МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК
СЕКЦІЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ПРОЕКТУВАННЯ

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: «Інструментарій контейнеризації програмних додатків»

за спеціальністю 122 «Комп'ютерні науки»,
освітньо-професійна програма «Інформаційні технології проектування»

Виконавець роботи: студент групи ІТ.м-91 Нечепорук Олександр Андрійович

Кваліфікаційну роботу
захищено на засіданні ЕК
з оцінкою «___» грудня 2020 р.

Науковий керівник _____ к.т.н., доц., Ващенко С.М.
(підпис)

Голова комісії _____ Шифрін Д.М.
(підпис)

Засвідчую, що у цій дипломній роботі немає
запозичень з праць інших авторів
без відповідних посилань.

Студент _____
(підпис)

Суми-2020

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук
Секція інформаційних технологій проектування
Спеціальність – 122 «Комп'ютерні науки»
Освітньо-професійна програма «Інформаційні технології проектування»

ЗАТВЕРДЖУЮ
Зав. секцією ІТП

_____ В.В. Шендрик
«__» _____ 2020 р.

ЗАВДАННЯ
на кваліфікаційну роботу магістра студентові

Нечепорук Олександр Андрійович
(прізвище, ім'я, по батькові)

1 Тема роботи *Інструментарій контейнеризації програмних додатків*

затверджені наказом по університету від « 26 » листопада 2020 р. № 1824-III

2 Строк подання студентом роботи «07» грудня 2020 р.

3 Вхідні дані до роботи технічні вимоги до інструментарію контейнеризації програмних продуктів

4 Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) аналіз предметної області, постановка задачі, моделювання інструментарію контейнеризації програмних додатків, розробка інструментарію контейнеризації програмних додатків.

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) актуальність роботи, обєкт та предмет, мета та задачі, дослідження та аналіз аналогів, функціональні вимоги, контекстна діаграма процесу роботи інструментарію, діаграма декомпозиції процесу роботи інструментарію, діаграма декомпозиції процесу переадресації системних викликів, діаграма варіантів використання, діаграма послідовності, конфігурація зберігання даних, вибір засобів реалізації, демонстрація роботи інструментарію, демо-контейнер, переадресація елементів файлової системи, переадресація для підпроцесів, переадресація елементів реєстру, висновки, практична значимість.

6. Консультанти випускної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

Дата видачі завдання _____.

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускного проекту	Термін виконання етапів проекту	Примітка
1	Дослідження існуючих моделей адміністрування та пакування програмного забезпечення	30.09.20 – 08.10.20	
2	Дослідження існуючих програмних рішень	09.10.20 – 14.10.20	
3	Визначення функціоналу програмного модуля	14.10.20 – 15.10.20	
4	Вибір технічних засобів для розробки	15.10.20 – 15.10.20	
5	Моделювання інструментарію	16.10.20 – 20.10.20	
6	Практична реалізація	21.10.20 – 16.11.20	
7	Тестування програмного модуля	17.11.20 – 27.11.20	
8	Оформлення документації	17.11.20 – 27.11.20	

Магістрант _____ Нечепорук О.А.
(підпис)

Керівник роботи _____ к.т.н., доц. Ващенко С.М.
(підпис)

РЕФЕРАТ

Тема кваліфікаційної роботи магістра: «Інструментарій контейнеризації програмних додатків».

Пояснювальна записка складається зі вступу, 4 розділів, висновків, списка використаних джерел із 45 найменувань, 3 додатки. Загальний обсяг роботи складає 113 сторінок, які містять 4 таблиці, 33 рисунка

Метою проекту є розробка інструментарію контейнеризації програмних додатків для його подальшого використання на віртуальних машинах у робочому середовищі компанії Apptimized Operations з мінімальним впливом на операційну систему задля зберігання стану операційної системи максимально наближеної до її початкового стану. Використання інструментарію дозволить скоротити ризик втрати потенціальних спільних файлів бібліотек чи системних ключів реєстру, які можуть бути присутні на віртуальних машинах із локально встановленими інструментами DevOps інженерів.

Актуальність даної роботи полягає в тому, що існує потреба в контейнеризації програмних додатків під час створення пакетів ПЗ з метою підвищення якості результатів пакування програмного забезпечення.

Результатом виконання проекту є розроблений інструментарій, що підтримує роботу контейнеризованого додатку використовуючи файл конфігурації та забезпечує безпечне використання програмного забезпечення на робочих станціях інженерів відділу розробки та пакування програмного забезпечення компанії Apptimized Operations.

Ключові слова: DevOps, контейнер, інструментарій, розробка, пакування, програмне забезпечення, EasyHook, DLL hooking, redirection, WinAPI, system call, application packaging.

ЗМІСТ

ВСТУП

Зважаючи на повсемісну діджиталізацію виробничих та бізнес процесів все частіше виникає потреба в розробці, налаштуванні та адмініструванні програмного забезпечення на робочих комп'ютерах. Враховуючи різноманітність програмних додатків в залежності від сфери використання та їх залежностей від зовнішніх бібліотек, фреймворків чи окремих продуктів існує проблема нераціонального використання дискових просторів робочих станцій користувачів та скупчення додаткових утиліт або продуктів, які не мають практичного застосування крім забезпечення працездатності необхідного програмного забезпечення. Окрім цього, актуальним є питання управління та адміністрування програмного забезпечення з мінімальним впливом на операційну систему користувача.

Існує певна кількість методів розповсюдження програмного забезпечення поміж комп'ютерами одного підприємства, а також безліч форматів зберігання та формування пакетів програмного забезпечення. Проаналізувавши існуючі методи та формати сформовано мету роботи - розробити інструментарій контейнеризації програмних додатків для зберігання програмних додатків з мінімальним впливом на операційну систему.

Об'єктом дослідження кваліфікаційної роботи є інформаційні технології зберігання, розповсюдження та запуску програмних додатків. Предметом дослідження можна вважати ті методи, технології та програмні засоби, що дозволяють розробити власний інструментарій контейнеризації програмних додатків.

Задля реалізації поставленої мети необхідно вирішити наступні задачі:

- проаналізувати предметну область та аналоги;
- обрати технічні та програмні засоби розробки;
- визначити функціонал та логіку інструментарію;
- створення алгоритмів та написання коду інструментарію;
- визначити модель зберігання даних віртуалізованих додатків;

- протестувати розроблений інструментарій.

З точки зору прикладного значення, розроблений інструментарій може та буде використаний у робочому середовищі відділу розробки та пакування програмного забезпечення компанії Apptimized Operations (додаток А), а також впроваджено у портал для клієнтів компанії.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Дослідження моделей адміністрування та пакування програмних додатків

Згідно зі статистикою популярності операційних систем (ОС) за 2020 рік, лідером серед десктопних ОС є Windows - більше 77% всіх комп'ютерів, наступними за кількістю встановлень є OS X – 17.7% та Linux – 1.7%. Такий розподіл є очікуваним та цілком логічним через специфіку використання систем сімейства Linux та апаратної залежності OS X. Також необхідно розуміти, що для деякої кількості систем розповсюдженим є принцип встановлення серверної частини на базі ОС Linux, а комп'ютери кінцевих користувачів в цій інформаційній системі можуть використовувати операційну систему від Microsoft [1].

Визначивши що на статистичній більшості робочих машин встановлена ОС Windows в даній кваліфікаційній роботі будуть розглянуті засоби розповсюдження, підготовки та встановлення програмного забезпечення актуальні саме для цієї операційної системи.

Перш за все, враховуючи специфіку адміністрування комп'ютерних мереж з Windows клієнтами, оптимальним варіантом розповсюдження програмного забезпечення є використання систем розгортання програмного забезпечення (Software Deployment System, SDS). Основними перевагами використання SDS є можливість планувати бізнес процеси з урахуванням життєвого циклу розробки програмного забезпечення, його гнучкості та значної економії витрат на процес розповсюдження та встановлення програмного забезпечення [2].

Розглянемо популярні рішення від Microsoft, це System Center Configuration Manager (SCCM)[3] та Microsoft Windows Intune[4]. Їх основні функції схожі, проте є певні відмінності. Обидві системи використовуються для дистрибуції програмних додатків, оновлень операційної системи, менеджменту програмного забезпечення, віддаленого керування; як системи моніторингу та зворотного зв'язку та інших

задач адміністрування. Головною відмінністю між цими системами є область їх використання.

SCCM встановлюється та запускається з фізичних серверів компанії, при тому, сервер має знаходитись в тій же мережі, що і клієнти. Можливість масштабування таких систем хоч і існує, проте це очевидно ускладнює її налаштування та подальшу підтримку.

Стосовно Intune, він є повністю хмарним сервісом, що з одного боку дозволяє об'єднувати комп'ютери різних мереж в єдину систему, проте повністю залежить від Інтернет з'єднання. З власного досвіду використання цієї системи можу сказати, що розповсюдження та видалення програмного забезпечення з «серверу» до кінцевих робочих станцій не є миттєвим, для запуску ініціалізації та застосування налаштувань необхідна певна кількість часу, навіть при ручному запуску відповідної служби. Крім затримки, також створюється додаткове навантаження на мережу, що може призвести до втрати швидкості передачі даних в мережі Інтернет [5].

Розглянуті рішення від Microsoft хоч і є популярними та зручними, проте мають і мінуси: надлишковий функціонал, складність налаштування, необхідність постійного адміністрування та вартість ліцензії на використання. Зваживши всі плюси та мінуси було прийнято до варіантів власний додаток під назвою Apptimized Workspace Launcher, що виконує роль агрегатора програмних додатків, як власної розробки, так і сторонніх розробників і, в той же час, є системою SDS клієнтського типу. Плюсом використання продукту власної розробки є безперешкодне його використання в комерційних цілях, а також можливість інтегрувати власний інструментарій та логіку одразу в систему розгортання програмного забезпечення [6].

Apptimized Launcher для зберігання пакетів програмного забезпечення використовує власний репозитарій, що доступний з віртуальних машин компанії, які в свою чергу, можуть використовуватися як в локальній мережі компанії, так і в мережі Інтернет. Встановлення додатків на робочих станціях відбувається шляхом

копіювання інсталяційних пакетів програмного забезпечення з подальшим його встановленням у автоматичному режимі.

Така модель постачання програмного забезпечення нас влаштовує, проте є недоліки поточної логіки встановлення додатків, так як даний метод потребує значного об’єму пам’яті, встановлення необхідних пререквізитів, та загалом суперечить принципам налаштування віртуальних машин інженерів-пакувальників ПЗ. Для ефективної роботи DevOps інженера при створенні пакету програмного додатку, звісно, необхідне спеціалізоване програмне забезпечення, але при цьому необхідно витримати баланс між необхідною кількістю утиліт та чистотою операційної систему, яка, в свою чергу, має бути максимально наближеною до стану системи «з коробки» [7].

Для вирішення описаної проблеми чистоти системи необхідно проаналізувати існуючі моделі пакування програмних додатків. Розглянемо технології віртуалізації, контейнеризації та класичного встановлення програм на кінцеву операційну систему.

Розпочнемо зі стандартного встановлення програмних додатків на систему так, як задумав виробник програмного забезпечення. Для зберігання та розповсюдження свого продукту розробники створюють інсталяційні пакети додатків використовуючи одну із існуючих технологій (наприклад: MSI, InnoSetup, InstallShield, NSIS та інші). Інсталяційний пакет, що не є Windows Installer (MSI файлом) частіше за все є архівом що саморозпаковується. Основними операціями під час встановлення ПЗ є розпакування архіву, копіювання файлів програми до файлової системи, запис ключів реєстру, а також можливе виконання певних спеціалізованих подій (системних команд, скриптових дій, запуск сервісів та інше) [8].

Очевидно, що для виробника програмного забезпечення раціональність використання ресурсів користувача та можливість портативного встановлення додатків не є першочерговою задачею, тому в даному випадку, технологія класичного встановлення програм використовуючи логіку розробника за замовчуванням зовсім не підходить для вирішення нашої задачі.

Деякі розробники надають можливість використовувати їх продукти портативно, тобто передбачають, що для повної функціональності додатку достатньо мати лише набір певних файлів в межах однієї директорії. Насправді, це майже ідеальний варіант для наших цілей, проте, далеко не всі програмні продукти мають можливість працювати портативно та разом з цим існує ймовірність надлишкового зберігання файлів, бібліотек та фреймворків спільних для декількох додатків одночасно в директоріях різних програм [9].

Отже, необхідно використовувати щось схоже на портативні інсталяції програмних додатків, проте з певними доопрацюваннями. Для того щоб вирішити недостатність портативності для більшості додатків звернемося до технологій віртуалізації та контейнеризації.

Розпочнемо з огляду технології віртуалізації. Віртуалізація – це процес створення програмного, віртуального подання або ж представлення будь-якої сутності, наприклад, віртуальних додатків, серверів, сховищ даних чи комп’ютерних мереж [10]. Це дозволяє використовувати повну потужність апаратного забезпечення, розподіляючи ресурси фізичної машини серед багатьох середовищ, віртуальних машин [11].

Віртуалізація стала першою технологією для оптимізації використання серверних ресурсів. Створені для наслідування апаратного забезпечення фізичного комп’ютера з налаштованою операційною системою, віртуальні машини та гіпервізори дають можливість запускати декілька комп’ютерів з різними операційними системами на базі єдиного фізичного сервера.

Віртуалізація неможлива без гіпервізора. Гіпервізор, або монітор віртуальної машини - це рівень програмного забезпечення чи прошивки, що забезпечує роботу декількох операційних систем паралельно, з індивідуальним доступом до спільних фізичних ресурсів сервера. Гіпервізор організовує та відокремлює доступні ресурси (обчислювальну потужність, оперативну пам’ять, дисковий простір, тощо), розподіляючи їх до кожної віртуальної машини за потреби [12].

Необхідно розуміти різницю між віртуальними машинами що використовуються в якості робочих станцій користувачами, які мають

користувацький інтерфейс та віртуальні машини що використовуються для віртуалізації додатків. Частіше за все вони не мають прямого доступу та користувацького інтерфейсу, а використовуються лише для запуску та роботи з віртуалізованим додатком.

Візуально для користувача кожна віртуальна машина в даному контексті представляє собою папку з файлами, кожен з яких можна модифікувати, переміщувати та копіювати засобами файлової системи операційної системи. Таким чином, віртуалізація дозволяє систематизувати дані програмних додатків, умовно, в межах однієї директорії; централізувати робочі навантаження та запускати кілька різних віртуальних додатків, навіть з різними системними вимогами на одній фізичній системі, що є значною перевагою перед локальним обладнанням.

Однак, віртуальні машини не позбавлені недоліків. Оскільки кожна віртуальна машина включає операційну систему та віртуальну копію необхідного обладнання, вона потребує значних об’ємів оперативної пам’яті та ресурсів центрального процесора. Через збільшення кількості віртуальних копій та споживаних ресурсів ускладнюється цикл розробки програмного забезпечення порівняно з локальною установкою програмного забезпечення. Складність налаштування віртуальних пакетів програмних додатків через необхідність врахування особливостей віртуального середовища, що в свою чергу ускладнює переміщення віртуальних машин між публічними хмарами, приватними хмарними сховищами та традиційними центрами зберігання та обробки даних. [13]

Контейнеризація стала основною тенденцією у розробці програмного забезпечення як альтернатива або супутник віртуалізації. Вона включає інкапсуляцію або упаковку програмного коду та всіх його залежностей, щоб він міг працювати рівномірно та послідовно у будь-якій інфраструктурі. Технологія швидко розвивається, що призводить до появи логічних переваг для розробників та DevOps інженерів, а також загальної інфраструктури програмного забезпечення.

Контейнеризація дозволяє розробникам створювати та розгортати програмні додатки швидше та безпечніше. За допомогою традиційних методів код розробляється в конкретному обчислювальному середовищі, яке при перенесенні на

нове місце часто призводить до помилок. Наприклад, коли розробник передає код із настільного комп'ютера на віртуальну машину або з Linux в операційну систему Windows. Контейнеризація усуває цю проблему, об'єднуючи код програми разом із відповідними файлами конфігурації, бібліотеками та залежностями, необхідними для її запуску. Цей єдиний пакет програмного забезпечення, або «контейнер» абстрагується від хостової операційної системи, тож він встановлюється окремо і стає портативним та може працювати на будь-якій платформі або хмарі [14].

Важливо розуміти, що контейнери не потребують окремих віртуальних машин, як всередині пакету, так і в цілому. Вони мають спільне ядро операційної системи з фізичною машиною та не вимагають додаткових зусиль на емуляцію операційної системи в кожному додатку. Контейнери є меншими за віртуальну машину і вимагають менше часу для запуску, що дозволяє використовувати більшу кількість контейнерів на тій самій обчислювальній потужності, що необхідно для однієї віртуальної машини. Це сприяє підвищенню ефективності роботи сервера і, в свою чергу, знижує витрати на апаратне забезпечення.

Розглянемо технологію контейнеризації на прикладі Docker – технологія яка була запущена у 2013 році як проект з відкритим кодом. Він використовував існуючі концепції та знання стосовно контейнерів, а в інфраструктурі Linux використовував відомі групи класів та простори імен. Технологія Docker унікальна, оскільки вона орієнтована на вимоги розробників та системних операторів, щоб відокремити залежності додатків від інфраструктури.

Успіх технології серед користувачів Linux привернув увагу Microsoft, що призвело до партнерства, яке, в свою чергу, дозволило адаптувати контейнери Docker та їх функціональність під вимоги та специфіку Windows [15].

Для кращого розуміння різниці між технологіями віртуалізації та контейнеризації коротко розглянемо принципи їх роботи.

Контейнери - це абстракція на програмному рівні, яка об'єднує в одну сутність основний програмний код і залежності. Кілька контейнерів можуть працювати на одній машині та спільно використовувати ядро ОС з іншими контейнерами, кожен з яких виконується як ізольований процес в просторі користувача. Візуальну схему

принципу роботи контейнеризованих пакетів та його взаємодію з операційною системою зображено на рисунку 1.1.

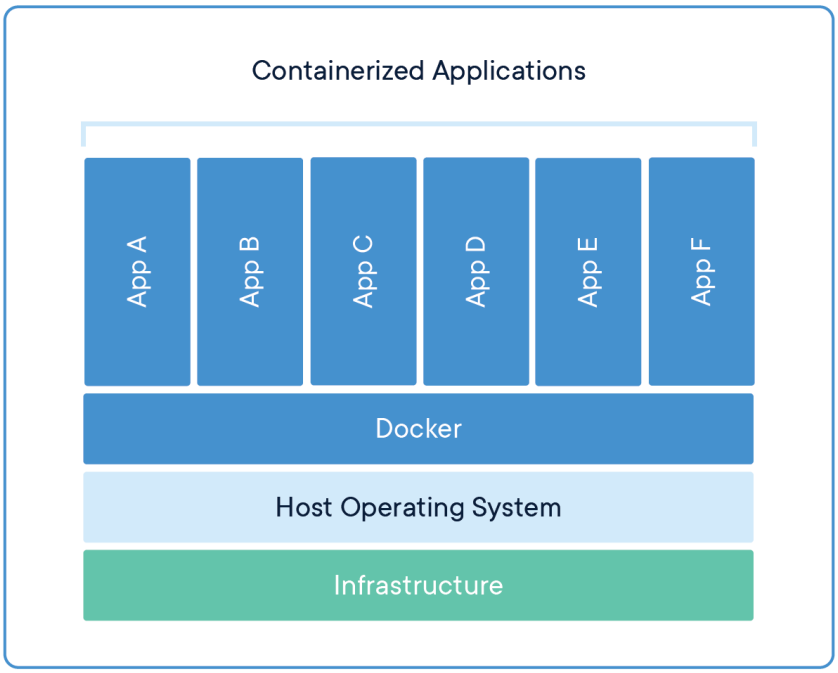


Рисунок 1.1 – Схема роботи контейнерів на ОС

Віртуальні машини - це абстракція фізичного обладнання, що перетворює один сервер на багато серверів. Гіпервізор дозволяє запускати кілька віртуальних машин на одній машині. Кожна віртуальна машина включає повну копію операційної системи, програми, необхідних бінарних файлів та бібліотек. Візуальну схему зображено на рисунку 1.2.

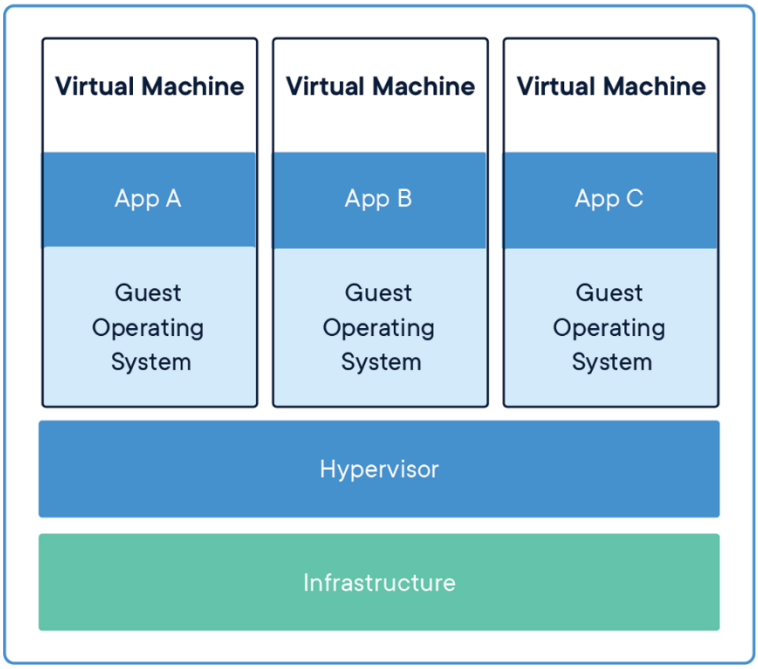


Рисунок 1.2 – Схема роботи віртуалізації на ОС

Контейнери займають менше місця, ніж віртуальні машини (розміри образів контейнерів, як правило, складають десятки МБ), можуть обробляти більше додатків і вимагати меншої кількості віртуальних машин та операційних систем. Проте, як і технологія віртуалізації, контейнери мають власні недоліки:

- контейнери для певних хостових версій операційних систем потребують підготовки контейнерів на аналогічній ОС. Контейнери розроблені на базі іншої версії ОС потребують іншого хоста або, в деяких окремих випадках, складнішого налаштування;
- так як операційна система є спільною для декількох контейнерів, вразливість в ядрі ОС може стати загрозою безпеки всіх контейнерів на хостовій машині;
- контейнеризація - відносно нове рішення, яке має різноманіття у способах реалізації та недостатню кількість кваліфікованих ресурсів, що робить процес впровадження контейнерів на підприємство складним.

1.2 Дослідження існуючих програмних рішень

Після проведеного аналізу сучасних моделей розповсюдження та встановлення програмних додатків необхідно визначитися з інструментами та методами контейнеризації додатків. Для цього розглянемо вже існуючі програмні рішення як контейнеризації, так і віртуалізації. Для порівняння функціональних ознак було обрано Docker та Cloudhouse як приклад реалізації технології контейнеризації та App-V серед рішень віртуалізації.

Microsoft Application Virtualization (більш відома як App-V) – рішення компанії Microsoft для віртуалізації програмних додатків. App-V є достатньо зрілою технологією, проте має багато прихильників навіть сьогодні, адже дозволяє адміністраторам пакувати додатки, що необхідно встановлювати локально на робочі комп'ютери у спеціалізовані керовані служби. Для підготовки пакетів зазвичай використовується App-V Sequencer, утиліта від Microsoft, або ж кастомні додатки, що використовують прикладні програмні інтерфейси та загальнодоступні бібліотеки технології [16].

Під час генерації App-V пакету засобами інструментів захоплення формується не лише структура файлів та ключів реєстру, а й віртуалізація операційної системи, симуляція апаратного забезпечення, інтерфейсів та всіх необхідних програмних залежностей для обраного додатку. Необхідно пам'ятати, що сутність віртуалізації є частиною саме пакету, а не зовнішніх обробників. Для встановлення App-V додатків та їх коректної роботи окрім гіпервізору (що, в принципі, є компонентом операційної системи) необхідним є також будь-який з App-V клієнтів. Для цього необхідно або увімкнути вбудований в систему (починаючи з Windows 10, version 1607) або ж встановити будь-який самостійний клієнт, наприклад App-V Configuration Manager [17].

Для App-V властиві всі переваги технології віртуалізації, які були описані раніше, зокрема, App-V навіть дозволяє запускати різні версії однієї програми на

одному комп’ютері, навіть якщо ці програми можуть конфліктувати, якщо їх встановлювати локально.

Однак, є кілька важливих речей, які слід врахувати, при впровадженні App-V. З одного боку, App-V передає повні двійкові файли додатків на комп'ютери користувачів, а не лише інформацію про натискання клавіш (введення) та відображення даних (виведення) за допомогою протоколу віддаленого робочого столу (RDP), як це відбувається при роботі з повноцінними віртуальними машинами RemoteApp. Необхідно пам’ятати, що App-V також вимагає окремої серверної технології окрім вже існуючої інфраструктури Windows Server.

Хоча Microsoft і продовжує вдосконалювати та підтримувати App-V (поточна версія 5.1), проте частота цих оновлень та їх сутність можна вважати профілактичними. Мається на увазі те, що дана технологія поступово перетворюється в застарілу та відходить на другий план. З одного боку, така зріла технологія означає, що вона пройшла випробування часом, є надійною та зрозумілою для адміністраторів та кінцевих користувачів, проте, очевидно, що налаштовувати систему використовуючи застарілі технології є не кращим варіантом [18].

На відміну від App-V, яка є зрілою технологією, Windows Containers - це нова технологія, яку Microsoft розробляє як частину Windows Server, починаючи з версії 2016. Windows Containers – це спроба Microsoft вдосконалити, розширити та використовувати контейнери Docker як на Windows Server, так і на Microsoft Azure.

Основна перевага використання Windows Containers полягає в тому, що він дозволяє компаніям, які мають Windows-орієнтовану інфраструктуру, використовувати ті ж самі переваги, що мають контейнерів Docker на системах сімейства Linux. Перевагами є швидке та легке розгортання, більша портативність додатків, ефективніше використання системних ресурсів та підтримку широкого кола платформ розробки [19].

Завдяки екосистемі контейнерів Microsoft, розробка та налаштування Docker контейнерів є набагато простішою за створення App-V пакетів, так як розробники впровадили інструмент контейнеризації одразу в операційну систему Windows

(доступно лише для Windows 10 Anniversary Update version 1607), а також можливе встановлення за допомогою PowerShell, або ж у складі популярного середовища розробки Microsoft Visual Studio чи як самостійний додаток. Створення контейнерів відбувається за допомогою спеціально підготовлених розробниками програмного забезпечення пакетів, або ж шляхом компіляції програмного коду одразу в контейнер[20].

Зважаючи на походження технології контейнеризації та, власне, рішення Docker, стає зрозумілим та логічним основне обмеження технології, що полягає у можливості контейнеризації лише додатків з інтерфейсом командного рядка. Так, основним мінусом Windows контейнерів є відсутність підтримки додатків з графічним користувацьким інтерфейсом, що визначає дану технологію такою, що не відповідає специфіці майбутнього використання в рамках середовища впровадження результатів кваліфікаційної роботи.

Враховуючи всі переваги використання контейнеризації як технології та основний недолік Docker'a, звернемо увагу на гібридну технологію Cloudhouse. Перш за все, це технологія контейнеризації Windows-додатків, основне призначення якої зворотна підтримка застарілих додатків на нових системах. Наприклад, існує деяка спеціалізована утиліта, що вже не підтримується розробником та не оновлювалась десяток років. Спочатку вона розроблювалась для операційної системи Windows XP, а певна потенціальна фірма може використовувати цю утиліту для вирішення її функціональних задач. Очевидно, що використовувати застарілу версію ОС не оптимально та навіть небезпечно, тому, створивши Cloudhouse контейнер існує можливість запускати та використовувати застарілі версії програмного забезпечення на сучасних системах, як локально, так і в хмарних сховищах [21].

