: обсяги фінансування проектних і налагоджувальних робіт; терміни розробки проектної документації по елементах системи; терміни виконання пусконалагоджувальних робіт.

У плані впровадження зазначаються організації (підрозділи), які відповідають за виконання робіт. Тому для впровадження розробленого програмного забезпечення в експлуатацію потрібна взаємодія програміста-розробника з користувачем, ІТР по обслуговуванню технічного обладнання, керівників різних рівнів, підрозділів, які будуть його експлуатувати.

Тому лише виконання вимог та правил МЕТОЗ дозволить в короткий термін часу провести необхідні реорганізаційні зміни та успішно впровадити систему в промислову експлуатацію.

Для ілюстрації виконання розробленим програмним забезпеченням основних функцій згідно ТЗ, наводимо скриншоти (рисунок 5.1, рисунок 5.2, рисунок 5.3, рисунок 5.4).

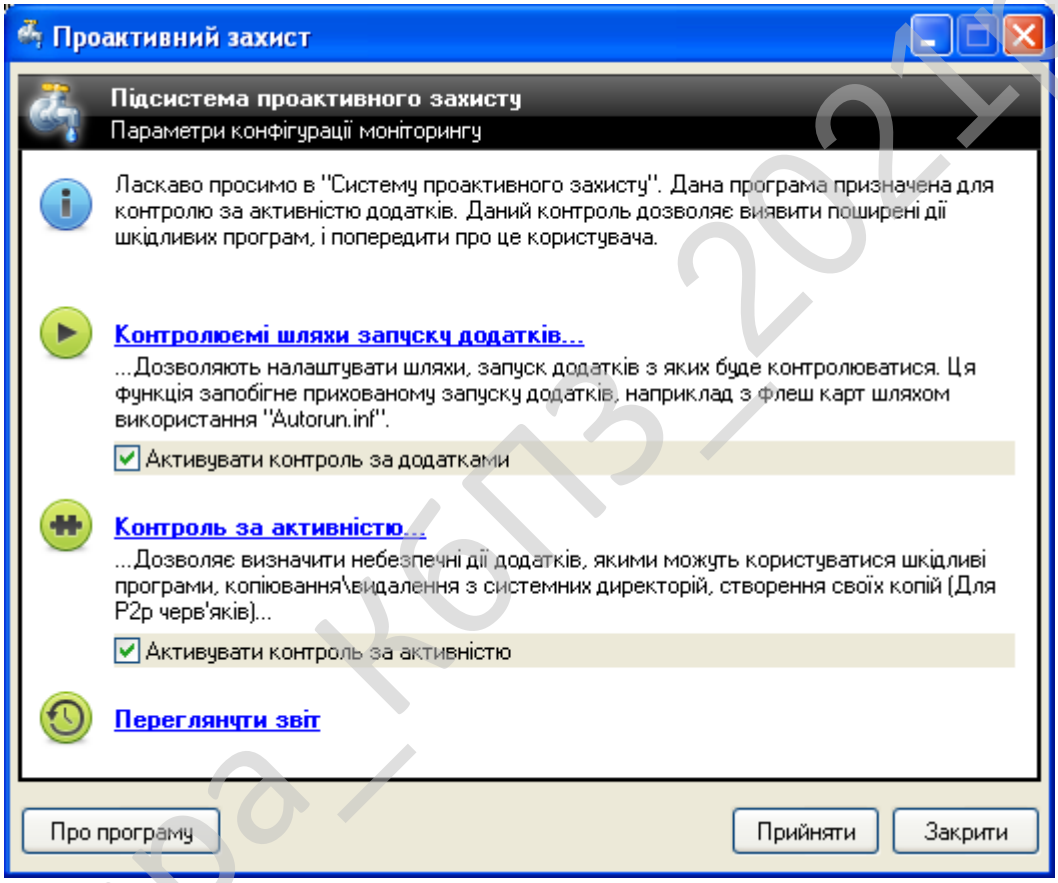

Рисунок 5.1 – Скриншот головної форми налаштування проактивної системи захисту 2

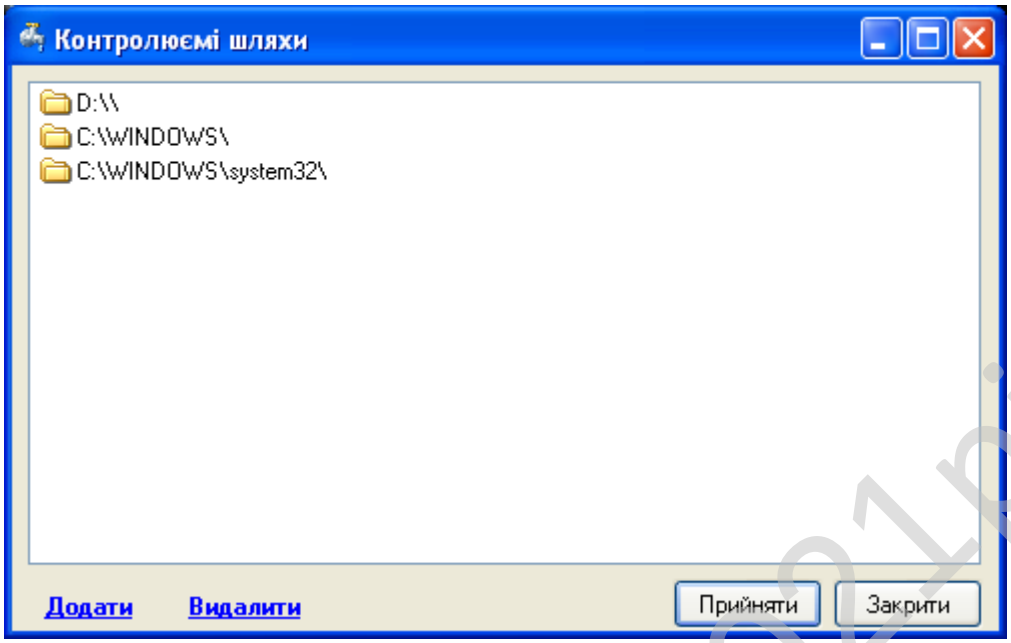

Рисунок 5.2 – Скриншот форми налаштування контролюємих шляхів

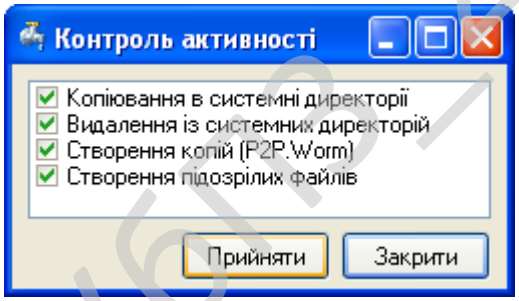

Рисунок 5.3 – Скриншот форми налаштування контролю активності

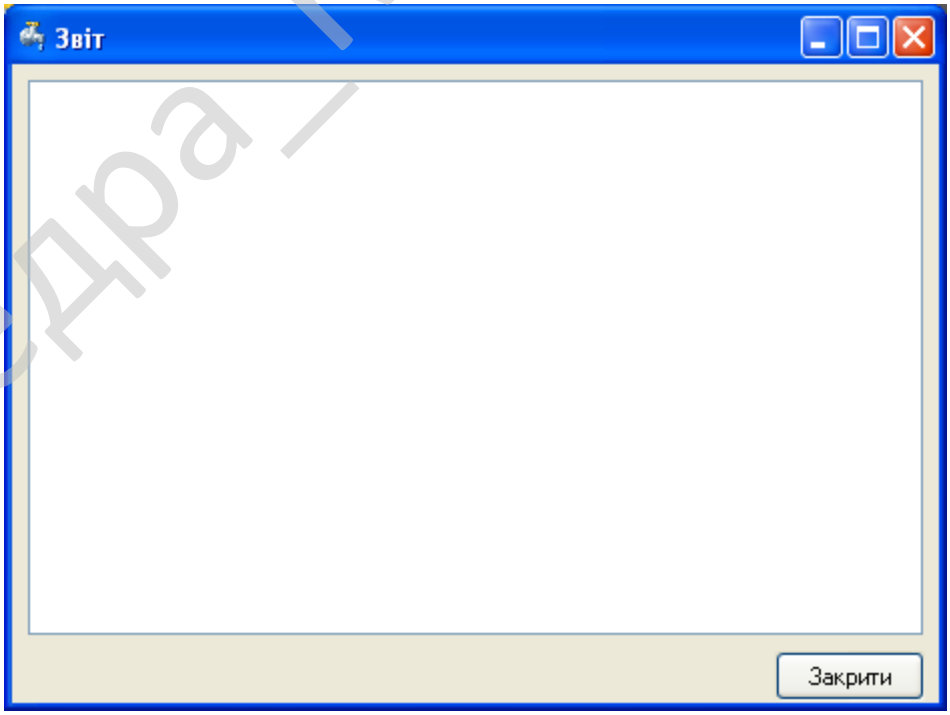

Рисунок 5.4 - Скриншот форми звіту

6 ОСНОВНІ ВИСНОВКИ

При розробці кваліфікаційної бакалаврської роботи згідно постановки задачі був виконаний наступний обсяг роботи:

– Розроблене та узгоджене технічне завдання та календарний план на розробку програмного забезпечення проактивного захисту на основі використання API-функцій (Додаток А до пояснювальної записки).

– Визначене призначення системи – захист інформаційних масивів, комп'ютерних систем користувача на основі використання розробленого ПЗ системи проактивного захисту, який полягає в виявлені загрози вірусної атаки до її появи. Область впровадження розробленої системи - будь-яка сфера діяльності людини, пов'язана з використанням ПК та сучасних комп'ютерних технологій.

– У розділі 2 пояснювальної записки проведено огляд існуючих систем-аналогів та інших джерел інформації щодо теми КБР з ціллю визначення позитивних якостей їх побудови для подальшого врахування при розробці програмного забезпечення системи. Проведено вибір та обґрунтування засобів для побудови системи, мови програмування, концепції розробки. Розроблена постановка задачі щодо реалізації технічного завдання.

– У розділі 3 пояснювальної записки проведено опис і обґрунтування проектних рішень щодо побудови архітектури майбутньої системи. Розглянуті компоненти її структури з ціллю визначення переліку функцій, виконання яких має забезпечити система. Розроблені та обґрунтовані: структурна та функціональна схеми системи; визначена взаємодія функцій та підфункцій програмного забезпечення. Визначені головні та підлеглі процеси та їх взаємодія в процесі роботи системи, побудована діаграма взаємодії процесів в системі. Визначена конфігурація мінімального комплексу технічних засобів, які забезпечать промислову експлуатацію системи.

– У розділі 4 пояснювальної записки надаються: теоретичні викладки та розрахунки, необхідні для розробки програмного забезпечення системи та які

підтверджують вірність програмних рішень. Розроблені блок-схеми основних алгоритмів системи та тексти програм ((Додаток В до пояснювальної записки). Визначені напрямки подальшого вдосконалення програмного забезпечення системи: доповнення головної програми новими модулями для розширення її функціональних можливостей. Визначені принципи захисту програмного забезпечення від несанкціонованого доступу зловмисників, що захочуть посягнути на авторські права.

