НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп'ютерних систем

До захисту допущено

Завідувач кафедри
_____ Віталій РОМАНКЕВИЧ
"___" червня 2021 р.

Дипломний проект

на здобуття ступеня бакалавра

по спеціальності **123 «Комп'ютерна інженерія»**

на тему: Засоби обробки аудіоданих драйверами APO ОС Windows

Виконав:

студент IV курсу, групи КВ-73

Браславець Олег Романович _____
_____
(підпис)

Керівник: Коляда Костянтин Вячеславович, _____
старший викладач кафедри СПіСКС _____
_____
(підпис)

Консультант з нормоконтролю, доц.каф.СПСКС, к.т.н. Клятченко Я.М.

_____ (підпис)

Рецензент_____
(підпис)

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2021 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп‘ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Спеціальність 123 «Комп’ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____Віталій РОМАНКЕВИЧ
(підпис) (ініціали, прізвище)
«___» червня 2021 р.

ЗАВДАННЯ
на дипломний проєкт студента
Браславець Олег Романович
(прізвище, ім’я, по батькові)

1. Тема проєкту Засоби обробки аудіоданих драйверами APO ОС Windows, керівник проєкту старший викладач кафедри СПСКС, к.т.н. Коляда К.В. ,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «25» травня 2021р. № 1331-С

2. Термін подання студентом проєкту

3. Вихідні дані до проєкту : науково-технічна література по технології APO, приклади APO у відкритому доступі, додаткова науково-технічна література по технологіям, які доцільно використовувати при написанні APO.

4. Зміст пояснювальної записки:
1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ОБҐРУНТУВАННЯ ТЕМИ ДИПЛОМНОГО ПРОЕКТУ
1.1Обгрунтування та вступ
1.2KMD
1.3 Програмне забезпечення звукових карт (Firmware)
1.4APO
1.5Порівняння і висновки
2. APO ЯК ПРОГРАМНИЙ МОДУЛЬ
3. ОСНОВНІ ЄТАПИ НАПИСАННЯ APO

3.1Приклади відкритих простих АРО рішень
3.2Основні методи АРО
4. НАЛАГОДЖЕННЯ АРО У СИСТЕМІ
4.1Встановлення АРО у систему
4.2Способи зв’язку з АРО
4.3Особливості налагодження програмного коду АРО
4.4Особливості логування програмного коду АРО
4.5Програмний продукт АРО

5. Перелік графічного матеріалу (із зазначенням обов’язкових креслеників, плакатів, презентацій тощо) :
- Взаємодія АРО.
- Алгоритм обробки окремих каналів.
- Проходження аудіобуферу.
- Виклики функцій АРО звуковою системою.

6. Консультанти розділів проєкту*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Клятченко Я.М., доц. каф. СПіСКС, к.т.н.		

7. Дата видачі завдання 20 жовтня 2021р.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів	Примітка
1.	Видача завдання на дипломне проєктування	30.10.2020	
2.	Вивчення літератури за тематикою проєкту	17.11.2020	
3.	Розроблення та узгодження технічного завдання	13.01.2021	
4.	Аналіз існуючих рішень	01.01.2021	
5.	Розроюка програмного модулю АРО	03.03.2021	
6.	Тестування швидкісті обробки даних	24.03.2021	
7.	Підготовка матеріалів графічної частини проєкту	09.04.2021	
8.	Оформлення документації дипломного проєкту	13.05.2021	

Студент _____ Браславець О.Р.
(підпис) (ініціали, прізвище)

* Консультантом не може бути зазначено керівника дипломного проекту.

Керівник проєкту _____ Коляда К.В.
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Об’єкт розробки – дослідження обробки аудіо даних через драйвери на базі APO у системах Windows.

Використання APO у системах Windows дозволяє легко створити та підтримувати проект обробки аудіо даних. APO дозволяє оброблювати окремі потоки аудіо сигналів, такі як: потоки надіслані однією програмою, групою програм, які використовують певний режим обробки звуку, а також зміни аудіо даних для всього звуку, який розташований на виході пристрою.

Були проаналізовані такі етапи розробки:

- написання основного каркасу;
- встановлення APO;
- способи зв'язку з APO;
- налагодження коду;
- логування коду;

В ході дослідження:

- виявлено недоліки а також переваги APO на фоні існуючих альтернативних рішень;
- визначено з чого складається і що собою являє APO;
- знайдено приклади APO, які знаходяться у відкритому доступі;
- визначено особливості основних методів APO;
- порівняно способи встановлення APO у ОС Windows;
- порівняно найрозповсюдженіші способи зв’язку між програмами, які доцільно використовувати при написані APO;
- визначено особливості налагодження програмного коду APO;
- визначено особливості логування програмного коду APO;

Ключові слова:

APO, процесінг, аудіо потік, обробка звуку, аудіо дані, драйвер, налагодження, логування, KMDF, INF, Windows.

SUMMARY

The object of development – study the processing of audio data through APO-based drivers on Windows systems.

Using APO on Windows systems makes it easy to create and maintain an audio data processing project. ARO allows you to process individual streams of audio signals, such as streams sent by a single program, a group of programs that use a specific audio processing mode, and changes the audio data for all the sound that is located at the output of the device.

The following stages of development were analyzed:

- writing the main framework;
- installation of ARO;
- methods of communication with the ARO;
- code debugging;
- code logging;

During the study:

- shortcomings and advantages of ARO against the background of existing alternative solutions are revealed;
- it is determined what the ARO consists of and what it is;
- examples of ARO that are in the public domain are found;
- features of the main methods of ARO are determined;
- compared ways to install APO in Windows;
- relatively the most common ways of communication between programs, which should be used when writing an APO;
- the peculiarities of ARO program code debugging are determined;
- the peculiarities of logging the APO program code are determined;

Keywords:

ARO, processing, audio stream, audio processing, audio data, driver, debugging, logging, KMDF, INF, Windows.

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045480.002 ТЗ	Засоби обробки	3		
			аудіоданих драйверами			
			APO ОС Windows			
			Технічне завдання			
	А4	ІАЛЦ.045480.003 ТП	Засоби обробки	2		
			аудіоданих драйверами			
			APO ОС Windows			
			Відомість технічного			
			проєкту			
	А4	ІАЛЦ.045480.004 ПЗ	Засоби обробки	54		
			аудіоданих драйверами			
			APO ОС Windows			
			Пояснювальна записка			

Змін.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.045480.001 ОА		
Розробив		Браславець О.Р.			Засоби обробки аудіоданих драйверами APO ОС Windows **Опис альбому**	Літ.	Аркуш 1 / Аркушів 3
Перевірив		Коляда К.В.					
Консульт.						КПІ ім. Ігоря Сікорського, ФПМ КВ-73	
Н. контроль		Клятченко Я.М.					
Зав. каф.		РомankевичВ.О.					

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045480.005 Д1	Засоби обробки	1		
			аудіоданих драйверами			
			APO ОС Windows			
			Взаємодія APO			
			Схема структурна			
	А4	ІАЛЦ.045480.006 Д2	Засоби обробки	1		
			аудіоданих драйверами			
			APO ОС Windows			
			Алгоритм обробки			
			окремих каналів			
			Схема алгоритму			
	А4	ІАЛЦ.045480.007 Д3	Засоби обробки	1		
			аудіоданих драйверами			
			APO ОС Windows			
			Проходження аудіобуферу			
			Схема структурна			

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045480.008 Д4	Засоби обробки	1		
			аудіоданих драйверами			
			APO ОС Windows			
			Виклики функцій			
			Схема структурна			
		Диск CD-ROM	Текст пояснювальної	1		
			записки			
			Графічний матеріал			

					ІАЛЦ.045480.001 ОА	Арк.
						3
Змін.	Арк.	№ докум.	Підпис	Дата		

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ РОЗРОБКИ

Назва розробки: «Засоби обробки аудіоданих драйверами APO ОС Windows».

Галузь застосування: обробка аудіоданих у ОС Windows.

2. ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на дипломне проєктування на здобуття першого (бакалаврського) рівня вищої освіти, затверджене кафедрою системного програмування і спеціалізованих комп'ютерних систем Національного технічного університету України «Київський Політехнічний Інститут імені Ігоря Сікорського».

3. МЕТА І ПРИЗНАЧЕННЯ РОБОТИ

Метою даного проекту є дослідження існуючих рішень обробки аудіоданих, визначення їх негативних та позитивних сторін, а також розробка власного APO.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом інформації є технічна та науково-технічна література, технічна документація, публікації у періодичних виданнях та електронні статті у мережі Інтернет.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до програмного продукту, що розробляється

- підтримка ОС Windows 10;
- підтримка обробки звуку всього вихідного сигналу;
- модифікація аудіопотоку;
- модифікація аудіопотоку окремо для кожного каналу;

- надання простих алгоритмів модифікації звуку.

5.2. Вимоги до апаратного забезпечення

- оперативна пам’ять: 2 Гб.

5.3. Вимоги до програмного забезпечення користувача

- ОС Windows 10.

6. ЕТАПИ РОЗРОБКИ

№ з/п	Назва етапів виконання дипломного прокту	Термін виконання етапів
1.	Видача завдання на дипломне проєктування	30.10.2020
2.	Вивчення літератури за тематикою проєкту	17.11.2020
3.	Розроблення та узгодження технічного завдання	13.01.2021
4.	Аналіз існуючих рішень	01.01.2021
5.	Розроюка програмного модулю АРО	03.03.2021
6.	Тестування швидкісті обробки даних	24.03.2021
7.	Підготовка матеріалів графічної частини проєкту	09.04.2021
8.	Оформлення документації дипломного проєкту	13.05.2021

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045480.004 ПЗ	Засоби обробки	54		
			аудіоданих драйверами			
			APO ОС Windows			
			Пояснювальна записка			
	А4	ІАЛЦ.045480.005 Д1	Засоби обробки	1		
			аудіоданих драйверами			
			APO ОС Windows			
			Взаємодія APO			
			Схема структурна			
	А4	ІАЛЦ.045480.006 Д2	Засоби обробки	1		
			аудіоданих драйверами			
			APO ОС Windows			
			Алгоритм обробки			
			окремих каналів			
			Схема алгоритму			

Змін.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.045480.003 ТП		
Розробив		Браславець О.Р.			Засоби обробки аудіоданих драйверами APO ОС Windows **Відомість технічного проекту**	Літ.	Аркуш 1 / Аркушів 2
Перевірив		Коляда К.В.					
Консульт.						КПІ ім. Ігоря Сікорського, ФПМ КВ-73	
Н. контроль		Клятченко Я.М.					
Зав. каф.		РоманкевичВ.О.					