Це можливо завдяки реалізації механізму переадресації системних викликів [22]. Наприклад, програма звертається до застарілої бібліотеки чи до файлу, що вже не входить до складу операційної системи. Сам контейнер складається з файлів, які встановлюються при звичайній інсталяції програми, спеціальної оболонки що забезпечує логіку роботи додатку в контейнері та

конфігураційних файлів, що містять серед інших, файли з вказаними параметрами переадресації. Під час виконання програми всередині контейнера у фоні виконується моніторинг системних викликів до ядра операційної системи, встановлено спеціальні обробники для захоплення дій пов’язаних зі зверненням до файлової системи та до реєстру Windows. Якщо моніторинг виявляє запит до ядра з викликом методу файлової системи чи реєстра, виконується перевірка, чи входить елемент що викликається до списку переадресації із конфігураційного файлу, якщо ні-запит залишається без змін, якщо ж існує співпадіння по фільтру, початкова адреса елементу замінюється на вказану у файлі та виконується dll-ін’єкція зі зміненою адресою. Таким чином, оболонка вказує оригінальній програмі де знайти той чи інший файл або ключ реєстра [23].

Отже, використовуючи технологію контейнеризації Cloudhouse, в результаті маємо директорію з усіма необхідними файлами та перевіреною системою обробки системних запитів, що дозволяє повністю ізолювати додаток від хостової системи. Завдяки системі переадресації, окрім зміни кореневої директорії додатку, можна змінити місце зберігання конфігураційних файлів та вказати необхідні залежності (як включені до контейнеру, так і зовнішні).

Проте, окрім позитивних сторін, маємо і недоліки. Перш за все-надлишковість функціоналу, який включає в себе додаткові плагіни віртуалізації, підтримку хмарних рішень сімейства Citrix та часткового використання хостової ОС. Також, для встановлення контейнерів Cloudhouse необхідний спеціальний токен-файл, що за умовами використання має бути індивідуальним та не передбачує корпоративного використання, окрім чого надсилає дані про використання контейнеру на хмарні сховища компанії розробника.

Отже, провівши детальний аналіз предметної області, технологій та прикладних рішень, зваживши всі сильні та слабкі сторони аналогів, було прийнято рішення розробити власний інструментарій для контейнеризації програмних додатків з подальшим використанням у корпоративних цілях.

2 ПОСТАНОВКА ЗАДАЧІ

2.1 Мета та задачі

Метою проекту є розробка інструментарію контейнеризації програмних додатків для його подальшого використання на віртуальних машинах у робочому середовищі компанії Apptimized Operations з мінімальним впливом на операційну систему задля зберігання стану операційної системи максимально наближеної до її початкового стану. Використання інструментарію дозволить скоротити ризик втрати потенціальних спільних файлів бібліотек чи системних ключів реєстру, які можуть бути присутні на віртуальних машинах із локально встановленими інструментами DevOps інженерів.

Перед початком роботи було отримано основні вимоги до функціоналу інструментарію. Перш за все, файли додатків мають зберігатися в єдиному сховищі разом зі всіма необхідними зовнішніми залежностями для полегшення адміністрування, встановлення та підтримки додатків на кінцевих робочих станціях. Також, існує необхідність в універсальному програмному рішенні, тобто забезпечити принцип багаторазового використання інструментарію без зміни його програмного коду в залежності від додатку, робота якого підтримується. Універсальність має забезпечуватися за допомогою файлу конфігурації, розташованим разом з файлами основного програмного додатку. Окрім файлів необхідно також передбачити обробку роботи додатків з директоріями файлів та з записами реєстру, оскільки більшість сучасних програмних рішень використовують реєстр ОС Windows для зберігання налаштувань чи корисної інформації.

Для кращого уявлення про роботу інструментарію контейнеризації було наведено схему обробки системних операцій, що представлена на рис. 2.1.

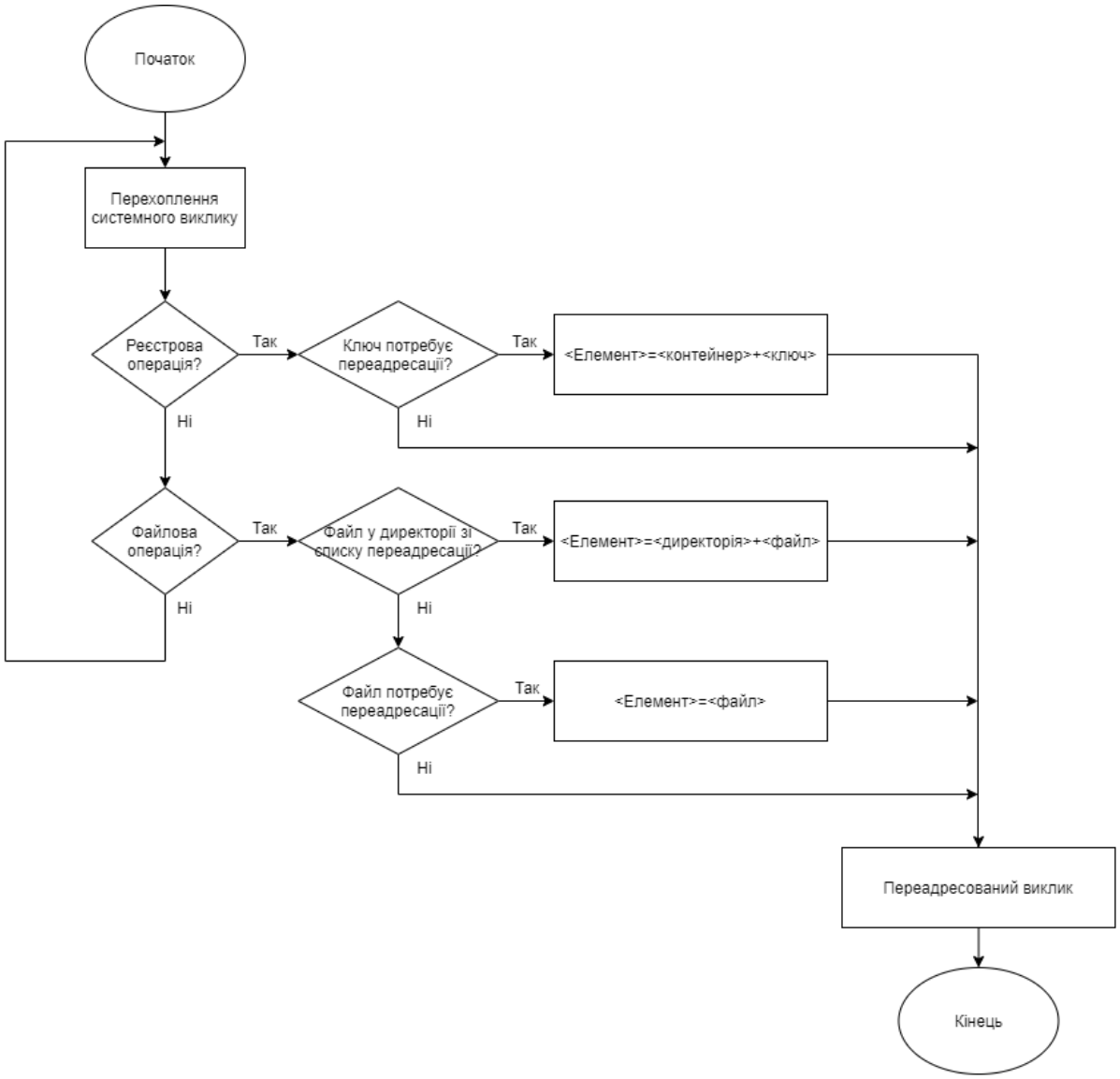


Рисунок 2.1 – Схема обробки системних операцій

Розроблений інструментарій виконуватиме наступні функції:

- зберігати файли програмних додатків у єдиній директорії-контейнері;
- забезпечувати роботу програмного додатку аналогічно до локального встановлення відповідно до його власних функціональних вимог;
- використовувати механізм моніторингу та перехвату системних викликів;
- редагувати системні виклики в залежності від конфігурації певного додатку;

- ведення логів для полегшення адміністрування контейнерів програмних додатків.

Так як інструментарій планується як окремий модуль Apptimized Workspace Launcher, то він не матиме власного інтерфейсу. Проте, після впровадження, результатом роботи будуть контейнеризовані додатки, з попередньо налаштованими конфігураціями переадресації викликів та загальних властивостей додатку, що будуть звертатися до обробника бізнес логіки, яка також буде частиною лаунчера. Таким чином, розроблений інструментарій не потребуватиме окремої утиліти обробки логіки, а використовуватиме вже наявний на всіх системах в робочому середовищі корпоративний додаток.

Для досягнення поставленої мети необхідно вирішити наступні задачі:

- обрати загальну парадигму пакування додатків та технологію обробки системних запитів;
- визначити функціонал інструментарію;
- обрати технічні засоби для розробки;
- обрати модель зберігання конфігурацій переадресацій;
- розробити код інструментарію;
- інтегрувати інструментарій в Apptimized Workspace Launcher;
- провести тестування інструментарію.

Задля ефективного виконання задач проекту було проведено планування робіт, результати якого наведені у додатку Б.

2.2 Вибір засобів реалізації

Проаналізувавши предметну область і порівнявши існуючі програмні рішення для контейнеризації та віртуалізації програмних продуктів було прийнято рішення про використання механізму переадресації системних викликів для забезпечення умови контейнеризації програмних додатків у окремій директорії в рамках файлової системи операційної системи кінцевого користувача.

В даному випадку, для реалізації механізму переадресації системних викликів необхідно використати систему моніторингу процесів. Так як написання власної подібної системи є достатньо масштабним проектом та не є реалізацією основної логіки, власне, редагування параметрів системних викликів, було прийнято рішення про використання інтерфейсу прикладного програмування EasyHook API [24].

Для взаємодії інструментарію з операційною системою використовується .NET Framework [25] через його мобільність та підтримку великої кількості методів для роботи з ОС. Також, інтерфейс EasyHook підтримує роботу зі збірками .NET, тож у випадку використання даного набору інструментів мінімізує виникнення помилок сумісності через підтримку розробника API стандартів фреймворку.

У якості мови розробки інструментарію було обрано об’єктно-орієнтовану мову C#. Основними перевагами використання обраної мови є підтримка роботи з .NET Framework, автоматична збірка сміття та механізми роботи зі сторонніми компонентами, що можуть бути представлені бібліотеками, API, зовнішніми проектами у вигляді набору файлів або ж як NuGet пакети.

Інтеграція NuGet пакетів у коді програми дозволяє з легкістю підключити та використовувати як елементи з вільними ліцензіями, так і з закритих депозитаріїв [26]. Для розроблюваного інструментарію у вигляді NuGet пакетів імплементовано такі модулі як: EasyHook-для моніторингу системних викликів; CommandLineParser – для реалізації перевірки та обробки аргументів виконуваного файлу [27]; та Serilog – логування процесів роботи інструментарію [28].

Враховуючи вищеописані інструменти, в якості середовища розробки обрано Microsoft Visual Studio. Вибір середовища був також обґрунтований наявністю вбудованого менеджеру роботи з NuGet пакетами, модулем для використання розподіленої системи контролю версій Git [29] та зручним інтерфейсом користувача.

За зберігання налаштувань додатків в контейнері відповідає маніфест файл стандарту XML. Враховуючи простоту редагування та достатність функціоналу, використання єдиного XML файлу є оптимальнішим за базу даних. Для роботи з маніфест файлом використовується клас XmlSerialized, що дозволяє зберігати дані у об’єкті даного класу для швидкого доступу при необхідності [30].

3 МОДЕЛЮВАННЯ ІНСТРУМЕНТАРІЮ КОНТЕЙНЕРИЗАЦІЇ ПРОГРАМНИХ ДОДАТКІВ

Інструментарій – єдина утиліта, набір програм або програмних інструментів, які використовуються для розробки та обслуговування інших додатків або баз даних[31]. В даному випадку, інструментарій що розробляється є саме таким, що підтримує та доповнює роботу сторонніх додатків. Сьогодні зазвичай використовуються локально встановлені версії програмного забезпечення необхідного для здійснення професійної діяльності DevOps інженера, або ж їх портативні версії, що створює певні незручності.

Створення інструментарію контейнеризації дозволить мінімізувати взаємодію з елементами операційної системи загального користування, що в свою чергу є вагомою перевагою для професійного використання в порівнянні з локально встановленим програмним забезпеченням. Впровадження та використання даного інструментарію дозволить зменшити ризики втрати необхідних елементів операційної системи в створюваних пакетах ПЗ, що означає підвищення загальної якості сервісу пакування.

3.1 Структурно-функціональне моделювання

Методологія IDEF0 є засобом аналізу та опису бізнес-процесів. Ідея IDEF0 полягає в тому, що бізнес-процес відображено у вигляді прямокутного блоку, до якого входять та з якого виходять стрілки. Блоки відображають процеси, операції чи завдання. Окрім блоків важливе значення мають сторони процесу та пов’язана з нею стрілка:

- вхід бізнес-процесу (зліва) – інформація, яка буде перетворена в ході виконання процесів;

- вихід бізнес-процесу (справа) – оброблена та перетворена інформація;
- управління бізнес-процесу (зверху) – інформація, яка визначає як саме має виконуватися бізнес-процес та як відбуватиметься перетворення входу на вихід;
- механізм бізнес-процесу (знизу) – те, що перетворює вхід у вихід: співробітники, техніка, тощо.

Для побудови діаграми IDEF0 «Робота інструментарію контейнеризації програмних додатків», було визначено наступний перелік даних:

- вхідні дані: директорія з файлами додатку, маніфест-файл налаштувань додатку, точка входу програмного додатку;
- вихідні дані: працездатність контейнеризованого додатку;
- управління процесу: принципи dll хукінгу, принципи роботи з CommandLineParser, принципи роботи з системними dll бібліотеками, а також принципи роботи з xml файлами;
- механізми процесу: Launcher, EasyHook, CommandLineParser, kernel32.dll, advapi32.dll, System.Xml.

Контекстна діаграма IDEF0 представлена на рис. 3.1.

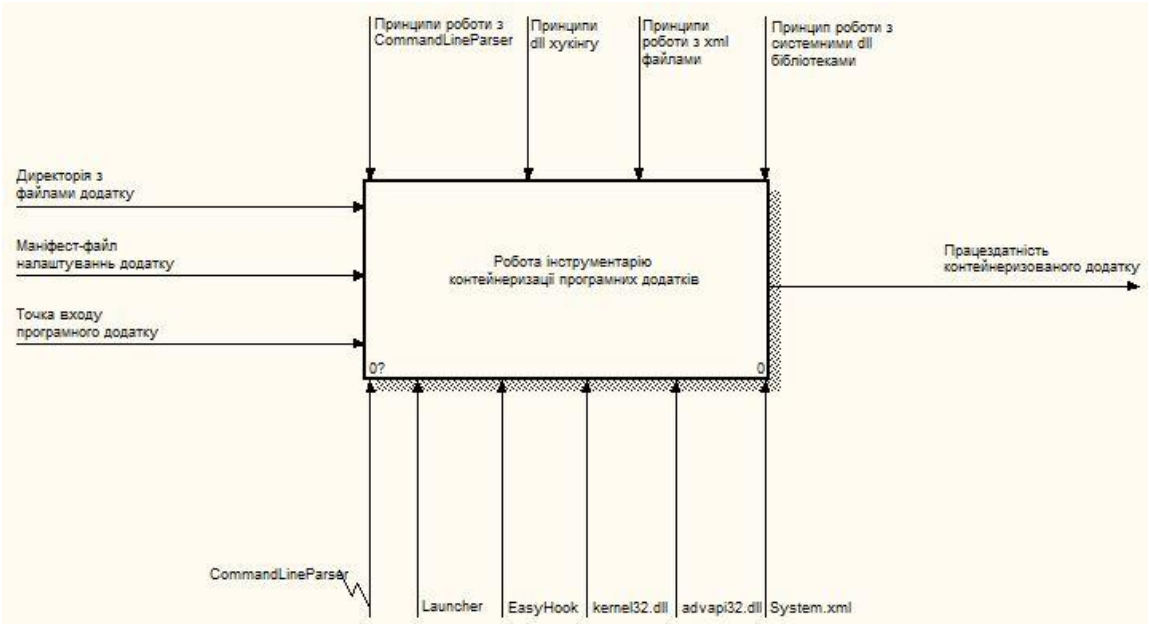


Рисунок 3.1 – Контекстна діаграма роботи інструментарію контейнеризації у нотації IDEF0

Для більш повного розуміння логіки роботи інструментарію, необхідно виконати декомпозицію.

Діаграму було розділено на три процеси:

- запуск основного процесу програмного додатку;
- встановлення dll хуків;
- переадресація системних викликів.

Для першого етапу було визначено наступний перелік даних:

- вхідні дані: точка входу програмного додатку, директорія з файлами додатку;
- вихідні дані: параметри процесу;
- управління процесу: принципи роботи з CommandLineParser;
- механізми процесу: Launcher, CommandLineParser.

Для другого етапу було визначено наступний перелік даних:

- вхідні дані: параметри процесу;
- вихідні дані: системний запит;
- управління процесу: принципи dll хукінгу;
- механізми процесу: EasyHook.

Для третього етапу було визначено наступний перелік даних:

- вхідні дані: системний запит;
- вихідні дані: працездатність контейнеризованого додатку;
- управління процесу: принципи dll хукінгу, принципи роботи з xml файлами, принципи роботи з системними dll бібліотеками;
- механізми процесу: EasyHook, kernel32.dll, advapi32.dll, System.Xml.

Діаграма декомпозиції роботи інструментарію представлена на рис. 3.2.

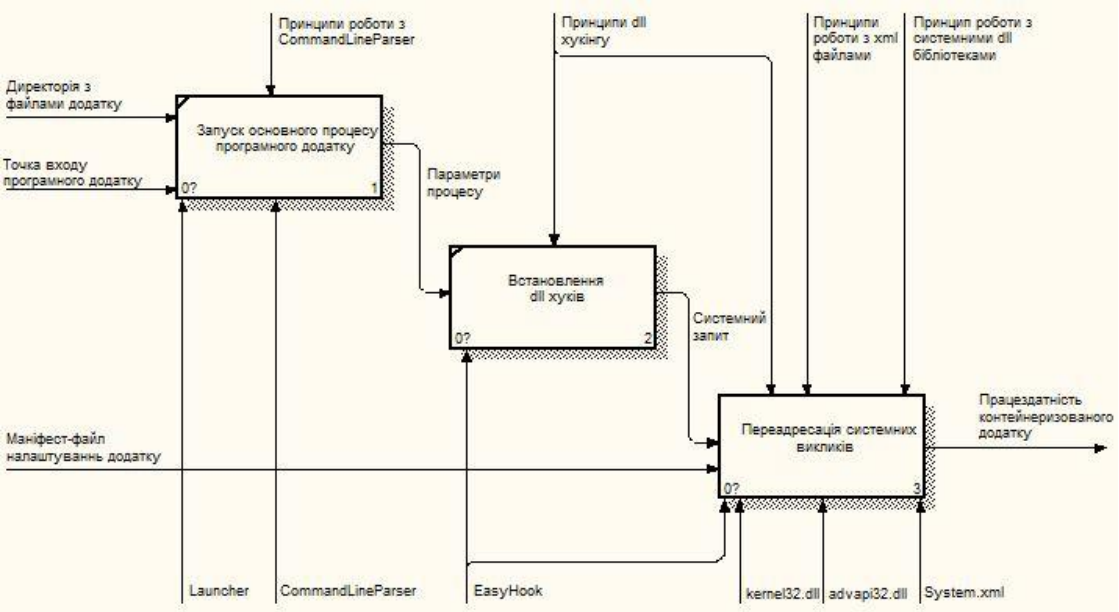


Рисунок 3.2 – Діаграма декомпозиції роботи інструментарію контейнеризації у нотації IDEF0

Для кращого розуміння процесу переадресації системних викликів проведемо декомпозицію даного етапу. Розділимо його також на три етапи:

- обробка вхідних параметрів системного запиту;
- пошук та обробка запитуваного системного елемента в маніфест файлі;
- виконання модифікованого системного виклику.

Для процесу обробки вхідних параметрів системного запиту визначено наступні дані:

- вхідні дані: системний запит;
- вихідні дані: параметри системного виклику;
- управління процесу: принципи dll хукінгу;
- механізми процесу: EasyHook.

Після того як інструментарій «спіймає» один із необхідних системних викликів, необхідно перевірити чи належить системний елемент що викликається до нашого програмного додатку. Цей процес забезпечено в наступному процесі - пошук та обробка запитуваного системного елемента в маніфест файлі, для якого визначено наступні дані:

- вхідні дані: параметри системного виклику;

- вихідні дані: модифіковані параметри системного виклику;
- управління процесу: принципи роботи з xml файлами;
- механізми процесу: System.Xml.

Після обробки параметрів системного виклику, у випадку якщо цільовий елемент системи має конфігурацію в маніфест-файлі, генеруємо набір параметрів виклику системної операції, який буде використовуватись в якості вхідних даних для наступного етапу роботи інструментарію.

Для процесу виконання модифікованого системного виклику:

- вхідні дані: модифіковані параметри системного виклику;
- вихідні дані: працездатність контейнеризованого додатку;
- управління процесу: принципи роботи з системними dll бібліотеками;
- механізми процесу: kernel32.dll, advapi32.dll.

Діаграма декомпозиції процесу переадресації системних викликів представлена на рис. 3.3.

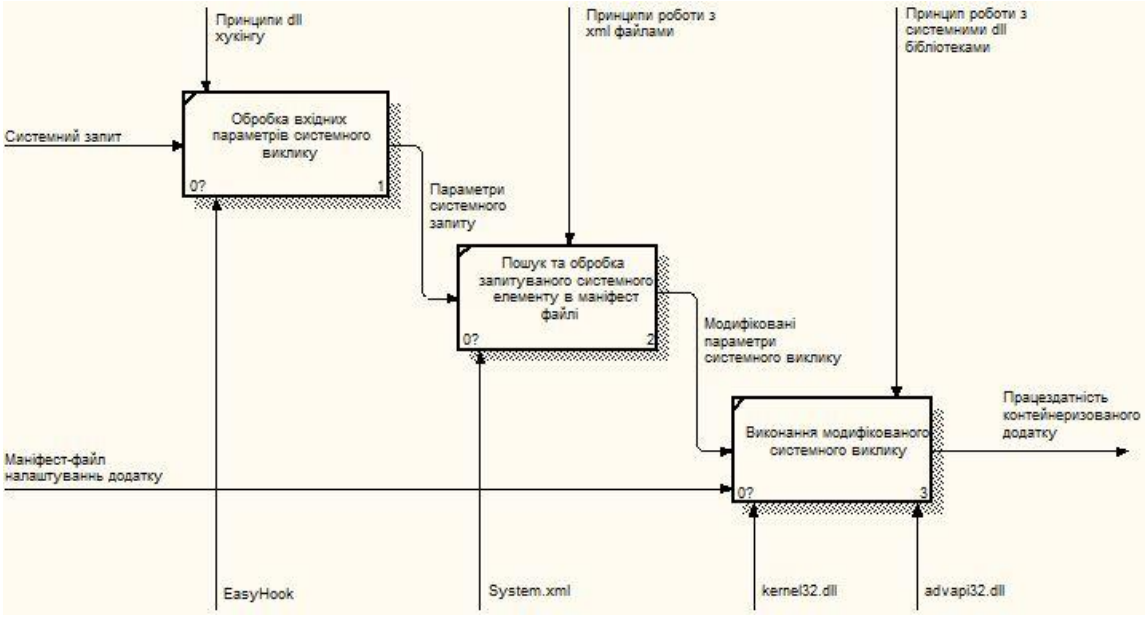


Рисунок 3.3 – Діаграма декомпозиції процесу переадресації системних викликів у нотації IDEF0

3.2 UML-моделювання

Діаграма варіантів використання (Use Case diagram) використовується для відображення взаємозв’язків між акторами та варіантами використання. Під акторами мається на увазі множина логічно пов’язаних ролей, що взаємодіють з прецедентами або сутностями. Актором може бути як людина, так і інша система, підсистема або клас, які представляють щось ззовні даної системи (сутності). Прецедент, або варіант використання – це опис множини послідовних подій, що виконуються в системі та призводять до певного результату. Варіант використання показує поведінку сутності, описуючи взаємодію між учасниками та системою. Прецедент не вказує на те, як досягається певний результат, а лише відображає що саме виконується.[32]

Для побудови діаграми варіантів використання інструментарію контейнеризації програмних додатків було визначено наступних акторів:

- Launcher – зовнішній програмний додаток, що буде використовувати інструментарій контейнеризації;
- файлова система – файлова система на комп’ютері чи віртуальній машині користувача, на якому зберігаються файли програмних додатків та маніфест-файл з налаштуваннями;
- модуль роботи з XML файлом конфігурації – модуль обробки маніфест-файлу з налаштуваннями контейнеризації для даного програмного додатку;
- EasyHook – бібліотека для роботи з системними викликами;
- ядро ОС – ядро операційної системи Windows.

Для визначених акторів можливі наступні варіанти використання:

- запуск основного процесу додатку – запуск додатку, що контейнеризовано, використовуючи точку входу;
- встановлення dll хуків – імплементація механізму моніторингу системних викликів в створений процес;

- обробка системних запитів – обробка та зберігання параметрів системних викликів;
- обробка маніфест файлу налаштувань – робота з xml маніфест-файлом для збору даних стосовно виклику конкретного системного елемента;
- dll ін’єкція до цільового системного виклику – виконання системного запиту з модифікованим заголовком відповідної dll бібліотеки.

Також, враховуючи наявність файлу налаштувань, додамо на діаграму артефакт для відображення його взаємодії з системою інструментарію.

Використовуючи дані визначених акторів, варіантів використання та артефакту було розроблено Use Case діаграму, рис. 3.4.

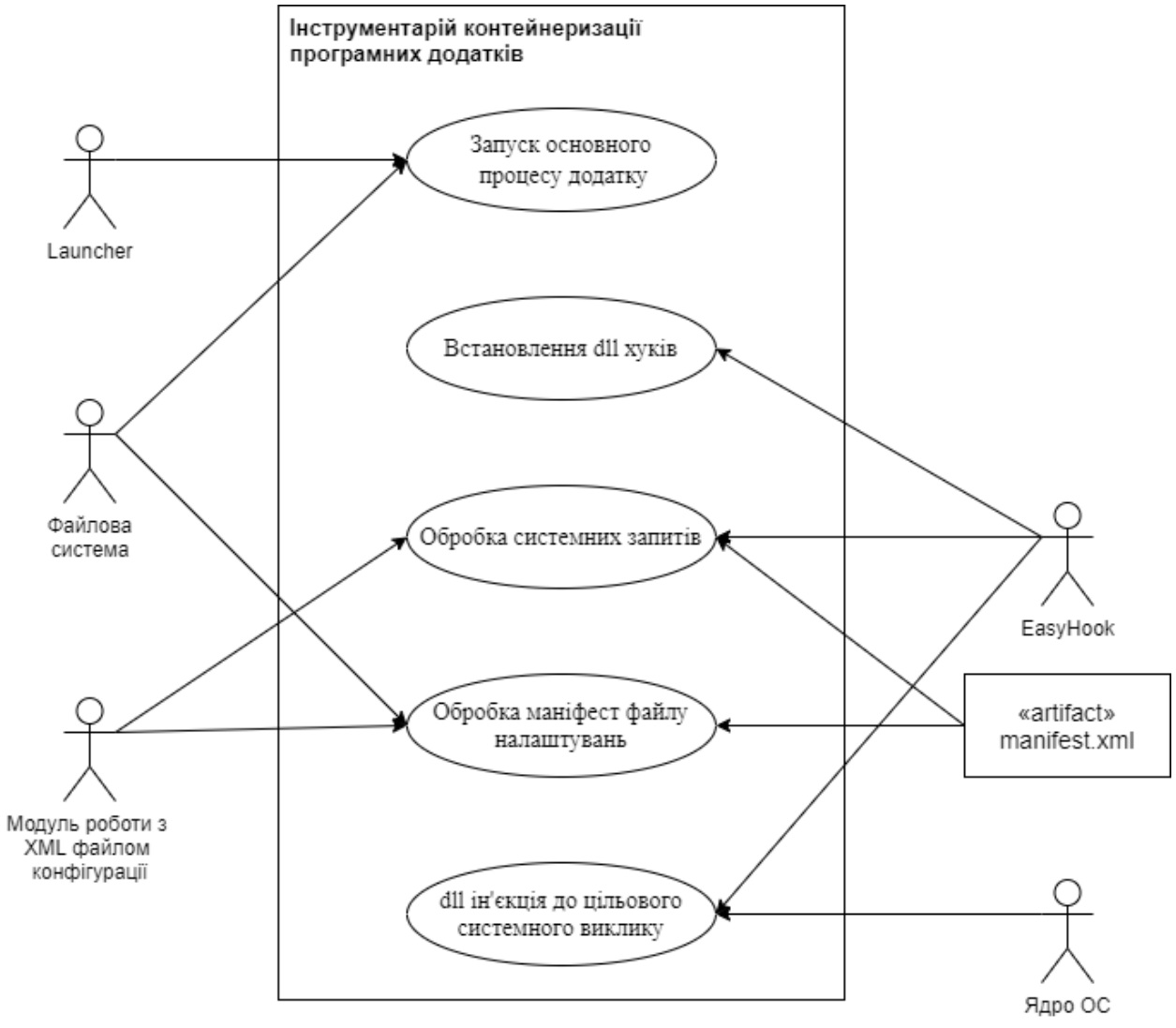


Рисунок 3.4 – Діаграма варіантів використання

Оскільки розроблюваний інструментарій не матиме користувацького інтерфейсу, додатково була розроблена діаграма послідовності з метою уточнення діаграми варіантів використання для більш детального відображення взаємодії між об’єктами в системі. Діаграма послідовності (Sequence diagram) зазвичай містить акторів, об’єкти, які взаємодіють в рамках сценарію, повідомлення, якими вони обмінюються та результати що повертаються [33].