– Розроблена методика впровадження системи в промислову експлуатацію: визначені складові організаційного та методичного забезпечень; надаються рекомендації по навчанню користувачів та обслуговуючого персоналу. Стисло надаються рекомендації користувачу щодо експлуатації системи, які ілюструються скриншотами (розділ 5 пояснювальної записки).

– В пояснювальній записці до КБР визначено та наведено: перелік скорочень, символів та спеціальних термінів ; список літератури, що була використана при розробці ПЗ системи та написанні пояснювальної записки.

Для більшої детального розуміння текстовий матеріал ілюстровано графічним матеріалом, що додається до даної пояснювальної записки в кількості 7 листів (формат А4).

Проведена дослідна експлуатація розробленої системи показала, що всі вимоги ТЗ виконані, розроблена система виконує всі означені ТЗ функції, що підтверджують позитивні відгуки користувачів організації, де проводилась дослідна експлуатація.

Отже, враховуючи вищеозначене, можемо зробити висновок, що робота по розробці програмного забезпечення системи проактивного захисту виконана в повному обсязі згідно ТЗ.

Додаток А
(обов'язковий)

Технічне завдання
Зміст

					КБР-123.21.0029.00.00.ТЗ			
Вим.	Арк.	№ документа	Підпис	Дата				
Розробив		Дзюбинський ОВ			***Програмне забезпечення системи кібербезпеки проактивного захисту на основі використання API-функцій***	Літ.	Аркуш	Аркушів
Перевірив		Савеленко О.К.				Б	1	6
						ЦНТУ КІ-18-ЗСК		
Н. Контр.		Гермак В.С.						
Затв.		Смірнов О.А..						

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку програмного забезпечення системи кібербезпеки проактивного захисту на основі використання API-функцій.

2 Підстава для розробки

Підставою для розробки служить завдання на кваліфікаційну бакалаврську роботу, видане на кафедрі програмного забезпечення (нак. № 37-02 від14.04.2021 року).

3 Мета та призначення розробки

Метою виконання КБР є розробка програмного забезпечення системи проактивного захисту на основі використання API-функцій.

4 Джерела розробки

Джерелом розробки цієї кваліфікаційної бакалаврської роботи є відносна стосовно теми література і існуючі системи 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:

– захист ПК від дій шкідливих програм;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс користувача.

5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинне 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

При реалізації ПЗ забезпечити виконання всіх правил, які стосуються стандартних викликів процедур, функцій, методів і форм, визначених відповідною технічною документацією на середовищем розробки.

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 18-20 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 персональному комп'ютері, працювати в ОС Windows та 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 стандартних бібліотек середовища Delphi, що працюють під управлінням ОС Windows.

5.8.1 Обладнання

Комп'ютер Intel Core™ i3 або сумісні з ним ПК з обсягом оперативної пам'яті не менше 1 Gb.

5.8.2 Мова програмування

Середовище Delphi.

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 та відповідних ДСТУ.

7 Перелік документів, що розробляються

– Структурна схема – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Блок-схеми алгоритмів роботи програми – 4 аркуші.
– Діаграма процесів – 1 аркуш.
– Пояснювальна записка – 84 аркуші.

10 Етапи розробки

10.1 Збір і обробка інформації по темі кваліфікаційної бакалаврської роботи. Постановка задачі на виконання дипломного проектування (складання ТЗ).

10.2 Проведення досліджень або експериментальних робіт для уточнення основних положень кваліфікаційної бакалаврської роботи.

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

10.4 Побудова схем взаємодії даних.

10.5 Створення прототипу ПЗ.

10.6 Віднаходження ПЗ, аналіз отриманих результатів.

10.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

11 Порядок контролю та приймання

11.1 Подання кваліфікаційної бакалаврської роботи на попередній захист 22.05.2021 р.

11.2 Подання кваліфікаційної бакалаврської роботи на захист__ .06.2021р.

Додаток Б
(обов'язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник кваліфікаційної бакалаврської роботи

_____ О.К.Савеленко

Програмне забезпечення системи кібербезпеки проактивного захисту на основі використання API-функцій