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045480.007 Д3	Засоби обробки	1		
			аудіоданих драйверами			
			APO ОС Windows			
			Проходження аудіобуферу			
			Схема структурна			
	А4	ІАЛЦ.045480.008 Д4	Засоби обробки	1		
			аудіоданих драйверами			
			APO ОС Windows			
			Виклики функцій			
			Схема структурна			
		Диск CD-ROM	Текст пояснювальної	1		
			записки			
			Графічний матеріал			

Змін.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.045480.003 ТП	Арк. 2

Пояснювальна записка
до дипломного проекту

на тему: Засоби обробки аудіоданих драйверами APO ОС Windows

Київ – 2021 року

ЗМІСТ

ДОДАТКИ

Додаток 1. Копії графічних матеріалів

- Взаємодія APO.
- Алгоритм обробки окремих каналів.
- Проходження аудіобуферу.
- Виклики функцій APO звуковою системою.

Додаток 2. Фрагменти програмного коду

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

APO – Audio Processing Object.

FL – Front left.

FR – Front right.

KMDF – Kernel-Mode Driver Framework.

INF – Setup Information file.

KMD – Kernel-Mode Driver.

IDE – Integrated development environment.

COM – Component Object Model.

DSP – Digital signal processing.

WinAPI – Windows Application Programming Interface.

USB – Universal Serial Bus.

ОС – Операційна система.

DLL – Dynamic Link Library.

SFX – Stream Effects Filter.

MFX – Mode Effects Filter.

EFX – Endpoint Effects Filter.

LFX – Local Effects Filter.

GFX – Global Effects Filter.

MS-PL – Microsoft Public License.

GPL – General Public License.

WMI – Windows Management Instrumentation.

API – Application programming interface.

- ВСТУП

APO – це метод реалізації обробки звуку у системах Windows. Існує певна кількість способів за допомогою яких у системах Windows можливо редагувати звук у режимі реального часу з мінімальною затримкою при обробці.

Спосіб процесінгу звуку через рішення APO надає можливість редагувати звук конкретних екземплярів аудіопотоків одночасно, як окремо так і усіх разом. Існує можливість зміни аудіопотоку, який надісланий однією програмою (одна програма може надсилати більше одного потоку звукових даних), групою програм, які використовують певний режим обробки звуку (існують режими обробки звуку, які вказують на характер аудіоданих, які слід обробляти), а також зміни аудіоданих для всього звуку, який розташований на виході одного пристрою.

Основною особливістю APO є легкість розробки. APO знаходиться у режимі виконання програмного коду призначеному для користувача – режим User Mode. Ця особливість, розташування програмного коду, дає змогу займатись налагодженням програмного коду напряму підключившись до нього. Враховуючи, що APO це драйвер, то можливість такого способу налагодження коду значно краща аніж використання тільки логування коду у випадку налагодження коду драйверів KMD.

Програмний код APO не має доступу до ресурсів ядра, не можливо нашкодити системі і привести її у несправний стан, що робить розробку і налагодження коду більш безпечним, порівняно з альтернативними рішеннями обробки звуку у реальному часі.

Таке рішення є простим у написанні та у легкості підтримки одразу великої кількості пристроїв, що значно краще аніж розроблювати окреме програмне забезпечення для кожного необхідного пристрою, який слід підтримувати.

Слід звернути увагу на реєстрацію APO у системі та на пристрої. Також APO має свої особливості налагодження програмного коду і логування програмного коду, які слід розглянути.

АРО має свої недоліки, їх необхідно визначити, і зрозуміти при яких обставинах слід використовувати інші рішення обробки аудіоданих у реальному часі.

1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ОБҐРУНТУВАННЯ ТЕМИ ДИПЛОМНОГО ПРОЕКТУ

1.1 Обґрунтування та вступ

Тематика обробки аудіоданих в першу чергу цікавить гейм-індустрію, а також медіа індустрію. Цими двома напрямами, використання обробки звуку, не обмежується. Обробка аудіоданих необхідна не лише для поліпшення звуку музики, але і для більш важливіших речей. В науці часто використовують алгоритми обробку звуку, найрозповсюдженіший серед них це алгоритм зменшення шуму. При дослідженні будь-якої місцевості, або чого завгодно нового, науковцям важливо міти відокремити аудіосигнал шуму, від аудіосигналу, напряму пов’язаного з об’єктом дослідження. Обробка аудіоданих це не лише обробка звуку який надсилається пристроєм (вихідний звук), а це також зміна та аналіз звуку який надходить у пристрої за допомогою мікрофонів (вхідний звук).

В першу чергу мета обробки звуку отриманого з мікрофону – поліпшення чіткості голосу. Для таких цілей розроблюються і використовуються алгоритми напрямку Noise Reduction – зменшення шуму. Ці алгоритми використовуються майже у будь-якому сучасному пристрої, який призначений для запису звуку (будь-який телефон та інші пристрої). За допомогою подібних алгоритмів можливо досягти чіткого та приємного голосу і відділити необхідну звукову інформацію від шуму, який виникає поруч. Навіть коли використовуються частота дискретизації зв’язку менше ніж або 16000Гц при розрядності 16 біт і менше, алгоритми Noise Reduction можуть виконати свою задачу. Такі низькі показники якості звуку використовуються коли потрібно зекономити трафік (або швидкість передачі даних занадто низька). При зв’язку на великі відстані можуть використовуватись ще гірші показники. Також алгоритми Noise Reduction для обробки сигналу отриманого з мікрофону використовуються при розпізнаванні голосу. Це складна технологія, яка потребує чіткого розбиття аудіоданих різної частоти на набори сигналів, які відповідають певним звукам.

Найважливішими алгоритмами обробки аудіоданих, які надсилає пристрій (тобто вихідний звуковий сигнал) є алгоритми 3D звуку, або алгоритми об'ємного звучання. Їх основною задачею є імітація об'ємного звуку, який призводить до відчуття реальної присутності серед об'єктів від яких надходить звук. Основними представниками використання таких алгоритмів є ігри. Але також ці алгоритми використовуються у науковій діяльності. Складність таких алгоритмів полягає в тому що потрібно відтворити звуки, які можуть надходити до вух слухача з будь-якої точки навколо нього, і цих звукових сигналів різних джерел може бути необмежена кількість, але при цьому можливо використати лише два реальні динаміки, які відтворюють звук (майже всі сучасні відтворювачі звукового сигналу – стерео, тобто мають два канали FL і FR). Також, при написанні алгоритмів об'ємного звучання, слід враховувати фізіологічні особливості людини: вуха, плечі, голова. Очевидно, що ці частини тіла є перешкодами для досягнення звуковими потоками вушних перетинок. Тому такі алгоритми враховують фізіологічні особливості тіла людини, щоб відтворювати звук якомога реалістичніше.

Не залежно від того, які алгоритми необхідно використати для обробки звуку є три основні питання, які слід визначити при виборі системи обробки аудіоданих:

1) Час обробки (складність алгоритму)

2) Можливість у реальному часі змінювати налаштування алгоритмів обробки

3) До чого слід застосовувати ці алгоритми (націлені пристрої)

Складність алгоритмів напряму впливає на час обробки звукових даних цими алгоритмами. Наприклад:

- Зміна звуку вимагає множення кожного елементу звуку на певний коефіцієнт. Отже час операцій обробки таким алгоритмом аудіоданих дорівнює кількості семплів помножених на час за котрий процесор виконає множення:

$T(n) = n * t$

Де T – загальний час обробки n семплів

t – час виконання операції множення.

- Використання алгоритмів об’ємного звуку потребують вирахування співвідношення гучності, яку буде передано на лівий канал (FL) до гучності, яку буде передано на правий канал (FR) в залежності від розташування віртуального джерела звуку (примітивне описання алгоритму об’ємного звучання). Кількість операцій при цьому, які використовуються для обробки одного семплу звуку значно більше а ніж одна дія множення.

Наслідками довгої обробки звуку є рипіння або зменшення швидкості відтворювання звукового сигналу пристроєм. При затримках обробки сигналу у систем, які контролюють обмеження часу обробки – система буде заповнювати необроблені дані тим сміттям, який вже був у буфері даних (тобто якщо не встигнути надіслати 480 семплів – всі семпли міститимуть сміття і у пристрій відтворювання звукового сигналу буде надіслано це сміття). У системах, які очікують прибуття обробленого буферу і не контролюють час обробки буде зменшена швидкість звуку настільки, наскільки повільно він буде оброблюватись. Тоді звук стане повільним, і не буде відповідати очікуваному результату. Обидва варіанти, повільної обробки сигналу, є незадовільними для будь-якої системи обробки звуку, так як відтворений звуковий сигнал не є тим сигналом який записаний.

Отже, система, яка буде оброблювати сигнал, повинна сама по собі досить швидко працювати, щоб мати можливість оброблювати складні алгоритми з великою кількістю операцій на один звуковий семпл у реальному часі. Кількість операцій котрі будуть використані для отримання буферу, його пересилання та перенаправлення, мікшування, та велика кількість інших операцій звукової системи повинні виконуватись досить швидко.

Час обробки звуку це сума необхідного часу для пристосування алгоритму до звукового буферу, який використовується, і час необхідний для роботи звукової системи, яка буде використана.

Наступне питання це можливість у реальному часі змінювати параметри обробки звукового сигналу. Якщо система повинна весь час працювати з одними і тими самими значеннями то доцільно використовувати для

цього вбудовані способи модифікації звуку (наприклад більша частина програмного забезпечення звукових карт). Але у випадку, коли необхідно змінювати параметри обробки звукового сигналу у реальному часі слід обирати систему, яка гнучка до спілкування з зовнішнім середовищем (іншими програмами, наприклад програмами користувацького інтерфейсу) або має легко підтримуваний спосіб передачі і отримання даних між програмами. Навіть, коли система вже має існуючий протокол зв'язку з зовнішніми програмами, слід враховувати складність їх модернізації, адже в майбутньому, може з'явитись необхідність використання нових параметрів для звукової обробки.

Останнє на що слід звернути увагу, при виборі системи обробки аудіоданих – до чого (яких пристроїв) слід використовувати алгоритми на які націлений проект. В першу чергу слід поділити пристрої на девайси виводу звуку і вводу. Адже мікрофон може отримувати один лише потік даних, а на пристрій виводу може бути надіслано величезна кількість потоків, які хтось повинен окремо (чи усі одразу з одним принципом обробки) обробити і об'єднати. Але якщо проекту необхідно обробляти лише остаточний потік звуку, який надходить до пристрою виводу, то таку задачу можливо реалізувати через будь-яку систему обробки звуку.

1.2 KMD

KMDF – Kernel-Mode Driver Framework. Це середовище драйверів, розроблена Microsoft в якості інструменту, що допомагає розробникам драйверів створювати і підтримувати драйвери пристроїв режиму ядра (KMD) [1].