Для побудови діаграми послідовності роботи інструментарію контейнеризації програмних додатків визначено актора – Launcher – зовнішній додаток, що використовується для запуску основного батьківського процесу контейнеризованого додатку.

У діаграмі були визначені наступні об’єкти:

- CommandLineParser – модуль обробки введення параметрів запуску процесу;
- Контейнеризований додаток – сутність, що містить файли програмного додатку та файли необхідні для роботи розробленого інструментарію;
- Делегат – клас та його об’єкти, що використовуються для обробки системних викликів;
- Manifest.xml – маніфест файл з налаштуваннями додатку;
- Kernel32.dll/advapi.dll – системні dll бібліотеки, що використовуються операційною системою для забезпечення зв’язків між ОС та її об’єктами (в даному випадку, об’єкти файлової системи, записи реєстру та процеси).

Між визначеними об’єктами відбувається обмін наступними повідомленнями:

- Запуск основного процесу;
- Встановлення dll хуків;
- Системний запит;
- Пошук та обробка даних;
- Перенаправлений системний запит.

Результатами, що повертаються є:

- Результат обробки даних;

- Результат виконання системного запиту;
- Видалення хуків та вихід з програмного додатку.

Враховуючи специфіку роботи інструментарію, певні дії виконуються асинхронно, що позначено на діаграмі блоком циклу.

Використовуючи дані визначених акторів, об’єктів, повідомлень та результатів було розроблено діаграму послідовності, рис. 3.5.

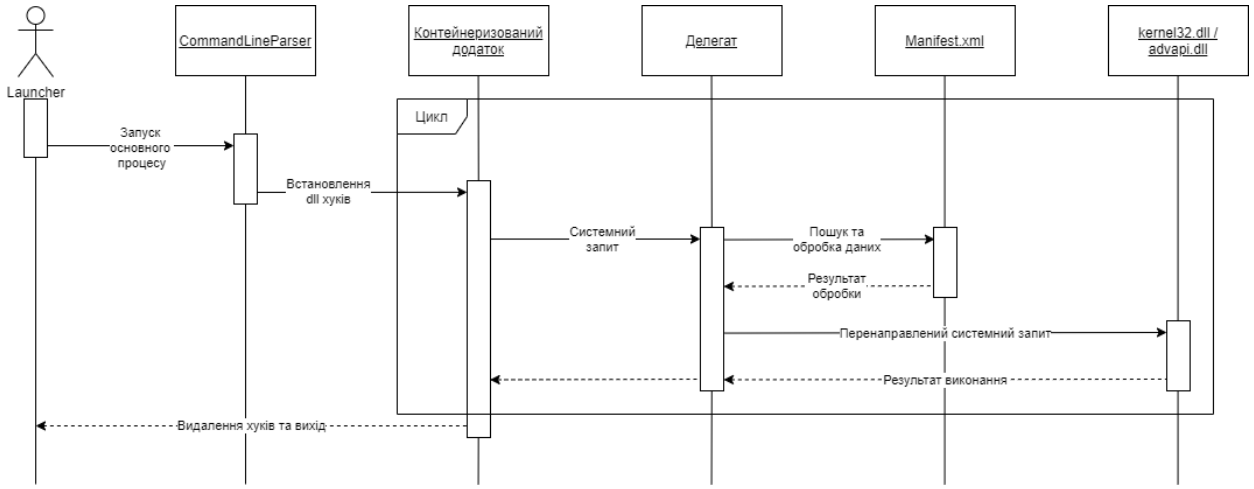


Рисунок 3.5 – Діаграма послідовності

3.3 Структура інформаційної моделі

Контейнер у складі розробленого інструментарію представляє собою директорію з файлами програмного додатку та маніфест-файл з необхідними конфігураціями. Щодо файлів, у контейнері зберігаються необхідні для коректної роботи набір файлі зберігаючи відповідну структуру директорій як при звичайному встановленні для спрощення навігації у випадку переадресації виклику файлів. Також, окрім файлів програм, у випадку необхідності, рекомендовано створити додаткову папку для файлів зовнішніх залежностей, які не обов’язково встановлювати на основну систему, а можна зберігати в контейнері та звертатися до них за необхідністю. Після розгортання, або ж встановлення контейнеру в робочому просторі користувача, файли контейнеру зберігаються в користувацькій директорії у папці лаунчера та не будуть взаємодіяти з файловою системою загального користування.

Для зберігання налаштувань контейнеризації використовується маніфест-файл формату XML. У даному випадку використання одного конфігураційного файлу є оптимальнішим, ніж використовувати бази даних через очевидну зручність та гнучкість налаштування необхідних параметрів, а також, найбільш важливо, для швидкості та легкості доступу до даних з будь-якої робочої станції у мережі. Значно швидше отримати дані з локального текстового файлу, ніж з бази даних в мережі, тож враховуючи важливість швидкого доступу до даних у випадку роботи з системними викликами це прискорить роботу контейнеризованих додатків в порівнянні з використанням БД.

Налаштування додатку зберігаються у файлі manifest.xml, який зберігається в контейнері поруч із директорією з його файлами. Структура конфігураційного файлу представляє собою систему тегів за принципом використання масивів для об’єднання тегів одного класу. Наприклад, існує тег <RegistryKey> для зберігання даних реєстру, оскільки їх зазвичай деяка кількість, то для зручної навігації по маніфест-файлу, вони об’єднані в умовний масив під тегом <RegistryKeys>.

Так як кожний тег відповідає за певний елемент ОС, очевидно що необхідно зберігати деякі властивості цих елементів. Для цього використовуються атрибути всередині відповідного тегу. Наприклад, для тегу <RegistryKey> існують наступні атрибути:

- Key – шлях ключа реєстру (наприклад, HKLM\SOFTWARE\7-Zip);
- Value – ім’я значення під вказаним ключем (наприклад, Path);
- Type – тип значення (наприклад, String).

Окрім атрибутів для зберігання основних даних для відповідного тегу використовується його значення (наприклад, C:\Program Files\7-Zip\). Отже, для наведеного вище прикладу маємо наступну структуру, зображену на рис. 3.6.

```
<RegistryKeys>
  <RegistryKey Key="HKLM\SOFTWARE\7-Zip" Value="Path" Type="String">C:\Program Files\7-Zip\</RegistryKey>
</RegistryKeys>
```

Рисунок 3.6 – Приклад використання xml структури у файлі налаштувань

Для забезпечення функціоналу переадресації системних викликів використано тег <Redirection>. Розглянемо більш детально структуру даного компонента за його атрибутами:

- Type – тип об’єкту що потребує переадресації (наприклад, директорія, файл чи реєстр);
- From – місце розташування елемента на хостовій системі при звичайній установці програмного додатку (наприклад, C:\Program Files\7-Zip\7z.exe);
- To – цільове місце розташування елемента всередині контейнера (наприклад, C:\Users\Alex\Launcher\PackageName\Files\Program Files\7-Zip\7z.exe).

Також, важливим для переадресації є тег <RegistryVirtualRoot> що містить в собі значення, яке відповідає за ключ реєстру для даного контейнеризованого додатку (наприклад, “HKCU\Software\Apptimized\Launcher\PkgName”) у який буде перенаправлено всі ключі реєстру, необхідні для роботи додатку, вказані у тегу

<RegistryKeys>.

Окрім функції переадресації системних викликів для коректної роботи програмних додатків можуть бути використані додаткові елементи, для зберігання яких використано наступні теги:

- <ProgramEntry> - для зберігання точок входу програмного додатку;
- <Shortcut> - для зберігання властивостей ярликів, які можуть використовувати як дані з <ProgramEntry>, так і виконувані файли поза контейнером;
- <Service> - для зберігання сервісів ОС Windows;
- <EnvironmentVariable> - для зберігання змінних оточення;
- <Scripts> - для зберігання інформації стосовно зовнішніх скрипт-файлів.

4 РОЗРОБКА ІНСТРУМЕНТАРІЮ КОНТЕЙНЕРИЗАЦІЇ ПРОГРАМНИХ ДОДАТКІВ

4.1 Розробка класу Manifest для зберігання конфігурацій

Для зберігання конфігурацій програмних додатків використовується xml маніфест-файл з описаною в розділі 3.3 «Структура інформаційної моделі» структурою. Для обробки даних з файлу було створено набір класів для забезпечення серіалізації даних – процесу перетворення об’єкта в потік байтів для зберігання об’єкта або передачі його в пам’ять, базу даних чи файл [34].

Під час програмної реалізації процесу серіалізації визначено основний клас Manifest, що містить атрибути, які, в свою чергу, зберігатимуть відповідні компоненти (теги) з xml файлу. Для підтримки роботи даного процесу використовується простір імен System.Xml.Serialization [35], що містить колекцію визначених класів для визначення та зберігання елементів XML формату, таких як: компоненти, масив тегів, атрибути, значення компонентів.

Клас XmlSerializer з описаного простору імен дозволяє серіалізувати та десеріалізувати об’єкти в xml документи та з них. Серіалізація дозволяє розробнику зберігати стан об’єкта та заново створювати його за потреби, забезпечуючи зберігання об’єктів, а також обмін даними. В даному випадку, перевагою використання процесу серіалізації є нормалізація та умовне спрощення роботи з файлом конфігурації, а також, скорочення часових та ресурсних витрат на обробку необхідних даних порівняно з обробкою xml файлу за допомогою звичайного синтаксичного розбору.

Перед програмною реалізацією класу Manifest необхідно звернути увагу на приблизну структуру та зміст файлу “manifest.xml”, що наведені на рис. 4.1.

```xml
<?xml version="1.0"?>
<WorkspacePackage xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" Id="redirect.test" Vendor="Apptimized"
                                                                                                                                                    Name="APEngine" Version="1.0.0">
  <Description>Application Packaging Engine</Description>
  <Icons>
    <Icon Width="130" Height="120" Location="Icons\Program.ico" />
  </Icons>
  <Scripts>
    <Script WorkingDirectory="Scripts\sample.ps1" Type="PowerShell" Sequence="PostInstallation" Elevated="false" Is32="true" />
  </Scripts>
  <Shortcuts>
    <Shortcut Id="ape" Location="%ProgramMenu%\Program" Command="Program\run.exe" Arguments="\r" Elevated="false">Shortcut to run app</Shortcut>
  </Shortcuts>
  <Categories>
    <Category>Category1</Category>
    <Category>Category2</Category>
  </Categories>
  <EnvironmentVariables>
    <EnvironmentVariable Name="TestVar" Type="Value" Value="value" />
  </EnvironmentVariables>
  <RegistryKeys>
    <RegistryKey Key="HKLM\Software\Apptimized\APE" Value="AutoUpdate" Type="String">Disabled</RegistryKey>
    <RegistryKey Key="HKCU\Software\Microsoft\TestKey" Value="User" Type="DWord">#0</RegistryKey>
  </RegistryKeys>
  <RegistryVirtualRoot>HKEY_CURRENT_USER\Software\Apptimized\Launcher\RedirectedApp</RegistryVirtualRoot>
  <Redirections>
    <Redirection Type="File" From="C:\!temp\testfile.txt" To="%DefaultDir%\Files\fileRedirect\testfile.txt" />
    <Redirection Type="File" From="%ProgramFiles%\Apptimized\Engine\APE.exe" To="Files\Apptimized\Engine\APE.exe" />
    <Redirection Type="Folder" From="%ProgramData%\Software" To="Files\ProgramData\Software" />
    <Redirection Type="Folder" From="C:\!temp\123" To="%DefaultDir%\Files\123" />
    <Redirection Type="Folder" From="%ProgramData%\Citrix" To="Files\ProgramData\Citrix" />
    <Redirection Type="Registry" From="HKEY_LOCAL_MACHINE\Software\Apptimized" />
    <Redirection Type="Registry" From="HKEY_CURRENT_USER\Software\Apptimized\Test" />
    <Redirection Type="Registry" From="HKEY_CURRENT_CONFIG\New Key #1" />
  </Redirections>
  <ProgramEntries>
    <ProgramEntry Id="id" Command="Files\Engine\APE.exe" Arguments="/run" Elevated="false">Run program entry</ProgramEntry>
  </ProgramEntries>
  <Services>
    <Service Name="tstserv" DisplayName="Test service" StartType="Auto" StartOnDeploy="false" Type="Win32">Files\lalal\servicefilepath\file.exe</Service>
  </Services>
</WorkspacePackage>
```

Рисунок 4.1 – Зміст файлу “manifest.xml”

Для елементів, що є одиничними (не є частиною масиву таких же об’єктів) або таких, що можуть бути описані стандартними типами даних, достатньо визначити атрибути класу Manifest, наприклад, такими є компоненти: Description (текстове описання програмного додатку, що контейнеризується), RegistryVirtualRoot (є атрибутом типу рядок), а також перелік атрибутів компонента WorkspacePackage, які також мають рядковий тип даних.

Що стосується інших елементів маніфест-файлу, для них визначені власні класи з необхідною структурою даних та відповідними атрибутами. Наприклад, основною необхідною інформацією для інструментарію контейнеризації є дані з тегу Redirections. Цей компонент є масивом об’єктів класу Redirection, який має власні атрибути Type (тип системного елемента що переадресовується), From (адреса елемента за якою до нього звертається системний виклик) і To (адреса, на яку необхідно переадресувати системний виклик).

Атрибути To та From мають рядковий тип даних, а Type приймає значення з власного перелічуваного типу даних RedirectionType, який, в даному випадку, містить наступні типи: File, Folder, Registry для ідентифікації системних елементів та відповідної їх обробки під час використання інструментарію.

Повний програмний код класу Manifest наведено у додатку В.

4.2 Розробка логіки роботи інструментарію

Після розробки моделі зберігання та обробки конфігураційних даних для програмних додатків переходимо до реалізації механізму переадресації системних викликів. Для цього спочатку необхідно застосувати бібліотеку EasyHook для моніторингу та перехоплення системних викликів з метою їх подальшого використання.

Для зберігання файлів EasyHook та інших спільних ресурсів що будуть використовуватися для забезпечення працездатності інструментарію контейнеризації створено проект SystemMonitor.Shared. Файли бібліотеки моніторингу викликів було додано до проекту за допомогою NuGet менеджер, завантаживши відповідний пакет.

Також, окрім зазначених файлів, даний проект містить власні перелічувані типи даних для зберігання системних операцій файлової системи (FileSystemOperationType), реєстру (RegistryOperationType) та процесів (ProcessOperationType), класи-інтерфейси для забезпечення поліморфізму при обробці відповідних подій файлової системи (FileSystemMonitorEvent), реєстру (RegistryKeyMonitorEvent, RegistryKeyValueMonitorEvent) і процесів (ProcessMonitorEvent), що наслідують публічний інтерфейс ISystemMonitorEvent.

Клас SystemMonitorLogger відповідає за створення сутності класу ILogger з NuGet пакету Serilog для забезпечення процесу логування операцій моніторингу та переадресації системних викликів, які зазначено у відповідних перелічуваних типах даних, описаних вище.

Останній клас даного проекту – SystemMonitorInterface. Він є інтерфейсом для використання зазначених ресурсів поза межами даного проекту, а також, водночас, має метод Ping для отримання статусу основного процесу контейнеризованого додатку, до якого були встановлені хуки системних викликів. Допоки інструментарій має доступ до даного методу виконується даний метод, якщо він досяжний отже існують об’єкти, що використовують цей клас та його об’єкти, тобто

завдяки даному методу інструментарій ідентифікує необхідність про завершення своєї роботи. Структуру проекту зображено на рис. 4.2.

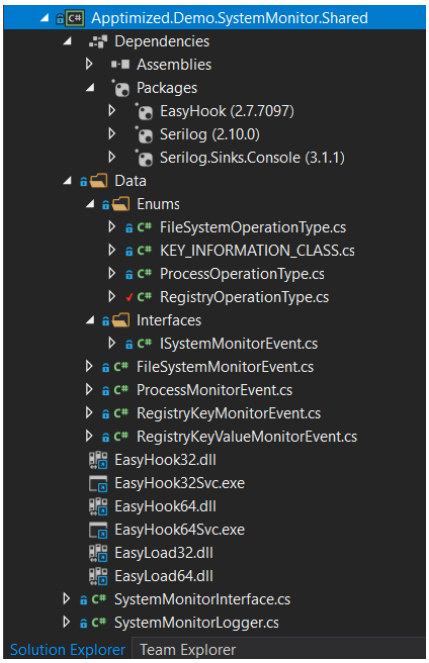


Рисунок 4.2 – Структура проекту “SystemMonitor.Shared”

Наступним кроком розробки було створено новий проект SystemMonitor.Hook, в якому описано логіку роботи основного функціоналу інструментарію контейнеризації. Перш за все, даний проект залежить від вище описаного проекту SystemMonitor.Shared та має доступ до його ресурсів, а також до проекту Launcher.Library, в рамках якого було реалізовано клас Manifest та класи від яких він залежить.

Проект SystemMonitor.Hook є відповідальним за забезпечення функціоналу моніторингу системи, встановлення хуків на певні елементи системи, обробки системних запитів в залежності від конфігураційного маніфест-файлу та виконання початкового системного запиту з модифікованими параметрами.

Перш за все, необхідно описати які саме системні виклики можливі в контексті роботи даного інструментарію. Виклики описані у вигляді статичних зовнішніх методів для зазначених dll бібліотек kernel32.dll та advapi32.dll у відповідних файлах класів Advapi32.cs та Kernel32.cs.

Наприклад, на рис. 4.3 зображено ініціалізацію функцій бібліотеки kernel32.dll для роботи з викликами операцій по роботі з елементами файлової системи, а саме створення файлу, видалення файлу, створення директорії та її видалення. Як можна побачити, кожна операція має власні параметри, що визначають сутність системного виклику. Всі використані зовнішні методи описані в документації до файлів-заголовків fileapi.h [36] та winreg.h [37] та нормалізовані для їх використання в контексті інструментарію та враховуючи відмінність у типізації даних мов C++ та C#.

```
[DllImport("kernel32.dll", CharSet = CharSet.Unicode, SetLastError = true,
    CallingConvention = CallingConvention.StdCall)]
1 reference
public static extern IntPtr CreateFileW(string filename, uint desiredAccess, uint shareMode,
    IntPtr securityAttributes, uint creationDisposition, uint flagsAndAttributes, IntPtr templateFile);

[DllImport("kernel32.dll", CharSet = CharSet.Unicode, SetLastError = true,
    CallingConvention = CallingConvention.StdCall)]
1 reference
public static extern IntPtr DeleteFileW(string filename);

[DllImport("kernel32.dll", CharSet = CharSet.Unicode, SetLastError = true,
    CallingConvention = CallingConvention.StdCall)]
1 reference
public static extern IntPtr CreateDirectoryW(string dirname, IntPtr securityAttributes);

[DllImport("kernel32.dll", CharSet = CharSet.Unicode, SetLastError = true,
    CallingConvention = CallingConvention.StdCall)]
1 reference
public static extern IntPtr RemoveDirectoryW(string dirname);
```

Рисунок 4.3 – Фрагмент файлу “Kernel32.dll”

Для опису хуків подій операційної системи використано внутрішній абстрактний клас AbstractBaseHook, що наслідує клас IDisposable, який в свою чергу дає можливість використовувати спільний виклик методу Dispose для звільнення виділеної пам’яті та видалення хуків з процесів після закінчення роботи інструментарію [38].

Клас AbstractBaseHook описує чергу з об’єктів класу ISystemMonitorEvent з проекту SystemMonitor.Shared та список з хуків класу LocalHook бібліотеки EasyHook у якості атрибутів класу. Стосовно методів даного класу, окрім конструктора, існують методи встановлення хуків InstallHook та InstallUniversalHook (що використовує InstallHook для встановлення хуків зважаючи

на різні кодування Ansi та Unicode, в залежності від операції що викликається), а також метод Dispose, описаний вище.

Від класу AbstractBaseHook наслідуються класи ProcessHook, FileSystemHook та RegistryHook. Клас ProcessHook використовується для моніторингу та хукінгу операції по створенню дочірніх процесів (CreateProcess) для встановлення хуків у процеси, що створюються основним процесом контейнеризованого додатку для подальшого моніторингу його системних викликів. Для цього використовується ім’я каналу вже створеного IPC сервісу – серверу міжпроцесорної взаємодії задля використання єдиного екземпляру об’єкта для логування результатів роботи інструментарію.

Встановлення dll хуку відбувається у конструкторі даного класу, який, в свою чергу, використовує делегат, що визначає необхідні для цього дії. Процес делегування реалізовано у делегаті CreateProcessDelegate, який вказує на власний метод CreateProcessHook, що виконує обробку вхідних параметрів початкового системного виклику створення процесу, де за допомогою регулярних виразів з повної команди виділяється ім’я процесу та окремо від нього необхідні параметри запуску. Далі виконується метод CreateAndInject тієї ж бібліотеки EasyHook для створення процесу з ін’єктованим обробником IPC сервісу замість звичайного запуску процесу.

Аналогічно, використовуючи принцип делегатів реалізовано класи FileSystemHook та RegistryHook для роботи з системними викликами операцій файлової системи та реєстру відповідно, проте відрізняється принцип обробки подій, що делегуються.

Перш за все, у конструкторах даних класів виконується метод LoadFromFile класу Manifest для серіалізації даних з xml файлу налаштувань до об’єкту даного класу.

У класі FileSystemHook для обробки системних запитів використовуються методи GetRedirectedFilePath та GetRedirectedFolderPath для перевірки на необхідність переадресації файлу чи директорії, залежно від запиту. Алгоритм їх роботи схожий, проте є певні відмінності. Для перевірки необхідності переадресації

виклику директорії порівняння власне директорії та її батьківських каталогів відбувається для тих записів перенаправлення, тип яких Folder. Якщо в результаті перевірки директорія співпадає зі вказаною в атрибуті From, отже існує необхідність заміни параметру виклику і метод повертає повний шлях до директорії з атрибуту To відповідного компонента Redirection маніфест-файлу.

Під час перевірки файлу на необхідність заміни параметрів використовуються компоненти Redirection з типом як File, так і Folder. Для перевірки файлу достатньо порівняти його реальний шлях зі значенням відповідного атрибуту, а для перевірки його приналежності до директорії, що переадресовується використовується алгоритм схожий на метод GetRedirectedFolderPath, проте з певними відмінностями пов’язаними з іменуванням файлів та побудовою шляху, що повертається методом.

Для обох методів реалізовано розв’язання повного шляху за допомогою методу ExpandEnvironmentVariables системного класу Environment. Це необхідно для запобігання зберігання в файлі конфігурації додатку шляху до елемента файлової системи, що містить жорстке кодування (hardcode, наприклад, «C:\Program Files» на комп’ютері з німецькою локалізацією та системним томом D матиме вигляд «D:\ Programme»). Використання даного методу дозволяє використовувати загальноприйняті змінні оточення, %ProgramFiles% для наведеного прикладу.

У процесі реалізації переадресації викликів реєстрових операцій загальний підхід до використання делегатів було збережено, проте є різниця у параметрах викликів. Якщо для об’єктів файлової системи одним із параметрів використовувався повний шлях до файлі або директорії, то під ас звернення до реєстру нам необхідно враховувати два параметри. Перший – вказівник на кореневий елемент реєстру, що може вказувати як на один із предвстановлених системою (такі як HKEY_LOCAL_MACHINE, HKEY_CURRENT_USER та інші), або ж містити батьківський ключ (наприклад, \REGISTRY\MACHINE\SOFTWARE\Manufacturer\ApplicationName). Другий корисний параметр виклику – власне ключ, який може бути як складеним ключем, так і одиничним.

Спочатку був розроблений метод визначення реального кореневого розділу реєстру з вказівника у методі KeyHandleToString. Наступним кроком був розроблений метод SetRedirectedKey в якому реалізовано механізм перевірки необхідності переадресації та подальшої обробки параметрів виклику. В даному методі викликається метод KeyHandleToString щоб одержати початковий ключ реєстру у зрозумілому для людини строковому вигляді. Далі відбувається пошук відповідності строкового вигляду ключа до компонентів Redirection з типом Registry маніфест-файлу.

Перед обробкою файлу конфігурації враховано ймовірність обробки як предвстановленого вигляду кореневого ключа, так і машинного вигляду, а також обробка ключів з розділу \REGISTRY\USER\SID, де SID – унікальний ідентифікатор конкретного користувача. Якщо SID ключа що викликається співпадає з ідентифікатором поточного користувача, відбувається заміна з гілки HKEY_USERS на HKEY_CURRENT_USER за правилами роботи реєстру Windows.

Аналогічно до обробки директорій, ключі реєстру також перевіряються рекурсивно щоб врахувати переадресацію всіх можливих батьківських елементів.

Метод SetRedirectedKey повертає новий вказівник, який відповідає гілці HKEY_CURRENT_USER у випадку необхідності та шлях до необхідного ключа, складений у відповідності до початкових параметрів. Якщо ж переадресація непотрібна, повертаються вхідні вказівник та рядковий параметр. Передача рядку відбувається за допомогою використання модифікатору out.

Після виконання відповідних методів для перевірки файлів, директорій або ключів реєстру виконуються початкові системні виклики, проте з модифікованими параметрами в залежності від конструкторів самих викликів.

Останнім класом в проекті SystemMonitor.Hook є SystemMonitorEntryPoint, який є спадкоємцем класу EasyHook.IEntryPoint. З його назви зрозуміло, що клас є вхідною точкою роботи механізму моніторингу системних викликів та використання інших файлів проекту, описаних вище.

В конструкторі SystemMonitorEntryPoint відбувається підключення до IPC клієнту, виконується метод Ping класу SystemMonitorInterface та ініціалізується

черга з об’єктів класу ISystemMonitorEvent. Окрім конструктору, клас має метод Run [39], що запускає стандартний цикл обробки повідомлень інструментарію у заданому в параметрах контексті IContext, визначеного бібліотекою EasyHook.

У методі Run використовується конструкція using для автоматичного виклику методу Dispose для вказаних класів, в цьому випадку – FileSystemHook, RegistryHook і ProcessHook. У тілі конструкції виконується цикл для роботи з чергою подій що моніторяться. Інструментарій буде виконуватися допоки метод Ping є доступним, в інакшому випадку обробник виключних ситуацій завершить роботу інструментарію. Використання конструкції using зображено на рис. 4.4.

```
using (new FileSystemHook(_queue))
using (new RegistryHook(_queue))
using (new ProcessHook(_queue))
{
    _monitorInterface.ReportHooksInstallation();

    RemoteHooking.WakeUpProcess();

    while (true)
    {
        _monitorInterface.Ping();

        while (_queue.TryDequeue(out var monitorEvent))
            _monitorInterface.ReportSystemEvent(monitorEvent);

        Thread.Sleep(500);
    }
}
```

Рисунок 4.4 – Конструкція using методу Run класу “SystemMonitorEntryPoint”

Повну структуру проекту SystemMonitor.Hook зображено на рис. 4.5.