Лістинг програми

Код документу 12

Носій: DVD-диск

Загальна кількість аркушів: 37

Літера: РП

Кропивницький – 2021 року

proControl.dpr

```
//файл проекту реалізації користувацької бібліотеки
library proControl;

uses
  Windows, pSender;
(* ------------------------------------------------------------------------ *)
const
  THREAD_ALL_ACCESS      = $001F03FF;
  THREAD_SUSPEND_RESUME  = $00000002;
  TH32CS_SNAPTHREAD      = $04;
  MutexName              = '_BLACKPROCESSCONTROLMUTEX';
  proCapt                = 'Проактивний захист';

const
  faReadOnly  = $00000001 platform;
  faHidden    = $00000002 platform;
  faSysFile   = $00000004 platform;
  faVolumeID  = $00000008 platform;
  faDirectory = $00000010;
  faArchive   = $00000020 platform;
  faSymLink   = $00000040 platform;
  faAnyFile   = $0000003F;

type
  PFunctionRestoreData = ^ TFunctionRestoreData;
  TFunctionRestoreData = packed record
    Address:Pointer;
    val1:Byte;
    val2:DWORD;
   end;

  TTHREADENTRY32 = packed record
    dwSize: DWORD;
    cntUsage: DWORD;
    th32ThreadID: DWORD;
    th32OwnerProcessID: DWORD;
    tpBasePri: Longint;
    tpDeltaPri: Longint;
    dwFlags: DWORD;
   end;

  TFileName = type string;
  TSearchRec = record
    Time: Integer;
    Size: Integer;
    Attr: Integer;
    Name: TFileName;
    ExcludeAttr: Integer;
    FindHandle: THandle  platform;
    FindData: TWin32FindData  platform;
  end;

  LongRec = packed record
    case Integer of
      0: (Lo, Hi: Word);
      1: (Words: array [0..1] of Word);
      2: (Bytes: array [0..3] of Byte);
  end;

Function OpenThread(dwDesiredAccess: dword; bInheritHandle: bool;
                    dwThreadId: dword): dword; stdcall; external 'kernel32.dll';
```

```
function CreateToolhelp32Snapshot(dwFlags, th32ProcessID: DWORD): dword;
         stdcall;external 'kernel32.dll';

function Thread32First(hSnapshot: THandle; var lpte: TThreadEntry32): BOOL;
         stdcall;external 'kernel32.dll';

function Thread32Next(hSnapshot: THandle; var lpte: TThreadENtry32): BOOL;
         stdcall;external 'kernel32.dll';

var
  SystemCreateProcW : TFunctionRestoreData;
  SystemCreateProcA : TFunctionRestoreData;
  SystemCreateFileA : TFunctionRestoreData;
  SystemCreateFileW : TFunctionRestoreData;
  SystemCopyFileA   : TFunctionRestoreData;
  SystemDeleteFileA : TFunctionRestoreData;
  {!}
  hDll          : integer;
  LastFile      : string;
  StrangeList   : string;
  CopyesList    : string;
  StrangeFiles  : integer = 0;
  CopyesNumber  : integer = 0;
  ShowDialog    : function(App,Path: PChar; dType: byte): integer; stdcall;
  ShowDialogPro : function (App,Problem: PChar; dType: byte): integer; stdcall;
  CurrentModule : array [0..MAX_PATH] of char;

(* ------------------------------------------------------------------------ *)
procedure InstallMutex;
var
  M:THandle;
begin
//створюємо м'ютекс
  m:=CreateMutex(0,false,MutexName);
  //якщо не вдало, то вихід
  if m=0 then exit;
end;
(* ------------------------------------------------------------------------ *)
function ExpandFileName(const FileName: string): string;
//розширяємо ім'я файлу
var
  FName: PChar;
  Buffer: array[0..MAX_PATH - 1] of Char;
begin
//змінюємо довжину рядка
  SetString(Result, Buffer, GetFullPathName(PChar(FileName), SizeOf(Buffer),
    Buffer, FName));
end;
(* ------------------------------------------------------------------------ *)
procedure FindClose(var F: TSearchRec);
begin
  //якщо структура відкрита
  if F.FindHandle <> INVALID_HANDLE_VALUE then
  begin
    //закрити структуру
    Windows.FindClose(F.FindHandle);
    F.FindHandle := INVALID_HANDLE_VALUE;
  end;
end;

function FindMatchingFile(var F: TSearchRec): Integer;
//знаходимо відповідність файлу
var
  LocalFileTime: TFileTime;
```

```
begin
  with F do
  begin
    while FindData.dwFileAttributes and ExcludeAttr <> 0 do
      //якщо немає наступного файлу
      if not FindNextFile(FindHandle, FindData) then
      begin
      //вихід
        Result := GetLastError;
        Exit;
      end;
    //перетворюємо час файлу
    FileTimeToLocalFileTime(FindData.ftLastWriteTime, LocalFileTime);
    FileTimeToDosDateTime(LocalFileTime, LongRec(Time).Hi,
      LongRec(Time).Lo);
      //записуємо дані файла
    Size := FindData.nFileSizeLow;
    Attr := FindData.dwFileAttributes;
    Name := FindData.cFileName;
  end;
  Result := 0;
end;

function FindFirst(const Path: string; Attr: Integer;
//знайти по масці
  var  F: TSearchRec): Integer;
const
  faSpecial = faHidden or faSysFile or faVolumeID or faDirectory;
begin
//любий атрибут
  F.ExcludeAttr := not Attr and faSpecial;
  //пошук і по імені, і по даті
  F.FindHandle := FindFirstFile(PChar(Path), F.FindData);
  if F.FindHandle <> INVALID_HANDLE_VALUE then
  begin
    //якщо знайдений, то 0
    Result := FindMatchingFile(F);
    if Result <> 0 then FindClose(F);
  end else
    Result := GetLastError;
end;
(* ---------------------------------------------------------------------------- *)
function StrPas(const Str: PChar): string;
//перетворення в рядок
begin
  Result := Str;
end;

function LowCase(ACh:Char): Char;
//якщо заголовочні символи, то перетворимо їх на прописні символи
begin
 Result := ACh;
 if (ACh >= 'A') and (ACh <= 'Z') then inc(Result, 32);
end;

function LowerCase(AStr:string): string;
// якщо є заголовочні символи в рядку, то перетворимо їх на прописні символи
var
 LI:Integer;
begin
 Result:=AStr;
 for LI := 1 to Length(Result) do Result[LI] := LowCase(Result[LI]);
end;
```

```
function FileAgeEx(const FileName: string): Integer;
var
  Handle: THandle;
  FindData: TWin32FindData;
  LocalFileTime: TFileTime;
begin
  //знаходимо хендл файла
  Handle := FindFirstFile(PChar(FileName), FindData);
  //якщо знайдений, то
  if Handle <> INVALID_HANDLE_VALUE then
  begin
    //закриваємо хендл
    Windows.FindClose(Handle);
    if (FindData.dwFileAttributes and FILE_ATTRIBUTE_DIRECTORY) = 0 then
    begin
      //переведення часових даних по створенню файла
      FileTimeToLocalFileTime(FindData.ftLastWriteTime, LocalFileTime);
      if FileTimeToDosDateTime(LocalFileTime, LongRec(Result).Hi,
        LongRec(Result).Lo) then Exit;
    end;
  end;
  Result := -1;
end;

function FileExistsEx(const FileName: string): Boolean;
//якщо не -1, то існує
begin
  Result := FileAgeEx(FileName) <> -1;
end;

function DirectoryExists( const Name: String): Boolean;
//якщо true, то знайдено каталог
asm
   PUSH EBX
        PUSH     EAX
   PUSH SEM_NOOPENFILEERRORBOX or SEM_FAILCRITICALERRORS
   CALL SetErrorMode
   XCHG EBX, EAX
        CALL     GetFileAttributes
        INC      EAX
        JZ       @@exit
        DEC      EAX
        {$IFDEF PARANOIA} DB $24, FILE_ATTRIBUTE_DIRECTORY {$ELSE} AND AL,
FILE_ATTRIBUTE_DIRECTORY {$ENDIF}
        SETNZ    AL
@@exit:
   XCHG EAX, EBX
   PUSH EAX
   CALL SetErrorMode
   XCHG EAX, EBX
   POP  EBX
end;
(* ---------------------------------------------------------------------------- *)
function GetCurrentModulePath: String;

begin
  //повний шлях до модуля
  GetModuleFileName(Hinstance, CurrentModule, MAX_PATH);
  Result := CurrentModule;
end;
(* ---------------------------------------------------------------------------- *)
function ExtractFileExt(AFile: String): String;
//визначаємо розширення файлу з рядка повного шляху до нього
var
```

```
  I,J: integer;
begin
  if Length(AFile)<>0 then
  begin
    J := 0;
    for I := Length(AFile) downto 1 do
      if AFile[I] = '.' then
      begin
        J := I;
        break;
      end;
    Result:=Copy(AFile,J,MaxInt);
  end else Result:='';
end;

function ExtractFilePath(APath:string):string;
//визначаємо шлях до файлу з рядка повного шляху до нього
var
 LI,LJ:Integer;
begin
 if Length(APath)<>0 then
 begin
  LJ:=0;
  for LI:=Length(APath) downto 1 do
   if APath[LI]='\' then
   begin
    LJ:=LI;
    Break;
   end;
  Result:=Copy(APath,1,LJ);
 end else Result:='';
end;

function ExtractFileName(APath:string):string;
//визначаємо ім'я файлу з рядка повного шляху до нього
var
 LI,LJ:Integer;
begin
 if Length(APath)<>0 then
 begin
  LJ:=0;
  for LI:=Length(APath) downto 1 do
   if APath[LI]='\' then
   begin
    LJ:=LI;
    Break;
   end;
  Result:=Copy(APath,LJ+1,MaxInt);
 end else Result:='';
end;
(* ------------------------------------------------------------------------ *)
function GetStrCn(Str: String; Count: integer): string;
// відсічення рядка на задану кількість символів
var
  i: integer;
begin
  Result := '';
  for i := 1 to Count do
    Result := Result + Str[i];
end;
(* ------------------------------------------------------------------------ *)
function RestoreLongName(fn: string): string;
//-----------------------------------------------------------------
// Відновлює довгі імена файлів по відомих коротких (8.3)
```

```
// Як аргумент приймає повний або неповний (в т.ч. відносний)
// шлях до файлу, наприклад 'C:\windows\ Робочий~1\ І так далі~1.LNK' або
// '..\..\COMMON~1\BORLAN~1\BDE\BDEREA~1.TXT'. Розуміє мережеві імена.
// Повертає повний(!) шлях типу 'C:\windows\ Робочий стіл\ і так далі.lnk',
// 'C:\program Files\common Files\borland Shared\bde\bdereadme.txt',
// '\computer\resource\folder with long name\file with long name.ext'
//---------------------------------------------------------------------
  function LookupLongName(const filename: string): string;
    var
    sr: TSearchRec;
  begin
    if FindFirst(filename, faAnyFile, sr) = 0 then
      Result := sr.Name
    else
      Result := ExtractFileName(filename);
    FindClose(sr);
  end;
  function GetNextFN: string;
    var
    i: integer;
  begin
    Result := '';
    if Pos('\\', fn) = 1 then
    begin
      Result := '\\';
      fn := Copy(fn, 3, length(fn) - 2);
      i := Pos('\', fn);
      if i <> 0 then
      begin
        Result := Result + Copy(fn, 1, i);
        fn := Copy(fn, i + 1, length(fn) - i);
      end;
    end;
    i := Pos('\', fn);
    if i <> 0 then
    begin
      Result := Result + Copy(fn, 1, i - 1);
      fn := Copy(fn, i + 1, length(fn) - i);
    end
    else begin
      Result := Result + fn;
      fn := '';
    end;
  end;

  var
  name: string;
begin
  //повертає повне ім'я файлу
  fn := ExpandFileName(fn);
  Result := GetNextFN;
  repeat
    name := GetNextFN;
    Result := Result + '\' + LookupLongName(Result + '\' + name);
  until length(fn) = 0;
end;
(* ------------------------------------------------------------------------ *)
function TempDir: string;
//визначаємо шлях до каталогу тимчасових файлів
var
      buf: packed array [0..4095] of Char;
begin
      GetTempPath(4096,buf);
  //переводимо в рядок
```

```
      Result := StrPas(buf);
      Result := buf;
  //відновлення довгого імені по заданому рядку
  Result := RestoreLongName(Result);
end;

function SysDir: string;
//визначаємо шлях до системного каталогу
var
      buf: packed array [0..4095] of Char;
begin
      GetWindowsDirectory(buf,4096);
      Result:=StrPas(buf);
      Result:=buf+'\system32\';
end;

function WinDir: string;
//визначаємо шлях до директорії ОС
var
      buf: packed array [0..4095] of Char;
begin
      GetWindowsDirectory(buf,4096);
      Result:=StrPas(buf)+'\';
end;
(* ------------------------------------------------------------------------- *)
Procedure StopThreads;
//зупинити потоки
var
 h, CurrTh, ThrHandle, CurrPr: dword;
 Thread: TThreadEntry32;
begin
//отримуємо "значення" ідентифікатора потока
 CurrTh := GetCurrentThreadId;
 //ідентифікатор потоку
 CurrPr := GetCurrentProcessId;
 //отримуємо знімок процесів
 h := CreateToolhelp32Snapshot(TH32CS_SNAPTHREAD, 0);
 //якщо успішно, то
 if h <> INVALID_HANDLE_VALUE then
   begin
    //отримуємо інформацію про перший процес
    Thread.dwSize := SizeOf(TThreadEntry32);
    if Thread32First(h, Thread) then
    repeat
     if (Thread.th32ThreadID <> CurrTh) and (Thread.th32OwnerProcessID = CurrPr)
then
      begin
       //відкрити процес
       ThrHandle := OpenThread(THREAD_SUSPEND_RESUME, false,
Thread.th32ThreadID);
        if ThrHandle>0 then
          begin
          //призупинити процес
           SuspendThread(ThrHandle);
           CloseHandle(ThrHandle);
          end;
       end;
    until not Thread32Next(h, Thread);
   CloseHandle(h);
   end;
end;

Procedure RunThreads;
//відновити процес
```

```
var
 h, CurrTh, ThrHandle, CurrPr: dword;
 Thread: TThreadEntry32;
begin
//отримуємо "значення" ідентифікатора потока
 CurrTh := GetCurrentThreadId;
 //ідентифікатор потоку
 CurrPr := GetCurrentProcessId;
  //отримуємо знімок процесів
 h := CreateToolhelp32Snapshot(TH32CS_SNAPTHREAD, 0);
  //якщо успішно, то
 if h <> INVALID_HANDLE_VALUE then
   begin
   //отримуємо інформацію про перший процес
    Thread.dwSize := SizeOf(TThreadEntry32);
    if Thread32First(h, Thread) then
    repeat
     if (Thread.th32ThreadID <> CurrTh) and (Thread.th32OwnerProcessID = CurrPr)
then
      begin
       //відкрити процес
       ThrHandle := OpenThread(THREAD_SUSPEND_RESUME, false,
Thread.th32ThreadID);
       if ThrHandle>0 then
         begin
         //відновити процес
          ResumeThread(ThrHandle);
          CloseHandle(ThrHandle);
         end;
       end;
    until not Thread32Next(h, Thread);
   CloseHandle(h);
   end;
end;

(* Установка перехвату функції по її покажчику

  Procedure: Setcodehook
  Procaddress,                    покажчик на цільову функцію
  Newprocaddress:pointer;         покажчик на функцію - перехватчик
  Restoredata:pfunctionrestoredata  Покажчик на міст до старої функції
  Result:    boolean              успішність *)

function SetCodeHook(ProcAddress, NewProcAddress: pointer;
RestoreDATA:PFunctionRestoreData):boolean;
var
  OldProtect, JMPValue:DWORD;
begin
  Result:=False;
  //змінюємо атрибути захисту вказаного регіону
  //вказаного адресного простору
  //PAGE_EXECUTE_READWRITE -  Надає права читання/запису пам'яті і
  //                          виконання коду для вказаного регіону.

  if not VirtualProtect(ProcAddress,5,PAGE_EXECUTE_READWRITE,OldProtect) then
exit;
  JMPValue := DWORD(NewProcAddress) - DWORD(ProcAddress) - 5;
  RestoreDATA^.val1:= Byte(ProcAddress^);
  RestoreDATA^.val2:= DWORD(Pointer(DWORD(ProcAddress)+1)^);
  RestoreDATA^.Address:=ProcAddress;
  byte(ProcAddress^):=$E9;
  DWORD(Pointer(DWORD(ProcAddress)+1)^):=JMPValue;
  Result:=VirtualProtect(ProcAddress,5,OldProtect,OldProtect);
end;
```

```
{-------------------------------------------------------------------------
   Установка перехвату функції по її імені

  Procedure: SetProcedureHook

  ModuleHandle:HMODULE;             Хендл модуля, в якому знаходиться цільова
функція
  ProcedureName:PChar;              Ім'я процедури
  NewProcedureAddress:Pointer;      Покажчик на процедуру - перехватчик
  RestoreDATA:PFunctionRestoreData  Покажчик на міст до старої функції

  Result:    Boolean                Успішність установки перехвату
-------------------------------------------------------------------------}
function
SetProcedureHook(ModuleHandle:HMODULE;ProcedureName:PChar;NewProcedureAddress:Po
inter;
   RestoreDATA:PFunctionRestoreData):Boolean;
var
  ProcAddress:Pointer;
begin
  ProcAddress:=GetProcAddress(ModuleHandle,ProcedureName);
  Result:=SetCodeHook(ProcAddress,NewProcedureAddress,RestoreDATA);
end;

(* Зняття перехвату функції
  Procedure: Unhookcodehook
  Restoredata:pfunctionrestoredata адреса моста до старої функції
  Result:    Boolean  *)

function UnHookCodeHook(RestoreDATA:PFunctionRestoreData):Boolean;
var
  ProcAddress:Pointer;
  OldProtect,JMPValue:DWORD;
begin
  Result:=False;
  ProcAddress:=RestoreDATA^.Address;
  if not VirtualProtect(ProcAddress,5,PAGE_EXECUTE_READWRITE,OldProtect) then
exit;
  Byte(ProcAddress^):=RestoreDATA^.val1;
  DWORD(Pointer(DWORD(ProcAddress)+1)^):=RestoreDATA^.val2;
  Result:=VirtualProtect(ProcAddress,5,OldProtect,OldProtect);
end;

(* Закрити процес по його PID - ідентифікатору *)
procedure TerminateProcessEx(PID: Integer);
var
 hProcess: integer;
begin
  hProcess := OpenProcess(PROCESS_TERMINATE, false, PID);
  if hProcess > 0 then
  begin
    TerminateProcess(hProcess, 0);
    CloseHandle(hProcess);
  end;
end;
(* ------------------------------------------------------------------------- *)
//показ діалогу з dll бібліотеки(завантаження/вивантаження) і