KMD – драйвери режиму ядра, дуже розповсюджені і широко використовуються у системах Windows. Драйвери рівня ядра можуть бути написані для таких цілей:

- Створення віртуального пристрою (часто використовуються для пристроїв, які фізично не підключені до машини, але інколи створюються як додатковий компонент більш складної системи, наприклад віртуальні аудіокабелі)

- Підміна реальних параметрів існуючого пристрою (зміна кількості каналів звукового пристрою, видалення або створення додаткової клавіши клавіатури та т.п.)
- Перехоплення запуску програм (часто використовують антивіруси, але в той самий час подібний функціонал можуть використовувати віруси, від яких потім досить складно позбутися)
- Обробка подій пристрою (не стандартний сценарій натискання певної клавіши клавіатури, перехоплення та зміна аудіосигналу та т.п.)
- Інше

Драйвери режиму ядра досить широко функціональні, тому складно назвати всі варіанти їх застосування. Це пояснюється їх правами доступу. Вони в прямому сенсі знаходяться в центрі всіх подій операційної системи і можуть контролювати ці події. Тобто за допомогою KMD драйверу можна обробляти та контролювати будь що у системі (слід бути дуже обережним, щоб не привести систему у несправний стан). Але саме це і створює складність написання такого драйверу, виникає великий ризик непередбачуваних наслідків.

Так як існує можливість написання драйвера рівня ядра для обробки звуку (обробка подій пристрою), то рішення KMD є альтернативним варіантом написання драйверу APO. Написання самого по собі KMD складна задача. Для недосвідченого програміста це досить ризиковий проект. KMD надає доступ до всіх компонентів комп'ютера, при бажанні, можна написати драйвер, який матиме можливість, фізично нашкодити машині, наприклад перегріти процесор, та привести його у стан несправності, адже процесор не може довгий час працювати при високій температурі, так як має свої фізичні обмеження. Але це малоймовірний випадок. Вірогідніше за все – ви пошкодите стан вашої операційної системи і навіть призведете її у несправний стан, доведеться перевстановлювати ОС із втратою усіх ваших даних.

Не дуже страшна помилка, для програм рівня користувача, наприклад вихід за межі виділеної пам’яті (помилка погана, але у режимі користувача оброблюється операційною системою), у драйвері режиму ядра може сильно нашкодити операційній системі, навіть зробити її несправною, так як за межами виділеної пам’яті можуть знаходитись необхідні системі дані, без яких вона не може виконувати певний функціонал. Одним з найчастіших випадків при розробці драйверу рівня ядра є BSOD [2].

Рисунок 1 – Попередження помилок драйверів ядра

В більшості випадків, виникнення BSOD, вистачить перезавантажити машину, щоб уникнути усіх наслідків несправної роботи вашого KMD драйверу. Але може статись так, що перезавантажитись система просто не зможе, наприклад якщо будуть пошкоджені дані, які відповідають за завантаження операційної системи. Тому весь процес розробки KMD, налагодження його програмного коду, а також його тестування – слід виконувати на віртуальній машині. Сервіси, які створюють віртуальні машини, обмежують об’єм даних на які можуть вплинути програми встановлені на віртуальну машину. Окрім самих даних основної операційної системи, до яких операційна система встановлена на віртуальну машину не може мати доступу, також обмежується доступ до фізичних елементів основної машини, таких як процесор, оперативна пам’ять та інші фізичні елементи. Тобто в більшості випадків, використання віртуальної машина захистить як данні основної операційної системи, так і доступ до фізичних елементів комп’ютера.

Розробка, перевірка та налагодження драйверів режиму ядра – обов’язково повинні відбуватись на віртуальних машинах. Виникнення BSOD на віртуальній

машині, у найгіршому випадку призведе до того, що доведеться перевстановлювати віртуальну операційну систему, що є набагато ліпшим розвитком подій, аніж необхідність заміни процесору.

Налагодження роботи KMD одне з найскладніших питань при розробці драйверів режиму ядра, воно істотно складніше, ніж налагодження звичайних програм користувацького режиму. Якщо для налагодження коду рівня користувача існує можливість використовувати стандартний дебагер, наприклад Visual Studio або дебагери інших середовищ IDE, то для налагодження коду режиму ядра потрібні спецзасоби [3]. Найчастіше доводиться використовувати програми, які лише виконують вивід спецповідомлень програмного коду – логування.

Ще одна складність, яка виникає при використанні драйверів KMD це його встановлення на операційну систему. Вирішення обмежити встановлення драйверів режиму ядра, досить розумне, враховуючи, що драйвери KMD можуть повністю контролювати операційну систему. Встановити будь-який драйвер на свою машину, не зважаючи на можливі складнощі і ризики – не складна задача. Для цього, досить відключити перевірку підписів драйверів і система почне використовувати драйвер, який ви встановили, так само, як вона б використовувала підписаний драйвер. Інше питання, як зробити так щоб будь-яка операційна система Windows мала довіру вашому KMD і використовувала його без обмежень. Операційні системи Windows 10 не завантажують нові драйвери режиму ядра, що не підписані в Windows Hardware Developer Center. Такі міри, були введені з виходом версії Windows 10 1607. Корпорація стверджує, що зміни потрібні для того, щоб зробити Windows безпечнішою операційною системою. На думку Microsoft, при введенні режиму завантаження тільки підписаних драйверів ядра ризик компрометації системи шкідливим програмним забезпеченням значно знижується [4]. Тому сертифіковане використання драйверу режиму ядра, на будь-якій машині, вимагає додаткових дій та часу для того, щоб драйвер розпізнавався як перевірений.

Розробка KMD вимагає досить великої уважності, а також не малого досвіду написання програмного коду. Окрім складності написання і налагодження роботи самого драйверу, існує складна задача налагодження роботи алгоритмів, які будуть використанні. Враховуючи, що помилки, які виникають при роботі KMD, в більшості випадків призводять до BSOD, правильність алгоритмів обробки звуку доведеться спочатку перевірити на додатковій програмі, з наближеними обставинами роботи, які можуть виникнути при роботі драйверу. Проект KMD це складна задача, яка потребує не лише сильних спеціалістів, але й велику кількість часу.

1.3 Програмне забезпечення звукових карт (Firmware)

Програмне забезпечення звукових карт – це програмний код який використовується в пристроях, які перетворюють цифровий аудіосигнал в аналоговий [5]. Окрім основної своєї задачі, звукові карти можуть модифікувати сигнал, який до них надходить, використовувати певні алгоритми обробки звукових сигналів. Майже усі звукові карти модифікують звуковий сигнал, що отримують. Це пояснюється тим, що кожний відтворювач звуку має свої фізичні властивості, тому певні частоти, відтворюванні одним пристроєм будуть звучати не так як їх відтворить інший пристрій. Виробники звукових карт намагаються наблизити звучання своїх пристроїв до більш реалістичного звучання (тобто у результаті більшість відтворюючих пристроїв звучать досить схоже), тому використовують для цього алгоритми еквалайзеру у своїх звукових картах(ці алгоритми збільшують / зменшують гучність певних частот звукового потоку). Складність обробки звуку через звукову карту можливо підвищити, і додати додаткові алгоритми.

З звуковими картами можливо покрити майже всі необхідні задачі обробки звуки, які можуть виникнути:

- Обробка звуку у реальному часі
- Використання будь-яких алгоритмів обробки
- Зміна параметрів обробки у реальному часі

- Обробка як вихідного сигналу так і вхідного
- Використання обробки незалежно від операційної системи

Також таке програмне забезпечення має свої плюси порівняно з альтернативними рішеннями:

- Неможливо нашкодити операційній системі (якщо звукова карта зовнішня)
- Не потрібно створювати сертифікати та підписувати програмне забезпечення

Але звукові карти мають і свої суттєві мінуси, через які таке рішення не використовується у всіх проектах.

По-перше – зміна параметрів обробки у реальному часі. Для того щоб пристрій із програмним забезпеченням обробки звуку мав можливість змінювати параметри обробки, він повинен мати певний протокол зв’язку, який також потрібно підтримувати (але тоді виникає залежність від операційної системи).

По-друге – при некоректній прошивці пристрою, його можна знищити, він перестане працювати. Часто при розробці прошивки на новий девайс, пристрої знищують партіями, при встановленні некоректного програмного забезпечення. Більш сучасні девайси виготовляються з розрахунком на таку ситуацію і мають захист від таких обставин, але так відбувається не з усіма пристроями, в більшості випадків лише пристрою преміум класу підтримують стійкість до несправних прошивок.

І по-третє, що є найголовнішим підтримка лише обмеженої кількості пристроїв. Одна і там сама прошивка може бути використана лише для пристроїв з ідентичними характеристиками. Тому розробка програмного забезпечення звукових карт пристроїв має досить обмежене використання.

Останній пункт є найважливішим при виборі способу обробки звукового сигналу. Такий тип програмного забезпечення частіше за все є досить примітивним – копіюється існуюча прошивка досить схожого пристрою та

модифікується в залежності від фізичних властивостей. І лише у тих випадках, коли мова іде про преміальний сегмент, і пристрій буде випущено великим тиражом, для нього може бути написане програмне забезпечення з великою кількістю алгоритмі обробки звуку, а також з програмним забезпечення, яке надає можливість змінювати параметри алгоритмів у реальному часі, і таке програмне забезпечення доводиться писати для кожної операційної системи окремо.

1.4 APO

Audio Processing Object (APO) – це хост-об'єкт COM, який містить алгоритми, що забезпечують особливий ефект цифрової обробки сигналів (DSP). Об'єкти обробки звуку (APO) забезпечують програмну обробку цифрових сигналів для аудіопотоків у системах Windows [6].

Насправді, кожен звуковий пристрій, котрий підключений до операційної системи Windows використовує об’єкт APO для обробки звуку. Якщо в базі драйверів операційної система не розташований драйвер APO з підтримкою саме цього пристрою, і з саме такими характеристиками підтримки як у націленого пристрою, то Windows буде використовувати для нього свій стандартний APO як і для всіх інших пристроїв для яких не буде знайдено додатково зареєстрованого APO.

Використання та розробка APO має величезну кількість позитивних сторін:

- Швидка обробка у реальному часі
- Використання будь-яких алгоритмів обробки
- Зміна параметрів обробки у реальному часі
- Обробка як вихідного сигналу так і вхідного
- Неможливо нашкодити операційній системі, адже код APO працює у режимі користувача

- Підтримка одним екземпляром APO одразу великої кількості пристроїв (можливо створити підтримку одразу всіх існуючих пристроїв з певними характеристиками)
- Обробка окремих потоків звуку роздільно (потоки надіслані однією програмою, групою програм, які використовують певний режим обробки звуку, а також зміни аудіоданих для всього звуку, який розташований на виході пристрою)

Але рішення APO для обробки звукового сигналу не є ідеальним рішенням (тому і не використовується усіма і усюди) і має свої суттєві недоліки:

- Залежність від операційної системи
- Необхідність підпису сертифікатів

Необіхдність підпису сертіфікатів не є великою проблемою, але вносить деякі складнощі при розробці. Будь-яка програма, яку планується використовувати на будь-якій машині з операційною системою Windows, повинна бути перевіреною і сертифікованою. Така необхідність існує задля того, щоб забезпечити захищеність операційної системи. Але як і драйвери KMD, під час розробки APO не потрібно при будь-якій зміні коду перепідписувати драйвер. На своїй машині, яка є машиною розробника, можна використовувати і тестувати роботоспроможність свого продукту двома способами:

- Відключивши перевірку підписів (робить це через можливість запуску операційної системи з певними прапорцями)
- Зробивши девелоперський підпис – самостійний підпис драйверу (його доведеться робити при кожній зміні коду, але він необхідний при перевірці повноцінних тестів встановлення і видалення вашего драйверу операційною системою Windows, так як встановлення драйверу я девелперським підписом більш наближений випадок до реального аніж встановлення з відключенням перевірки підписів)

Слід більш детально розглянути можливість використання одного єкземпляру APO для декількох пристроїв. Так як APO це спосіб обробки звуку який підтримує Windows то майже весь процесс підтримки пристрою операційна система бере на себе. Наприклад одне й те саме APO буде працювати з усіма мікрофонами, які пієднуються через USB вхід до машини та мають одноканальний звуковий вхід. Для такої широкої підримки пристроїв достатньо написати APO яке слід встановлювати на всі пристрої типу USB Media, які мають одноканальний вхід (більшість мікрофонів одноканальні). В такому випадку це APO можна використовувати для обрабки звуку великої кількості мікрофонів різних компаній і моделей. І коли слід модифікувати вже написаний APO (додати новий алгоритм, змінити існуючий) то доведеться змінювати досить малу частину програмного забезпечення одразу для всіх підритмуємих цим APO пристроїв. Але в той самий час APO підтримується лише операційною системою Windows, тому таке програмне забезпечення не може бути використанним на інших операційних системах.