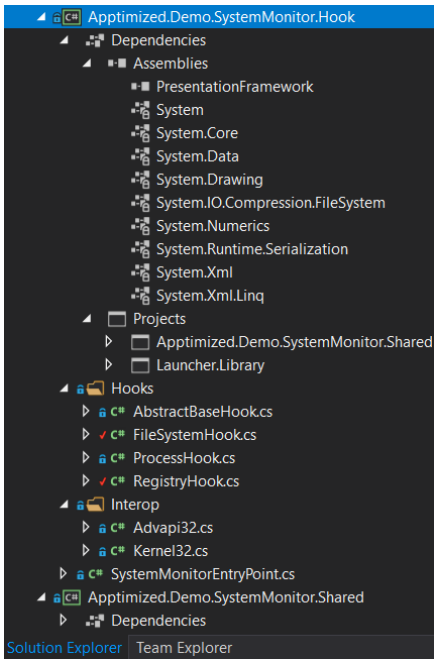


Рисунок 4.5 – Структура проекту “SystemMonitor.Hook”

За забезпечення взаємодії з зовнішнім середовищем та іншими програмними додатками, а також за можливість запуску відповідає проект SystemMonitor. Даний клас містить клас Program, в якому реалізовано метод Main, який використовується у якості точки входу в інструментарій.

Використовуючи розроблені файли InjectOptions.cs та RunOptions.cs, що визначають аргументи командного рядка для виклику інструментарію з відповідними параметрами (за обробку яких відповідає NuGet пакет CommandLineParser), в залежності від їх значень виконуються метод InjectIntoExisting або CreateAndInject. За назвами методів зрозуміло, що перший використовується для встановлення хуків до вже існуючого процесу за його ідентифікатором, а інший забезпечує запуск вказаного процесу з вказаними аргументами, для якого також будуть встановлені хуки.

Окрім запуску або імплементації до вказаного процесу в методі Main відбувається ініціалізація сутностей IPC серверу, об’єкту класу відповідального за логування, а також очікування завершення роботи головного процесу додатку. Зі структурою проекту можна ознайомитися на рис. 4.6.

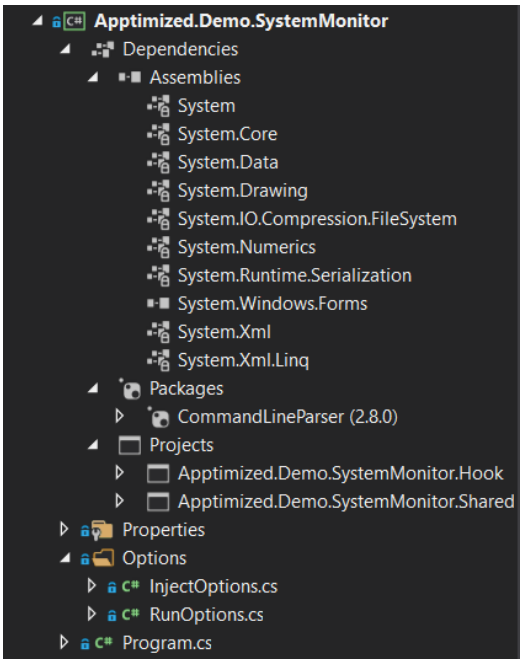


Рисунок 4.6 – Структура проекту “SystemMonitor”

Повний програмний код наведено у додатку В.

4.3 Результат роботи інструментарію

Враховуючи відсутність графічного інтерфейсу користувача, що не передбачений для інструментарію контейнеризації програмних додатків, для демонстрації результатів роботи розробленого інструментарію буде використано демонстрацію виводу логування в консольному вигляді у режимі реального часу, а також, власне результати роботи функціоналу переадресації системних викликів.

Для демонстрації роботоспроможності механізмів переадресації викликів елементів файлової системи та операції створення процесу використано програмний додаток Ghisler Total Commander [40]. Попередньо було підготовлено тестовий контейнеризований додаток, що представлено скомпільованими файлами розробленого інструментарію, набором файлів необхідних для роботи додатку в

окремій директорії Files, що зображено на рис. 4.7, а також підготовано xml файл конфігурації, із зазначеними на рис. 4.8 налаштуваннями переадресації.

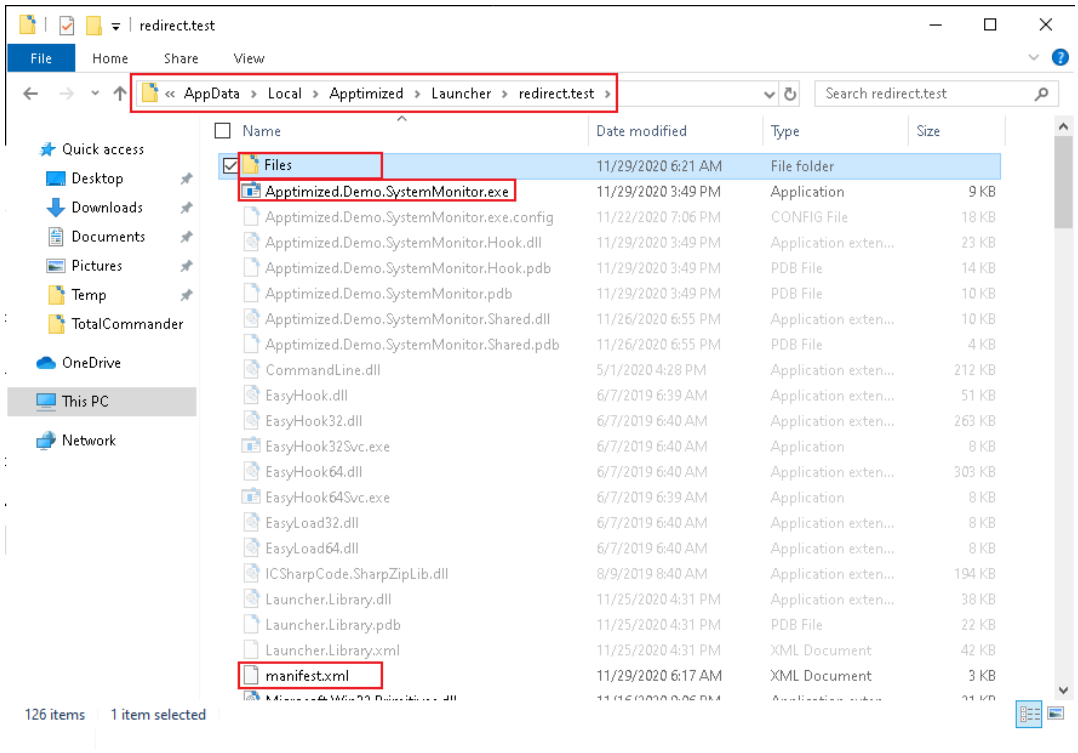


Рисунок 4.7 – Вміст директорії контейнеризованого додатку

```
<?xml version="1.0"?>
<WorkspacePackage xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" Id="red
  <Description>Application Packaging Engine</Description>
  <Icons>
    <Icon Width="130" Height="120" Location="Icons\Program.ico" />
  </Icons>
  <Shortcuts>
    <Shortcut Id="total" Location="%ProgramMenu%\Program" Command="Program\run.exe" Arguments="\r" Elevated="false">Shortcut
  </Shortcuts>
  <Categories>
    <Category>Category1</Category>
    <Category>Category2</Category>
  </Categories>
  <EnvironmentVariables>
    <EnvironmentVariable Name="TestVar" Type="Value" Value="value" />
  </EnvironmentVariables>
  <RegistryKeys>
    <RegistryKey Key="HKLM\Software\Apptimized\Engine" Value="AutoStart" Type="String">Disabled</RegistryKey>
  </RegistryKeys>
  <RegistryVirtualRoot>HKCU\Software\Apptimized\Launcher\totalcmd</RegistryVirtualRoot>
  <Redirections>
    <Redirection Type="Folder" From="%ProgramFiles%\TotalCommander" To="%DefaultDir%\Files" />
    <Redirection Type="Folder" From="C:\!temp\dirToRedirect" To="%DefaultDir%\Files\dirToRedirect" />
    <Redirection Type="Folder" From="C:\!temp\fileRedirect" To="%DefaultDir%\Files\fileRedirect" />
    <Redirection Type="File" From="C:\!temp\testfile.txt" To="%DefaultDir%\Files\fileRedirect\testfile.txt" />
  </Redirections>
  <ProgramEntries>
    <ProgramEntry Id="id" Command="Files\Engine\APE.exe" Arguments="/run" Elevated="false">Run program entry</ProgramEntry>
  </ProgramEntries>
  <Services>
    <Service Name="tstserv" DisplayName="Test service" StartType="Auto" StartOnDeploy="false" Type="Win32">Files\lalal\servi
  </Services>
</WorkspacePackage>
```

Рисунок 4.8 – Перелік переадресацій для демонстраційного додатку

Після запуску контейнеризованого додатку, використовуючи розроблений інструментарій за допомогою команди “.\Apptimized.Demo.SystemMonitor.exe run -p ".\Files\TOTALCMD64.EXE"” відображається вікно програмного додатку Total Commander в такому ж вигляді, як і при звичайній інсталяції програми.

Для демонстрації роботи інструментарію під час роботи з файловою системою, були використані операції створення директорії, переміщення файлу та видалення файлу.

Спершу, намагаємось створити директорію “C:\!temp\dirToRedirect”, яка вказана в маніфест-файлі як та, для якої є необхідність провести переадресацію на “%DefaultDir%\Files\dirToRedirect”, де плейсхолдер %DefaultDir% є позначенням директорії з контейнеризованим додатком, розв’язування якого реалізовано в інструментарії. Після виконання операції створення директорії жодних змін у каталозі “C:\!temp” не відбулося, проте в “%localappdata%\Apptimized\Launcher\redirect.test\Files” було створено каталог dirToRedirect. Вміст директорій після даної операції зображено на рис. 4.9.

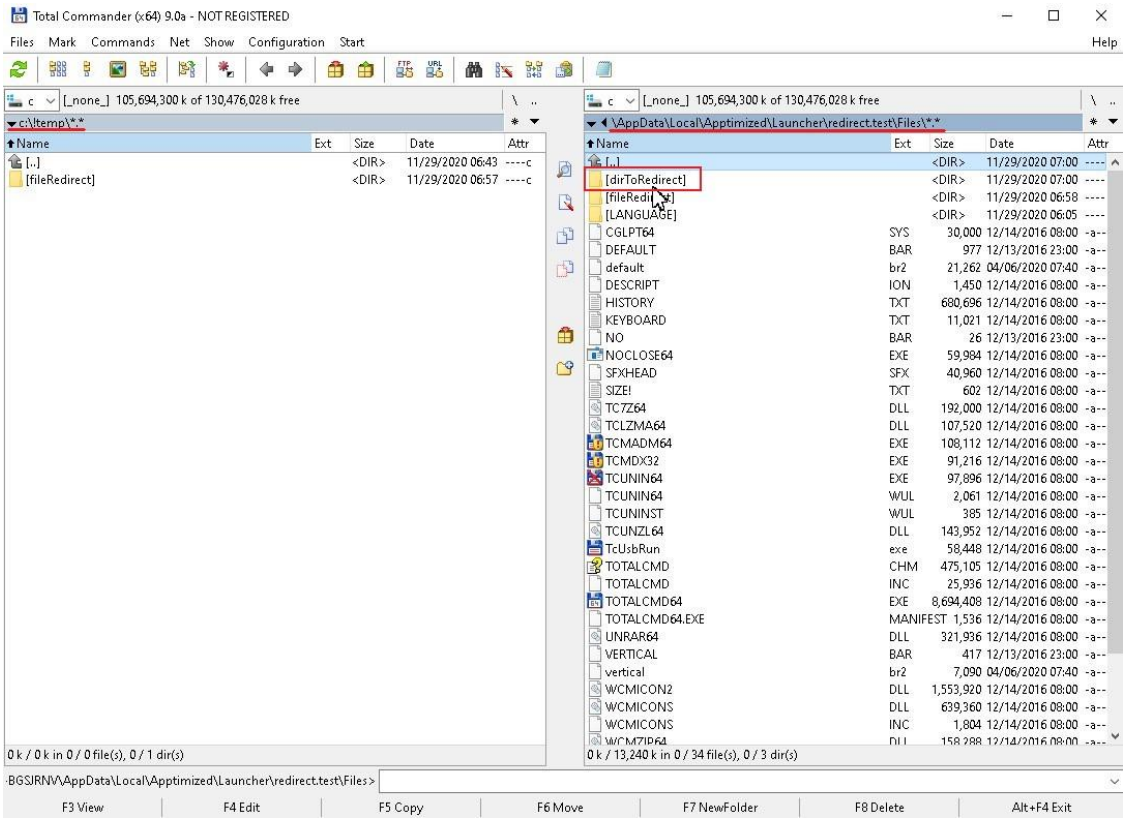


Рисунок 4.9 – Результат переадресації виклику створення директорії

Наступним тестовим сценарієм є переміщення файлу до директорії, що переадресовується. В даному випадку, переміщуємо файл до директорії “C:\!temp\fileRedirect”, результат очікуваний – файл не з'явився у даній директорії, а був переміщений до вказаної в xml файлі налаштувань папки “%localappdata%\Apptimized\Launcher\redirect.test\Files\fileRedirect”. Даний сценарій показує роботоспроможність механізму перевірки необхідності переадресації файлів не лише за типом File компоненту Redirection в маніфест-файлі, а демонструє виконання логіки перевірки шляху до файлу рекурсивно по компонентам типу Folder. Результат операції переміщення файлу у вигляді вмісту цільової та перенаправленої директорії зображено на рис. 4.10.

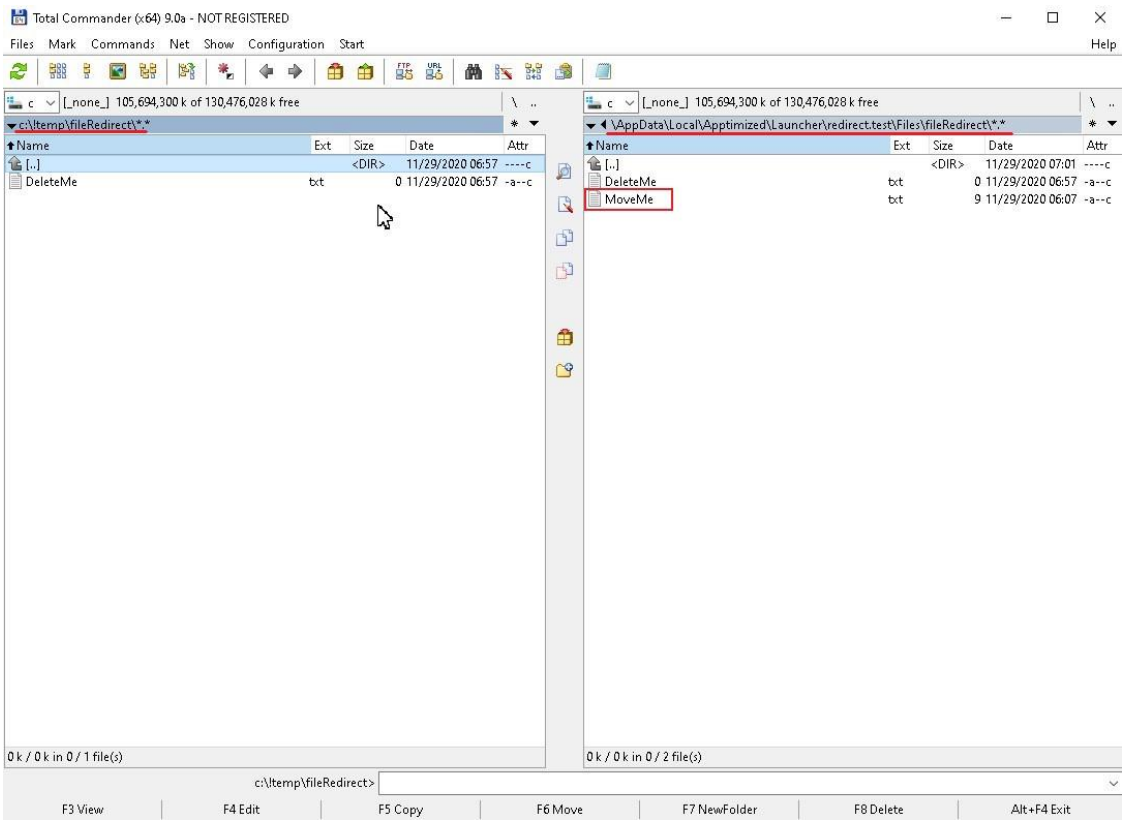


Рисунок 4.10 – Результат переадресації операції переміщення файлу

Останнім демонстраційним прикладом роботи інструментарію з файловою системою є обробка процесу видалення файлу. Видаляючи файл з директорії “C:\!temp\fileRedirect”, в результаті файл видаляється з переадресованої директорії

“%localappdata%\Apptimized\Launcher\redirect.test\Files\fileRedirect”, а в цільовій папці файл не було видалено. Результат зображено на рис. 4.11.

Результати роботи також підтверджуються логуванням у вигляді системного виведення в консоль у режимі реального часу. На рис. 4.12 зображено результат логування з виділеними операціями створення директорії, переміщення та видалення файлу, а також процес обробки маніфест-файлу з конфігураціями переадресацій додатку.

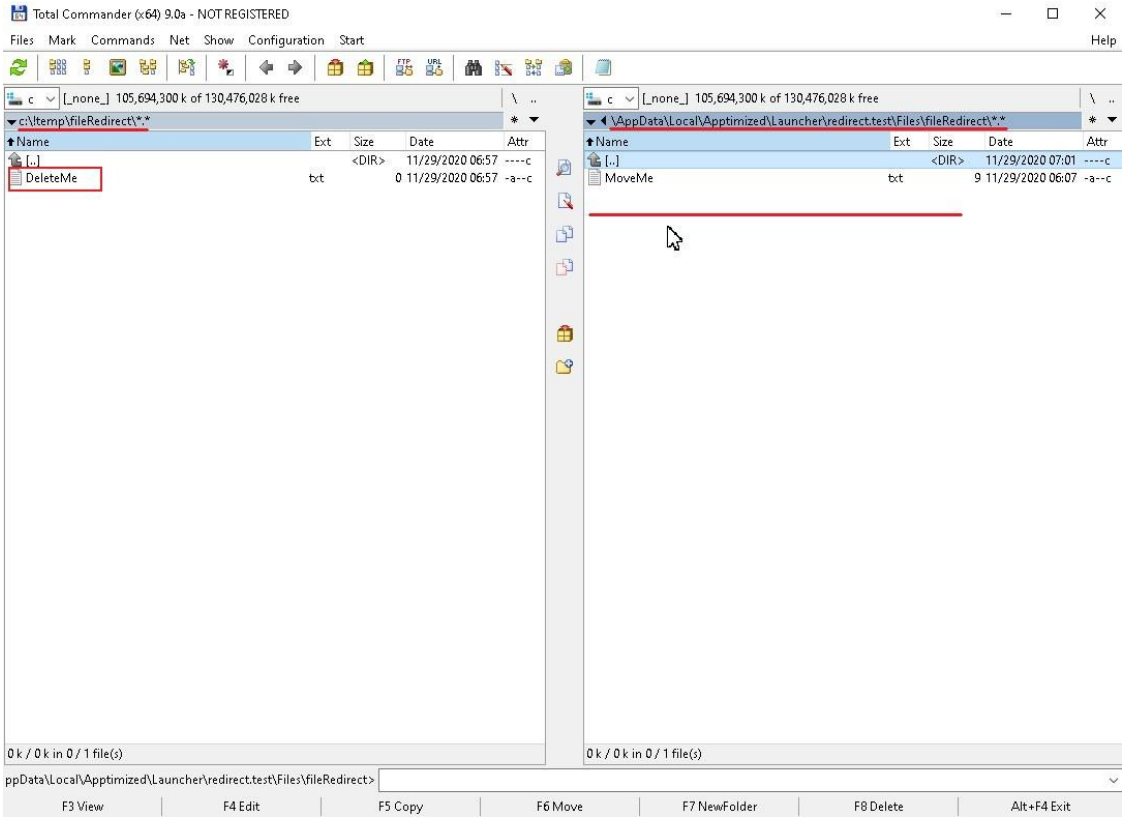


Рисунок 4.11 – Результат переадресації операції видалення файлу

Встановлення хуків та обробка системних викликів відбувається не лише для основного процесу контейнеризованого додатку, а й для його підпроцесів. Реалізація цього функціоналу була обов’язковою, адже більшість програмних додатків не обмежується використанням лише одного виконуваного файлу та може використовувати декілька процесів для забезпечення власної працездатності. Для демонстрації даного функціоналу також було використано Total Commander у якості

основного додатку, в якому було викликано Windows додаток Notepad для створення файлу.

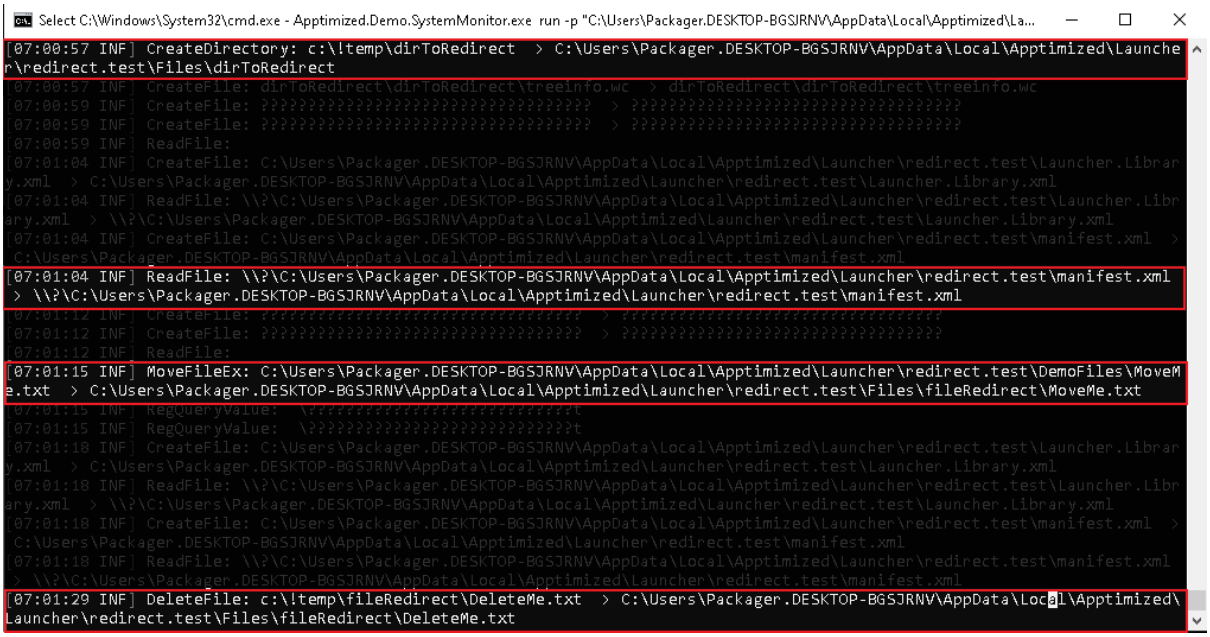


Рисунок 4.12 – Консоль логування операцій

Аналогічно до попередніх тестових сценаріїв, виконуємо запуск додатку за допомогою інструментарію. Наступним кроком запускаємо додаток Notepad за допомогою кнопки на панелі інструментів, рис 4.13.

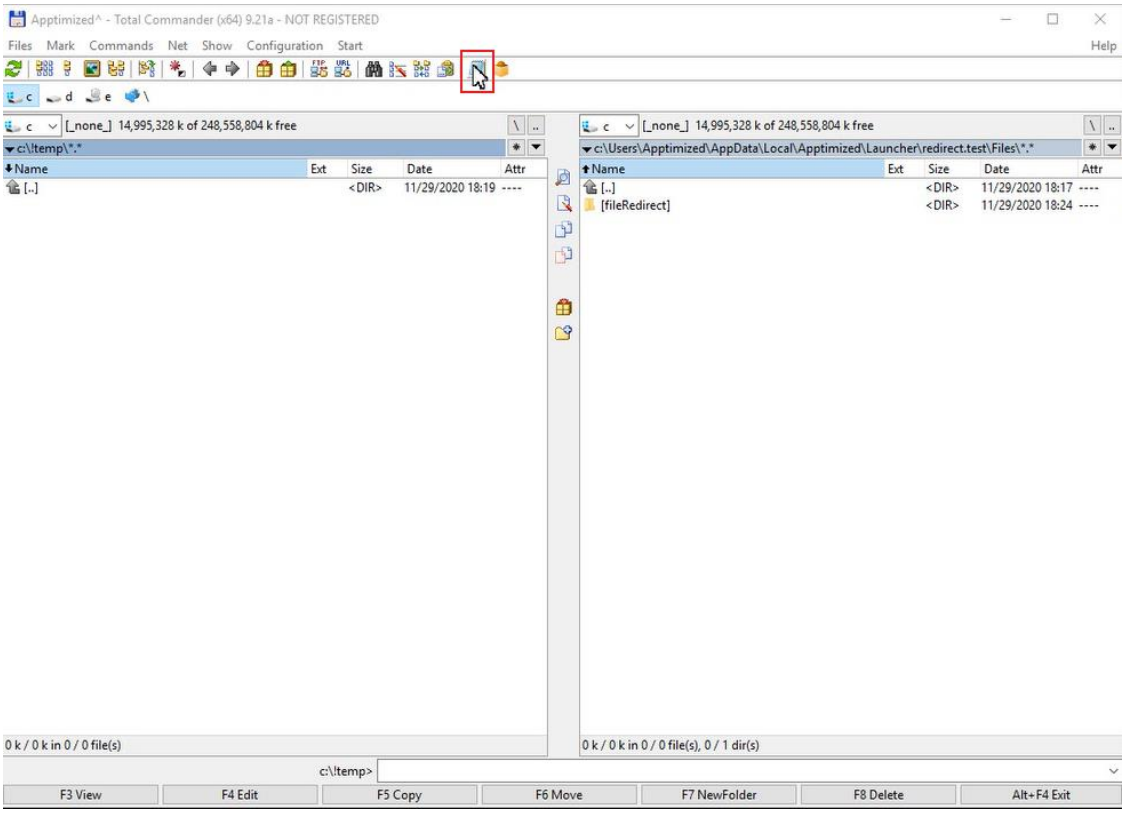


Рисунок 4.12 – Запуск Notepad

Після запуску нового процесу створюється системний виклик операції ProcessCreate, який було перехоплено та створено з інтегрованими хуками. На рис. 4.13 зображено логування встановлення хуків для підпроцесів.

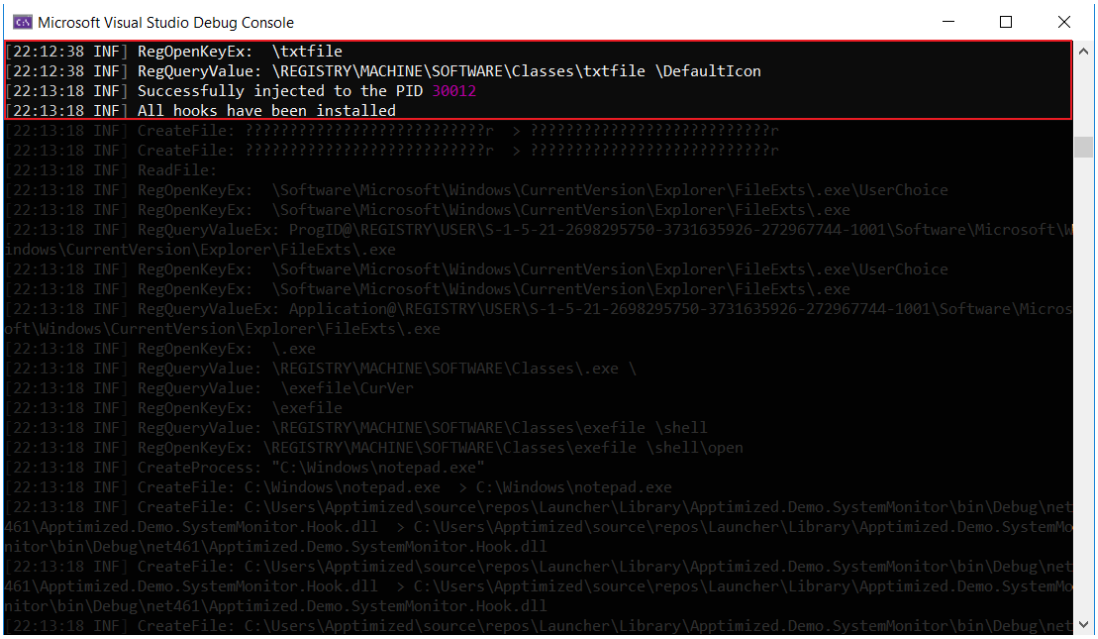


Рисунок 4.13 – Ін’єкція в підпроцес та встановлення хуків

Після запуску блокноту заповнюємо його тестовим контентом та зберігаємо його як “C:\!temp\testfile.txt” (рис. 4.14). Результат очікуваний: файл не зберігся за вказаною адресою, а був перенаправлений у відповідності до налаштувань маніфест-файлу до директорії “%localappdata%\Apptimized\Launcher\redirect.test\Files\fileRedirect”, що зображено на рис. 4.15.