//обробка програмного додатку, стан якого контролюється

function ShowProDialog(Path: String; dType: byte) : integer;
var
  PID: dWord;
begin
```

```
  hDll :=
LoadLibrary(PChar(ExtractFilePath(GetCurrentModulePath)+'proDialog.dll'));
  if hDll <> 0 then begin
    try
    //виведення вікна - підозріла активність
      @ShowDialogPro:=GetProcAddress(hDll, 'ShowDialogPro');
      //читаємо дані вибору користувача
      Result :=
ShowDialogPro(pChar(ExtractFileName(ParamStr(0))),PChar(Path),dType);
      case Result of
      //закрити процес
        3 : begin
              PID := GetCurrentProcessId;
              if PID <> 0 then begin
                TerminateProcessEx(PID);
              end;
            end;
       //відправка повідомлення для внесення додатку до білого списку додатків
        9 : sendstring(proCapt,PChar('#LOG2#'+ParamStr(0)));
      end;
    finally
    //очищаємо пам'ять
      FreeLibrary(hDll);
    end;
  end;
end;

function ShowCanRunDialog(Path: String) : integer;
begin
  hDll :=
LoadLibrary(PChar(ExtractFilePath(GetCurrentModulePath)+'proDialog.dll'));
  if hDll <> 0 then begin
    try
    //виведення вікна - запуск додатку
      @ShowDialog:=GetProcAddress(hDll, 'ShowDialog');
      Result := ShowDialog(pChar(ExtractFileName(ParamStr(0))),PChar(Path),0);
    finally
    //очистка пам'яті
      FreeLibrary(hDll);
    end;
  end;
end;
(* ------------------------------------------------------------------------- *)
function CanRun(FName, CmdLine: String): boolean;
  var
  i: integer;
  j: integer;
  S: string;
  cmd : String;
begin
  Result := True;
    case ShowCanRunDialog(FName) of
      -1 : Result := false;
      //білий список
       9 : begin
             Result := true;
             sendstring(proCapt,PChar('#LOG2#'+FName));
             exit;
           end;
       //дозволений одноразовий запуск
       6 : begin
             Result := true;
             sendstring(proCapt,PChar('#LOG1#'+FName));
             exit;
```

```
            end;
        //запуск додатку не дозволений
        3 : begin
              Result := False;
              sendstring(proCapt,PChar('#LOG3#'+FName));
              exit;
            end;
    end;

end;
(* ---------------------------------------------------------------------------- *)
function ExistsInList(Str: String; List: String) : Boolean;
//перевіряємо присутність шаблону в рядку
//якщо присутній, то true
begin
  Result := False;
  if pos(Str,List) <> 0 then Result := true;
end;

function DeleteInvalidInList(var List: String) : Boolean;
//якщо файл існує, то true
const
  Return = #13#10;
var
  i: integer;
  tmp,tmp1 : String;
begin
  Result := False;
  i    := 1;
  tmp1 := List+Return;
  List := '';
  while i <> 0 do begin
    i := pos(Return,tmp1);
    tmp := copy(tmp1,1,i-1);
    delete(tmp1,1,i+1);
    if FileExistsEx(tmp) then List := List + tmp + Return;
  end;
  Result := true;
end;

function CanDeleteFile(FileName: String) : boolean;
//видалення з системної директорії (так чи ні?)
var
  ClearName: String;
  ext      : String;
  DirName  : String;
  S        : String;
begin
  Result := False;
  //відновлення скорочених імен та розбір шляху
  //файла - відокремлюємо назву додатка і його шлях
  //визначення розширення файлу
  LastFile  := RestoreLongName(FileName);
  Result    := false;
  ClearName := LowerCase(ExtractFileName(FileName));
  Ext       := LowerCase(ExtractFileExt(FileName));
  DirName   := LowerCase(ExtractFilePath(FileName));
  ClearName := GetStrCn(ClearName,length(ClearName)-length(ext));
  delete(dirname,1,(length(DirName)-1)-Length(ClearName));
  //якщо системна директорія
  if (LowerCase(ExtractFilePath(FileName)) = LowerCase(WinDir))  or
     (LowerCase(ExtractFilePath(FileName)) = LowerCase(TempDir)) or
     (LowerCase(ExtractFilePath(FileName)) = LowerCase(SysDir))  then begin
     //відправка повідомлення для статистики
```

```
       sendstring(proCapt,PChar('#LOG6#'+FileName));
       //виведення діалогу "Підозріла активність"
       ShowProDialog(FileName,5);
     end;
end;

function CanCreateFile(FileName: String) : boolean;
//обробка при створенні або копіюванні файла в системній директорії
var
  ClearName: String;
  ext      : String;
  DirName  : String;
begin
 //відновлення скорочених імен та розбір шляху
  //файла - відокремлюємо назву додатка і його шлях
  //визначення розширення файлу
  LastFile  := RestoreLongName(FileName);
  Result    := false;
  ClearName := LowerCase(ExtractFileName(FileName));
  Ext       := LowerCase(ExtractFileExt(FileName));
  DirName   := LowerCase(ExtractFilePath(FileName));
  ClearName := GetStrCn(ClearName,length(ClearName)-length(ext));
  delete(dirname,1,(length(DirName)-1)-Length(ClearName));
  //якщо розширення файла відноситься до запускаємого, то
  if (ext = '.exe') or (ext = '.scr') or (ext = '.bat') or (ext = '.cmd') or
(ext = '.com') then
  //якщо системна директорія
  if (LowerCase(ExtractFilePath(FileName)) = LowerCase(WinDir))  or
     (LowerCase(ExtractFilePath(FileName)) = LowerCase(TempDir)) or
     (LowerCase(ExtractFilePath(FileName)) = LowerCase(SysDir))  then begin
     //відправка повідомлення в файл статистики
       sendstring(proCapt,PChar('#LOG5#'+FileName));
       //виведення діалогу "Підозріла активність"
       //спроба копіювання в систему
       ShowProDialog(FileName,3);
     end;
   //якщо файли запускаємі
  if (ext = '.exe') or (ext = '.scr') or (ext = '.bat') or (ext = '.cmd') or
(ext = '.com') then
  if (DirName = ClearName+'\') or
(DirectoryExists(ExtractFilePath(FileName)+ClearName)) then begin
    if not ExistsInList(FileName, StrangeList) then begin
      inc(StrangeFiles);
      StrangeList := StrangeList + #13#10 + FileName;
    end;
    //якщо створено більше 5 копій
    if StrangeFiles > 5 then begin
    //відправка повідомлення до файлу статистики
      sendstring(proCapt,PChar('#LOG4#'+ParamStr(0)));
       //виведення діалогу "Підозріла активність"
       //можливо P2P вірус
      ShowProDialog(StrangeList,4);
      Result := true;
      Exit;
    end;
  end;

  if Length(DirName) = 3 then
    if LowerCase( ExtractFileName(FileName)) = 'autorun.inf' then begin
   //відправка повідомлення до файлу статистики
      sendstring(proCapt,PChar('#LOG7#'+FileName));
    //виведення діалогу "Підозріла активність"
    //створення підозрілого файлу
      ShowProDialog(FileName,0);
```

```
    end else
    if LowerCase( ExtractFileName(FileName)) = 'explorer.exe' then begin
      //відправка повідомлення до файлу статистики
      sendstring(proCapt,PChar('#LOG7#'+FileName));
      //виведення діалогу "Підозріла активніть"
      //створення підозрілого файлу
      ShowProDialog(FileName,6);
    end;
end;

function CanCopySelf(FileName: String) : boolean;
//обробка при копіюванні додатку
var
  ClearName: String;
  ext      : String;
  DirName  : String;
begin
 //відновлення скорочених імен та розбір шляху
  //файла - відокремлюємо назву додатка і його шлях
  //визначення розширення файлу
  LastFile  := RestoreLongName(FileName);
  Result    := false;
  ClearName := LowerCase(ExtractFileName(FileName));
  Ext       := LowerCase(ExtractFileExt(FileName));
  DirName   := LowerCase(ExtractFilePath(FileName));
  ClearName := GetStrCn(ClearName,length(ClearName)-length(ext));
  delete(dirname,1,(length(DirName)-1)-Length(ClearName));

  DeleteInvalidInList(CopyesList);
  if not ExistsInList(FileName,CopyesList) then begin
    CopyesList := CopyesList + FileName + #13#10;
    Inc(CopyesNumber);
  end;
  //якщо більше 7 копій, то
  if CopyesNumber > 7 then begin
  //відправка повідомлення
    sendstring(proCapt,PChar('#LOG4#'+ParamStr(0)));
     //виведення діалогу "Підозріла активніть"
      //можливо P2P вірус
    ShowProDialog(CopyesList,1);
    Exit;
  end;

  if (DirName = ClearName+'\') or
(DirectoryExists(ExtractFilePath(FileName)+ClearName)) then begin
    if not ExistsInList(FileName, StrangeList) then begin
      inc(StrangeFiles);
      StrangeList := StrangeList + FileName + #13#10;
    end;
    //якщо більше 5 копій
    if StrangeFiles > 5 then begin
      //відправка повідомлення в файл статистики
      sendstring(proCapt,PChar('#LOG4#'+ParamStr(0)));
      //виведення діалогу "Підозріла активніть"
      //можливо P2P вірус
      ShowProDialog(StrangeList,4);
      Result := true;
      Exit;
    end;
  end;
  //якщо в системну директорію
  if (LowerCase(ExtractFilePath(FileName)) = LowerCase(WinDir))  or
     (LowerCase(ExtractFilePath(FileName)) = LowerCase(TempDir)) or
     (LowerCase(ExtractFilePath(FileName)) = LowerCase(SysDir))  then begin
```

```
       //відправка повідомлення
       sendstring(proCapt,PChar('#LOG5#'+FileName));
      //виведення діалогу "Підозріла активніть"
      //спроба копіювання в системну директорію
       ShowProDialog(FileName,2);
     end;
end;
(* ------------------------------------------------------------------------ *)
//обробка пастки на
//відкриття файлу чи каталогу
function CreateFileANext(lpFileName: PAnsiChar; dwDesiredAccess, dwShareMode:
DWORD;
                         lpSecurityAttributes: PSecurityAttributes;
dwCreationDisposition, dwFlagsAndAttributes: DWORD;
                         hTemplateFile: THandle): THandle; stdcall;
var
  isNewFile: Boolean;
begin
  //наявність файла
  if not FileExistsEx(lpFileName) then isNewFile := true else
  begin
    isNewFile := false;
  end;
  //знімаємо перехват функції
  UnHookCodeHook(@SystemCreateFileA);

  try
    //виконуємо функцію відкриття
    result := CreateFileA(lpFileName,dwDesiredAccess,dwShareMode,
                          lpSecurityAttributes,dwCreationDisposition,
                          dwFlagsAndAttributes,hTemplateFile);

  except
  end;
   //якщо файл є і він створюється
  if FileExistsEx(lpFileName) and isNewFile then begin
    //обробляємо файл
    CanCreateFile(lpFileName);
    LastFile := RestoreLongName(lpFileName);
  end;
    //ставимо пастку
    SetCodeHook(SystemCreateFileA.Address,@CreateFileANext,@SystemCreateFileA);
end;
(* ------------------------------------------------------------------------ *)
//обробка пастки на
//відкриття файлу чи каталогу
function CreateFileWNext(lpFileName: PWideChar; dwDesiredAccess, dwShareMode:
DWORD;
                         lpSecurityAttributes: PSecurityAttributes;
dwCreationDisposition, dwFlagsAndAttributes: DWORD;
                         hTemplateFile: THandle): THandle; stdcall;
var
  isNewFile: Boolean;
begin
  //визначаємо: створюємо файл чи ні
  if not FileExistsEx(lpFileName) then isNewFile := true else
  begin
    isNewFile := false;
  end;
  //знімаємо пастку
  UnHookCodeHook(@SystemCreateFileW);

  try
  //виконуємо команду
```

```
    result := CreateFileW(lpFileName,dwDesiredAccess,dwShareMode,
                          lpSecurityAttributes,dwCreationDisposition,
                          dwFlagsAndAttributes,hTemplateFile);

  except
  end;
  //якщо файл є і він новий, то
  if FileExistsEx(lpFileName) and isNewFile then begin
  //обробка події створення через функцію захисту
    CanCreateFile(lpFileName);
    LastFile := RestoreLongName(lpFileName);
  end;
    //відновлюємо пастку
    SetCodeHook(SystemCreateFileW.Address,@CreateFileWNext,@SystemCreateFileW);

end;
(* ------------------------------------------------------------------------ *)
function DeleteFileCallBack(lpFileName: PChar): BOOL; stdcall;
//обробка пастки, що виникає при команді видалення файлу
begin
  //знімаємо пастку
  UnHookCodeHook(@SystemDeleteFileA);
  //обробка системою проактивного захисту
  CanDeleteFile(lpFileName);
  try
    //видалення файлу
    result := Windows.DeleteFile(lpFileName);
  except
  end;
  //відновлення пастки
  SetCodeHook(SystemDeleteFileA.Address,@DeleteFileCallBack,@SystemDeleteFileA);
end;
(* ------------------------------------------------------------------------ *)
//обробка пастки, що виникає при обробці системної функції копіювання файлу
function CopyFileANext(lpExistingFileName, lpNewFileName: PChar; bFailIfExists:
BOOL): BOOL; stdcall;
begin
  //знімаємо пастку
  UnHookCodeHook(@SystemCopyFileA);
  try
    //виконуємо копіювання
    Result := CopyFile(lpExistingFileName,lpNewFileName,bFailIfExists);
  except
  end;

  if LowerCase(RestoreLongName(lpExistingFileName)) = LowerCase(ParamStr(0))
then
  begin
    //обробка системою проактивного захисту
    CanCopySelf(RestoreLongName(lpNewFileName));
  end else CanCreateFile(RestoreLongName(lpNewFileName));
  //встановлення пастки
  SetCodeHook(SystemCopyFileA.Address,@CopyFileANext,@SystemCopyFileA);
end;
(* ------------------------------------------------------------------------ *)
//обробка пастки на системну подію відкриття процесу
function CreateProcessWCallback(appName, cmdLine: pwidechar;
                                processAttr, threadAttr: PSecurityAttributes;
                                inheritHandles: bool; creationFlags: dword;
                                environment: pointer; currentDir: pwidechar;
                                const startupInfo: TStartupInfo;
                                var processInfo: TProcessInformation) : bool;
stdcall;
var appNameA,cmdLineA: string;
```

```
begin
    appNameA:=appName;
    cmdLineA:=cmdLine;
    //обробка системою проактивного захисту
    if not CanRun(appName, cmdLine) then begin
      Result := true;
      Exit;
    end;
    //знімаємо пастку
    UnHookCodeHook(@SystemCreateProcW);
    try
    //виконуємо системну команду
      result := CreateProcessW(appName, cmdLine, processAttr, threadAttr,
                               inheritHandles, creationFlags,
                               environment, currentDir,
                               startupInfo, processInfo);
    except
    end;
    //встановлюємо пастку

SetCodeHook(SystemCreateProcW.Address,@CreateProcessWCallback,@SystemCreateProcW
);
end;
(* ------------------------------------------------------------------------ *)
//обробка пастки на системну подію відкриття процесу
function CreateProcessACallback(appName, cmdLine: pchar;
                                processAttr, threadAttr: PSecurityAttributes;
                                inheritHandles: bool; creationFlags: dword;
                                environment: pointer; currentDir: pchar;
                                const startupInfo: TStartupInfo;
                                var processInfo: TProcessInformation) : bool;
stdcall;
begin
    //обробка системою проактивного захисту
    if not CanRun(appName, cmdLine) then begin
      Result := true;
      Exit;
    end;
    //знімаємо пастку
    UnHookCodeHook(@SystemCreateProcA);
    try
    //виконуємо системну команду
      result := CreateProcessA(appName, cmdLine, processAttr, threadAttr,
                               inheritHandles, creationFlags,
                               environment, currentDir,
                               startupInfo, processInfo);
    except
    end;
    // встановлюємо пастку

SetCodeHook(SystemCreateProcA.Address,@CreateProcessACallback,@SystemCreateProcA
);
end;
(* ------------------------------------------------------------------------ *)
(* ------------------------------------------------------------------------ *)
{$R *.res}
(* ------------------------------------------------------------------------ *)
begin
  StopThreads();
  InstallMutex;
    //встановлення перехвату по імені функцій

    //на створення нового процесу
```

```
SetProcedureHook(GetModuleHandle('kernel32.dll'),'CreateProcessW',@CreateProcess
WCallback,@SystemCreateProcW);

SetProcedureHook(GetModuleHandle('kernel32.dll'),'CreateProcessA',@CreateProcess
ACallback,@SystemCreateProcA);
    //на створення або відкриття файлу чи каталогу

SetProcedureHook(GetModuleHandle('kernel32.dll'),'CreateFileA',@CreateFileANext,
@SystemCreateFileA);

SetProcedureHook(GetModuleHandle('kernel32.dll'),'CreateFileW',@CreateFileWNext,
@SystemCreateFileW);
    //на копіювання файлу з одного місця на друге

SetProcedureHook(GetModuleHandle('kernel32.dll'),'CopyFileA',@CopyFileANext,@Sys
temCopyFileA);
    //на видалення файлу

SetProcedureHook(GetModuleHandle('kernel32.dll'),'DeleteFileA',@DeleteFileCallBa
ck,@SystemDeleteFileA);
  RunThreads();
end.
```