Так як і всі інші способи обробки звуку APO потребує створення системи зв’язку, для зміни параметрів обробки звуку у реальному часі. APO знаходиться у користувацькому режимі, тому вибір способів зв’язку досить великий. Існує можливість обирати яким саме способом забезпечити комунікацію між програмою, яка надає зручний спосіб зміни параметрів обробки звуку та розробленим APO. Можна обирати між більш захищенними способами (наприклад власний сервіс зі своїм закодованим типом зв’язку) або меншзахищеним способом (наприклад звичайний файл в якому змінюються параметри).

APO досить легкий у розробці. Майже всю необхідну інформацію надають Microsoft на офіційному сайті – docs.microsoft.com. Також надаються приклади пустих APO (мінімальнонеобхідний код для роботи APO). Для написання APO достатньо бути знайомим з мовами програмування C та C++, а також з особливостями функціоналу WinAPI. Необхідні знання налагодження коду (так

як APO знаходиться у користуватському режимі то до коду можливо під’єнатись у реальному часі, і такий вид налогодження коду більш звичний для звичаного програмісту у порівнянні з налагодженням коду KMD драйверів).

1.5 Порівняння і висновки

Для порівняння APO з існуючими рішеннями було обрано методи обробки звукового сигналу KMD та метод програмного забезпечення звукових карт (Firmware). Існують ще способи для обробки звуку, але три способи, які розглянуті в цьому дипломі є найбільш ефективними, зручними і простими. Також якщо розглядати обробку звукового сигналу в інших операційних системах, то такі методи як APO та KMD будуть значно програвати деяким методам обробки звуку інших операційних систем. Кожна система має свої плюсі і мінуси, а у системах Windows часто доводиться мати справу з низькорівневими мовами програмування, які значно складніше для розуміння та розробки ніж високорівневі мови програмування, які можуть бути використанні для подібних задач у інших операційних системах.

З таблиці порівняння найбільш розповсюджених методів обробки звуку видно, що використання APO для обробки звукового сигналу має більше позитивних та менше негативних сторін порівняно з іншими методами. APO виграє у таким моментах як:

- Зміна параметрів у реальному часі
- Ризик пошкодження системи
- Має можливість обробки окремих потоків звукових сигналів
- Налагодження коду
- Необхідний рівень знань значно нижчий порівняно з іншими методами

Таблиця 1 – Порівняння найбільш розповсюджених методів обробки звуку

	Обробка у реальному часі	Використання будь-яких алгоритмів	Зміна параметрів у реальному часі	Обробка вхідного і вихідного сигналу	Ризик пошкодження системи	Підтримка різних пристроїв без необхідності зміну коду	Можливість обробки окремих потоків	Залежність від операційної системи	Необхідність підпису	Налагодження коду	Необхідний рівень знань	Кількість позитивних	Кількість прийнятних	Кількість негативних
KMD	Можлива	Можлива	Складна задача по створення зв'язку	Можлива	Дуже вірогідний при розробці	Усі пристрої з однаковими характеристиками	Існує але досить складно	Тільки Windows	Обов'язково	Досить складно і велика ймовірність пошкодження системи	Досить високий	3	2	6
Firmware	Можлива	Можлива	Складна задача по створення зв'язку	Можлива	Малоймовірно	Певні моделі одного виробника	Неможливо так як оброблюється кінечний або початковий сигнал	Будь-яка система	Немає	Досить складно	Досить високий	5	2	4
APO	Можлива	Можлива	Легше створення зв'язку	Можлива	Малоймовірно	Усі пристрої з однаковими характеристиками	Існує	ільки Window	Обов'язково	Так само як звичайни код	Середній	6	3	2

В основному, можна сказати, що APO значно легший у розробці. Для написання і підтримки APO не потрібно бути спеціалістом високого рівня, його розробка може відбуватися на власній робочій машині а не на віртуальній з якою завжди виникають певні проблеми і складнощі, знаходження помилок у коді APO майже не відрізняється від налагодження звичайного коду режиму користувача, при необхідності додання нових технологій, програміст майже не обмежується у виборі їх реалізації (тоді як з KMD та Firmware вибір досить вузький).

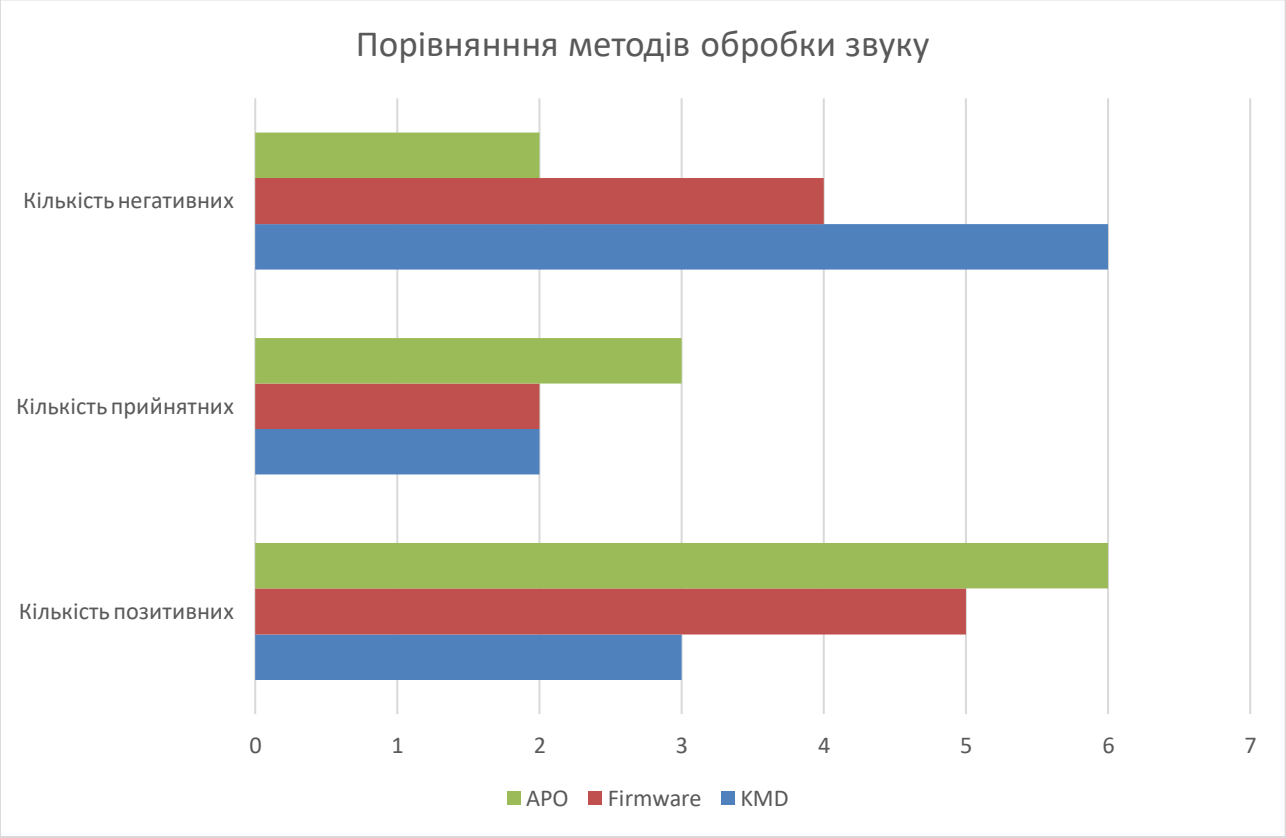

Рисунок 2 – Порівняння методів обробки звуку

Рисунок 3 – Характеристика APO

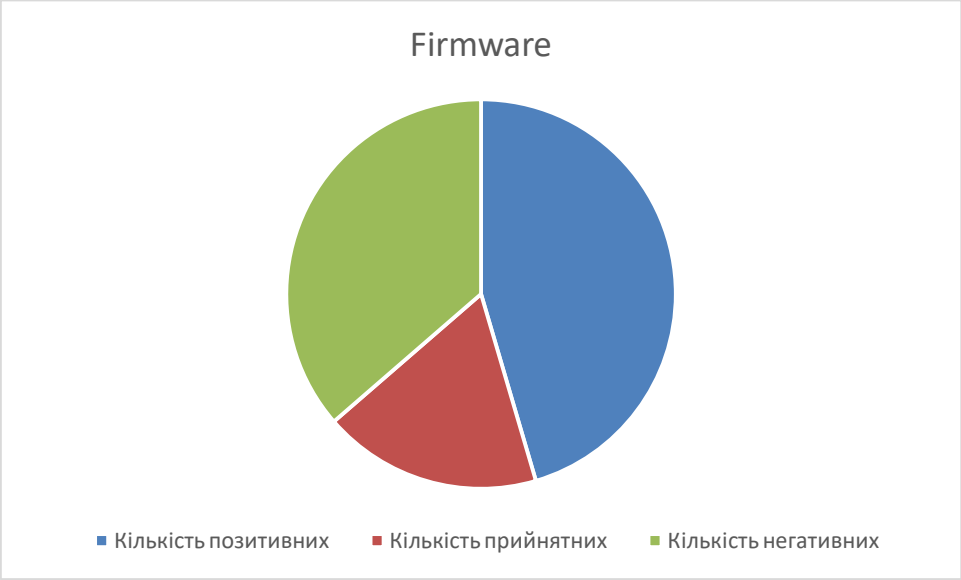

Рисунок 4 – Характеристика Firmware

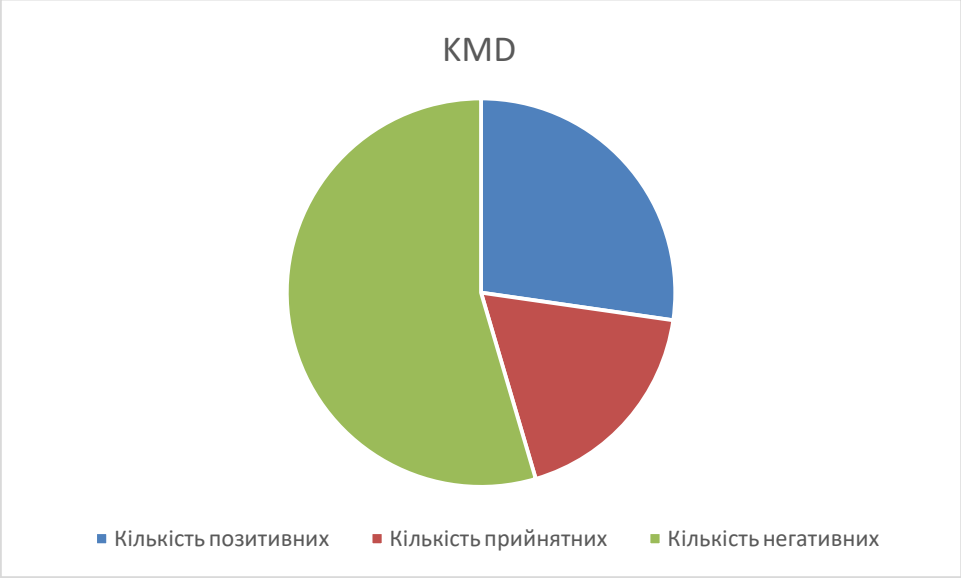

Рисунок 5 – Характеристика KMD

Отримавши кількісне порівняння позитивних сторін, де результати розподілені наступним чином:

Таблиця 2 – Співвідношення позитивних і негативних сторін різних методів обробки звуку

	Позитивні	Негативні
KMD	3	6
Firmware	5	4
APO	6	2

Робимо висновки, що APO більш вдалий вибір так як має більше позитивних і менше негативних сторін. Але це не так. Слід розуміти, коли який метод обробки звуку більш доцільніший.

Не дивлячись на складність розробки Firmware, та необхідність модифікувати програмне забезпечення майже для кожного пристрою окремо – такий метод дуже розповсюджений і необхідний. Цей метод розробки актуальний для виробників пристроїв. Виробнику необхідно розроблювати програмне забезпечення своїх звукових карт і модифікувати в ньому обробку звуку, і все за для того щоб їх пристроїм користувалось більше людей, не залежно від операційної системи, яку використовують користувачі. В таких випадках, компанія завжди має налагоджений процес розробки Firmware для своїх пристроїв, і тому при розробці прошивки для нового пристрою вистачає лише зміни деяких частин коду вже використовуваних прошивок. Коли виникає необхідність розробки прошивки з нуля, слід чітко розуміти, що це ризикова задача, дуже складна і її результат буде актуальним лише для одного або декількох пристроїв з однаковими характеристиками.

KMD в нашому порівнянні програє іншим методам по всім пунктам. Він є найскладнішим у розробці, так як потребує досить суттєвих знань. Також драйвери такого типу несуть суттєвий ризик пошкодження системи, тому їх розробка, а також налагодження роботи коду повинно відбуватись на віртуальній машині, що також має свої негативні сторони. Але драйвери типу KMD широко

розповсюдженні і їх розробкою займається велика кількість спеціалістів. Для обробки звукового сигналу драйвери режиму ядра використовуються рідко. Найбільш розповсюдженні ситуації розробки драйверів KMD:

- Існує драйвер KMD, який призначений для обробки звуку і його слід модифікувати
- Окрім обробки звуку потрібен драйвер, котрий буде виконувати ще певні цілі, які можливо виконати лише у режимі ядра (тоді можна писати один драйвер, котрий буде поєднувати дві задачі – обробка звукового сигналу, та іншу необхідну ціль)
- Потрібно розробити віртуальний звуковий девайс і до його звукового потоку слід застосовувати алгоритми обробки звуку

Розроблювати драйвер KMD лише для обробки звукового сигналу – не аргументовано затратна і ризикована задача. Якщо немає необхідності щоб ваша програма виконувала додаткові дії окрім процесінгу звуку – слід звернути увагу на більш простіші варіанти систем обробки звуку.

Варіант APO перемагає у порівнянні з досить значною кількістю позитивних і досить малою кількість негативних сторін розробки. Але не завжди слід використовувати саме систему розробки звуку APO. Випадка коли НЕ слід використовувати APO:

- Необхідність підтримки операційної системи, яка не є ОС Windows
- Вже існує драйвер типу KMD і його легко можна модифікувати для підтримки нової задачі обробки звукового сигналу

В усіх інших випадках доцільно використовувати систему обробки звуку APO.

KMD – слід використовувати, коли вже існує драйвер KMD, який досить легко модифікувати для підтримки нової задачі обробки звукового сигналу. Або,

коли необхідно, щоб ваш драйвер виконував якусь задачу, яка може бути реалізована лише у драйвері режиму ядра.

Firmware – слід використовувати, коли існує необхідність підтримки ОС окрім Windows. А також коли ви вже маєте відкритий код прошивки і його легко модифікувати для підтримки нової задачі обробки звукового сигналу.

APO – слід використовувати в усіх інших випадках.

2. APO ЯК ПРОГРАМНИЙ МОДУЛЬ

APO являє собою модуль .dll - Dynamic Link Library. Цей модуль може мати всередині себе один або більше різних APO, які будуть використовуватись для різних цілей. ОС Windows налаштована на роботу з APO таким чином – APO це клас, об'єкт якого створює звукова система Windows, коли виникає така необхідність. Операційна система викликає методи, які прописані в APO для виконання конкретних дій у конкретних ситуаціях. Після того, як створений об'єкт APO вже не потрібний (APO використовувалось для обробки звукового сигналу однієї програми і програма закінчила свою роботу) буде викликаний деструктор об'єкта APO. Так як система викликає методи APO то клас APO повинен мати ці методи. APO успадковується від класу CBaseAudioProcessingObject і одразу має всі необхідні методи, які може викликати аудіосистема Windows. Для того щоб ваше APO відрізнялось від стандартного APO, який ви успадковуєте, потрібно перевизначити ті методи, які вам необхідно модифікувати.

Тобто програма APO – це успадкований клас стандартної моделі APO, який ви модифікуєте для певних цілей. Надалі модифіковані вами методи будуть викликатись звуковою системою і будуть використовуватись модифіковані вами алгоритми для обробки звукового сигналу.

Як видно зі схеми проходження аудіобуферу (Додаток 1) APO розташоване після програм, які надають аудіопотік, та перед драйверами режиму ядра. Таким чином повний шлях звукового аудіопотоку складається з:

- Надсилання програмою аудіопотоку
- Обробка аудіоданих за допомогою APO
  - Обробка APO SFX
  - Обробка APO MFX
  - Обробка APO EFX
- Обробка аудіобуферу за допомогою драйверів режиму ядра
- Обробка аудіобуферу за допомогою Firmware пристрою
- Надсилання аудіопотоку на динаміки пристрою

Шлях обробки аудіопотоку досить довгий. Такий широкий вибір, етапів, в котрих може бути оброблений звук, необхідний для підтримки обробки аудіосигналу на рівні користувача, системи та самого пристрою.

В залежності від задач, які повинно виконувати APO, воно буває трьох видів:

- SFX – Stream Effects Filter
- MFX – Mode Effects Filter
- EFX – Endpoint Effects Filter

Також існують LFX(Local Effects Filter) та GFX(Global Effects Filter), але вони використовувались у старих системах Windows і зараз просто мають свої аналоги при використанні LFX = EFX і GFX = EFX. Різниця лише у реєстрації їх у системі [7].

SFX використовується для обробки сигналу, який надсилається однією програмою (також програма може надіслати більше одного окремого звукового потоку, і ці потоки будуть оброблюватись окремими екземплярами об’єктів APO SFX) до його мікшування з іншими потоками. Також цей етап обробки несе відповідальність за перетворення звуку однієї кількості каналів в звук з іншою кількістю каналів (наприклад, коли програма запускає восьми канальний трек, а пристрій має лише два канали – необхідно перетворити вісім каналів у два канали так щоб було збережено інформацію усіх восьми каналів, алгоритми об’ємного звуку займаються подібними задачами). Важливо знати, що одночасно у системі буде створено така кількість об’єктів APO SFX, як кількість запущених аудіопотоків на пристрої на котрому встановлене APO SFX.

MFX використовується для обробки сигналу, який був змікшований з усіх аудіопотоків, які було надіслано на пристрій, на котрому встановлено APO MFX (після етапу обробки SFX), але тільки тих потоків, які було запущено з одним й тим самим режимом обробки звуку. Існують режими:

- Raw
- Default

- Movies
- Media
- Speech
- Communications
- Notification

Таких режимів всього 7. Програма може обрати з яким режимом вона хоче запустити свій аудіопотік. Якщо для пристрою існує і використовується APO MFX, а також якщо в ньому присутня підтримка вказаного режиму, то APO MFX буде оброблювати звук, який запущено з цим режимом певним чином, для потоку аудіоданих у який буде змікшовано всі потоки запущені з таким самим режимом обробки звуку на цьому пристрої. В усіх інших випадках буде використано режим Default [8].

EFX використовується для обробки сигналу, який був змікшований з усіх аудіопотоків, які були надіслані на пристрій на котрому було встановлено APO EFX (після етапу обробки MFX). Для одного пристрою може бути створений і використаний лише один об’єкт APO EFX. Якщо APO EFX встановлено на N пристроях, і на всіх N пристроях буде одночасно використовуватись програвання звукового сигналу, то буде одночасно працювати N об’єктів APO EFX.

Зі схеми взаїмодії APO (Додаток 1) видно з якими потоками які APO працюють. Кількість об’єктів АПО SFX дорівнює кількості потоків, котрі були ініціалізовані програмами (кількість об’єктів АПО SFX НЕ дорівнює кількості програм, тому що одна програма може ініціалізувати безліч звукових потоків). Для кожного пристрою і для кожного режиму з якими були ініціалізовані потоки, створено свій об’єкт APO MFX (так як для першого пристрою було запущено потоки з трьома різними режимами і для другого девайсу було запущено потоки з трьома різними режимами то кількість APO MFX = 6). В кінці, керування обробкою передається до APO EFX (APO EFX усього два об’єкти, так як всього для двох пристроїв були ініціалізовані звукові потоки). Слід зауважити, що місця в котрих пересікаються лінії направлення одного й того самого кольору, це місця

в яких відбувається мікшування потоків звуку в один. За мікшування звуку відповідає не APO, а звукова система Windows. Таку детальну схему обробки звуку не можливо створити з використанням Firmware. Існує можливість перехопленняkmd потоків звуку до мікшування (тобто чисті потоки звуку, які направленні від програми до АПО SFX) з використанням KMD драйверу, але це набагато складніше аніж написання самого драйвера режиму ядра, який націлений лише на обробку звукового сигналу, який направлений на вихід звукового пристрою, а також це невиправдана витрата ресурсів.