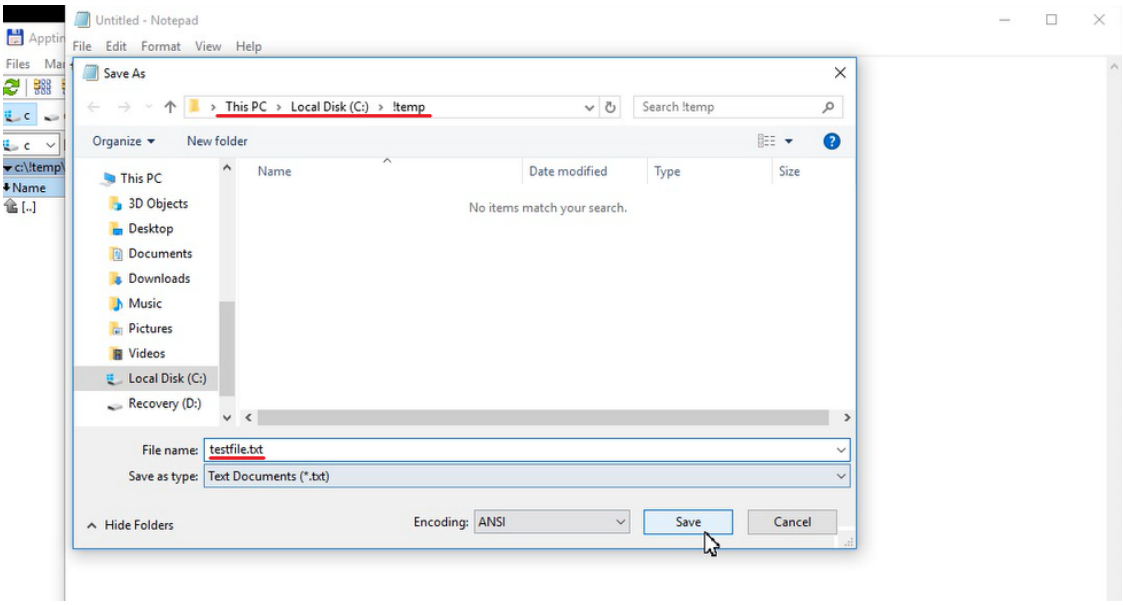


Рисунок 4.14 – Зберігання файлу

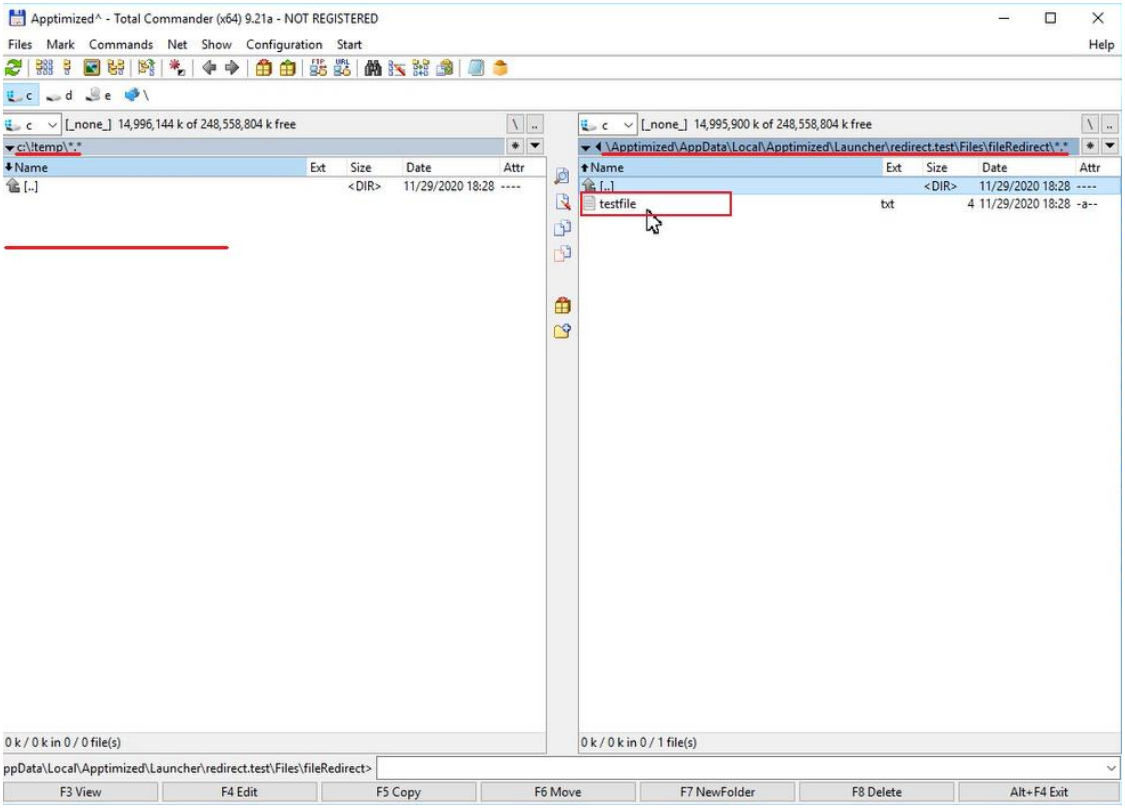


Рисунок 4.15 – Результат переадресації операції створення файлу

Також маємо логування даної операції на рис. 4.16.

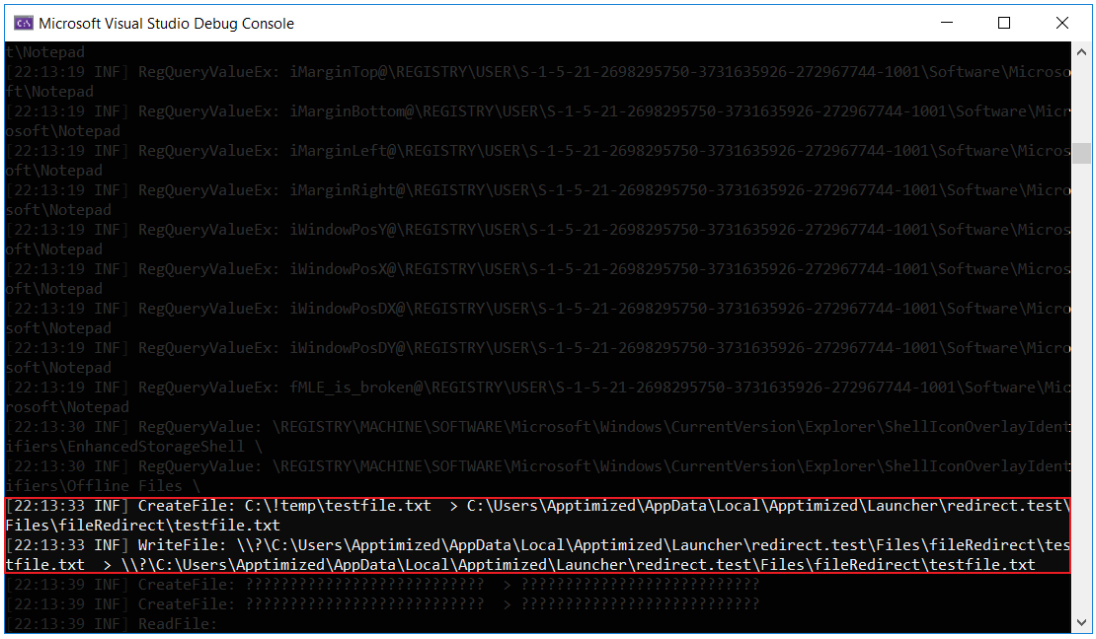


Рисунок 4.16 – Створення та зберігання переадресованого файлу

Для демонстрації взаємодії розробленого інструментарію з викликами елементів реєстру було використано стандартну утиліту Windows Registry Editor для створення, перейменування та видалення ключів і значень.

Для тесту створимо ключ за адресою “HKEY_LOCAL_MACHINE\SOFTWARE\Apptimized”, який має переадресацію згідно з налаштуваннями файлу manifest.xml. За замовчуванням ім’я нового ключа “New Key #1”, перейменовуємо його на ”TestKey”. Всередині створеного ключа додаємо строкове значення та перейменовуємо з “New Value #1” на “StringValue”. Також створимо підключ зі стандартним ім’ям. Отже, маємо структуру ключа, що зображена на рис. 4.17.

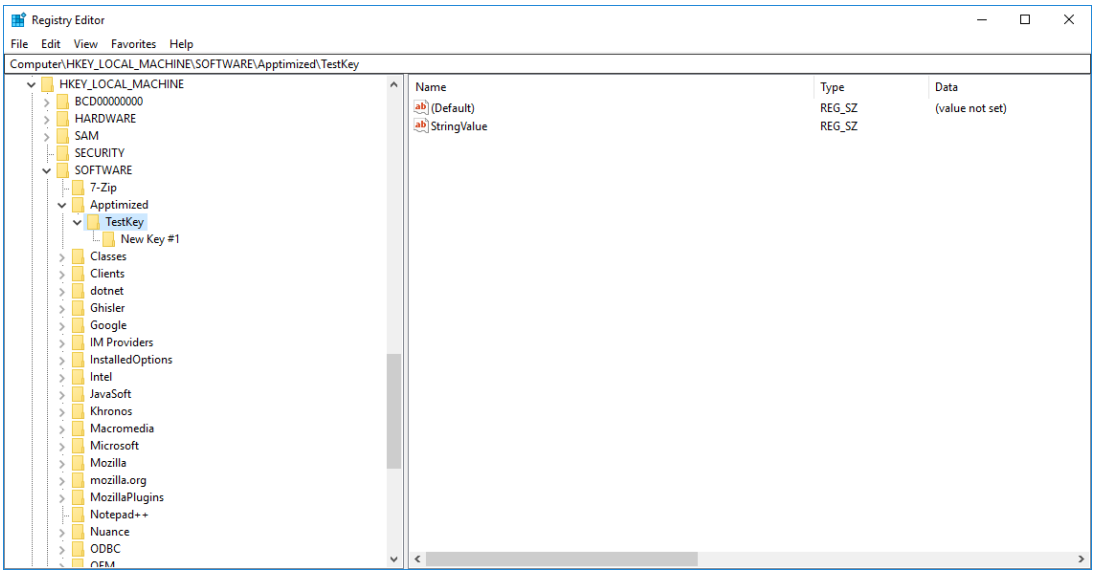


Рисунок 4.17 – Структура цільового ключа

Відображення переадресованих ключів можливо, на відміну від файлів, і в цільовому ключі, адже для адресації ключів реєстру використовується інший механізм та при відображенні приймають участь функції, що теж можуть бути перенаправлені. Отже, візуально маємо вигляд ніби дії безрезультатні, проте, перевіривши ключ, що зберігає перенаправлені елементи (вигляд перед видаленням ключа TestKey зображено на рис. 4.18) та виведення логування (рис. 4.19) можна

впевнитися в успіху роботи інструментарію та зробити висновок, що всі системні виклики для роботи з елементами реєстру були перенаправлені.

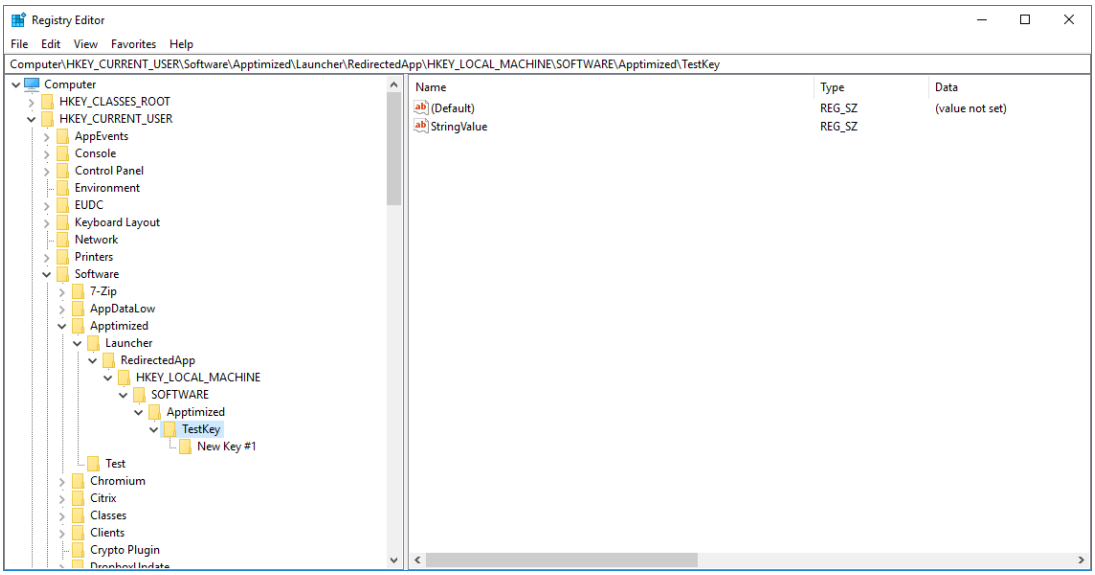


Рисунок 4.18 – Структура цільового ключа

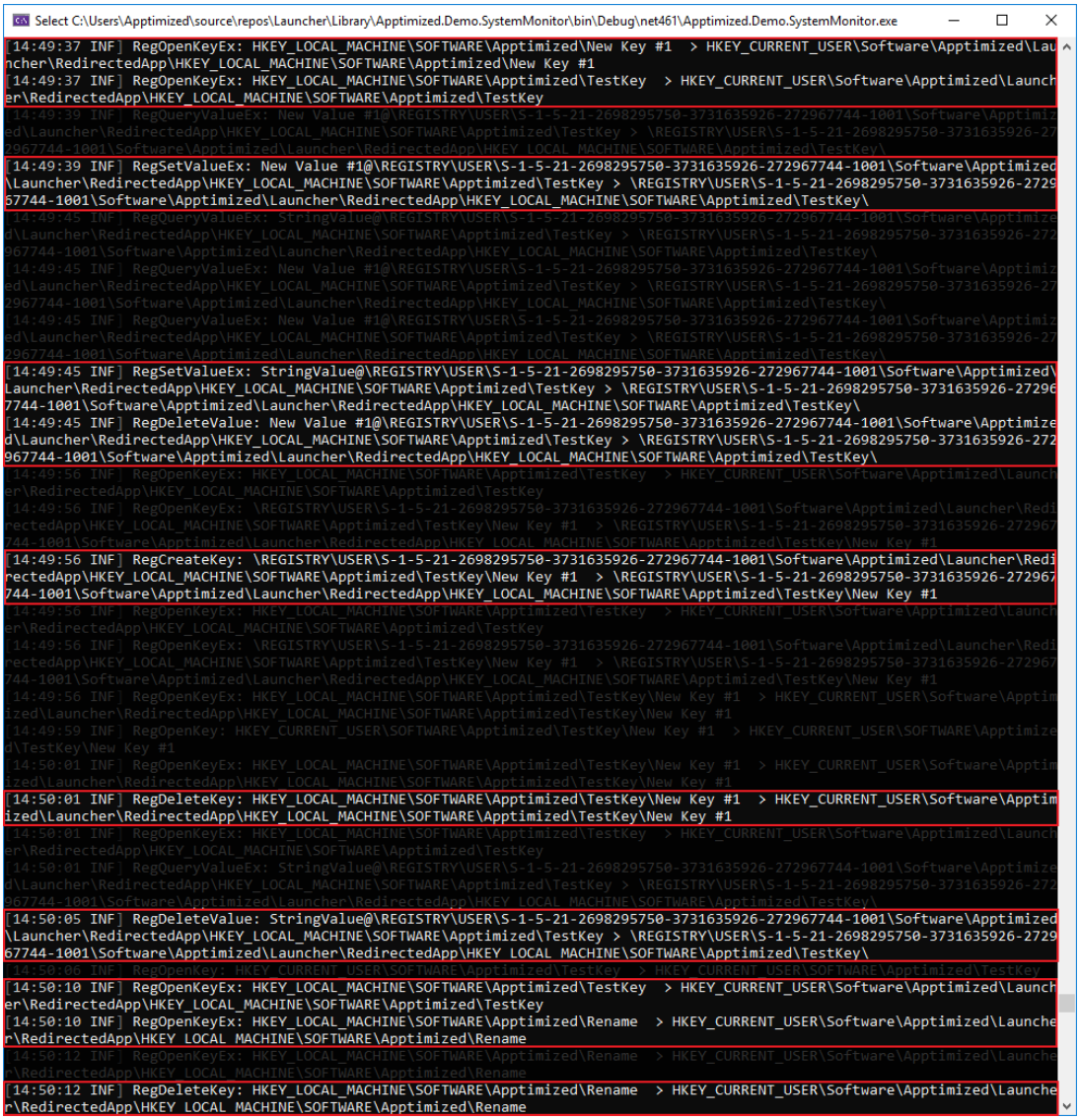


Рисунок 4.19 – Логування роботи інструментарію з операціями реєстру

ВИСНОВКИ

Під час виконання дипломного проекту був розроблений інструментарій контейнеризації програмних додатків. Продукт проекту розроблено з метою подальшого його використання у компанії Apptimized Operations як працівниками відділу пакування та розробки програмного забезпечення, так і клієнтами порталу.

У процесі виконання проекту була обґрунтована актуальність роботи, проаналізовано існуючі моделі адміністрування та пакування програмних додатків включно з інсуючими програмними рішеннями, їх переваги і недоліки; визначено функціональні вимоги до інструментарію що розробляється та обрано засоби реалізації.

Було проведено процес деталізації мети проекту та виконано планування робіт, за результатами якого розроблено календарний план, який дозволяє визначити часві рамки проекту, а також визначено перелік ризиків під час виконання проекту.

У процесі моделювання виконано структурно-функціональне моделювання у нотації IDEF0, проведено декомпозицію роботи інструментарію контейнеризації та процесу переадресації системних викликів зокрема. У цьому розділі також були проаналізовані варіанти використання за допомогою Use Case діаграми, розроблено діаграму послідовності з метою уточнення взаємодії між об’єктами в системі та описано структуру інформаційної моделі проекту.

Враховуючи результати моделювання була виконана розробка інструментарію попередньо обравши засоби реалізації. Інструментарій було розроблено засобами об'єктно-орієнтованої мови C# з використанням .NET Framework, а також NuGet пакетів EasyHook, CommandLineParser та Serilog у середовищі розробки Microsoft Visual Studio 2019.

Під час розробки було створено та серіалізовано у інструментарій структуру файлу конфігурації програмного додатку стандартом XML; використовуючи API EasyHook розроблено систему моніторингу системних викликів та подальшу їх

обробку, враховуючи особливості роботи операційної системи Windows, а також оброблено всі можливі виключні ситуації під час роботи інструментарію.

Розроблений інструментарій відповідає поставленим функціональним вимогам, а саме, реалізовано функціонал:

- зберігання файлів програмних додатків у єдиній директорії-контейнері;
- забезпечується робота програмного додатку аналогічно до локального встановлення відповідно до його власних функціональних вимог;
- було використано механізм моніторингу та перехвату системних викликів;
- під час роботи інструментарію системні виклики оброблюються в залежності від конфігурації певного додатку;
- ведення логів для полегшення адміністрування контейнерів програмних додатків.

Після завершення розробки було проведено тестування продукту в компанії Apptimized Operations та за його результатами отримано акт впровадження від відділу пакування та розробки програмного забезпечення, який наведено у додатку А.

Використання розробленого інструментарію контейнеризації програмних продуктів дозволить покращити показники роботи та скоротити ризик втрати потенціальних спільних файлів бібліотек чи системних ключів реєстру, які можуть бути присутні на віртуальних машинах із локально встановленими інструментами DevOps інженерів під час створення пакетів програмного забезпечення.

СПИСОК ЛІТЕРАТУРИ

1 Desktop Operating System Market Share Worldwide [Електронний ресурс] – режим доступу: https://gs.statcounter.com/os-market-share/desktop/worldwide/#monthly-202001-202009-bar

2 What is Software Deployment? [Електронний ресурс] – режим доступу: https://www.itarian.com/itcm-software-deployment.php

3 System Center 2019 [Електронний ресурс] – режим доступу: https://www.microsoft.com/en-us/system-center

4 What is Microsoft Intune – Azure [Електронний ресурс] – режим доступу: https://docs.microsoft.com/en-us/mem/intune/fundamentals/what-is-intune

5 System Center Configuration Manager vs. Intune [Електронний ресурс] – режим доступу: https://www.softwareone.com/en/blog/all-articles/2019/12/09/system-center-configuration-manager-vs-intune

6 Workspace Launcher [Електронний ресурс] – режим доступу: https://docs.apptimized.com/books/manual/page/workspace-launcher

7 Application Packaging Best Practices [Електронний ресурс] – режим доступу: https://www.itninja.com/blog/view/application-packaging-best-prac-tices

8 12 Best installers for windows programs as of 2020 [Електронний ресурс] – режим доступу: https://www.slant.co/topics/4794/~installers-for-windows-programs

9 Installations [Електронний ресурс] – режим доступу: http://www.manifold.net/doc/mfd9/installations.htm

10 What is virtualization technology & virtual machine? [Електронний ресурс] – режим доступу: https://www.vmware.com/solutions/virtualization.html

11 What is virtualization? [Електронний ресурс] – режим доступу: https://www.redhat.com/en/topics/virtualization/what-is-virtualization

12 Containerization vs. Virtualization: What’s the Difference? [Електронний ресурс] – режим доступу: https://www.burwood.com/blog-archive/containerization-vs-virtualization

13 Desktop Virtualization: A Pros and Cons List [Електронний ресурс] – режим доступу: https://www.mtm.com/desktop-virtualization-pros-cons-list/

14 Containerization Explained [Електронний ресурс] – режим доступу: https://www.ibm.com/cloud/learn/containerization

15 What is a Container? [Електронний ресурс] – режим доступу: https://www.docker.com/resources/what-container

16 Overview of Application Virtualization [Електронний ресурс] – режим доступу: https://docs.microsoft.com/en-us/microsoft-desktop-optimization-pack/appv-v4/overview-of-application-virtualization

17 What is App-V and how does it work? [Електронний ресурс] – режим доступу: https://www.tmurgent.com/TmBlog/?p=2489

18 App-V or Windows Containers: Which is right for you? [Електронний ресурс] – режим доступу: http://techgenix.com/app-v-or-windows-containers/

19 About Windows Containers [Електронний ресурс] – режим доступу: https://docs.microsoft.com/en-us/virtualization/windowscontainers/about/

20 Build and tun your first Docker Windows Server container [Електронний ресурс] – режим доступу: https://www.docker.com/blog/build-your-first-docker-windows-server-container/

21 Application Compatibility Packages for new Server Migration [Електронний ресурс] – режим доступу: https://cloudhouse.com/products/

22 Usermode System Call hooking [Електронний ресурс] – режим доступу: https://www.malwaretech.com/2014/06/usermode-system-call-hooking-betabo.html

23 Windows DLL Injection Into Process Using KnownDlls [Електронний ресурс] – режим доступу: https://www.apriorit.com/dev-blog/257-dll-injection

24 EasyHook - Documentation [Електронний ресурс] – режим доступу: http://easyhook.github.io/documentation.html

25 .NET [Електронний ресурс] – режим доступу: https://dotnet.microsoft.com/

26 What is NuGet? [Електронний ресурс] – режим доступу: https://www.nuget.org/

27 CommandLineParser [Електронний ресурс] – режим доступу: https://github.com/commandlineparser/commandline

28 Serilog - Simple .NET logging [Електронний ресурс] – режим доступу: https://serilog.net/

29 Git [Електронний ресурс] – режим доступу: https://git-scm.com/

30 Сериализация в XML. XmlSerializer [Електронний ресурс] – https://metanit.com/sharp/tutorial/6.4.php

31 Toolkit [Електронний ресурс] – режим доступу: https://www.pcmag.com/encyclopedia/term/toolkit

32 Diagram of use cases (UseCase diagram) [Електронний ресурс] – режим доступу: https://flexberry.github.io/en/fd_use-case-diagram.html

33 Sequence diagram [Електронний ресурс] – режим доступу: https://flexberry.github.io/en/fd_sequence-diagram.html

34 Serialization (C#) [Електронний ресурс] – режим доступу: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/serialization/

35 System.Xml.Serialization Namespace [Електронний ресурс] – режим доступу: https://docs.microsoft.com/en-us/dotnet/api/system.xml.serialization

36 Fileapi.h header [Електронний ресурс] – режим доступу: https://docs.microsoft.com/en-us/windows/win32/api/fileapi/

37 Winreg.h header [Електронний ресурс] – режим доступу: https://docs.microsoft.com/en-us/windows/win32/api/winreg/

38 Implement a Dispose method [Електронний ресурс] – режим доступу: https://docs.microsoft.com/en-us/dotnet/standard/garbage-collection/implementing-dispose

39 Application.Run Method [Електронний ресурс] – режим доступу: https://docs.microsoft.com/en-us/dotnet/api/system.windows.forms.application.run

40 Total Commander - home [Електронний ресурс] – режим доступу: https://www.ghisler.com/

41 What is a Work Breakdown Structure? [Електронний ресурс] – режим доступу: https://www.workbreakdownstructure.com

42 What is Work Breakdown Structure? [Електронний ресурс] – режим доступу: https://www.visual-paradigm.com/guide/project-management/what-is-work-breakdown-structure

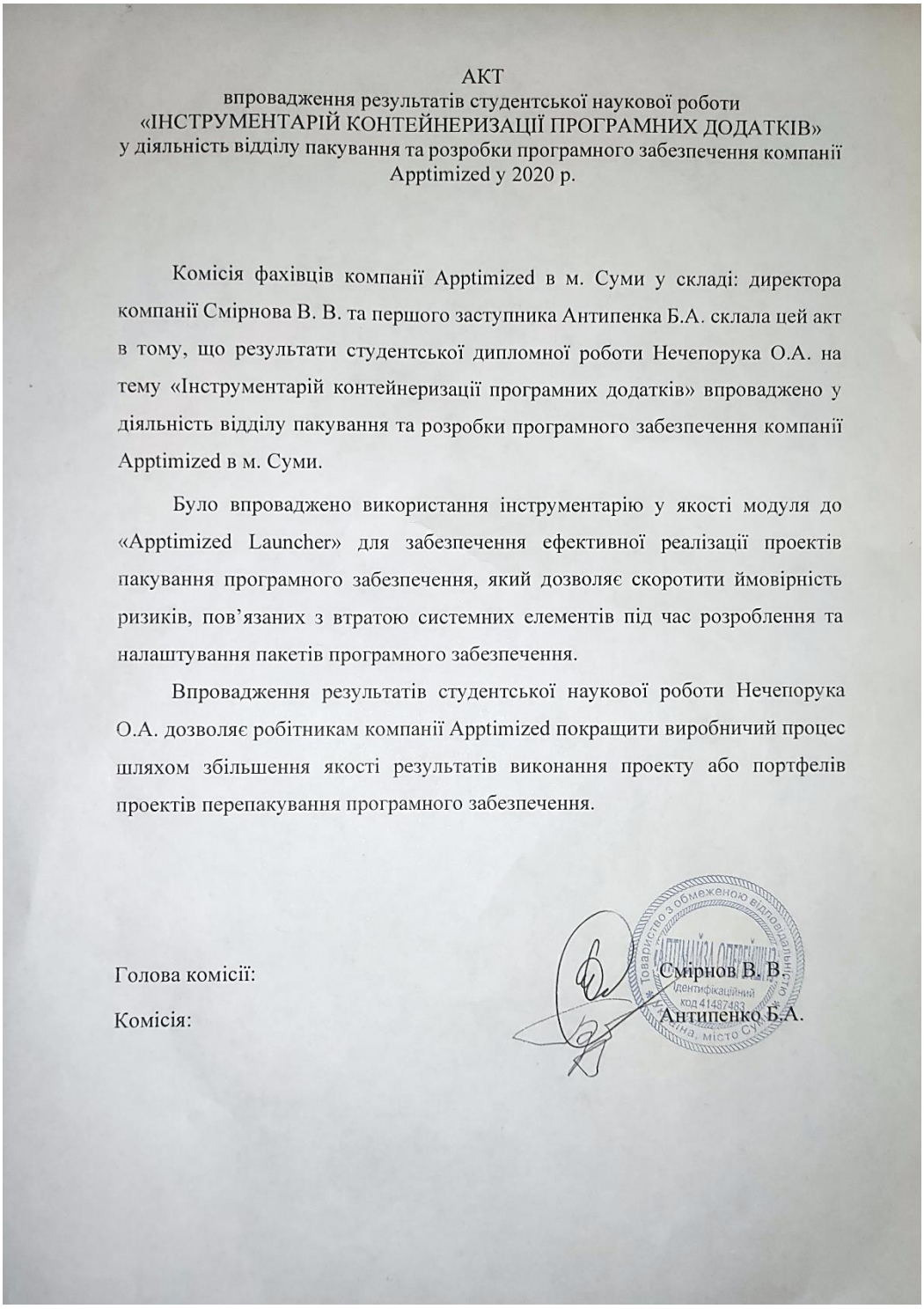
43 Управління проектами. Односпрямована структуризація — створення робочої структури проекту [Електронний ресурс] – режим доступу: https://library.if.ua/book/96/6604.html

44 What Is an Organizational Breakdown Structure (OBS)? [Електронний ресурс] – режим доступу: https://smallbusiness.chron.com/implications-organizational-culture-project-structure-73039.html

45 Що таке діаграма Ганта та як вона використовується у бізнес-плані [Електронний ресурс] – режим доступу: http://monetary-flow.com/shto-take-dagrama-ganta-ta-yak-vona-vikoristovutysya-u-bznes-planuvann/

Додаток А

Акт впровадження



АКТ
впровадження результатів студентської наукової роботи
«ІНСТРУМЕНТАРІЙ КОНТЕЙНЕРИЗАЦІЇ ПРОГРАМНИХ ДОДАТКІВ»
у діяльність відділу пакування та розробки програмного забезпечення компанії Apptimized у 2020 р.