PC.dpr

```
program PC;
//файл проекту системи проактивного захисту (моніторинг процесів)
uses
  Forms,
  Unit1 in 'Unit1.pas' {Form1};

{$R *.RES}

begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

Unit1.pas

```
{$S-,R-,B-}
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Menus,
  ExtCtrls, Grids, StdCtrls,
  Forms, Dialogs, tlHelp32,  FileCtrl, ComCtrls,
  shellApi, psApi,
  { Для автозавантаження:} activeX,shlObj,comobj;

const
info=
'Модуль відображення процесів'#13#10+
'Ловить вірусні процеси, які були помічені як шкідливі і видаляє їх.'#13#10+
'Можна натиснути праву кнопку миші на процесі і вказати його тип:'#13#10+
'шкідливий - відразу видаляти'#13#10+
'корисний - ніколи не видаляти'#13#10+
'Ctrl - не видаляти тільки, якщо натиснута Ctrl при запуску'#13#10+
'Можна використовувати для заборони запуску сторонніми'#13#10+
'або для заборони запуску 2-ої копії'#13#10#13#10+
'Усі права захищені Copyright 2021 '#13#10;

MyTrayIcon = WM_USER + 1;

type
  TForm1 = class(TForm)
    Timer1: TTimer;
    StringGrid1: TStringGrid;
    Label1: TLabel;
    Label2: TLabel;
    PopupMenu1: TPopupMenu;
    N1: TMenuItem;
    N2: TMenuItem;
    N3: TMenuItem;
    N4: TMenuItem;
    N5: TMenuItem;
    N6: TMenuItem;
    N7: TMenuItem;
    N8: TMenuItem;
    N9: TMenuItem;
    N10: TMenuItem;
    N11: TMenuItem;
    N14: TMenuItem;
    N15: TMenuItem;
    N16: TMenuItem;
    Ctrl: TMenuItem;
    N17: TMenuItem;
    N12: TMenuItem;
    HIGHPRIORITYCLASS1: TMenuItem;
    IDLEPRIORITYCLASS1: TMenuItem;
    NORMALPRIORITYCLASS1: TMenuItem;
    REALTIMEPRIORITYCLASS1: TMenuItem;
    procedure Timer1Timer(Sender: TObject);
    procedure StringGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
      R: TRect; State: TGridDrawState);
    procedure FormCreate(Sender: TObject);
    procedure FormCloseQuery(Sender: TObject; var CanClose: Boolean);
    procedure StringGrid1MouseDown(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure StringGrid1KeyDown(Sender: TObject; var Key: Word;
      Shift: TShiftState);
```

```
    procedure FormResize(Sender: TObject);
    procedure N1Click(Sender: TObject);
    procedure N2Click(Sender: TObject);
    procedure N3Click(Sender: TObject);
    procedure N4Click(Sender: TObject);
    procedure N6Click(Sender: TObject);
    procedure N8Click(Sender: TObject);
    procedure N9Click(Sender: TObject);
    procedure N10Click(Sender: TObject);
    procedure N11Click(Sender: TObject);
    procedure N14Click(Sender: TObject);
    procedure N15Click(Sender: TObject);
    procedure PopupMenu1Popup(Sender: TObject);
    procedure N16Click(Sender: TObject);
    procedure CtrlClick(Sender: TObject);
    procedure N17Click(Sender: TObject);
    procedure HIGHPRIORITYCLASS1Click(Sender: TObject);
    procedure NORMALPRIORITYCLASS1Click(Sender: TObject);
    procedure IDLEPRIORITYCLASS1Click(Sender: TObject);
    procedure REALTIMEPRIORITYCLASS1Click(Sender: TObject);
    procedure CreateProcessTree();
    procedure AddChild(до: integer);
    procedure AddWindow(_hwnd: longint);//pr: cardinal);//
    procedure tvwProcDblClick(Sender: TObject);
    procedure tvwProcClick(Sender: TObject);
    procedure btnRecreateClick(Sender: TObject);
    procedure btnPriorClick(Sender: TObject);

  protected
  procedure WMGetSysCommand(var Message :TMessage); message WM_SYSCOMMAND;

  private
    { Private declarations }
    procedure MTIcon(var a: TMessage); message MyTrayIcon;
  public
    { Public declarations }
  end;
 //описуємо структуру процесу
type
tm= record
 //назва процесу
 n   :string;
 //шлях до exe - файлу
 path:string;
 //активність процесу
 a:boolean;
 //видалити
 k:boolean;
 //w-шкідливий p-корисний n-не знаю c-ctrl
 t:char;
 //номер по порядку у однакових
 q:byte;
 //дата появи процесу
 d:integer;
 //<>0 - встановити пріоритет
 pr:dWord;
end;

var
Form1: TForm1;
//одномірний масив структури процесу
m:array of tm;
p:array of tm;
p1:tm;
```