3. ОСНОВНІ ЄТАПИ НАПИСАННЯ APO

3.1 Приклади відкритих простих APO рішень

В інтернеті можна знайти відкриті приклади APO, ці APO не несуть в собі досить складних алгоритмів, тому знаходяться під ліцензією MS-PL та GPL. Обидва види ліцензії призначені для публічного використання. Код проектів з ліцензіями MS-PL та GPL можна використовувати навіть у комерційних проектах, але власники коду під цією ліцензією не несуть відповідальність за працездатність та правильність виконання цього коду, а також не дають жодних гарантій. Тобто код може буде використаний будь-ким, але вся відповідальність кладеться на користувача цього коду.

Почати написання APO слід з прикладів, тому що розумним а також рекомендованим рішенням для написання власного APO є модифікація існуючих прикладів. «Використання зразка коду SYSVAD Swap APO як шаблон може прискорити процес розробки APO. Приклад Swap - це зразок, який був розроблений для ілюстрації деяких функцій об'єктів обробки звуку» [9]. Це логічно, модифікувати приклад APO, тому що будь-який APO повинен містити мінімально необхідний код, котрий визначає сам клас APO та деякі перевизначені його методи.

Відкритими прикладами APO є приклади розроблені Microsoft, а також приклади безіменних розробників, які вирішили поділитись своєю працею з іншими людьми:

- DelayAPO (https://github.com/microsoft/Windows-driver-samples/tree/master/audio/sysvad/APO/DelayAPO)
- KWSApo (https://github.com/microsoft/Windows-driver-samples/tree/master/audio/sysvad/APO/KWSApo)
- SwapAPO (https://github.com/microsoft/Windows-driver-samples/tree/master/audio/sysvad/APO/SwapAPO)
- EqualizerAPO (https://github.com/mirror/equalizerapo/tree/master/EqualizerAPO)

Ці приклади можуть бути використанні для їх модифікації, з ціллю розробки власного APO. В прикладі EqualizerAPO реалізовані навіть деякі серйозні алгоритми такі як еквалайзер та інші.

3.2 Основні методи APO

Основними методами APO, які вам скоріше за все прийдеться перевизначити у своєму проекті є такі методи (написано у порядку виклику методів звуковою системою Windows):

- Initialize
- IsInputFormatSupported (тільки для SFX)
- IsOutputFormatSupported (тільки для SFX)
- LockForProcess
- APOProcess
- UnlockForProcess
- Деструктор вашого APO

При перевизначені методів APO існує два шляхи реалізації:

- Написання коду всього методу самостійно
- Виклик існуючого методу з класу CBaseAudioProcessingObject та додавання до методу власного коду

Виглядає це наступним чином:

- Випадок самостійного написання

```
STDMETHODIMP MyAPOClass::LockForProcess(UINT32 u32NumInputConnections,
    APO_CONNECTION_DESCRIPTOR** ppInputConnections,
    UINT32 u32NumOutputConnections,
APO_CONNECTION_DESCRIPTOR** ppOutputConnections)
```

```
{
    //Логіка прописана вами самостійно
}
```

- Випадок виклику існуючого методу

```
STDMETHODIMP MyAPOClass::LockForProcess(UINT32 u32NumInputConnections,
    APO_CONNECTION_DESCRIPTOR** ppInputConnections,
    UINT32 u32NumOutputConnections, APO_CONNECTION_DESCRIPTOR** ppOutputConnections)
{
HRESULT hr = S_OK;
hr = CBaseAudioProcessingObject::LockForProcess(u32NumInputConnections,
    ppInputConnections, u32NumOutputConnections, ppOutputConnections);
        IF_FAILED_JUMP(hr, Exit);

Exit:
        return hr;
}
```

Обидва варіанти є правильними, і вибір між ними здійснюється в залежності від потреб вашого класу APO.

Initialize – метод, який викликається для ініціалізації об’єкту класу APO. Тут слід ініціалізувати всі необхідні данні для роботи APO. Метод може бути викликаний багаторазово для одного й того самого потоку звуку, це відбувається тому що сторона виклику може з першого разу ініціалізувати аудіопотік з неправильними даними, тоді ініцілізація буде завершена з помилкою і стороні виклику доведеться викликати ініціалізіцію потворно.

IsInputFormatSupported (тільки для SFX) – викликається стороною ініціалізації аудіопотоку, з ціллю перевірки підтримки APO певних характеристик вхідного аудіопотоку (кількість каналів, канальна маска та інше). В цьому методі слід вказувати, які саме канальні перетворення підтримує ваше APO в інакшому випадку, сторона виклику зможе ініціалізувати аудіопотік тільки з тими характеристиками, які відповідають вашому пристрою.

IsOutputFormatSupported (тільки для SFX) – викликається стороною ініціалізації аудіопотоку, з ціллю перевірки підтримки APO певних характеристик вихідного аудіопотоку. Працює так само, як і IsInputFormatSupported() метод.

LockForProcess – викликається аудіосистемою при завершені всіх необхідних етапів ініціалізації а також всіх перевірок підтримки форматів. Коли цей метод завершається, це свідчить про те що об'єкт APO повністю готовий до обробки аудіопотоку (об'єкт знає і підтримує всі характеристики аудіопотоку а також для цього об'єкту вже ініціалізовані всі необхідні ресурси). В цьому методі слід виконати всі остаточні ініціалізації, які будуть необхідні при роботі обробки звуку.

APOProcess – викликається аудіосистемою у РЕАЛЬНОМУ ЧАСІ для обробки буферу аудіоданих. Саме в цьому методі виконується процесінг звуку (накладання всіх необхідних алгоритмів). Слід зауважити, що це метод, який викликається у реальному часі, а це означає, що затримки у його виконанні призведуть до затримки передачі аудіоданих на пристрій, і в результаті виникне рипіння. Також до цього методу накладаються обмеження (вони не є обов'язковими, і програми буде працювати з недотриманням цих обмежень, але в такому випадку ймовірність виникнення помилок значно зростає):

- Код повинен бути якнайбільш оптимізований
- Не повинні використовуватися ніякі засоби блокування (мьютекси та подібне)
- Не можна використовувати динамічне виділення пам'яті

Щодо оптимізованого коду, та заборони на динамічне виділення пам’яті все очевидно. Не оптимізовані алгоритми, а також виділення пам’яті займають багато часу, тому в результаті звуковий сигнал не буде встигати оброблюватись а отже на пристрій буде надсилатись сміття, яке буде чутно як рипіння.

Блокування може призвести до усього блокування потоку обробки звуку, що очевидно погано для програмного забезпечення, яке повинно оброблювати звук [10].

UnlockForProcess – метод протилежний методу LockForProcess. Слід звільнити усі виділені ресурси, а також завершити процес з усіма об’єктами взаємодії з якими було розпочато цей процес.

Деструктор вашого APO – виконується звільнення ресурсів виділених під час ініціалзіцаї. Є завершальним етапом після якого не можуть бути використанні жодні методи та данні об’єкту APO. Якщо після завершення виконання цього методу не були звільненні якісь ресурси то відбудеться витік пам’яті.

Зі схеми викликів функцій APO звуковою системою (Додаток 1) видно в якій послідовності викликаються метода APO звуковою системою:

- Initialize
- IsInputFormatSupported (тільки для SFX)
- IsOutputFormatSupported (тільки для SFX)
- LockForProcess
- APOProcess
- UnlockForProcess
- Деструктор вашого APO

Також на схемі вказано, що методи: APOProcess та Деструктор не повертають у аудіо систему ніяких значень. Вони використовуються для обробки переданих параметрів (буферу аудіосигналу) та для звільнення використовуваних APO ресурсів.

Усі інші методи повертають HRESULT – результат виклику методу.

Виклик таких методів як: Initialize, IsInputFormatSupported, IsOutputFormatSupported, LockForProcess та UnlockForProcess відбувається до поки не буде повернено результат виконання методу S_OK (виконано без помилок), або до поки викликаюча сторона не вирішить зупинити спроби виклику цих методів.

Метод APOProcess, викликається протягом усього часу, до поки не буде завершена обробка аудіосигналу.

4. НАЛАГОДЖЕННЯ АРО У СИСТЕМІ

4.1 Встановлення АРО у систему

За допомогою технології INF

Windows використовує файли інформації установки INF для установки наступних компонентів на пристрій:

- Один або кілька драйверів, що підтримують пристрій
- Конфігурація для конкретного пристрою або налаштування для переведення пристрою в оперативний режим.

Файл INF - це текстовий файл, що містить всю інформацію, яку компоненти установки пристрою використовують для встановлення драйвера. Windows встановлює драйвери за допомогою файлів INF. Інформація таких файлів включає наступне:

- Ім'я та місцезнаходження драйверу
- Інформація про версію драйвера
- Інформація, яку слід занести до реєстру

Також існує можливість використання файлів INX для автоматичного створення файлів INF [11].

Використання файлів INF для встановлення АРО драйверів дуже зручний і досить розповсюджений спосіб. Він має свої сильні сторони:

- Легкість написання
- Легкість підтримки
- Задачу виявлення необхідності, та встановлення драйверу ОС Windows повністю бере на себе
- Виконує всі необхідні дії для реєстрації драйверу у системі

Майже у всіх випадках слід використовувати файли INF для встановлення драйверів APO. В інтернеті можна знайти багато прикладів написаних INF файлів. Також завантаживши будь-який проект який використовує для встановлення своїх драйверів технологію INF ви отримуєте файл INF у папці з усіма файлами. INF файли досить розповсюдженні, тому в інтернеті можна знайти відповіді на питання, котрі у вас виникають, в більшості випадків ви не будете першим у кого виникне подібна проблема.

Порівняно з наступним способом встановлення драйверу у систему, спосіб INF файлу має один суттєвий плюс - задачу виявлення необхідності, та встановлення драйверу ОС Windows повністю бере на себе. Це означає, що вам досить вказати в вашому INF файлі лише унікальний ідентифікатор пристрою, на котрий повинен встановлюватись APO драйвер. Усі інші задачі виконає ОС Windows:

- Розпізнає підключення нового пристрою
- Виявить необхідний INF файл, який слід використати для цього пристрою
- Виконає реєстрацію драйверу
- Копіювання драйверу до необхідного шляху у системі
- Заповнить всі необхідні реєстри для реєстрації драйверу на необхідному пристрої
- Перезавантажить аудіосистему, щоб дії вступили в силу (APO одразу почне використовуватись звуковою системою)

Використання INF файлів значно спрощує встановлення APO драйверів у систему, а також на девайс. При правильно написаному INF файлі, розробнику не потрібно турбуватись за взаємодію APO драйверу з системою. Але такий спосіб має один мінус – необхідність підписання INF файлу та драйверу. Це не є тим мінусом через, який слід шукати альтернативу INF файлу, але слід розуміти, що ваш драйвер APO повинен бути перевірений та зареєстрований.

Ручне встановлення

Ще одним способом встановлення APO драйверу є ручне встановлення. Таке встановлення використовується значно рідше. У проекті, який вище розглянутий як приклад написання APO драйверу, EqualizerAPO використовується ручне встановлення APO драйверу.