Комісія фахівців компанії Apptimized в м. Суми у складі: директора компанії Смірнова В. В. та першого заступника Антипенка Б.А. склала цей акт в тому, що результати студентської дипломної роботи Нечепорука О.А. на тему «Інструментарій контейнеризації програмних додатків» впроваджено у діяльність відділу пакування та розробки програмного забезпечення компанії Apptimized в м. Суми.

Було впроваджено використання інструментарію у якості модуля до «Apptimized Launcher» для забезпечення ефективної реалізації проектів пакування програмного забезпечення, який дозволяє скоротити ймовірність ризиків, пов'язаних з втратою системних елементів під час розроблення та налаштування пакетів програмного забезпечення.

Впровадження результатів студентської наукової роботи Нечепорука О.А. дозволяє робітникам компанії Apptimized покращити виробничий процес шляхом збільшення якості результатів виконання проекту або портфелів проектів перепакування програмного забезпечення.

Голова комісії: Смірнов В. В.

Комісія: Антипенко Б.А.

Рисунок А.1 – Акт впровадження

Додаток Б

Планування робіт

Б.1 Деталізація мети методом SMART

Метою дипломного проекту є розробка інструментарію контейнеризації програмних додатків для операційних систем сімейства Windows. Розроблений інструментарій працюватиме самостійно, без втручання користувача, отже не матиме окремого користувацького інтерфейсу. Принцип контейнеризації додатків буде забезпечений, власне, розробленим інструментарієм у якості окремого модуля агрегатора програмних додатків Apptimized Launcher. За допомогою розробленого інструментарію реалізується логіка переадресації системних викликів до вказаних файлів/ключів реєстру/процесів зазначених у окремому xml файлі-маніфесті, що в свою чергу дозволяє використовувати програмний додаток у власному контейнері без взаємодії з файловою системою та загалом з хостовою операційною системою.

Результати деталізації методом SMART наведено у табл. Б.1.

Таблиця Б.1 – Деталізація мети методом SMART

Specific (конкретна)	Розробити інструментарій контейнеризацію програмних додатків.
Measurable (вимірювання)	Оскільки даний проект не є комерційним, то результатом його роботи є оцінка замовника.

Продовження таблиці Б.1

Achievable (досяжна, узгоджена)	Ціль даного проекту вважається досяжною, оскільки розробник володіє необхідними навичками у створенні Windows-додатків засобами мови C# та має необхідні знання у сфері технологій виокремлення та переадресації системних викликів, а також мови розмітки XML. Мета була узгоджена з вимогами та потребами замовника.
Relevant (реалістична)	Для реалізації продукту проекту є всі необхідні технічні та програмні засоби (середовище розробки Microsoft Visual Studio, система контролю версій Git), доступ до системних викликів засобами бібліотеки EasyHook. Розробник достатньо кваліфікований для виконання поставлених задач.
Time-framed (обмежена в часі)	Інструментарій розроблюється з обмеженням у часі на основі сформованого календарного плану.

Б.2 Планування змісту структури робіт IT-проекту (WBS)

Розбиття задачі на менші завдання - це загальноприйнята техніка продуктивності, яка використовується, щоб зробити роботу більш продуктивною та керованою. Для проектів система розподілу (декомпозиції) робіт (WBS) - це інструмент, який використовує цю техніку, і є одним з найважливіших документів управління проектами. Він допомагає виконати декомпозицію робіт, впорядкувати етапи їх виконання та забезпечити узгодженість етапів проекту.[41]

Система декомпозиції робіт (WBS) - це орієнтована на результати ієрархічна декомпозиція роботи, що виконується проектною командою для досягнення цілей

проекту та створення необхідних результатів. WBS є основою ефективного планування проекту, його виконання, контролю, моніторингу та звітності. Вся робота, яка міститься в WBS, повинна бути визначена, оцінена, запланована та передбачена бюджетом.

WBS розроблена для встановлення загального розуміння обсягу проекту. Це ієрархічний опис роботи, яку потрібно виконати для завершення результатів проекту. Кожен спадний рівень в WBS представляє дедалі детальніший опис результатів проекту.

Перші два рівні WBS (кореневий вузол та рівень 2) визначають набір запланованих результатів, які в сукупності та виключно представляють 100% обсягу проекту. На кожному наступному рівні діти батьківського вузла колективно і виключно представляють 100% сфери дії їх батьківського вузла.[42]

Розробка WBS зазвичай має відбуватися на початку проекту і перед детальним плануванням проекту і задач. WBS є попереднім етапом, основою для розробки мережевих і календарних планів, які потребують повного переліку всіх робіт за проектом, які можна отримати, маючи пакети робіт. WBS наочно демонструє весь обсяг робіт і місце окремих виконавців.

Основні етапи розробки WBS:[43]

- визначення ступеня деталізації проектних;
- визначення кількості рівнів (як правило, три-чотири, для сучасних компаній чотири – оптимально);
- розробка структури кожного рівня;
- підготовка опису елементів WBS (коротка назва кожної складової WBS);
- формування системи кодування;
- проведення зворотних обчислень.

Виконуємо декомпозицію робіт для проекту. Діаграма WBS представлена на рис. Б.1.

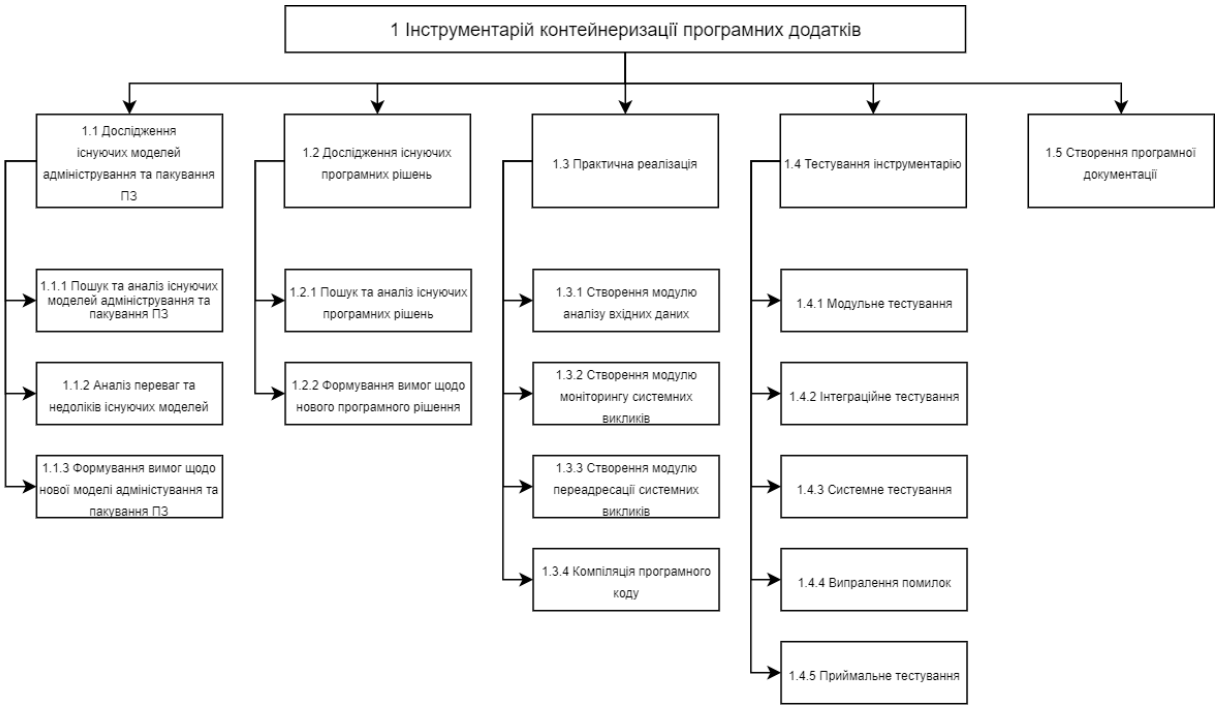


Рисунок Б.1 – WBS структура проекту

Б.3 Організаційна структура проекту (OBS)

OBS-структура проекту – організаційна структура виконавців (організацій) проекту. Визначається за переліком пакетів робіт нижнього рівня кожної гілки WBS-структури. Представляється відповідальними особами. Організаційна структура представляє собою графічне відображення учасників проекту та їх відповідальних осіб, які задіяні в реалізації проекту. На верхньому рівні OBS розташована команда проекту. На наступному рівні фіксуються виконавці: організації, відділи тощо. Потім, рівнем нижче, для кожного виконавця вказують прізвища конкретних осіб, які будуть відповідати за виконання елементарних робіт WBS.[44]

Об'єднання робочої та організаційної структури дає можливість планувати і контролювати етапи роботи. Кожна відповідальна особа в цій ієрархії може мати власний набір методик, підходів і звітів за своїми сферами відповідальності. За

специфікою WBS, розподіл задач здійснюється до робочого пакету, який виконується окремою групою, а OBS, в свою чергу, демонструє групу відповідальних осіб, які виконують найнижчий рівень робіт в WBS. Таким чином, маючи WBS та OBS можна контролювати роботи найнижчого рівня WBS та визначати відповідальних на певних етапах робіт. Організаційна структура дипломного проекту зображена на рис. Б.2.

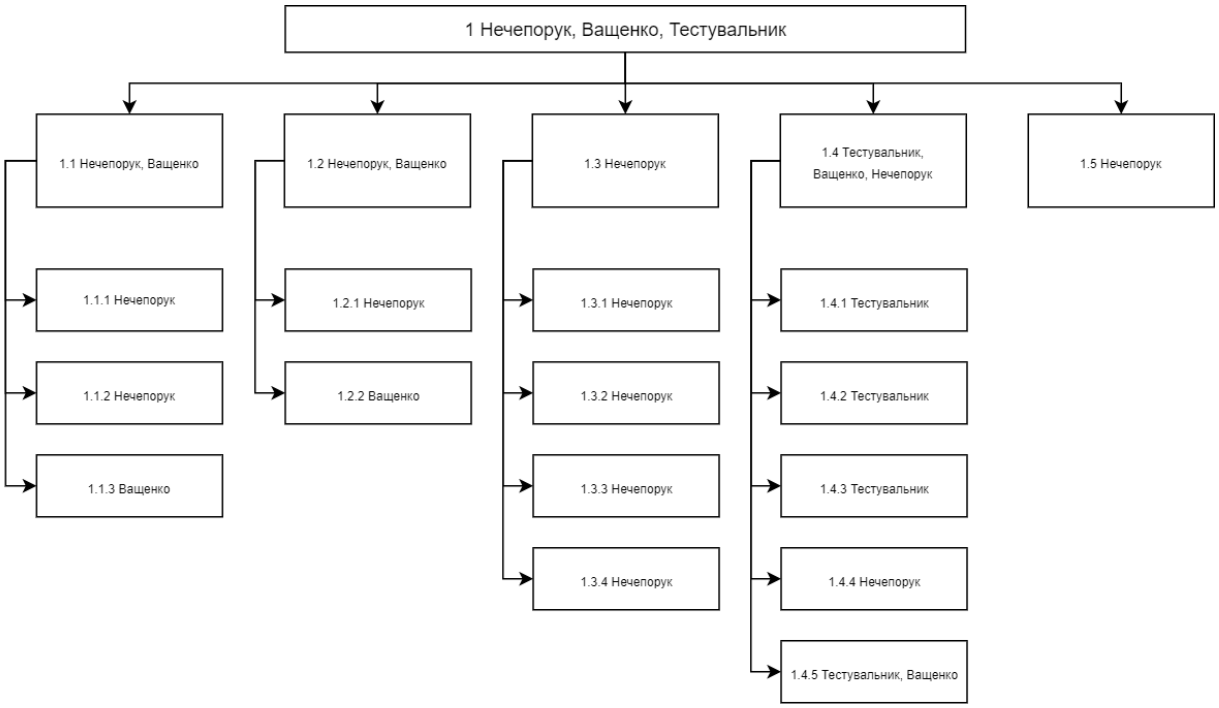


Рисунок Б.2 – OBS структура проекту

Б.4 Побудова календарного графіку виконання ІТ-проекту

Діаграма Ганта вважається якісним інструментом для відображення цілей та задач. Управління проектами з діаграмами Ганта засноване на форматі гістограм, що допомагає відслідковувати відсоток робіт, виконаних по кожному етапу роботи. Керівникам проектів дуже важливо правильно розподілити завдання і бути впевненими в тому, що проект буде завершений вчасно. Основна увага діаграм

Ганта зосереджена на процентному завершенні кожного завдання. Крім того, діаграми Ганта необхідна для проектів з невеликою кількістю взаємопов'язаних завдань.[45]

Загалом, діаграма Ганта є зручним інструментом планування та контролю дотримання термінів виконання завдань на різних етапах виконання проекту. За допомогою діаграм Ганта можна побачити:

- які завдання включає в себе проект;
- дати початку та закінчення певного етапу;
- тривалість завдань;
- відповідальні особи на конкретних етапах;
- послідовність етапів.

За допомогою MS Project була розроблена діаграма Ганта, яка у вигляді гістограми відображає тривалість кожного процесу, що був визначений на етапі формування WBS. Діаграма Ганта представлена на рис. Б.3.

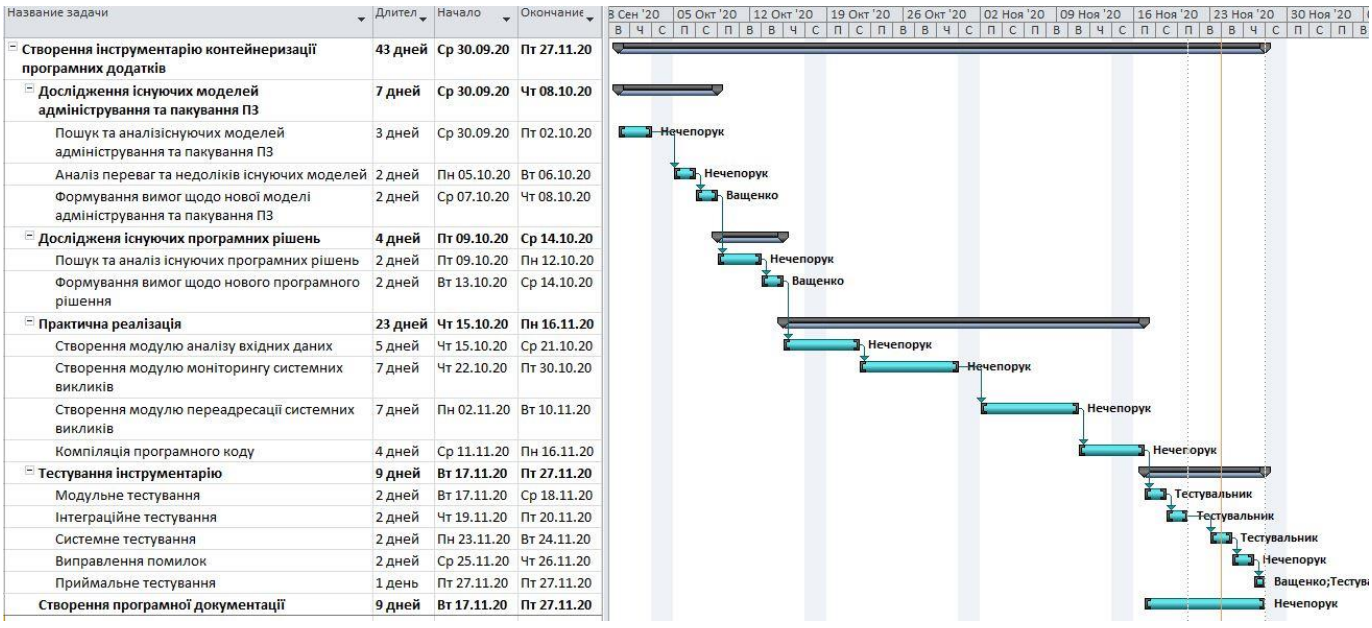


Рисунок Б.3 – Діаграма Ганта

Б.5 Управління ризиками проекту

При плануванні робіт над проектом необхідно враховувати ризики, що можуть негативно вплинути на час та якість розробки продукту.

Для даного проекту були виділені наступні ризики:

- R1 – Форс-мажорні обставини;
- R2 – Людський фактор;
- R3 – Апаратно-програмні збої;
- R4 – Неправильне розподілення ресурсів;
- R5 – Зміна вимог;
- R6 - Зміна цілей в ході реалізації проекту;
- R7 - Неякісно проведене тестування.

Таблиця Б.2 – Класифікація ризиків

№	**Ризик**	**Ймовірність виникнення**	**Обсяг втрат**
1	Форс-мажорні обставини	2	3
2	Людський фактор	1	2
3	Апаратно-програмні збої	1	3
4	Неправильне розподілення ресурсів	3	4
5	Зміна вимог	2	5
6	Зміна цілей в ході реалізації проекту	2	2
7	Неякісне проведення тестування	3	5

Використовуючи дану класифікацію, була побудована матриця ризиків, що представлена в таблиці Б.3.

Таблиця Б.3 – Матриця ризиків

Ймовірність виникнення							
		5	**10**	**15**	**20**	**25**	**Неприпустимі ризики**
		4	**8**	**12**	**16**	**20**	
	4, 7	**3**	**6**	**9**	**12**	**15**	**Виправдані ризики**
	1, 5, 6	**2**	**4**	**6**	**8**	**10**	
	2, 3	**1**	**2**	**3**	**4**	**5**	**Допустимі ризики**
			2, 6	**1, 3**	**4**	**5, 7**	
	Обсяг втрат						

Визначимо рівні ризиків та ступінь їх дії.

Рівні можуть бути:

– допустимі 1<R<4;

– виправдані 5<R<10;

– недопустимі 11<R<25.

Ступінь дії ризиків:

– ті, що можна проігнорувати 1<R<4;

– незначні 5<R<8;

– помірні 9<R<10;

– істотні 11<R<16;

– критичні 17<R<25.

На основі цих даних була виконана оцінка ступенів та рівнів для кожного ризику в проекті. Результати роботи представлені в табл. Б.4.

Таблиця Б.4 – Визначення ступенів та рівнів ризиків

№	Ризик	Ймовірність виникнення	Обсяг втрат	Індекс ризику	Рівень ризику	Ступінь дії
1	Форс-мажорні обставини	2	3	6	Виправданий	Незначний

Продовження таблиці Б.4

2	Людський фактор	1	2	2	Допустимий	Проігнорувати
3	Апаратно-програмні збої	1	3	3	Допустимий	Проігнорувати
4	Неправильне розподілення ресурсів	3	4	12	Неприпустимий	Істотний
5	Зміна вимог	2	5	10	Виправданий	Помірний
6	Зміна цілей в ході реалізації проекту	2	2	4	Допустимий	Проігнорувати
7	Неякісне проведення тестування	3	5	15	Неприпустимий	Істотний

Ризик неякісного проведення тестування можна уникнути при побудові календарного плану додавши часу на цей етап. Щодо неправильного розподілення ресурсів, його можна уникнути при глибокому аналізі задач та ретельному плануванню робіт.

Додаток В

Лістинг програмного коду

Файл Manifest.cs

```
using System.Collections.Generic;
using System.IO;
using System.Xml.Serialization;
namespace Launcher.Library.Models
{
    /// <summary>
    /// Workspace Launcher package.
    /// </summary>
    [XmlRoot(ElementName = "WorkspacePackage")]
    public class Manifest
    {
        /// <summary>
        /// Description in a free form. To be shown in UI.
        /// </summary>
        [XmlElement(ElementName = "Description")]
        public string Description { get; set; }
        /// <summary>
        /// Icons for a package.
        /// </summary>
        [XmlArray("Icons")]
        [XmlArrayItem("Icon", typeof(Icon))]
        public List<Icon> Icons { get; set; }
        /// <summary>
        /// Scripts to be launched during the installation.
        /// </summary>
        [XmlArray("Scripts")]
        [XmlArrayItem("Script", typeof(Script))]
        public List<Script> Scripts { get; set; }
        /// <summary>
        /// Shortcuts for the application.
        /// </summary>
        [XmlArray("Shortcuts")]
```

```
[XmlArrayItem("Shortcut", typeof(Shortcut))]
public List<Shortcut> Shortcuts { get; set; }
/// <summary>
/// Categories for the application.
/// </summary>
[XmlArray("Categories")]
[XmlArrayItem("Category", typeof(string))]
public List<string> Categories { get; set; }
/// <summary>
/// Categories for the application.
/// </summary>
[XmlArray("FileActions")]
[XmlArrayItem("FileAction", typeof(FileAction))]
public List<FileAction> FileActions { get; set; }
/// <summary>
/// Categories for the application.
/// </summary>
[XmlArray("EnvironmentVariables")]
[XmlArrayItem("EnvironmentVariable", typeof(EnvironmentVariable))]
public List<EnvironmentVariable> EnvironmentVariables { get; set; }
/// <summary>
/// Application registry keys.
/// </summary>
[XmlArray("RegistryKeys")]
[XmlArrayItem("RegistryKey", typeof(RegistryKey))]
public List<RegistryKey> RegistryKeys { get; set; }
/// <summary>
/// Registry virtual root for redirections.
/// </summary>
[XmlElement(ElementName = "RegistryVirtualRoot")]
public string RegistryVirtualRoot { get; set; }
/// <summary>
/// Files and registry keys redirections.
/// </summary>
[XmlArray("Redirections")]
[XmlArrayItem("Redirection", typeof(Redirection))]
public List<Redirection> Redirections { get; set; }
/// <summary>
/// Application entry points.
/// </summary>
[XmlArray("ProgramEntries")]
```

```
[XmlArrayItem("ProgramEntry", typeof(ProgramEntry))]
public List<ProgramEntry> ProgramEntries { get; set; }
/// <summary>
/// Application services.
/// </summary>
[XmlArray("Services")]
[XmlArrayItem("Service", typeof(Service))]
public List<Service> Services { get; set; }
/// <summary>
/// Unique application id in a format com.apptimized.ape.
/// </summary>
[XmlAttribute(AttributeName = "Id")]
public string Id { get; set; }
/// <summary>
/// Application normalized vendor name.
/// </summary>
[XmlAttribute(AttributeName = "Vendor")]
public string Vendor { get; set; }
/// <summary>
/// Application normalized name.
/// </summary>
[XmlAttribute(AttributeName = "Name")]
public string Name { get; set; }
/// <summary>
/// Application version in a semver format.
/// </summary>
[XmlAttribute(AttributeName = "Version")]
public string Version { get; set; }

public void Save(string fileName)
{
   using (var stream = new FileStream(fileName, FileMode.Create))
   {
      var XML = new XmlSerializer(typeof(Manifest));
      XML.Serialize(stream, this);
   }
}

public static Manifest LoadFromFile(string fileName)
{
   using (var stream = new FileStream(fileName, FileMode.Open))
```

```
            {
                var XML = new XmlSerializer(typeof(Manifest));
                return (Manifest)XML.Deserialize(stream);
            }
        }
    }
}
```

Файл Redirection.cs

```
using System.Xml.Serialization;
using Launcher.Library.Enums;

namespace Launcher.Library.Models
{
    /// <summary>
    /// Files, folders and registry keys redirections.
    /// </summary>
    public class Redirection
    {
        /// <summary>
        /// Redirected object type.
        /// </summary>
        [XmlAttribute(AttributeName = "Type")]
        public RedirectionType Type { get; set; }

        /// <summary>
        /// Initial target object path.
        /// </summary>
        [XmlAttribute(AttributeName = "From")]
        public string From { get; set; }

        /// <summary>
        /// Redirected object path.
        /// </summary>
        [XmlAttribute(AttributeName = "To")]
        public string To { get; set; }
    }
}
```

Файл RedirectionType.cs

```
namespace Launcher.Library.Enums
{
    /// <summary>
    /// Registry Value type.
    /// </summary>
    public enum RedirectionType
    {
        /// <summary>
        /// File redirection.
        /// </summary>
        File,

        /// <summary>
        /// Redirection for whole folder.
        /// </summary>
        Folder,

        /// <summary>
        /// Registry key redirection.
        /// </summary>
        Registry
    }
}
```

Файл ISystemMonitorEvent.cs

```
namespace Apptimized.Demo.SystemMonitor.Shared.Data.Interfaces
{
    public interface ISystemMonitorEvent
    {

    }
}
```

Файл FileSystemMonitorEvent.cs

```
using System;
using Apptimized.Demo.SystemMonitor.Shared.Data.Enums;
using Apptimized.Demo.SystemMonitor.Shared.Data.Interfaces;

namespace Apptimized.Demo.SystemMonitor.Shared.Data
{
   [Serializable]
   public class FileSystemMonitorEvent : ISystemMonitorEvent
   {
      public FileSystemOperationType Type { get; set; }

      public string Location { get; set; }
      public string NewLocation { get; set; }

      public FileSystemMonitorEvent()
      {

      }

      public FileSystemMonitorEvent(FileSystemOperationType type, string location)
      {
         Type = type;
         Location = location;
      }

      public FileSystemMonitorEvent(FileSystemOperationType type, string location, string newLocation)
      {
         Type = type;
         Location = location;
         NewLocation = newLocation;
      }

      public override string ToString() =>
         $"{Type}: {Location} {(string.IsNullOrEmpty(NewLocation) ? "" : $" > {NewLocation}")}";
   }
}
```

Файл ProcessMonitorEvent.cs

```
using System;
using Apptimized.Demo.SystemMonitor.Shared.Data.Enums;
using Apptimized.Demo.SystemMonitor.Shared.Data.Interfaces;

namespace Apptimized.Demo.SystemMonitor.Shared.Data
{
    [Serializable]
    public class ProcessMonitorEvent : ISystemMonitorEvent
    {
        public ProcessOperationType Type { get; set; }

        public string lpCommandLine { get; set; }
        public string lpApplicationName { get; set; }

        public ProcessMonitorEvent()
        {

        }

        public ProcessMonitorEvent(ProcessOperationType type, string command)
        {
            Type = type;
            lpCommandLine = command;
        }

        public ProcessMonitorEvent(ProcessOperationType type, string command, string appname)
        {
            Type = type;
            lpCommandLine = command;
            lpApplicationName = appname;
        }

        public override string ToString() =>
            $"{Type}: {lpCommandLine} {(string.IsNullOrEmpty(lpApplicationName) ? "" : $" for {lpApplicationName}")}";
    }
}
```