```
mf: array of char;
f:file;
t:textFile;
hn,wn,tn,ln,rrr:integer;
qm:integer=0;
qp:integer=0;
//попередне значення qm
qmp:integer=0;
//виділений рядок
sr:integer=0;
//координати меню
xm,ym:integer;
//колонка сортування
aCol:integer;
wTimer:boolean=false;
nr:byte=0;
td:string;
s:string;
//exeName;
en:string;
ini:string;
//колонка сортування
sCol:byte;
//сортування по спаданню
su:boolean;
// >0 було запущене видалення, щоб не додавати
zKill:byte=0;
//було запущене видалення через меню
zKillM:boolean=false;
NID: TNotifyIconData;
// видаляти усі нові
uwn:boolean=false;
// нові вважати шкідливими
nw :boolean=false;
// Ctrl не натиснуто
nCtrl:boolean=true;

// реалізація модуля
implementation

{$R *.DFM}
PROCEDURE wHide;
// згорнути у трей
begin
Application.ShowMainForm := false;
//дескриптор структури
with NID do begin
  //розмір запису
  cbSize := SizeOf(TNotifyIconData);
  //дескриптор вікна
  Wnd := form1.handle;
  //номер піктограми
  uId := 1;
  //включаємо усі прапорці
  uFlags := NIF_ICON or NIF_MESSAGE or NIF_TIP;
  //ідентифікатор повідомленння - відправляється при знаходженні
  //маніпулятора мишки в зоні піктограми
  uCallbackMessage := MyTrayIcon;
  //дескриптор піктограми
  hIcon := Application.Icon.Handle;
  //зміст спливаючого повідомлення
  szTip := 'Стежу за процесами';
end;
//додаємо піктограму в область трей
```

```
Shell_NotifyIcon(NIM_ADD, @NID);
end;

procedure TForm1.WMGetSysCommand(var Message : TMessage) ;
//обробка повідомлення при мінімізації вікна
begin
if (Message.wParam = SC_MINIMIZE) then begin
// Application.Minimize;
 Shell_NotifyIcon(NIM_ADD, @NID);
 visible:=false;
end
else inherited;
end;

PROCEDURE uqp;
begin
//збільшуємо лічильник на 1
inc(qp);
if qp>=length(p) then setLength(p,length(p)+100);
end;

FUNCTION ok(qs:integer):string;
//визначаємо вибраний (активний процес на формі)
var m10:byte;
begin
m10:=qs mod 10;
if (qs in [11..19]) or (m10 in [0,5..9]) then ok:='ов' else
if m10=1 then ok:='' else ok:='а';
end;

PROCEDURE wl1;
//вивід на форму повідомлення про кількість процесів
begin
form1.label1.caption:='Помню '+intToStr(qp)+' процесс'+ok(qp);
end;

PROCEDURE pAlign;
//встановлюємо розмір структури процесів stringGrid
begin
with form1 do begin
  stringGrid1.height:=clientHeight-stringGrid1.top;
  stringGrid1.width :=clientWidth;
end;
end;

function KillProcess(PROCESSID: DWORD): boolean;
//видалити процес
var
hProcess: THandle;
hToken: THandle;
Priv,PrivOld: TOKEN_PRIVILEGES;
cbPriv: DWORD;
dwError: DWORD;
begin
Result:=False;
//визначаємо по ідентиіфкатору дескриптор процесу
hProcess:=OpenProcess(PROCESS_TERMINATE,false,ProcessID);
if hProcess = 0 then
 begin
  cbPriv:=SizeOf(PrivOld);
  // Для Win2k. Отримуємо токен процесу
  OpenThreadToken(GetCurrentThread,TOKEN_QUERY or
TOKEN_ADJUST_PRIVILEGES,false,hToken);
```

```
  OpenProcessToken(GetCurrentProcess,TOKEN_QUERY or
TOKEN_ADJUST_PRIVILEGES,hToken);
  //дозволяємо привілеї
  Priv.PrivilegeCount:=1;
  Priv.Privileges[0].Attributes:=SE_PRIVILEGE_ENABLED;
  //Отримуємо LUID (locally unique identifier) привілеї
  LookupPrivilegeValue(nil,'SeDebugPrivilege',Priv.Privileges[0].Luid);
  //Додаемо привілегію до вибраного процесу
  AdjustTokenPrivileges(hToken,false,Priv,SizeOf(Priv),PrivOld,cbPriv);
  //отримуємо дескриптор процесу для його завершення
  hProcess:=OpenProcess(PROCESS_TERMINATE,false,ProcessID);
  dwError:=GetLastError;
  cbPriv:=0;
  //Видаляємо привілей
  AdjustTokenPrivileges(hToken,false,PrivOld,SizeOf(PrivOld),nil,cbPriv);
  CloseHandle(hToken);
 end;
 //Завершуємо процес
 TerminateProcess(hProcess,$FFFFFFFF);
 CloseHandle(hProcess);
 Result:=true;
end;

FUNCTION CtrlDown: Boolean;
//якщо натиснута клавіша  Ctrl,  то true
begin
result:=GetKeyState(VK_CONTROL)< 0;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
//обробка події таймера
var
handler:thandle;
data:tagPROCESSENTRY32;
 pID,dwError:DWORD;
 hProc:THandle;
 Buf:Array [0..255] Of Char;
 i:integer;

Function obr:string;
var
s,path:string;
q,j:integer;
proc:thandle;
//запущено зараз
zs:boolean;

begin
s:=trim(ansiLowerCase(data.szExeFile));
//отримуємо хендл
hProc:=OpenProcess(PROCESS_ALL_ACCESS,True,data.th32ProcessID);
//отримати повний шлях до файлу
GetModuleFileNameEx(hProc,0,Buf,256);
path:=ansiLowerCase(Buf);

if (path<>'') and(path[1]<>'?')
   //поділяємо повний шлях на складові
   then path:=extractFileDir(path)
   else path:=extractFileDir(s);
s:=extractFileName(s); //для 98
q:=0;

for j:=1 to qm do if s=m[j].n then inc(q);
//збільшення значення лічильника
```

```
inc(qm);
if qm>=length(m) then setLength(m,length(m)+100);
//заповнюємо структуру
m[qm].q:=q;
m[qm].n:=s;
m[qm].t:=' ';
//добавка нових  процесів в структуру контроля
j:=1;
while (j<=qp) and((s<>p[j].n)or(q<>p[j].q)) do inc(j);

zs:=(qm>qmp) and(j<=qp);
if zs and(p[j].t='w') and CtrlDown then p[j].t:=' ';
//якщо реальних процесів більше, чим контролюємих, то
if j>qp then begin
  uqp;
  //заповнюємо структуру
  p[qp].q:=q;
  p[qp].n:=s;
  p[qp].t:=' ';
  //якщо умова виконується, то позначаємо процес, як шкідливий
  if nw and nCtrl then p[qp].t:='w';
  //заповнюємо структуру далі
  //запис дати
  p[qp].d:=DateTimeToFileDate(now);
  p[j].path:='';
  p[j].k:=uwn and nCtrl;
  p[j].pr:=0;
end;

p[j].a:=true;
//встановити пріоритет
if p[j].pr<>0 then begin
  SetPriorityClass(hProc,p[j].pr);
  p[j].pr:=0;
end;
//якщо шлях до файлу не вказано, то прийняти з структури
if path<>'' then p[j].path:=path;
//якщо натиснута Ctrl - значення структури процесу
//покажчика видалити, переводимо в true
if zs and (p[j].t='c') and nCtrl then p[j].k:=true;
//якщо процес визначений як шкідливий, то видаленння
if (p[j].t='w') or p[j].k then KillProcess(data.th32ProcessID);
//покажчика видалити, переводимо в false
p[j].k:=false;
end;

//*********************************************************
BEGIN
if wTimer then exit;
wTimer:=true;

if nr=1 then begin
  StringGrid1.RowCount:=qp+1;
  StringGrid1.rePaint;

  if hn>0 then begin height:=hn; hn:=0; width:=wn end;
end;

handler:=createtoolhelp32snapshot(TH32CS_SNAPALL,0);
data.dwSize:=sizeOf(data);
if handler>0 then begin
if process32first(handler,data) then begin
  qm:=0;
  nCtrl:=not CtrlDown;
```

```
  for i:=1 to qp do p[i].a:=false;
  repeat
    obr;
  until not process32next(handler,data);
  qmp:=qm;
end;
CloseHandle(handler);
end;

if qp>0 then begin wl1; nr:=1; end;

wTimer:=false;
end;


PROCEDURE usr(aRow:integer);
begin
//обробка мітки на формі, що відображає вибраний процес
with form1.label2 do
if (aRow>0) and(aRow<=qp) then begin
  sr:=aRow;
  //об*єкт видимий
  visible:=true;
  //виводимо ім*я процесу на мітку на форму
  caption:=p[sr].n;
end
else caption:='';
end;

PROCEDURE kc(r,g,b:integer);
var c,z:integer; m:array[1..4] of byte absolute c;

Function o(z:integer):integer;
begin
//обмеження на вихід за межі кольору
if z>255 then o:=255 else if z<0 then o:=0 else o:=z;
end;

begin
//корегування кольору
c:=form1.stringGrid1.canvas.brush.color;
m[1]:=o(m[1]+r);
m[2]:=o(m[2]+g);
m[3]:=o(m[3]+b);
form1.stringGrid1.canvas.brush.color:=c;
end;

procedure TForm1.StringGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
  R: TRect; State: TGridDrawState);
var
s:string;
cw:integer;
begin
if qp=0 then exit;
with StringGrid1.Canvas do begin
pen.color:=$C0C0CC;
if (aRow<qp) then r:=rect(r.left,r.top,r.right,r.bottom+1);
if (aCol<stringGrid1.colCount-1) then r:=rect(r.left,r.top,r.right+1,r.bottom);
s:='';
if aRow=0 then begin
  brush.color:=$B0C2E0;//$B0C0D0;
  pen.color:=$A0A0AA;
  font.color:=$F0FFFF;
  rectangle(r);
```