Ручне встановлення драйверу означає, що програміст сам повинен:

- Відстежити підключення нового пристрою у систему
- Виявити необхідність встановлення APO драйверу на цей пристрій
- Занести у реєстри необхідну інформацію про реєстрацію APO у системі
- Занести у реєстри необхідну інформацію про реєстрацію APO на конкретний пристрій
- Обробити необхідність перезавантаження операційної системи/пристрою/аудіосистеми

У випадку використання ручного встановлення драйверу APO слід виконати всі пункти наведені вище. Будь-яким способом потрібно визначати на який пристрій встановлювати драйвер. Необхідно провести реєстрацію APO як у системі так і на пристрої, так само як це робить ОС Windows за допомогою INF файлу. І в останню чергу слід визначити, що ви будете перезавантажувати і як, щоб система почали використовувати зареєстроване вами APO.

Плюси використання ручного встановлення APO драйверу:

- Відсутня необхідність підпису драйверу
- Можливість встановлення APO драйверу на віртуальні девайси

При використанні такого способу зникає необхідність підпису вашого APO драйверу, так як ви самі виконуєте встановлення його у систему і система не буде перевіряти цей драйвер на надійність.

Щодо віртуальних девайсів – це основна причина використання ручного встановлення. Віртуальні пристрої це пристрої, які фізично не під'єднані до машини. Ці пристрої створюються драйверами режиму ядра і цей же драйвер регулюэ взаємодію з цим пристроєм. Прикладами таких віртуальних пристроїв є:

- Віртуальний аудіокабель
- Блютуз пристрій
- Інші

Проблема виникає тому що так як ці пристрої це драйвери режиму ядра, то насправді вони вже використовують один INF файл. Коли ви спробуєте встановити на такий пристрій ваш APO драйвер за допомогою технології встановлення INF то ви приведете пристрій у несправний стан (до наступного перезавантаження операційної системи/пристрою/аудіосистеми). Тому у випадку, коли необхідно використати APO драйвер з віртуальним пристроєм, доведеться використовувати ручну реєстрацію. Так як для системи віртуальний пристрій з точки зору реєстрів – звичайний пристрій, то достатньо відтворити реєстрацію у реєстрі вашого APO драйверу так само, як це виконує ОС Windows.

Ручна реєстрація набагато складніше ніж звичайна реєстрація через INF файл, так як потребує виконання всіх необхідних дій програмістом. Але інколи не можливо обійтись без ручної реєстрації, наприклад у ситуації з віртуальними пристроями.

4.2 Способи зв'язку з APO

Windows Registry

Windows Registry – являє собою систему певних баз даних, в якій зберігаються та з якої вилучаються додатки і компоненти системи. Дані, що зберігаються в реєстрі, залежать від версії Microsoft Windows. Додатки використовують API реєстру для отримання, зміни або видалення даних реєстру [12].

Реєстри – дуже зручний спосіб зберігання інформації. Будь-яка програма встановлена у ОС Windows зберігає у реєстрах свою інформацію, мінімально це:

- Назва програми
- Версія програми
- Постачальник програми
- Дата встановлення програми
- Як слід видаляти програму
- Та інше.

Всі ці пункти, є мінімальною базою даних, яка створюється для кожної встановленої програми, і надалі використовується і модифікується системою при необхідності. Але окрім службової інформації, кожна програма має право створити свій розділ у реєстрі і зберігати там свої данні. Наприклад, можна зберігати, певні налаштування користувача, шляхи до необхідних файлів та папок у системі, також можна зберегти навіть самі файли. За допомогою реєстрів можна настроїти взаємодію системи з вашою програмою, наприклад додати програму до автозапуску, додати перевірку оновлень і т.п.

Реєстри це сильний і досить зручний інструмент, який використовує ОС Windows. І він може бути використаний для зв’язку з APO через іншу програму зі зручним для користувача інтерфейсом. Зв’язок з APO необхідний для зміни параметрів обробки звуку у реальному часі.

WinAPI надає простий функціонал для роботи з реєстрами. Краще за все використовувати для взаємодії з реєстрами клас CRegKey. CRegKey – клас, який надає методи для управління записами в системному реєстрі [13]. Найбільш розповсюдженими і найбільш кориснішими методами класу CRegKey можна вважати:

- Open – відкриття розділу
- Create – створення розділу
- DeleteSubKey – видалення розділу

- DeleteValue – видалення ключа
- QueryValue (та йому подібні) – вилучення ключа
- SetValue (та йому подібні) – запис ключа
- NotifyChangeKeyValue – відстеження змін у реєстрі

Тобто клас CRegKey має можливість створювати та видаляти розділи а також ключі, можливість встановлювати та вилучати значення певних ключів, і найголовніше, надає зручний спосіб відстежувати зміни ключів у розділі. Також для взаємодії з ключами можуть бути використанні додаткові методи, подібні методам QueryValue / SetValue, але з уточненням типу даних з яким необхідно взаємодіяти:

- Binary
- DWORD
- GUID
- MultiString
- QWORD
- String

NotifyChangeKeyValue – повідомляє викликаючу сторону про зміни атрибутів або вмісту відкритого розділу реєстру [14]. Цей метод дозволяє створити очікування зміни реєстрів у додатковому потоці. В такому випадку викликаюча сторона може не витрачати процесорний час на перевірку зміни параметрів у реальному часі. Коли через дії виконані іншою програмою, дані необхідних розділів будуть змінені, керування кодом повернеться до додаткового потоку викликаючої сторони, вона зможе обновити необхідні данні, і після цього знову перейти в режим очікування зміни даних необхідного розділу реєстру. Код очікування зміни необхідного розділу може виглядати наступним чином:

```
void WaitChanges()
```

```
{
        CRegKey key;
        key.Open(HKEY_LOCAL_MACHINE, L"SOFTWARE\\MyApp");

        while (true)
        {
                key.NotifyChangeKeyValue(TRUE,
REG_NOTIFY_CHANGE_LAST_SET, NULL, FALSE);

                if (key == NULL)
                        return;

                //зчитування з реєстру необхідних даних і використання їх
        }
}
```

Використання Windows Registry як способу зв'язку програми із програмним кодом APO – це дуже простий і зручний спосіб, який легко використовувати навіть недосвідченому програмісту. Використання класу CRegKey надає можливість створювати та видаляти розділи а також ключі, встановлювати та вилучати значення певних ключів, і найголовніше, надає зручний спосіб відстежувати зміни ключів у розділі.

Використання власного сервісу

Сервіс – програма, що запускається автоматично системою при запуску Windows і виконується незалежно від статусу користувача. Має спільні риси з концепцією демонів в Unix [15]. Одним з варіантів зв'язку вашої програми з APO у ОС Windows є написання власного сервісу. Сервіс в даному випадку буде програмою, яка безкінечно працює у фоновому режимі, та автоматично запускається при завантажені операційної системи.

Сервіс повинен отримувати та передавати інформації між програмами (між кодом APO та програмою з зручним користувацьким інтерфейсом для зміни параметрів обробки звуку). Така програма повинна очікувати надходження та отримувати повідомлення. Ці повідомлення краще ще й шифрувати за своїм власним принципом шифрування. Після чого надсилати необхідне повідомлення у іншу програму за своїм власним протоколом.

Подібний функціонал у Windows може бути написаний за допомогою спільної пам’яті (Shared Memory) [16]. Такий функціонал надає можливість створити спільну пам’ять і керувати нею. В цю пам’ять, один процес може записувати данні (повідомлення оформлене за певними правилами та можливо зашифроване), а інший процес може зчитувати ці данні.

Подібний сервіс складний у написанні. Необхідно створити досить вдало спроектований протокол зв’язку, а також спосіб передачі повідомлень та сигналізування відправки повідомлення. І порівняно з варіантом зв’язку за допомогою Windows Registry, використання яких може бути написано одразу у коді програми з зручним користувацьким інтерфейсом для зміни параметрів обробки звуку, сервіс це окрема програма. Тобто потрібно написати три програми:

- АПО
- Програма користувацького інтерфейсу
- Програма сервісу

Тобто час розробки, а також тестування – значно більший ніж при використанні Windows Registry.

Сервіс має досить суттєву позитивну сторону – безпека. Дані збережені у Windows Registry можуть бути зчитані, змінені, та видалені будь-якою програмою з правами адміністратора (хоча в офіційній документації і написано «Ви не повинні редагувати дані реєстру, які не належать вашій програмі, за винятком випадків крайньої необхідності» [12], але все ж таки така можливість існує, і ваші дані можуть бути модифіковані іншою програмою).

Зміна ваших даних буде означати втрату або їх актуальності або повну втрату даних. А якщо код APO не пристосований до втрати певних розділів у реєстрі, то в найгіршому випадку він може перестати коректно працювати. Сервіс в свою чергу є більш захищеним. Не дивлячись на те, що спільна пам'ять, яку ви будете використовувати – це теж пам'ять комп'ютера, яку при бажанні можна змінити, зробити це не так просто. Протокол створення та використання спільної пам'яті, для роботи вашого процесу, може бути унікальним, а також може використовувати шифрування, в такому випадку, використання не коректних даних не може відбутись.

4.3 Особливості налагодження програмного коду APO

APO – код, який працює у режимі користувача. Як і будь, який код у режимі користувача, код APO можна налагоджувати у реальному часі, спостерігаючи за виконанням програмою поставленої задачі, за зміною параметрів та іншим. Найзручнішою програмою для розробки APO є Visual Studio. Звичайно, код APO можна розробляти і у будь-якому іншому середовищі, але Visual Studio має підтримку розробки драйверів, і надає певні допоміжні властивості при розробці.

Так як APO це файл динамічної бібліотеки (.dll), а також викликається він не вашою програмою, то тестування працездатності його коду, дещо відрізняється від налагодження звичайного коду. Для початку написаний APO слід встановити у системі на націлений для розробки пристрій. Після встановлення APO на пристрій, та запуску нашої DLL звуковою системою Windows, наш APO буде використовуватись як модуль процесом Audiodg.exe (Audio D evice G raph). Audiodg керує передачею звукових потоків у середовищі аудіографу.

Враховуючи, що модуль APO запускається процесом, який ми не можемо контролювати, для налагодження роботи коду APO слід під'єднуватись до викликаючого процесу (Audiodg.exe) у реальному часі. Ви можете використовувати «Attach to Process» у середовищі Visual Studio для

налагодження запущених програм на локальних або віддалених комп'ютерах, одночасного налагодження декількох процесів, налагодження програм, що не були створені в Visual Studio, або налагодження будь-якої програми, яку ви не запустили з Visual Studio з підключеним налагоджувачем [17]. За допомогою «Attach to Process» можна під’єднатись до APO, який було запущено іншою програмою (Audiodg.exe), і налагоджувати код таким самим чином, якби налагодження відбувалось з програмою запущеною у середовищі Visual Studio.