Файл RegistryKeyMonitorEvent.cs

```
using System;
using Apptimized.Demo.SystemMonitor.Shared.Data.Enums;
using Apptimized.Demo.SystemMonitor.Shared.Data.Interfaces;

namespace Apptimized.Demo.SystemMonitor.Shared.Data
{
   [Serializable]
   public class RegistryKeyMonitorEvent : ISystemMonitorEvent
   {
      public RegistryOperationType Type { get; set; }

      public string Key { get; set; }

      public string RedirectedRoot { get; set; }
      public RegistryKeyMonitorEvent()
      {

      }

      public RegistryKeyMonitorEvent(RegistryOperationType type, string key)
      {
         Type = type;
         Key = key;
      }

      public RegistryKeyMonitorEvent(RegistryOperationType type, string key, string root)
      {
         Type = type;
         Key = key;
         RedirectedRoot = root;
      }

      public override string ToString() =>
         $"{Type}: {Key} {(string.IsNullOrEmpty(RedirectedRoot) ? "" : $" > {RedirectedRoot}")}";
   }
}
```

Файл RegistryKeyValueMonitorEvent.cs

```
using System;
using Apptimized.Demo.SystemMonitor.Shared.Data.Enums;

namespace Apptimized.Demo.SystemMonitor.Shared.Data
{
    [Serializable]
    public class RegistryKeyValueMonitorEvent : RegistryKeyMonitorEvent
    {
        public string ValueName { get; set; }

        public RegistryKeyValueMonitorEvent()
        {

        }

        public RegistryKeyValueMonitorEvent(RegistryOperationType type, string key, string valueName) : base(type, key)
        {
            ValueName = valueName;
        }

        public RegistryKeyValueMonitorEvent(RegistryOperationType type, string key, string valueName, string root) : base(type, key)
        {
            ValueName = valueName;
            RedirectedRoot = root;
        }

        public override string ToString() =>
            $"{Type}: {(string.IsNullOrEmpty(ValueName) ? "Default" : ValueName)}@{Key}{(string.IsNullOrEmpty(RedirectedRoot) ? "" : $" > {RedirectedRoot}")}";
    }
}
```

Файл ProcessOperationType.cs

```
using System;
```

```
namespace Apptimized.Demo.SystemMonitor.Shared.Data.Enums
{
   [Serializable]
   public enum ProcessOperationType
   {
      CreateProcess
   }
}
```

Файл FileSystemOperationType.cs

```
using System;

namespace Apptimized.Demo.SystemMonitor.Shared.Data.Enums
{
   [Serializable]
   public enum FileSystemOperationType
   {
      CreateFile,
      DeleteFile,
      CreateDirectory,
      RemoveDirectory,
      ReadFile,
      WriteFile,
      MoveFile,
      MoveFileEx,
      OpenFile
   }
}
```

Файл RegistryOperationType.cs

```
using System;

namespace Apptimized.Demo.SystemMonitor.Shared.Data.Enums
{
   [Serializable]
   public enum RegistryOperationType
   {
      RegOpenKeyEx,
```

```
        RegOpenKey,
        RegCreateKey,
        RegDeleteKeyEx,
        RegDeleteKey,
        RegDeleteTree,
        RegGetValue,
        RegQueryValue,
        RegQueryValueEx,
        RegSetKeyValue,
        RegSetValue,
        RegSetValueEx,
        RegDeleteValue,
        RegDeleteKeyValue
    }
}
```

Файл KEY_INFORMATION_CLASS.cs

```
using System;

namespace Apptimized.Demo.SystemMonitor.Shared.Data.Enums
{
    [Serializable]
    public enum KEY_INFORMATION_CLASS
    {
        KeyBasicInformation,           // A KEY_BASIC_INFORMATION structure is supplied.
        KeyNodeInformation,            // A KEY_NODE_INFORMATION structure is supplied.
        KeyFullInformation,            // A KEY_FULL_INFORMATION structure is supplied.
        KeyNameInformation,            // A KEY_NAME_INFORMATION structure is supplied.
        KeyCachedInformation,          // A KEY_CACHED_INFORMATION structure is supplied.
        KeyFlagsInformation,           // Reserved for system use.
        KeyVirtualizationInformation,  // A KEY_VIRTUALIZATION_INFORMATION structure is supplied.
        KeyHandleTagsInformation,      // Reserved for system use.
        MaxKeyInfoClass                // The maximum value in this enumeration type.
    }
}
```

Файл SystemMonitorLogger.cs

```
using Serilog;

namespace Apptimized.Demo.SystemMonitor.Shared
{
   public static class SystemMonitorLogger
   {
      private static ILogger _instance;
      public static ILogger Instance => _instance ?? (
         _instance = new LoggerConfiguration()
            .WriteTo.Console()
            .MinimumLevel.Verbose()
            .CreateLogger());
   }
}
```

Файл SystemMonitorInterface.cs

```
using System;
using Apptimized.Demo.SystemMonitor.Shared.Data.Interfaces;
using Serilog;
namespace Apptimized.Demo.SystemMonitor.Shared
{
   public class SystemMonitorInterface : MarshalByRefObject
   {
      private readonly ILogger _logger = SystemMonitorLogger.Instance;
      public void ReportHooksInstallation() =>
         _logger.Information("All hooks have been installed");
      public void ReportInjection(int processId) =>
         _logger.Information("Successfully injected to the PID {pid}", processId);
      public void ReportSystemEvent(ISystemMonitorEvent monitorEvent)
      {
         _logger.Information(monitorEvent.ToString());
      }
      public void Ping()
      {
      }
   }
}
```

Файл Advapi32.cs

```
using System;
using System.Runtime.InteropServices;

namespace Apptimized.Demo.SystemMonitor.Hook.Interop
{
   internal static class Advapi32
   {
      [DllImport("advapi32.dll", CharSet = CharSet.Unicode, SetLastError = true,
         CallingConvention = CallingConvention.StdCall)]
      public static extern IntPtr RegOpenKeyExW(IntPtr hKey, string subKey, int ulOptions, int samDesired,
         IntPtr hkResult);

      [DllImport("advapi32.dll", CharSet = CharSet.Unicode, SetLastError = true,
         CallingConvention = CallingConvention.StdCall)]
      public static extern IntPtr RegOpenKeyW(IntPtr hKey, string subKey, IntPtr hkResult);

      [DllImport("advapi32.dll", CharSet = CharSet.Unicode, SetLastError = true,
         CallingConvention = CallingConvention.StdCall)]
      public static extern IntPtr RegCreateKeyW(IntPtr hKey, string subKey, IntPtr hkResult);

      [DllImport("advapi32.dll", CharSet = CharSet.Unicode, SetLastError = true,
         CallingConvention = CallingConvention.StdCall)]
      public static extern IntPtr RegDeleteKeyExW(IntPtr hKey, string subKey, int samDesired, int reserved);

      [DllImport("advapi32.dll", CharSet = CharSet.Unicode, SetLastError = true,
         CallingConvention = CallingConvention.StdCall)]
      public static extern IntPtr RegDeleteKeyW(IntPtr hKey, string subKey);

      [DllImport("advapi32.dll", CharSet = CharSet.Unicode, SetLastError = true,
         CallingConvention = CallingConvention.StdCall)]
      public static extern IntPtr RegDeleteTreeW(IntPtr hKey, string subKey);

      [DllImport("advapi32.dll", CharSet = CharSet.Unicode, SetLastError = true,
         CallingConvention = CallingConvention.StdCall)]
      public static extern IntPtr RegGetValueW(IntPtr hKey, string subKey, string value, uint flags, out uint type,
         IntPtr vData, out uint pData);

      [DllImport("advapi32.dll", CharSet = CharSet.Unicode, SetLastError = true,
         CallingConvention = CallingConvention.StdCall)]
```

```
        public static extern IntPtr RegQueryValueW(IntPtr hKey, string subKey, IntPtr lpData, ref int lpcbData);

        [DllImport("advapi32.dll", CharSet = CharSet.Unicode, SetLastError = true,
            CallingConvention = CallingConvention.StdCall)]
        public static extern IntPtr RegQueryValueExW(IntPtr hKey, string valueName, int reserved, out uint type,
            IntPtr data, out uint cbData);

        [DllImport("advapi32.dll", CharSet = CharSet.Unicode,
            SetLastError = true, CallingConvention = CallingConvention.StdCall)]
        public static extern IntPtr RegSetKeyValueW(IntPtr hKey, string subKey, string valueName, uint type,
            IntPtr data, uint cbData);

        [DllImport("advapi32.dll", CharSet = CharSet.Unicode, SetLastError = true,
            CallingConvention = CallingConvention.StdCall)]
        public static extern IntPtr RegSetValueW(IntPtr hKey, string subKey, uint type, IntPtr data, uint cbData);

        [DllImport("advapi32.dll", CharSet = CharSet.Unicode, SetLastError = true,
            CallingConvention = CallingConvention.StdCall)]
        public static extern IntPtr RegSetValueExW(IntPtr hKey, string valueName, uint reserved, uint type, IntPtr data,
            uint cbData);

        [DllImport("advapi32.dll", CharSet = CharSet.Unicode, SetLastError = true,
            CallingConvention = CallingConvention.StdCall)]
        public static extern IntPtr RegDeleteValueW(IntPtr hKey, string valueName);

        [DllImport("advapi32.dll", CharSet = CharSet.Unicode, SetLastError = true,
            CallingConvention = CallingConvention.StdCall)]
        public static extern IntPtr RegDeleteKeyValueW(IntPtr hKey, string subKey, string valueName);
    }
}
```

Файл Kernel32.cs

```
using System;
using System.Runtime.InteropServices;
using System.Text;

namespace Apptimized.Demo.SystemMonitor.Hook.Interop
{
    internal static class Kernel32
    {
```

```
[StructLayout(LayoutKind.Sequential)]
public struct PROCESS_INFORMATION
{
   public IntPtr hProcess;
   public IntPtr hThread;
   public int dwProcessId;
   public int dwThreadId;
}

[DllImport("kernel32.dll", CharSet = CharSet.Unicode, SetLastError = true,
   CallingConvention = CallingConvention.StdCall)]
public static extern IntPtr MoveFileExW(string existingFileName, string newFileName, uint dwFlags);

[DllImport("kernel32.dll", CharSet = CharSet.Unicode, SetLastError = true,
   CallingConvention = CallingConvention.StdCall)]
public static extern IntPtr CreateFileW(string filename, uint desiredAccess, uint shareMode,
   IntPtr securityAttributes, uint creationDisposition, uint flagsAndAttributes, IntPtr templateFile);

[DllImport("kernel32.dll", CharSet = CharSet.Unicode, SetLastError = true,
   CallingConvention = CallingConvention.StdCall)]
public static extern IntPtr DeleteFileW(string filename);

[DllImport("kernel32.dll", CharSet = CharSet.Unicode, SetLastError = true,
   CallingConvention = CallingConvention.StdCall)]
public static extern IntPtr CreateDirectoryW(string dirname, IntPtr securityAttributes);

[DllImport("kernel32.dll", CharSet = CharSet.Unicode, SetLastError = true,
   CallingConvention = CallingConvention.StdCall)]
public static extern IntPtr RemoveDirectoryW(string dirname);

[DllImport("kernel32.dll", SetLastError = true, CallingConvention = CallingConvention.StdCall)]
public static extern bool ReadFile(IntPtr hFile, IntPtr lpBuffer, uint nNumberOfBytesToRead,
   out uint lpNumberOfBytesRead, IntPtr lpOverlapped);

[DllImport("kernel32.dll", CharSet = CharSet.Unicode, SetLastError = true,
   CallingConvention = CallingConvention.StdCall)]
[return: MarshalAs(UnmanagedType.Bool)]
public static extern bool WriteFile(IntPtr hFile, IntPtr lpBuffer, uint nNumberOfBytesToWrite,
   out uint lpNumberOfBytesWritten, IntPtr lpOverlapped);

[DllImport("kernel32.dll", CharSet = CharSet.Unicode, SetLastError = true,
```

```
        CallingConvention = CallingConvention.StdCall)]
    public static extern IntPtr MoveFileW(string existingFileName, string newFileName);

    [DllImport("Kernel32.dll", SetLastError = true, CharSet = CharSet.Auto)]
    public static extern uint GetFinalPathNameByHandle(IntPtr hFile,
        [MarshalAs(UnmanagedType.LPTStr)] StringBuilder lpszFilePath, uint cchFilePath, uint dwFlags);

    [DllImport("Kernel32.dll", SetLastError = true, CharSet = CharSet.Auto)]
    public static extern IntPtr OpenFile(string filename, IntPtr lpReOpenBuff, uint uStyle);

    [DllImport("Kernel32.dll", SetLastError = true, CharSet = CharSet.Auto)]
    public static extern bool CreateProcess(string lpApplicationName, string lpCommandLine, ref IntPtr lpProcessAttributes,
        ref IntPtr lpThreadAttributes, bool bInheritHandles, uint dwCreationFlags, IntPtr lpEnvironment, string lpCurrentDirectory,
        ref IntPtr lpStartupInfo, out PROCESS_INFORMATION lpProcessInformation);
  }
}
```

Файл AbstractBaseHook.cs

```
using System;
using System.Collections.Concurrent;
using System.Collections.Generic;
using Apptimized.Demo.SystemMonitor.Shared.Data.Interfaces;
using EasyHook;

namespace Apptimized.Demo.SystemMonitor.Hook.Hooks
{
  internal abstract class AbstractBaseHook : IDisposable
  {
    protected readonly ConcurrentQueue<ISystemMonitorEvent> Queue;

    private readonly List<LocalHook> _hooks = new List<LocalHook>();

    protected AbstractBaseHook(ConcurrentQueue<ISystemMonitorEvent> queue)
    {
      Queue = queue;
    }

    protected void InstallHook(string moduleName, string symbolName, Delegate hookDelegate)
```

```
        {
            var hook = LocalHook.Create(
                LocalHook.GetProcAddress(moduleName, symbolName), hookDelegate, this);

            hook.ThreadACL.SetExclusiveACL(new[] {0});

            _hooks.Add(hook);
        }

        protected void InstallUniversalHook(string moduleName, string symbolName, Delegate hookDelegate)
        {
            InstallHook(moduleName, symbolName + "A", hookDelegate);
            InstallHook(moduleName, symbolName + "W", hookDelegate);
        }

        public void Dispose()
        {
            foreach (var hook in _hooks)
                hook?.Dispose();
        }
    }
}
```

Файл ProcessHook.cs

```
using System;
using System.Collections;
using System.Collections.Concurrent;
using System.Collections.Generic;
using System.Runtime.InteropServices;
using System.Text;
using Apptimized.Demo.SystemMonitor.Hook.Interop;
using Apptimized.Demo.SystemMonitor.Shared;
using Apptimized.Demo.SystemMonitor.Shared.Data;
using Apptimized.Demo.SystemMonitor.Shared.Data.Enums;
using Apptimized.Demo.SystemMonitor.Shared.Data.Interfaces;
using EasyHook;
using System.Text.RegularExpressions;
using System.Windows;

namespace Apptimized.Demo.SystemMonitor.Hook.Hooks
```

```
{
    [UnmanagedFunctionPointer(CallingConvention.StdCall, CharSet = CharSet.Unicode, SetLastError = true)]
    internal delegate void CreateProcessDelegate(string lpApplicationName, string lpCommandLine, ref IntPtr lpProcessAttributes,
        ref IntPtr lpThreadAttributes, bool bInheritHandles, uint dwCreationFlags, IntPtr lpEnvironment, string lpCurrentDirectory,
        ref IntPtr lpStartupInfo, Kernel32.PROCESS_INFORMATION lpProcessInformation);

    internal class ProcessHook : AbstractBaseHook
    {
        private static string _hookPath;
        private static string _channelName;

        //public void SetChannelName(string channel) { _channelName = channel; }
        public ProcessHook(ConcurrentQueue<ISystemMonitorEvent> queue) : base(queue)
        {
            // https://docs.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-createprocessa
            // https://docs.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-createprocessw
            InstallUniversalHook("kernel32.dll", "CreateProcess", new CreateProcessDelegate(CreateProcessHook));
        }

        private void CreateProcessHook(string lpApplicationName, string lpCommandLine, ref IntPtr lpProcessAttributes,
            ref IntPtr lpThreadAttributes, bool bInheritHandles, uint dwCreationFlags, IntPtr lpEnvironment, string lpCurrentDirectory,
            ref IntPtr lpStartupInfo, Kernel32.PROCESS_INFORMATION lpProcessInformation)
        {
            Queue.Enqueue(new ProcessMonitorEvent(ProcessOperationType.CreateProcess, lpCommandLine));

            _hookPath = typeof(SystemMonitorEntryPoint).Assembly.Location;
            _channelName = SystemMonitorEntryPoint.GetChannelName();

            string processName = "";
            string arguments = "";
            try
            {
                MatchCollection matches = Regex.Matches(lpCommandLine, @"^(.*"" )");
                if (matches.Count > 0)
                {
                    //If executable path contains quotes
                    processName = matches[0].Value.Remove(matches[0].Value.Length - 2, 2);
```

```
            processName = processName.Remove(0,1);
            arguments = lpCommandLine.Remove(0, processName.Length + 3);
        }
        else
        {
            //If executable path doesn't contain quotes
            processName = Regex.Matches(lpCommandLine, @"^\S*")[0].Value;
            arguments = lpCommandLine.Remove(0, processName.Length + 1);
        }
      }
      catch (Exception exception)
      {
        SystemMonitorLogger.Instance.Error(exception,
            "Critical error occured while injecting to {path} with arguments {arguments}",
            processName, arguments);
      }

     // MessageBox.Show($"full cmd='{lpCommandLine}'");
     // MessageBox.Show($"proc='{processName}'; \nparams='{arguments}'");
      try
      {
        RemoteHooking.CreateAndInject(
            processName,
            arguments,
            0,
            InjectionOptions.DoNotRequireStrongName,
            _hookPath,
            _hookPath,
            out var processId,
            _channelName);
        RemoteHooking.WakeUpProcess();
      }
      catch (Exception exception)
      {
        SystemMonitorLogger.Instance.Error(exception,
            "Critical error occured while injecting to {path} with arguments {arguments}",
            processName, arguments);
      }

    }
}
```

```
}
```

Файл FileSystemHook.cs

```
using System;
using System.Collections.Concurrent;
using System.IO;
using System.Runtime.InteropServices;
using System.Text;
using System.Text.RegularExpressions;
using Apptimized.Demo.SystemMonitor.Hook.Interop;
using Apptimized.Demo.SystemMonitor.Shared;
using Apptimized.Demo.SystemMonitor.Shared.Data;
using Apptimized.Demo.SystemMonitor.Shared.Data.Enums;
using Apptimized.Demo.SystemMonitor.Shared.Data.Interfaces;
using Launcher.Library.Enums;
using Launcher.Library.Models;

namespace Apptimized.Demo.SystemMonitor.Hook.Hooks
{
    [UnmanagedFunctionPointer(CallingConvention.StdCall, CharSet = CharSet.Unicode, SetLastError = true)]
    internal delegate IntPtr CreateFileDelegate(string filename, uint desiredAccess, uint shareMode,
        IntPtr securityAttributes, uint creationDisposition, uint flagsAndAttributes, IntPtr templateFile);

    [UnmanagedFunctionPointer(CallingConvention.StdCall, CharSet = CharSet.Unicode, SetLastError = true)]
    internal delegate IntPtr DeleteFileDelegate(string filename);

    [UnmanagedFunctionPointer(CallingConvention.StdCall, CharSet = CharSet.Unicode, SetLastError = true)]
    internal delegate IntPtr CreateDirectoryDelegate(string dirname, IntPtr securityAttributes);

    [UnmanagedFunctionPointer(CallingConvention.StdCall, CharSet = CharSet.Unicode, SetLastError = true)]
    internal delegate IntPtr RemoveDirectoryDelegate(string dirname);

    [UnmanagedFunctionPointer(CallingConvention.StdCall, SetLastError = true)]
    internal delegate bool ReadFileDelegate(IntPtr hFile, IntPtr lpBuffer, uint nNumberOfBytesToRead,
        out uint lpNumberOfBytesRead, IntPtr lpOverlapped);

    [UnmanagedFunctionPointer(CallingConvention.StdCall, CharSet = CharSet.Unicode, SetLastError = true)]
    [return: MarshalAs(UnmanagedType.Bool)]
    internal delegate bool WriteFileDelegate(IntPtr hFile, IntPtr lpBuffer, uint nNumberOfBytesToWrite,
        out uint lpNumberOfBytesWritten, IntPtr lpOverlapped);
```

```
        [UnmanagedFunctionPointer(CallingConvention.StdCall, CharSet = CharSet.Unicode, SetLastError = true)]
        internal delegate IntPtr MoveFileDelegate(string existingFileName, string newFileName);

        [UnmanagedFunctionPointer(CallingConvention.StdCall, CharSet = CharSet.Unicode, SetLastError = true)]
        internal delegate IntPtr MoveFileExDelegate(string existingFileName, string newFileName, uint dwFlags);

        [UnmanagedFunctionPointer(CallingConvention.StdCall, CharSet = CharSet.Unicode, SetLastError = true)]
        internal delegate IntPtr OpenFileDelegate(string filename, IntPtr lpReOpenBuff, uint uStyle);

        internal class FileSystemHook : AbstractBaseHook
        {
            public FileSystemHook(ConcurrentQueue<ISystemMonitorEvent> queue) : base(queue)
            {
                _manifest = Manifest.LoadFromFile(Path.GetDirectoryName(typeof(FileSystemHook).Assembly.Location) +
@"\manifest.xml");

                // https://docs.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-createfilea
                // https://docs.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-createfilew
                InstallUniversalHook("kernel32.dll", "CreateFile", new CreateFileDelegate(CreateFileHook));

                // https://msdn.microsoft.com/en-us/library/windows/desktop/aa365467(v=vs.85).aspx
                InstallHook("kernel32.dll", "ReadFile", new ReadFileDelegate(ReadFileHook));

                // https://msdn.microsoft.com/en-us/library/windows/desktop/aa365747(v=vs.85).aspx
                InstallHook("kernel32.dll", "WriteFile", new WriteFileDelegate(WriteFileHook));

                // https://docs.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-createdirectorya
                // https://docs.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-createdirectoryw
                InstallUniversalHook("kernel32.dll", "CreateDirectory",
                    new CreateDirectoryDelegate(CreateDirectoryHook));

                // https://docs.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-removedirectorya
                // https://docs.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-removedirectoryw
                InstallUniversalHook("kernel32.dll", "RemoveDirectory",
                    new RemoveDirectoryDelegate(RemoveDirectoryHook));

                // https://docs.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-deletefilea
                // https://docs.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-deletefilew
                InstallUniversalHook("kernel32.dll", "DeleteFile", new DeleteFileDelegate(DeleteFileHook));

                // https://docs.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-movefilea
```

```
            // https://docs.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-movefilew
            InstallUniversalHook("kernel32.dll", "MoveFile", new MoveFileDelegate(MoveFileHook));

            // https://docs.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-movefileexa
            // https://docs.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-movefileexw
            InstallUniversalHook("kernel32.dll", "MoveFileEx", new MoveFileExDelegate(MoveFileExHook));

            // https://docs.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-openfile
            InstallHook("kernel32.dll", "OpenFile", new OpenFileDelegate(OpenFileHook));
        }
        private static Manifest _manifest;
        private static string GetRedirectedFilePath(string _filename)
        {
            try
            {
                string pathToFind = Regex.Matches(_filename, @"\w{1}:\\.*$")[0].Value;
                string dirPath = Path.GetDirectoryName(pathToFind);

                string prefix = pathToFind.Remove(_filename.Length - pathToFind.Length);

                foreach (Redirection redirection in _manifest.Redirections)
                {
                    switch (redirection.Type)
                    {
                        case RedirectionType.Folder:
                            if (dirPath.Equals(Environment.ExpandEnvironmentVariables(redirection.From),
StringComparison.OrdinalIgnoreCase))
                                return $"{prefix}" +
                                    $"{Environment.ExpandEnvironmentVariables(redirection.To).Replace("%DefaultDir%",
Environment.ExpandEnvironmentVariables(@"%localappdata%\Apptimized\Launcher\" + _manifest.Id))}" +
                                    $"\\{Path.GetFileName(pathToFind)}";
                            else
                            {   // Check if subfolders fit the mask
                                string sub = Path.GetDirectoryName(dirPath);
                                while (!(string.IsNullOrEmpty(Path.GetDirectoryName(sub))))
                                {
                                    if (sub.Equals(Environment.ExpandEnvironmentVariables(redirection.From),
StringComparison.OrdinalIgnoreCase))
                                    {
                                        return $"{prefix}" +
```

```
                                $"{Environment.ExpandEnvironmentVariables(redirection.To).Replace("%DefaultDir%",
Environment.ExpandEnvironmentVariables(@"%localappdata%\Apptimized\Launcher\" + _manifest.Id))}" +
                                $"{pathToFind.Remove(0,
Environment.ExpandEnvironmentVariables(redirection.From).Length)}";
                            }
                            sub = Path.GetDirectoryName(sub);
                        }
                    }
                    break;
                case RedirectionType.File:
                    if (pathToFind.Equals(Environment.ExpandEnvironmentVariables(redirection.From),
StringComparison.OrdinalIgnoreCase))
                        return $"{prefix}" +
                            $"{Environment.ExpandEnvironmentVariables(redirection.To).Replace("%DefaultDir%",
Environment.ExpandEnvironmentVariables(@"%localappdata%\Apptimized\Launcher\" + _manifest.Id))}";
                    break;
                default:
                    break;
            }
        }
    }
    catch
    {
        return _filename;
    }
    return _filename;
}

private static string GetRedirectedFolderPath(string _foldername)
{
    try
    {
        string pathToFind = Regex.Matches(_foldername, @"\w{1}:\\.*$")[0].Value;

        string prefix = pathToFind.Remove(_foldername.Length - pathToFind.Length);

        foreach (Redirection redirection in _manifest.Redirections)
        {
            if (redirection.Type == RedirectionType.Folder)
            {
```

```
                    if (pathToFind.Equals(Environment.ExpandEnvironmentVariables(redirection.From),
StringComparison.OrdinalIgnoreCase))
                        return $"{prefix}" +
                            $"{Environment.ExpandEnvironmentVariables(redirection.To).Replace("%DefaultDir%",
Environment.ExpandEnvironmentVariables(@"%localappdata%\Apptimized\Launcher\" + _manifest.Id))}";
                    else
                    {
                        string sub = pathToFind;
                        while (!(string.IsNullOrEmpty(Path.GetDirectoryName(sub))))
                        {
                            if (sub.Equals(Environment.ExpandEnvironmentVariables(redirection.From),
StringComparison.OrdinalIgnoreCase))
                            {
                                return $"{prefix}{Environment.ExpandEnvironmentVariables(redirection.To)}" +
                                    $"{pathToFind.Remove(0,
Environment.ExpandEnvironmentVariables(redirection.From).Length)}";
                            }
                            sub = Path.GetDirectoryName(sub);
                        }
                    }
                }
            }
            catch
            {
                return _foldername;
            }
            return _foldername;
        }

        private IntPtr CreateFileHook(string filename, uint desiredAccess, uint shareMode, IntPtr securityAttributes,
            uint creationDisposition, uint flagsAndAttributes, IntPtr templateFile)
        {
            string target = GetRedirectedFilePath(filename);

            Queue.Enqueue(new FileSystemMonitorEvent(FileSystemOperationType.CreateFile, filename, target));
            return Kernel32.CreateFileW(target, desiredAccess, shareMode, securityAttributes, creationDisposition,
                flagsAndAttributes, templateFile);
        }

        private IntPtr DeleteFileHook(string filename)
```

```
        {
            string target = GetRedirectedFilePath(filename);

            Queue.Enqueue(
                new FileSystemMonitorEvent(FileSystemOperationType.DeleteFile, filename, target));
            return Kernel32.DeleteFileW(target);
        }

        private IntPtr CreateDirectoryHook(string dirname, IntPtr securityAttributes)
        {
            string target = GetRedirectedFolderPath(dirname);
            Queue.Enqueue(
                new FileSystemMonitorEvent(FileSystemOperationType.CreateDirectory, dirname, target));

            return Kernel32.CreateDirectoryW(target, securityAttributes);
        }

        private IntPtr RemoveDirectoryHook(string dirname)
        {
            string target = GetRedirectedFolderPath(dirname);
            Queue.Enqueue(
                new FileSystemMonitorEvent(FileSystemOperationType.RemoveDirectory, dirname, target));

            return Kernel32.RemoveDirectoryW(target);
        }

        private bool ReadFileHook(IntPtr hFile, IntPtr lpBuffer, uint nNumberOfBytesToRead,
            out uint lpNumberOfBytesRead, IntPtr lpOverlapped)
        {
            var fileNameBuffer = new StringBuilder(255);

            Kernel32.GetFinalPathNameByHandle(hFile, fileNameBuffer, 255, 0);

            string target = GetRedirectedFilePath(fileNameBuffer.ToString());

            Queue.Enqueue(
                new FileSystemMonitorEvent(FileSystemOperationType.ReadFile, fileNameBuffer.ToString(), target));

            return Kernel32.ReadFile(hFile, lpBuffer, nNumberOfBytesToRead, out lpNumberOfBytesRead,
lpOverlapped);
        }
```

```
        private bool WriteFileHook(IntPtr hFile, IntPtr lpBuffer, uint nNumberOfBytesToWrite,
            out uint lpNumberOfBytesWritten, IntPtr lpOverlapped)
        {
            var fileNameBuffer = new StringBuilder(255);

            Kernel32.GetFinalPathNameByHandle(hFile, fileNameBuffer, 255, 0);
            string target = GetRedirectedFilePath(fileNameBuffer.ToString());

            Queue.Enqueue(
                new FileSystemMonitorEvent(FileSystemOperationType.WriteFile, fileNameBuffer.ToString(), target));

            return Kernel32.WriteFile(hFile, lpBuffer, nNumberOfBytesToWrite, out lpNumberOfBytesWritten,
lpOverlapped);
        }

        private IntPtr MoveFileHook(string existingFileName, string newFileName)
        {
            existingFileName = GetRedirectedFilePath(existingFileName);
            newFileName = GetRedirectedFilePath(newFileName);
            Queue.Enqueue(
                new FileSystemMonitorEvent(FileSystemOperationType.MoveFile, existingFileName, newFileName));

            return Kernel32.MoveFileW(existingFileName, newFileName);
        }

        private IntPtr MoveFileExHook(string existingFileName, string newFileName, uint dwFlags)
        {
            existingFileName = GetRedirectedFilePath(existingFileName);
            newFileName = GetRedirectedFilePath(newFileName);
            Queue.Enqueue(
                new FileSystemMonitorEvent(FileSystemOperationType.MoveFileEx, existingFileName, newFileName));

            return Kernel32.MoveFileExW(existingFileName, newFileName, dwFlags);
        }

        private IntPtr OpenFileHook(string filename, IntPtr lpReOpenBuff, uint uStyle)
        {
            string target = GetRedirectedFilePath(filename);
            Queue.Enqueue(
                new FileSystemMonitorEvent(FileSystemOperationType.OpenFile, filename, target));
```

```
            return Kernel32.OpenFile(target, lpReOpenBuff, uStyle);
        }
    }
}
```