```
  case aCol of
    0:s:='Процес';
    1:s:='Тип';
    2:s:='Стан';
    3:s:='З'явилось';
    4:s:='Nпп';
    5:s:='Шлях до файлу';
  end;
end
else begin
  font.color:=$8088A8;
  if gdSelected in State then begin
    brush.color:=$D8E8FF; usr(aRow);
  end
  else brush.color:=$F0F8FF;

  if aRow<=qp then with p[aRow] do begin
// текст
  case aCol of
   0: s:=n;
   1: case t of
       'p':s:='корисний'; 'w':s:='шкідливий';'c':s:='Ctrl'; else s:='не знаю';
      end;
   2: if a then s:='активний' else s:='спить';
   3: s:=FormatDateTime('dd.mm.yy hh:nn',FileDateToDateTime(d));
   4: if q>0 then s:=intToStr(q);
   5: begin s:=path;
       cw:=stringGrid1.ColWidths[5]-14;
       while (s<>'') and(textWidth(s) >cw) do setLength(s,length(s)-1);
       if length(s) <Length(path) then s:=s+'...';
      end;
  end;

// колір
  case t of
    'p':begin kc(-10,5,10);  font.color:=$BB8888 end;
    'w':begin kc(10,-20,-20); font.Color:=$FF end;
  end;
  if a then
  if aCol=2 then begin kc(80,-10,-20);font.color:=$8888DD end
  else kc(5,-5,-5);

  end;
  rectangle(r);
  if gdSelected in State then begin
     pen.Color:=$A0C0FF;
     moveTo(r.left,r.top+1); lineTo(r.right,r.top+1);
     moveTo(r.left,r.bottom-2); lineTo(r.right,r.bottom-2);
  end;
end;
textOut(r.left+3,r.top+3,s);
end;

nr:=0;
end;

procedure TForm1.CreateProcessTree();
var
//  AllPr: array of TProcessEntry32;
  i,j,k: integer;
  PE   : TProcessEntry32;
  IsRoot: boolean;
//  NodeNew: TTreeNode;
begin
```

```
 n:=0;
 ihw:=0;
 thrCount:=0;
 SH := CreateToolHelp32SnapShot(Th32cs_SnapAll, 0);
//виділення місця під структуру
pe.dwSize:=sizeof(PE);
 Process32First(SH, PE);
 repeat                  //найти всі процеси
   SetLength(all,n+2);
   all[n]:=PE;
   inc(thrCount,all[n].cntThreads);
   inc(n);
 until not Process32Next(SH, PE);
 //побудова дерева
 i:=0;
 DT_hWnd :=GetDesktopWindow;  //////////
 repeat
   //пошук кореневого елементу
   IsRoot:=true;
   for j:=0 to n-1 do
     if (all[i].th32ParentProcessID=all[j].th32ProcessID) and (i<>j)
     then
      begin
       IsRoot:=false;
       break;
      end;
   //побудова гілки
   if IsRoot
   then
    begin
     tvwProc.Items.AddObject(nil,ExtractFileName(all[i].szExeFile),@all[i]);
     tvwProc.Items[tvwProc.Items.Count-1].ImageIndex:=1;
     AddChild(i);
    end;
   inc(i);
  until i>=n;
 CloseHandle(SH);
 Form1.Caption:='Монітор - '+'Процесів: '+IntToStr(n)+
                ', Потоків: '+IntToStr(thrCount);
end;

// здійснює побудову гілки дерева від кореневого
// елементу до листя
procedure TForm1.AddChild(до: integer);
var
  m,i: integer;
begin
  i:=0;
  m:=tvwProc.Items.Count;//додається завжди до останнього елементу
  pr:=all[k].th32ProcessID;//для AddWindow
  AddWindow(DT_hwnd);
  repeat
    if (all[i].th32ParentProcessID=all[k].th32ProcessID) and(i<>k) then begin
      tvwProc.Items.AddChildObject (tvwProc.Items[m-1],
                                    ExtractFileName(all[i].szExeFile),@all[i]);
      tvwProc.Items[tvwProc.Items.Count-1].ImageIndex:=1;
      AddChild(i);
    end;
    inc(i);
  until i>=n;
end;

// додавання дерева вікон до д. процесів (перед викликом
// вказати ID процесу і Handle)
```

```
procedure TForm1.AddWindow(_hwnd: longint);
var
  m: integer;//  hw: longint;
begin
  m:=tvwProc.Items.Count-1;
  _hwnd:=GetWindow(_hWnd,GW_CHILD);
  while (_HWnd<>0) do begin
    thID:=GetWindowThreadProcessId(_hWnd,@prId);
    if prID=pr then begin       //якщо вікно належить процесу
      buflen:=GetWindowText(_HWnd,@buf,MaX_);
      if (buflen >0) then begin //якщо біля вікна є ім'я
        inc(ihw);
        hw[ihw]:=_hwnd;
        tvwProc.Items.AddChildObject(tvwProc.Items[m],String(buf),@hw[ihw]);
        tvwProc.Items[tvwProc.Items.Count-1].ImageIndex:=2;
        AddWindow(_hwnd);
      end;
    end;
    _hwnd :=GetWindow(_hwnd, GW_HWNDNEXT);
  end;
end;

procedure TForm1.btnRecreateClick(Sender: TObject);
begin
  sgrInfo.Cells[1,0]:='';  sgrInfo.Cells[1,1]:='';  sgrInfo.Cells[1,2]:='';
  while tvwProc.items.Count<>0 do begin
    tvwProc.Items[0].Delete;
    Application.ProcessMessages;
  end;
  CreateProcessTree();
end;

end;

// завершення процесу з використанням його ідентифікатора
procedure TForm1.btnTerminateClick(Sender: TObject);
var
  procId,ExCode : Cardinal;
  Hp : Thandle;   th: byte;
begin
  th:=ppe(tvwProc.Selected.data)^.cntThreads;
  Procid:=StrToInt64(edID.Text);
  Hp:=OpenProcess( PROCESS_TERMINATE,  //  прапор  доступу
     false  // handle inheritance flag
     procid );    // process identifier
  if TerminateProcess(Hp,ExCode) then begin
    tvwProc.Selected.Delete;

    dec(n);
    dec(thrCount,th);
    Form1.Caption:='Монітор - '+'Процесів: '+IntToStr(n)+
                ', Потоків: '+IntToStr(thrCount);
  end
  else
    ShowMessage('Неможливо завершити процес');
  tvwProc.Repaint;
end;

procedure TForm1.tvwProcDblClick(Sender: TObject);
begin
  if not ((ppe(tvwProc.Selected.Data)^.cntThreads=0)or
  (ppe(tvwProc.Selected.Data)^.pcPriClassBase>32)) then begin
    edID.Text:=IntToStr(Ppe(tvwProc.Selected.Data)^.th32ProcessID);
    btnTerminate.Enabled:=true;
```

```
  end;
end;

procedure TForm1.tvwProcClick(Sender: TObject);
var
  cpos: TPoint;
  tr: TRect;
begin
  edID.Text:='';
  if (ppe(tvwProc.Selected.Data)^.cntThreads=0) or
     (ppe(tvwProc.Selected.Data)^.pcPriClassBase>32) then begin

    //ВІКНО

    btnPrior.Enabled:=false;
    btnTerminate.Enabled:=false;
    if IsWindowVisible(longint(tvwProc.Selected.Data^)) then begin
      label1.Caption:='видиме';
      btnHideW.Enabled:=true;
      btnShowW.Enabled:=false;
    end
    else begin
      label1.Caption:='скрите';
      btnShowW.Enabled:=true;
      btnHideW.Enabled:=false;
    end;
    sgrInfo.Cells[1,0]:='';  sgrInfo.Cells[1,1]:='';  sgrInfo.Cells[1,2]:='';
  end

  else begin
    //ПРОЦЕС
    label1.Caption:='';
    btnHideW.Enabled:=false;
    btnShowW.Enabled:=false;
    btnPrior.Enabled:=true;
    btnTerminate.Enabled:=false;
    //
    sgrInfo.Cells[1,0]:=Ppe(tvwProc.Selected.Data)^.szExeFile;
    sgrInfo.Cells[1,1]:=IntToStr(Ppe(tvwProc.Selected.Data)^.pcPriClassBase);
    sgrInfo.Cells[1,2]:=IntToStr(Ppe(tvwProc.Selected.Data)^.cntThreads);
    case Ppe(tvwProc.Selected.Data)^.pcPriClassBase of
      4: cbPrior.ItemIndex:=0;//IDLE_PRIORITY_CLASS;
      8: cbPrior.ItemIndex:=1;//NORMAL_PRIORITY_CLASS;
      13:cbPrior.ItemIndex:=2;//HIGH_PRIORITY_CLASS;
      24:cbPrior.ItemIndex:=3;//REALTIME_PRIORITY_CLASS    ;
    else
      cbPrior.ItemIndex:=-1;
    end;

  end;
end;
procedure TForm1.tvwProcKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
 if (key=38) or(key=40) then tvwProcClick(Self);
end;

procedure TForm1.btnHideWClick(Sender: TObject);
begin
  ShowWindow(longint(tvwProc.Selected.Data^),SW_HIDE);
  btnHideW.Enabled:=false;
  btnShowW.Enabled:=true;
  label1.Caption:='сховано';
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
begin
 CreateProcessTree();
 sgrInfo.ColWidths[1]:=500;
 sgrInfo.Cells[0,0]:='Ім'я';
 sgrInfo.Cells[0,1]:='Пріоритет';
 sgrInfo.Cells[0,2]:='Потоків';
end;

procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
var
 i: integer;
begin
 all:=nil;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
 tvwProc.FullExpand;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
 tvwProc.FullCollapse;
end;

Function wel(c:char):string;
var se:string; i:integer;
begin
//видалення елементу до символу з s
se:='';
i:=1;
while (i<=length(s)) and (s[i]<>c) do begin
  //готуємо рядок
  se:=se+s[i];
  //інкремент лічильника
  inc(i);
end;
//видаляємо пробіли та керуючі символи
wel:=trim(se);
//видаляємо визначену кількість сиволів
delete(s,1,i);
end;

procedure TForm1.FormCreate(Sender: TObject);
//при відкритті форми
var
 sr,z:string;
 g:byte;
 dt:tdatetime;
 i:integer;
 //tc:cardinal;
begin
//if ctrlDown then qp:=0;
//координати вікна
hn:=500; wn:=525; ln:=360; tn:=120;
//висота вікна
height:=56;
//короткий формат дати
ShortDateFormat:='dd.MM.yyyy';
//розмір лінії
stringGrid1.GridLineWidth:=0;
//задаємо колір компоненти
```

```
stringGrid1.color:=$F0F8FF;
color:=$DFEFFF;
//колір фона мітки
label1.font.color:=$97A8DB;
label2.font.color:=$7788BB;
//інформація про запускаємий файл
en:=application.exeName;
//визначення повного шляху до запущеного додатку
ini:=extractFilePath(en)+'PC.ini';
//з'єднання
assignFile(t,ini);
{$i-}reset(t);{$i+}
//якщо помилка, то перезаписати файл
if ioresult<>0 then rewrite(t) else
while not eof(t) do begin
//поки не досягли кінця файлу, читаємо в рядок
  readln(t,s);
  //знаходимо знак =
  i:=pos('=',s);
  //при знаходженні знака TAB запуск лічильника
  if pos(#9,s) >0 then begin
    uqp;
    //заповнюємо структуру
    with p[qp] do begin
      wel(#9);
      n:=wel(#9);
      sr:=wel(#9); if sr='' then t:=' ' else t:=sr[1];
      sr:=wel(#9);
      d:=DateTimeToFileDate(StrToDateTime(sr));
      dt:=FileDateToDateTime(d);
      q:=strToIntDef(wel(#9),0);
      path:=wel(#9);
    end;
  end
  else
  if i>0 then begin
    sr:=copy(s,1,i-1);
    z:=copy(s,i+1,$FFFF);
    if sr='Висота' then hn:=strToIntDef(z,500);
    if sr='Ширина' then wn:=strToIntDef(z,325);
    if sr='Вліво'   then ln:=strToIntDef(z,360);
    if sr='Вгору'   then tn:=strToIntDef(z,120);
    if sr='Видаляти нові' then uwn:=bool(strToIntDef(z,0));
    if sr='Нові - шкідливі' then nw:=bool(strToIntDef(z,0));
  end;
end;
//розмір вікна
left:=ln;
top:=tn;
//закрити файл
closeFile(t);
//згорнути у трей
wHide;
if (paramCount=0) or (paramStr(1) <>'/h') then begin
  //видалити структуру з трей
  Shell_NotifyIcon(NIM_DELETE, @NID);
  //відобразити форму
  show
end;
end;

PROCEDURE save;
//запис у файл
var i:integer; si:string;
```

```
begin
with form1 do begin
assignFile(f,ini);
reset(f,1);
s:=info+
'Висота='+inttostr(height)+#13#10+
'Ширина='+inttostr(width)+#13#10+
'Вгору='+inttostr(top)+#13#10+
'Вліво='+inttostr(left)+#13#10+
'Видаляти нові='+intToStr(ord(uwn))+#13#10+
'Нові - шкідливі='+inttostr(ord(nw))+#13#10;

//запис даних у файл налаштувань
for i:=1 to qp do with p[i] do begin
  si:=n; while length(si) <20 do si:=si+' ';
  s:=s+'Процес='#9+si+#9+t+#9+
    FormatDateTime('dd.mm.yy hh:nn',FileDateToDateTime(d))+#9+
    intToStr(q)+#9+path+#13#10;
  blockWrite(f,s[1],length(s));
  s:='';
end;
//обрізаємо файл до поточної позиції
truncate(f);
//закрити файл
closeFile(f);
end;
end;

procedure TForm1.FormCloseQuery(Sender: TObject; var CanClose: Boolean);
//запис стану перед закриттям
begin
save;
end;

PROCEDURE sort;
var
i,j1,g1,r:integer;  ns:boolean;
Function srw:boolean;
begin
r:=j1+g1;
with p[j1] do
  case aCol of
    //по 1 колонці
    0: if n<p[r].n then srw:=su else
       if n>p[r].n then srw:=ns else srw:=false;
    //по 2 колонки
    1: if t<p[r].t then srw:=su else
       if t>p[r].t then srw:=ns else srw:=false;
    //по 3 колонки
    2: if a<p[r].a then srw:=su else
       if a>p[r].a then srw:=ns else srw:=false;
    //по 4 колонки
    3: if d<p[r].d then srw:=su else
       if d>p[r].d then srw:=ns else srw:=false;
    //по 5 колонок
    4: if q<p[r].q then srw:=su else
       if q>p[r].q then srw:=ns else srw:=false;
    //по 6 колонок
    5: if path<p[r].path then srw:=su else
       if path>p[r].path then srw:=ns else srw:=false;
  end;
end;

begin
```

```
//сортування
g1:=qp;
ns:=not su;
while g1>0 do begin
  g1:=g1 div 2;
  i:=g1;
  while i<qp do begin
    j1:=i-g1+1;
    while (j1>=1) and srw do begin
      r:=j1+g1;
      p1:=p[j1]; p[j1]:=p[r]; p[r]:=p1;
      dec(j1,g1)
    end;
    inc(i);
  end;
end;
end;

procedure TForm1.StringGrid1MouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
  //обробка події натиснення маніпулятора мишки
var aRow,xm,ym:integer;   r: TGridRect;

begin
StringGrid1.MouseToCell(X, Y, aCol, aRow);
if arow=0 then begin
  if aCol=sCol then su:=not su else su:=false;
  sCol:=aCol;
  sort; StringGrid1.rePaint; exit;
end;

usr(aRow);
r.top:=aRow; r.bottom:=aRow; r.left:=0; r.right:=StringGrid1.ColCount-1;
StringGrid1.selection:=r;
nr:=2;
StringGrid1.rePaint;
//якщо права кнопка натиснута, то виклик контекстного меню
if button=mbRight then begin
  N6.visible:=not p[sr].a;
  popupMenu1.popup(X+left,Y+top+StringGrid1.top+30);
end;
end;

procedure TForm1.StringGrid1KeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
  //обробка натиснення клавіш з клавіатури
begin
if key=112{F1} then showMessage(info);
nr:=2; StringGrid1.rePaint;
end;

procedure TForm1.FormResize(Sender: TObject);
//зміна розміру форми
begin
pAlign;
end;

PROCEDURE uTip(t:char);
//обробка рядкового параметра
begin
p[sr].t:=t;
nr:=2;
//перемалювання вікна
form1.StringGrid1.rePaint;
```

```
end;

procedure TForm1.N1Click(Sender: TObject);
//обробка контекстного меню "Корисний"
begin uTip('p') end;

procedure TForm1.N2Click(Sender: TObject);
//обробка контекстного меню "Шкідливий"
begin uTip('w') end;

procedure TForm1.N3Click(Sender: TObject);
//обробка контекстного меню "Не знаю"
begin uTip(' ') end;

procedure TForm1.CtrlClick(Sender: TObject);
//натиснута Ctrl
begin uTip('c') end;

procedure TForm1.N4Click(Sender: TObject);
//Забути (поновити дані, процес)
var i:integer;
begin
//зменшуємо на активний
dec(qp);
for i:=sr to qp do p[i]:=p[i+1];
nr:=2;
form1.StringGrid1.rePaint;
wl1;
end;

procedure TForm1.N6Click(Sender: TObject);
//обробка контексного меню "Відновити"
begin
//визначаємо шлях і програму запуску
s:=p[sr].path+'\'+p[sr].n;
//запускаємо
ShellExecute(0,'open', pChar(s),nil,pChar(p[sr].path),SW_NORMAL);
end;

PROCEDURE obrF(imf:string);
//лічильник
begin
inc(rrr);
end;

var
DirBytes : int64;
ns:string;
function scan(Dir:string):integer;
//визначення файлу процеса
var
sr: TSearchRec;
e,r : string;
begin
if Copy(Dir,Length(Dir),1)='\' then r:= '' else r:='\';
if FindFirst(Dir+r+'*.*',faAnyFile,sr)= 0 then begin
  repeat
    if FileExists(Dir+r+sr.Name) then begin
      DirBytes:=DirBytes+sr.Size;
      if ansiLowerCase(sr.name)=ns then
      s:=dir;
      e:=lowerCase(extractFileExt(sr.name));
      if (e='.dll') or(e='.exe') then
      inc(rrr);
```

```
//    Dir+r+sr.Name - файл
    end
    else
    if DirectoryExists(Dir+r+sr.Name) then begin
      if (sr.Name<>'.') and (sr.Name<>'..') then
        scan(Dir+r+sr.Name);
    end;
  until FindNext(sr) <>0;
end;
FindClose(sr);
end;

procedure TForm1.N11Click(Sender: TObject);
//визначення шляху до програми, що створила процес
var path:array[0..MAX_PATH] of char;
begin
//отримати шлях
GetWinDowsDirectory(path,MAX_PATH);
DirBytes:=0;
ns:=p[sr].n;
s:='';
scan(strPas(path));
//якщо шлях не пустий
if s<>'' then
begin p[sr].path:=s;
//перемалювати
stringGrid1.repaint;
end;
end;

FUNCTION pp(id:word):string;
var
PIDList:PItemIDList;
path:array[0..MAX_PATH] of char;
begin
//отримання типового шляху
pp:='';
if SHGetSpecialFolderLocation(form1.handle,id,PIDList)=NOERROR
then begin
  SHGetPathFromIDList(PIDList,Path);
  pp:=strPas(path)+'\';
end;
end;

PROCEDURE CreateLink(const pathObj,PathLink,Descript,Params,pathI: string);
var
shellLink:IshellLink;
iObject:IUnknown;
LinkFile:iPersistFile;
begin
  iObject:=createComObject(CLSID_ShellLink);
  LinkFile:=IObject as IPersistFile;
  ShellLink:=IObject as IShellLink;
  with ShellLink do begin
    SetPath(PChar(pathObj));
    SetArguments(PChar(params));
    SetDescription(PChar(Descript));
    SetIconLocation(PChar(pathI),0);
  end;
  LinkFile.Save(PWChar(WideString(PathLink)),false);
end;

PROCEDURE startUP(z:boolean; pa:string);
var
```

```
lnk:string;
begin
//запис в автозавантаження з параметром pa
s:=pp(CSIDL_STARTUP);
if s<>'' then begin
  lnk:=s+'\'+'PC_Auto.lnk';
  if z then CreateLink(en,lnk,'',pa,en)
       else deleteFile(lnk);
end;
end;

procedure TForm1.N8Click(Sender: TObject);
//контексне меню встановити, автозапуск вибраного
begin
startUP(true,'')
end;
procedure TForm1.N9Click(Sender: TObject);
//вимкнути автозапуск
begin
startUP(false,'')
end;

procedure TForm1.N10Click(Sender: TObject);
//запускати в фоновому режимі
begin
startUP(true,'/h')
end;

procedure TForm1.MTIcon(var a: TMessage);
var P: TPoint;
begin
if (a.lParam=WM_LBUTTONDOWN) or(a.lParam=WM_RBUTTONUP) then begin
  show;
  SetForegroundWindow(Handle);
  application.processMessages;
//видалення значка
  Shell_NotifyIcon(NIM_DELETE, @NID);
end;
end;

procedure TForm1.N14Click(Sender: TObject);
//обробка контекстного меню
//знищити поточний
begin
p[sr].k:=true
end;

procedure TForm1.N15Click(Sender: TObject);
//контекстне меню
//знищити усі нові
begin
uwn:=not uwn;
end;

procedure TForm1.PopupMenu1Popup(Sender: TObject);
begin
N15.checked:=uwn;
N16.checked:=nw;
end;

procedure TForm1.N16Click(Sender: TObject);
//знищити нові шкідливі
begin
nw:=not nw;
```

```
end;

procedure TForm1.N17Click(Sender: TObject);
//виведення довідки
begin
showMessage(info);
end;

procedure TForm1.HIGHPRIORITYCLASS1Click(Sender: TObject);
//високий пріоритет
begin
p[sr].pr:=HIGH_PRIORITY_CLASS;
end;

procedure TForm1.NORMALPRIORITYCLASS1Click(Sender: TObject);
//нормальний пріоритет
begin
p[sr].pr:=NORMAL_PRIORITY_CLASS
end;

procedure TForm1.IDLEPRIORITYCLASS1Click(Sender: TObject);
//нижче нормального
begin
p[sr].pr:=IDLE_PRIORITY_CLASS;
end;

procedure TForm1.REALTIMEPRIORITYCLASS1Click(Sender: TObject);
//встановлення пріоритетів реального часу
begin
p[sr].pr:=REALTIME_PRIORITY_CLASS;
end;

end.
```