Файл RegistryHook.cs

```
using System;
using System.Collections.Concurrent;
using System.IO;
using System.Linq;
using System.Runtime.InteropServices;
using Apptimized.Demo.SystemMonitor.Hook.Interop;
using Apptimized.Demo.SystemMonitor.Shared.Data;
using Apptimized.Demo.SystemMonitor.Shared.Data.Enums;
using Apptimized.Demo.SystemMonitor.Shared.Data.Interfaces;
using Launcher.Library.Enums;
using Launcher.Library.Models;

namespace Apptimized.Demo.SystemMonitor.Hook.Hooks
{
    [UnmanagedFunctionPointer(CallingConvention.StdCall, CharSet = CharSet.Unicode, SetLastError = true)]
    internal delegate IntPtr RegOpenKeyExDelegate(IntPtr hKey, string subKey, int ulOptions, int samDesired,
        IntPtr hkResult);

    [UnmanagedFunctionPointer(CallingConvention.StdCall, CharSet = CharSet.Unicode, SetLastError = true)]
    internal delegate IntPtr RegOpenKeyDelegate(IntPtr hKey, string subKey, IntPtr hkResult);

    [UnmanagedFunctionPointer(CallingConvention.StdCall, CharSet = CharSet.Unicode, SetLastError = true)]
    internal delegate IntPtr RegCreateKeyDelegate(IntPtr hKey, string subKey, IntPtr hkResult);

    [UnmanagedFunctionPointer(CallingConvention.StdCall, CharSet = CharSet.Unicode, SetLastError = true)]
    internal delegate IntPtr RegDeleteKeyExDelegate(IntPtr hKey, string subKey, int samDesired, int reserved);

    [UnmanagedFunctionPointer(CallingConvention.StdCall, CharSet = CharSet.Unicode, SetLastError = true)]
    internal delegate IntPtr RegDeleteKeyDelegate(IntPtr hKey, string subKey);

    [UnmanagedFunctionPointer(CallingConvention.StdCall, CharSet = CharSet.Unicode, SetLastError = true)]
    internal delegate IntPtr RegDeleteTreeDelegate(IntPtr hKey, string subKey);
```

```
        [UnmanagedFunctionPointer(CallingConvention.StdCall, CharSet = CharSet.Unicode, SetLastError = true)]
        internal delegate IntPtr RegGetValueDelegate(IntPtr hKey, string subKey, string value, uint flags, out uint type,
            IntPtr vData, out uint pData);

        [UnmanagedFunctionPointer(CallingConvention.StdCall, CharSet = CharSet.Unicode, SetLastError = true)]
        internal delegate IntPtr RegQueryValueDelegate(IntPtr hKey, string subKey, IntPtr lpData, ref int lpcbData);

        [UnmanagedFunctionPointer(CallingConvention.StdCall, CharSet = CharSet.Unicode, SetLastError = true)]
        internal delegate IntPtr RegQueryValueExDelegate(IntPtr hKey, string valueName, int reserved, out uint type,
            IntPtr data, out uint cbData);

        [UnmanagedFunctionPointer(CallingConvention.StdCall, CharSet = CharSet.Unicode, SetLastError = true)]
        internal delegate IntPtr RegSetKeyValueDelegate(IntPtr hKey, string subKey, string valueName, uint type,
            IntPtr data, uint cbData);

        [UnmanagedFunctionPointer(CallingConvention.StdCall, CharSet = CharSet.Unicode, SetLastError = true)]
        internal delegate IntPtr RegSetValueDelegate(IntPtr hKey, string subKey, uint type, IntPtr data, uint cbData);

        [UnmanagedFunctionPointer(CallingConvention.StdCall, CharSet = CharSet.Unicode, SetLastError = true)]
        internal delegate IntPtr RegSetValueExDelegate(IntPtr hKey, string valueName, uint reserved, uint type,
            IntPtr data, uint cbData);

        [UnmanagedFunctionPointer(CallingConvention.StdCall, CharSet = CharSet.Unicode, SetLastError = true)]
        internal delegate IntPtr RegDeleteValueDelegate(IntPtr hKey, string valueName);

        [UnmanagedFunctionPointer(CallingConvention.StdCall, CharSet = CharSet.Unicode, SetLastError = true)]
        internal delegate IntPtr RegDeleteKeyValueDelegate(IntPtr hKey, string subKey, string valueName);

        internal class RegistryHook : AbstractBaseHook
        {
            public RegistryHook(ConcurrentQueue<ISystemMonitorEvent> queue) : base(queue)
            {
                _manifest = Manifest.LoadFromFile(Path.GetDirectoryName(typeof(FileSystemHook).Assembly.Location) +
@"\manifest.xml");
                // https://docs.microsoft.com/en-us/windows/win32/api/winreg/nf-winreg-regopenkeya
                // https://docs.microsoft.com/en-us/windows/win32/api/winreg/nf-winreg-regopenkeyw
                InstallUniversalHook("advapi32.dll", "RegOpenKey", new RegOpenKeyDelegate(RegOpenKeyHook));

                // https://docs.microsoft.com/en-us/windows/win32/api/winreg/nf-winreg-regopenkeyexa
                // https://docs.microsoft.com/en-us/windows/win32/api/winreg/nf-winreg-regopenkeyexw
```

```
            InstallUniversalHook("advapi32.dll", "RegOpenKeyEx", new
RegOpenKeyExDelegate(RegOpenKeyExHook));

            // https://docs.microsoft.com/en-us/windows/win32/api/winreg/nf-winreg-regcreatekeya
            // https://docs.microsoft.com/en-us/windows/win32/api/winreg/nf-winreg-regcreatekeyw
            InstallUniversalHook("advapi32.dll", "RegCreateKey", new RegCreateKeyDelegate(RegCreateKeyHook));

            // https://docs.microsoft.com/en-us/windows/win32/api/winreg/nf-winreg-regcreatekeyexa
            // https://docs.microsoft.com/en-us/windows/win32/api/winreg/nf-winreg-regcreatekeyexw
            InstallUniversalHook("advapi32.dll", "RegCreateKeyEx", new RegCreateKeyDelegate(RegCreateKeyHook));

            // https://docs.microsoft.com/en-us/windows/win32/api/winreg/nf-winreg-regdeletekeyexa
            // https://docs.microsoft.com/en-us/windows/win32/api/winreg/nf-winreg-regdeletekeyexw
            InstallUniversalHook("advapi32.dll", "RegDeleteKeyEx", new
RegDeleteKeyExDelegate(RegDeleteKeyExHook));

            // https://docs.microsoft.com/en-us/windows/win32/api/winreg/nf-winreg-regdeletekeya
            // https://docs.microsoft.com/en-us/windows/win32/api/winreg/nf-winreg-regdeletekeyw
            InstallUniversalHook("advapi32.dll", "RegDeleteKey", new RegDeleteKeyDelegate(RegDeleteKeyHook));

            // https://docs.microsoft.com/en-us/windows/win32/api/winreg/nf-winreg-regdeletetreea
            // https://docs.microsoft.com/en-us/windows/win32/api/winreg/nf-winreg-regdeletetreews
            InstallUniversalHook("advapi32.dll", "RegDeleteTree", new RegDeleteTreeDelegate(RegDeleteTreeHook));

            // https://docs.microsoft.com/en-us/windows/win32/api/winreg/nf-winreg-reggetvaluea
            // https://docs.microsoft.com/en-us/windows/win32/api/winreg/nf-winreg-reggetvaluew
            InstallUniversalHook("advapi32.dll", "RegGetValue", new RegGetValueDelegate(RegGetValueHook));

            // https://docs.microsoft.com/en-us/windows/win32/api/winreg/nf-winreg-regqueryvaluea
            // https://docs.microsoft.com/en-us/windows/win32/api/winreg/nf-winreg-regqueryvaluew
            InstallUniversalHook("advapi32.dll", "RegQueryValue", new
RegQueryValueDelegate(RegQueryValueHook));

            // https://docs.microsoft.com/en-us/windows/win32/api/winreg/nf-winreg-regqueryvalueexa
            // https://docs.microsoft.com/en-us/windows/win32/api/winreg/nf-winreg-regqueryvalueexw
            InstallUniversalHook("advapi32.dll", "RegQueryValueEx",
                new RegQueryValueExDelegate(RegQueryValueExHook));

            // https://docs.microsoft.com/en-us/windows/win32/api/winreg/nf-winreg-regsetkeyvaluea
            // https://docs.microsoft.com/en-us/windows/win32/api/winreg/nf-winreg-regsetkeyvaluew
```

```
            InstallUniversalHook("advapi32.dll", "RegSetKeyValue", new
RegSetKeyValueDelegate(RegSetKeyValueHook));

            // https://docs.microsoft.com/en-us/windows/win32/api/winreg/nf-winreg-regsetvaluea
            // https://docs.microsoft.com/en-us/windows/win32/api/winreg/nf-winreg-regsetvaluew
            InstallUniversalHook("advapi32.dll", "RegSetValue", new RegSetValueDelegate(RegSetValueHook));

            // https://docs.microsoft.com/en-us/windows/win32/api/winreg/nf-winreg-regsetvaluea
            // https://docs.microsoft.com/en-us/windows/win32/api/winreg/nf-winreg-regsetvaluew
            InstallUniversalHook("advapi32.dll", "RegSetValueEx", new
RegSetValueExDelegate(RegSetValueExHook));

            // https://docs.microsoft.com/en-us/windows/win32/api/winreg/nf-winreg-regdeletevaluea
            // https://docs.microsoft.com/en-us/windows/win32/api/winreg/nf-winreg-regdeletevaluew
            InstallUniversalHook("advapi32.dll", "RegDeleteValue", new
RegDeleteValueDelegate(RegDeleteValueHook));

            // https://docs.microsoft.com/en-us/windows/win32/api/winreg/nf-winreg-regdeletekeyvaluea
            // https://docs.microsoft.com/en-us/windows/win32/api/winreg/nf-winreg-regdeletekeyvaluew
            InstallUniversalHook("advapi32.dll", "RegDeleteKeyValue",
                new RegDeleteKeyValueDelegate(RegDeleteKeyValueHook));
        }

        [DllImport("ntdll.dll", SetLastError = true, CharSet = CharSet.Unicode)]
        private static extern int ZwQueryKey(IntPtr hKey, KEY_INFORMATION_CLASS KeyInformationClass, IntPtr
lpKeyInformation, int Length, out int ResultLength);

        private static string KeyHandleToString(IntPtr handle)
        {
            switch (handle.ToInt64())
            {
                case -2147483648:
                    return "HKEY_CLASSES_ROOT";
                case -2147483643:
                    return "HKEY_CURRENT_CONFIG";
                case -2147483647:
                    return "HKEY_CURRENT_USER";
                case -2147483646:
                    return "HKEY_LOCAL_MACHINE";
                case -2147483645:
                    return "HKEY_USERS";
```

```
                default:
                    String result = String.Empty;
                    IntPtr pKNI = IntPtr.Zero;

                    int needed = 0;
                    int status = ZwQueryKey(handle, KEY_INFORMATION_CLASS.KeyNameInformation, IntPtr.Zero, 0,
out needed);
                    if ((UInt32)status == 0xC0000023)   // STATUS_BUFFER_TOO_SMALL
                    {
                        pKNI = Marshal.AllocHGlobal(sizeof(UInt32) + needed + 4);
                        status = ZwQueryKey(handle, KEY_INFORMATION_CLASS.KeyNameInformation, pKNI, needed,
out needed);
                        if (status == 0)    // STATUS_SUCCESS
                        {
                            char[] bytes = new char[2 + needed + 2];
                            Marshal.Copy(pKNI, bytes, 0, needed);
                            result = new String(bytes, 2, (needed / 2) - 2);
                        }
                    }
                    Marshal.FreeHGlobal(pKNI);
                    return result;
                }
            }

            private static Manifest _manifest;
            string newSubKey = "";

            private static IntPtr SetRedirectedKey(IntPtr handle, string subkey, out string newSub)
            {
                try
                {
                    // Trim hKey, because it is always HKCU
                    string virtualRoot = _manifest.RegistryVirtualRoot.Replace("HKEY_CURRENT_USER\\", "");
                    string root = KeyHandleToString(handle);
                    IntPtr newHandle = IntPtr.Zero;
                    newSub = "";

                    // Replace machine code to "human-readable"
                    root = root.Replace(@"\REGISTRY\MACHINE", @"HKEY_LOCAL_MACHINE");
                    string SID = System.Security.Principal.WindowsIdentity.GetCurrent().User.ToString();
                    root = root.Replace($"\\REGISTRY\\USER\\{SID}\\", @"HKEY_CURRENT_USER\");
```

```
        root = root.Replace(@"\REGISTRY\USER\", @"HKEY_USERS\");

        // Parse Manifest
        foreach (Redirection redirection in _manifest.Redirections)
        {
            if (redirection.Type == RedirectionType.Registry)
            {
                // Check if subkeys fit the mask recursevly
                string sub = root + (string.IsNullOrEmpty(subkey) ? "" : $"\\{subkey}");
                while (!(string.IsNullOrEmpty(Path.GetDirectoryName(sub))))
                {
                    if (sub.Equals(redirection.From, StringComparison.OrdinalIgnoreCase))
                    {
                        newHandle = new IntPtr(unchecked((int)0x80000001));
                        newSub = virtualRoot + "\\" + root + (string.IsNullOrEmpty(subkey) ? "" : $"\\{subkey}");
                        return newHandle;
                    }
                    sub = Path.GetDirectoryName(sub);
                }
            }
        }
    }
    catch (Exception e) {   }
    //If not redirected
    newSub = subkey;
    return handle;
}


private IntPtr RegOpenKeyExHook(IntPtr hKey, string subKey, int ulOptions, int samDesired, IntPtr hkResult)
{
    IntPtr newHandle = SetRedirectedKey(hKey, subKey, out newSubKey);
    Queue.Enqueue(new RegistryKeyMonitorEvent(RegistryOperationType.RegOpenKeyEx,
        $"{KeyHandleToString(hKey)}\\{subKey}", $"{KeyHandleToString(newHandle)}\\{newSubKey}"));

    return Advapi32.RegOpenKeyExW(newHandle, newSubKey, ulOptions, samDesired, hkResult);
}

private IntPtr RegOpenKeyHook(IntPtr hKey, string subKey, IntPtr hkResult)
{
    IntPtr newHandle = SetRedirectedKey(hKey, subKey, out newSubKey);
```

```
    Queue.Enqueue(new RegistryKeyMonitorEvent(RegistryOperationType.RegOpenKey,
        $"{KeyHandleToString(hKey)}\\{subKey}", $"{KeyHandleToString(newHandle)}\\{newSubKey}"));

    return Advapi32.RegOpenKeyW(newHandle, newSubKey, hkResult);
}

private IntPtr RegCreateKeyHook(IntPtr hKey, string subKey, IntPtr hkResult)
{
    IntPtr newHandle = SetRedirectedKey(hKey, subKey, out newSubKey);
    Queue.Enqueue(new RegistryKeyMonitorEvent(RegistryOperationType.RegCreateKey,
        $"{KeyHandleToString(hKey)}\\{subKey}", $"{KeyHandleToString(newHandle)}\\{newSubKey}"));

    return Advapi32.RegCreateKeyW(newHandle, newSubKey, hkResult);
}

private IntPtr RegDeleteKeyExHook(IntPtr hKey, string subKey, int samDesired, int reserved)
{
    IntPtr handle = SetRedirectedKey(hKey, subKey, out newSubKey);
    Queue.Enqueue(new RegistryKeyMonitorEvent(RegistryOperationType.RegDeleteKeyEx,
        $"{KeyHandleToString(hKey)}\\{subKey}", $"{KeyHandleToString(handle)}\\{newSubKey}"));
    return Advapi32.RegDeleteKeyExW(handle, newSubKey, samDesired, reserved);
}

private IntPtr RegDeleteKeyHook(IntPtr hKey, string subKey)
{
    IntPtr handle = SetRedirectedKey(hKey, subKey, out newSubKey);
    Queue.Enqueue(new RegistryKeyMonitorEvent(RegistryOperationType.RegDeleteKey,
        $"{KeyHandleToString(hKey)}\\{subKey}", $"{KeyHandleToString(handle)}\\{newSubKey}"));
    return Advapi32.RegDeleteKeyW(handle, newSubKey);
}

private IntPtr RegDeleteTreeHook(IntPtr hKey, string subKey)
{
    IntPtr handle = SetRedirectedKey(hKey, subKey, out newSubKey);
    Queue.Enqueue(new RegistryKeyMonitorEvent(RegistryOperationType.RegDeleteTree,
        $"{KeyHandleToString(hKey)}\\{subKey}", $"{KeyHandleToString(handle)}\\{newSubKey}"));
    return Advapi32.RegDeleteTreeW(handle, newSubKey);
}

private IntPtr RegGetValueHook(IntPtr hKey, string subKey, string value, uint flags, out uint type,
    IntPtr vData, out uint pData)
```

```
        {
            IntPtr newHandle = SetRedirectedKey(hKey, subKey, out newSubKey);
            Queue.Enqueue(new RegistryKeyValueMonitorEvent(RegistryOperationType.RegGetValue,
                $"{KeyHandleToString(hKey)}\\{subKey}", value,
$"{KeyHandleToString(newHandle)}\\{newSubKey}"));

            return Advapi32.RegGetValueW(newHandle, newSubKey, value, flags, out type, vData, out pData);
        }

        private IntPtr RegQueryValueHook(IntPtr hKey, string subKey, IntPtr lpData, ref int lpcbData)
        {
            IntPtr newHandle = SetRedirectedKey(hKey, subKey, out newSubKey);
            Queue.Enqueue(new RegistryKeyMonitorEvent(RegistryOperationType.RegQueryValue,
                $"{KeyHandleToString(hKey)}\\{subKey}", $"{KeyHandleToString(newHandle)}\\{newSubKey}"));

            return Advapi32.RegQueryValueW(newHandle, newSubKey, lpData, ref lpcbData);
        }

        private IntPtr RegQueryValueExHook(IntPtr hKey, string valueName, int reserved, out uint type, IntPtr data,
            out uint cbData)
        {
            IntPtr newHandle = SetRedirectedKey(hKey, "", out newSubKey);
            Queue.Enqueue(new RegistryKeyValueMonitorEvent(RegistryOperationType.RegQueryValueEx,
                KeyHandleToString(hKey), valueName, $"{KeyHandleToString(newHandle)}\\{newSubKey}"));
            return Advapi32.RegQueryValueExW(newHandle, valueName, reserved, out type, data, out cbData);
        }

        private IntPtr RegSetKeyValueHook(IntPtr hKey, string subKey, string valueName, uint type, IntPtr data,
            uint cbData)
        {
            IntPtr newHandle = SetRedirectedKey(hKey, subKey, out newSubKey);
            Queue.Enqueue(new RegistryKeyValueMonitorEvent(RegistryOperationType.RegSetKeyValue,
                $"{KeyHandleToString(hKey)}\\{subKey}", valueName,
$"{KeyHandleToString(newHandle)}\\{newSubKey}"));
            return Advapi32.RegSetKeyValueW(newHandle, newSubKey, valueName, type, data, cbData);
        }

        private IntPtr RegSetValueHook(IntPtr hKey, string subKey, uint type, IntPtr data, uint cbData)
        {
            IntPtr newHandle = SetRedirectedKey(hKey, subKey, out newSubKey);
            Queue.Enqueue(new RegistryKeyMonitorEvent(RegistryOperationType.RegSetValue,
```

```
            $"{KeyHandleToString(hKey)}\\{subKey}", $"{KeyHandleToString(newHandle)}\\{newSubKey}"));

        return Advapi32.RegSetValueW(newHandle, newSubKey, type, data, cbData);
    }

    private IntPtr RegSetValueExHook(IntPtr hKey, string valueName, uint reserved, uint type, IntPtr data,
        uint cbData)
    {
        IntPtr newHandle = SetRedirectedKey(hKey, "", out newSubKey);
        Queue.Enqueue(new RegistryKeyValueMonitorEvent(RegistryOperationType.RegSetValueEx,
            $"{KeyHandleToString(hKey)}", valueName, $"{KeyHandleToString(newHandle)}\\{newSubKey}"));
        return Advapi32.RegSetValueExW(newHandle, valueName, reserved, type, data, cbData);
    }
    private IntPtr RegDeleteValueHook(IntPtr hKey, string valueName)
    {
        IntPtr newHandle = SetRedirectedKey(hKey, "", out newSubKey);
        Queue.Enqueue(new RegistryKeyValueMonitorEvent(RegistryOperationType.RegDeleteValue,
            $"{KeyHandleToString(hKey)}", valueName, $"{KeyHandleToString(newHandle)}\\{newSubKey}"));
        return Advapi32.RegDeleteValueW(newHandle, valueName);
    }
    private IntPtr RegDeleteKeyValueHook(IntPtr hKey, string subKey, string valueName)
    {
        IntPtr newHandle = SetRedirectedKey(hKey, "", out newSubKey);
        Queue.Enqueue(new RegistryKeyValueMonitorEvent(RegistryOperationType.RegDeleteKeyValue,
            $"{KeyHandleToString(hKey)}", valueName, $"{KeyHandleToString(newHandle)}\\{newSubKey}"));
        return Advapi32.RegDeleteKeyValueW(newHandle, newSubKey, valueName);
    }
  }
}
```

Файл SystemMonitorEntryPoint.cs

```
using System.Collections.Concurrent;
using System.Threading;
using Apptimized.Demo.SystemMonitor.Hook.Hooks;
using Apptimized.Demo.SystemMonitor.Shared;
using Apptimized.Demo.SystemMonitor.Shared.Data.Interfaces;
using EasyHook;

namespace Apptimized.Demo.SystemMonitor.Hook
{
```

```
public class SystemMonitorEntryPoint : IEntryPoint
{
    private readonly SystemMonitorInterface _monitorInterface;
    private readonly ConcurrentQueue<ISystemMonitorEvent> _queue;
    private static string _channelName;
    public static void SetChannelName(string channel) { _channelName = channel;
    public static string GetChannelName() { return _channelName; }

    public SystemMonitorEntryPoint(RemoteHooking.IContext context, string channelName)
    {
        _monitorInterface = RemoteHooking.IpcConnectClient<SystemMonitorInterface>(channelName);
        _monitorInterface.Ping();
        _channelName = channelName;
        _queue = new ConcurrentQueue<ISystemMonitorEvent>();
    }
    public void Run(RemoteHooking.IContext context, string channelName)
    {
        _monitorInterface.ReportInjection(RemoteHooking.GetCurrentProcessId());
        try
        {
            using (new FileSystemHook(_queue))
            using (new RegistryHook(_queue))
            using (new ProcessHook(_queue))
            {
                _monitorInterface.ReportHooksInstallation();
                RemoteHooking.WakeUpProcess();
                while (true)
                {
                    _monitorInterface.Ping();

                    while (_queue.TryDequeue(out var monitorEvent))
                        _monitorInterface.ReportSystemEvent(monitorEvent);

                    Thread.Sleep(500);
                }
            }
        }
        catch
        {
            // ignored
        }
```

```
            finally
            {
                LocalHook.Release();
            }
        }
    }
}
```

Файл InjectOptions.cs

```
using CommandLine;
namespace Apptimized.Demo.SystemMonitor.Options
{
    [Verb("inject", HelpText = "Inject into the existing process")]
    public class InjectOptions
    {
        [Option('p', "process-id", Required = true)]
        public int ProcessId { get; set; }
    }
}
```

Файл RunOptions.cs

```
using CommandLine;
namespace Apptimized.Demo.SystemMonitor.Options
{
    [Verb("run", HelpText = "Start process from path and inject into it")]
    public class RunOptions
    {
        [Option('p', "path", Required = true)]
        public string Path { get; set; }
        [Option('a', "arguments", Required = false, Default = "")]
        public string Arguments { get; set; } = string.Empty;
    }
}
```

Файл Program.cs

```
using System;
```

```
using System.Diagnostics;
using Apptimized.Demo.SystemMonitor.Hook;
using Apptimized.Demo.SystemMonitor.Options;
using Apptimized.Demo.SystemMonitor.Shared;
using CommandLine;
using EasyHook;
using Serilog;
namespace Apptimized.Demo.SystemMonitor
{
   internal static class Program
   {
      private static string _channelName;
      private static string _hookPath;

      public static int Main(string[] args)
      {
         _hookPath = typeof(SystemMonitorEntryPoint).Assembly.Location;
         RemoteHooking.IpcCreateServer<SystemMonitorInterface>(
            ref _channelName, System.Runtime.Remoting.WellKnownObjectMode.Singleton);
         var logger = SystemMonitorLogger.Instance;
         SystemMonitorEntryPoint.SetChannelName(_channelName);
         var processId = Parser.Default.ParseArguments<InjectOptions, RunOptions>(args)
            .MapResult(
               (InjectOptions o) => InjectIntoExisting(logger, o),
               (RunOptions o) => CreateAndInject(logger, o),
               errors => 0);
         if (processId == 0)
            return 1;

         logger.Information(
            "System Monitor is running. It will exit together after the child process.");
         Process
            .GetProcessById(processId)
            .WaitForExit();
         return 0;
      }

      private static int InjectIntoExisting(ILogger logger, InjectOptions options)
      {
         try
         {
```

```
            RemoteHooking.Inject(
                options.ProcessId,
                InjectionOptions.DoNotRequireStrongName,
                _hookPath,
                _hookPath,
                _channelName);
            return options.ProcessId;
        }
        catch (Exception exception)
        {
            logger.Error(exception,
                "Critical error occured while injecting to PID {pid}", options.ProcessId);
            return 0;
        }
    }

    private static int CreateAndInject(ILogger logger, RunOptions options)
    {
        try
        {
            RemoteHooking.CreateAndInject(
                options.Path,
                options.Arguments,
                0,
                InjectionOptions.DoNotRequireStrongName,
                _hookPath,
                _hookPath,
                out var processId,
                _channelName);
            return processId;
        }
        catch (Exception exception)
        {
            logger.Error(exception,
                "Critical error occured while injecting to {path} with arguments {arguments}",
                options.Path, options.Arguments);
            return 0;
        }
    }
  }
}
```