Налагодження коду APO, дещо складніше, аніж налагодження коду запущеного середовищем у якому відбувається налагодження. Але все ж таки існує спосіб «Attach to Process», який значно легший за налагодження коду KMD або Firmware.

4.4 Особливості логування програмного коду APO

Після випуску будь-якого програмного продукту розробник має підтримувати свій продукт ще деякий час. Найчастіше для цього їх програмний продукт пише спеціальний звіт про свою роботу. У нього потрапляє інформація про помилки, збої, несправності та іншу статистику, вже працюючого програмного забезпечення. Процес запису цієї інформації називають логуванням. За допомогою логування розробник може зібрати статистику про те, як працювало його програмне забезпечення на комп’ютері користувача. Це допомагає знайти чому, де і коли виникли несправності у роботі програмного чи апаратного забезпечення. Частіше за все, при виникненні помилки, користувач має відправляти файл з текстом логування (так званий log-файл) розробнику, щоб той міг його прочитати та зрозуміти як зробити, щоб ця несправність більше не з’являлась. Найчастіше людина яка відповідає за відсилання лог-файлів є не зовсім звичайним користувачем, а наприклад системним адміністратором [18].

Логування досить корисне при розробці APO. Після написання APO на власній машині, налагодження його роботи на власній машині з власними пристроями – при використанні того самого драйверу з іншим пристроєм та на іншій машині можуть виникнути проблеми. По-перше так може статись, тому що

одна людина не може швидко перевірити усі варіанти роботи з APO на усіх машинах та на усіх пристроях. Тому тестувальник, може зіткнутись з проблемою, з якою розробник не мав би можливості зустрітись. З кожною версією ОС Windows, в систему додаються зміни, які часто направлені на захист системи, а отже з новою версією операційної системи можуть з'явитись нові обмеження для коду APO. І навпаки, може статись так, що код написаний на більш новішій версії використовує компоненти, яких немає в більш старих версіях.

Логування необхідно для будь-якого великого проекту, для того щоб збирати інформацію, яка в подальшому допоможе в налагоджені роботи коду APO.

Більшість програм можна логувати багатьма способами:

- Вивід інформації у консольне вікно
- Вивід інформації тільки у режимі налагодження
- Вивід інформації у файл
- Вивід інформації у базу даних
- Вивід інформації у віддалену базу даних
- Та багато інших способів

Зважаючи на те, що APO це драйвер, а також те, що деякі логи потрібно виводити у режимі реального часу, не всі існуючі способи будуть доцільні при логуванні APO. Microsoft рекомендують використовувати готовий спосіб логування коду WPP Software Tracing.

WPP Software Tracing – програма, яка доповнює і покращує трасування подій WMI, додаючи способи спростити процес керування постачальника трасування. Це ефективний механізм для постачальника трасування, що дозволяє реєструвати виконавчі повідомлення в реальному часі. Записані в журнал повідомлення згодом можуть бути перетворені в інформацію зручну для розуміння [19]. WPP логування слід використовувати при роботі з:

- Логуванні драйверу режиму ядра (KMD)

- Логуванні драйверу користувацького режиму
- Логуванні програми або бібліотеки динамічного компонування (DLL)

WPP Tracing надає можливість:

- Вивід повідомлення у вигляді тексту
- Перетворення значення змінної / змінних у текст і виведення їх разом з текстом
- Групування повідомлень за їх характером

Повідомлення надіслані за допомогою системи WPP Tracing, захоплюються програмою traceview.exe. TraceView (TraceView.exe) налаштовує і контролює сеанси трасування і відображає відформатовані повідомлення трасування з сеансів трасування в реальному часі і журналів [20]. В програмі TraceView можна налаштувати, повідомлення якого модуля слід захоплювати та відображати, можна відсортувати повідомлення за певним значенням. Програма зручна і інтуїтивно зрозуміла, тому нею досить просто користуватись тестувальникам, а отже звіти з логуванням будуть досить корисними для налагоджування коду.

Враховуючи, що код APO може використовуватись одночасно необмеженою кількістю потоків, налагодження коду при прямому підключені неймовірно складне, адже при зупинці коду, ви зупиняєте одразу всі потоки процесу Audiodg і вам доводиться переглядати правильність виконання коду одразу усіма потоками. Тому логування коду APO необхідне при його розробці. До логування кожного повідомлення, можна додати унікальний ідентифікатор потоку, щоб відрізняти потоки між собою. Також рекомендується додавати до кожного логу при логуванні – адресу об’єкту APO. Коли у коді виникнуть проблеми, і один з об’єктів APO стане несправним, буде достатньо переглянути логи лише відповідні адресі цього об’єкту APO.

4.5 Програмний продукт APO

Програмний продукт APO з назвою CBORRenderAPOEFX (Додаток 2). Назва складається з:

- C – вказує, що це клас
- Унікальна назва компанії (BOR – ініціали розробника)
- Напрямок використання Render (Потік рендеринга аудіо. Аудіодані передаються з програми в кінцеве аудіопристрій, який візуалізує потік [21], або просто пристрій виводу аудіоданих)
- APO – вказує, що це клас успадковується від класу CBaseAudioProcessingObject
- EFX – вказує на тип обробки аудіоданих

APO CBORRenderAPOEFX має всі необхідні методи для функціонування у режимі обробки аудіоданих EFX:

- CBORRenderAPOEFX
- ~CBORRenderAPOEFX
- Initialize
- LockForProcess
- APOProcess
- UnlockForProcess
- GetLatency
- GetEffectsList
- ValidateAndCacheConnectionInfo

Методи GetLatency, GetEffectsList та ValidateAndCacheConnectionInfo не були розглянуті раніше:

- GetLatency – отримує затримку кінцевої точки аудіо [22].

- GetEffectsList – використовується для отримання списку ефектів обробки звуку, які в даний час активні, і зберігає подію, про яку буде повідомлено, якщо список зміниться [23].
- ValidateAndCacheConnectionInfo – виділяє пам'ять для зберігання деталей формату, наприклад, кількості каналів, частоти дискретизації, глибини дискретизації і маски каналу [9].

Також APO CBORRenderAPOEFX має поля класу:

- m_AudioProcessingMode
- m_spAPOEndpointProperties
- m_spAPOSystemEffectsProperties
- sm_RegProperties
- m_UncompOutputFormat
- m_UncompInputFormat
- m_EffectsLock
- m_hEffectsChangedEvent

APO CBORRenderAPOEFX у методі CBORRenderAPOEFX::APOProcess() для всіх пристроїв у котрих кількість каналів не дорівнює двом – виконує копіювання буферу аудіоданих за допомогою функції CopyFrames (використовується макрос CopyMemory для копіювання буферу). Для пристроїв з двома каналами, виконує присвоєння каналу FL значення 0 (функціонал мьют – заглушення звуку), для FR каналу зменшує гучність на 30 децибел. Таким чином у APO CBORRenderAPOEFX показана реалізація такої функціональності як:

- Обробка окремих випадків характеристик пристроїв
- Обробка окремих каналів пристрою різними алгоритмами
- Використання функціоналу мьют
- Використання функціоналу зміни гучності

Для обробки окремих каналів слід використовувати «Алгоритм обробки окремих аудіоканалів» (Додаток 1). Зі схеми видно, що послідовність дій алгоритму наступна:

1) Для кожного каналу, котрий слід обробити виконати перехід до блоку 2.
2) Якщо в канальній масці вхідного аудіосигналу присутній канал до якого застосовуються дії то перейти до блоку 3, якщо відсутній канал то перейти до блоку 1.
3) Копіювати вхідний буфер.
4) Змістити позицію вхідного буфера на позицію першого елементу каналу, який слід обробити
5) Для кожного фрейму виконати перехід до блоку 6.
6) Використати алгоритми обробки до семплу на котрий вказує буфер.
7) Змістити позицію буферу на кількість каналів вхідного буфера

Таким чином буде використано алгоритм обробки звукового потоку для конкретного каналу. Також, якщо необхідно оброблювати усі канали одночасно, цей алгоритм може бути модифіковано:

- Копіювати вхідний буфер
- Для кожного фрейму виконати наступні дії:
    - Для кожного каналу вхідного буферу виконати наступні дії:
        - Використати алгоритми обробки до семплу на котрий вказує буфер
        - Змістити позицію буферу на один семпл

Модифікованим алгоритмом буде оброблено одразу всі канали вхідного аудіобуферу.

Якщо є необхідність оброблювати одночасно всі канали аудіопотоку, але по різному, то на етапі «Для кожного каналу вхідного буферу виконати наступні дії», модифікованого алгоритму, слід в залежності від порядкового номеру каналу пристосовувати для нього конкретний алгоритм. Саме такий алгоритм було використано для обробки окремих аудіоканалів, різними алгоритмами у методі CBORRenderAPOEFX::APOProcess().

- ВИСНОВКИ

При написанні даного дипломного проекту було розглянуто обробку аудіоданих через драйвери на базі APO у системах Windows, а також альтернативні існуючі рішення обробки аудіоданих.

В процесі дослідження отримані такі результати:

1. Виявлено недоліки а також переваги APO на фоні існуючих альтернативних рішень. Основними недоліками APO є:

- Залежність від операційної системи
- Необхідність підпису

В той самий час, APO має велику кількість переваг:

- Існує можливість зміни параметрів у реальному часі
- Ризик пошкодження системи досить низький
- Має можливість обробки окремих потоків звукових сигналів
- Налагодження коду досить легке порівняно з альтернативними варіантами
- Необхідний рівень знань значно нижчий порівняно з іншими методами

Варіант APO перемагає у порівнянні існуючих рішень обробки звуку, з досить значною кількістю позитивних і досить малою кількість негативних сторін розробки. Але не завжди слід використовувати саме систему розробки звуку APO. Випадка коли НЕ слід використовувати APO:

- Необхідність підтримки операційної системи, яка не є ОС Windows
- Вже існує драйвер типу KMD і його легко можна модифікувати для підтримки нової задачі обробки звукового сигналу

В усіх інших випадках доцільно використовувати систему обробки звуку APO.

2. Визначено, що APO - це модуль .dll, який може мати всередині себе один або більше різних APO. APO успадковується від класу CBaseAudioProcessingObject і одразу має всі необхідні методи, які може викликати аудіосистема Windows.

В залежності від задач, які повинно виконувати APO, воно буває трьох видів:

- SFX – Stream Effects Filter
- MFX – Mode Effects Filter
- EFX – Endpoint Effects Filter

3. Знайдено приклади APO, які знаходяться у відкритому доступі (тобто їх можна використовувати, як основу для написання власного APO). Визначено особливості основних методів APO.

4. Порівняно способи встановлення APO у ОС Windows, та виявлено при яких обставинах який спосіб слід використовувати.

5. Порівняно найрозповсюдженіші способи зв'язку між програмами, які доцільно використовувати при написані APO.

6. Визначено особливості налагодження програмного коду APO.

7. Визначено особливості логування програмного коду APO, та необхідність логування APO.

За результатами досліджень, APO є досить зручним методом обробки аудіоданих, який слід використовувати у випадках, окрім випадків, коли:

- Необхідність підтримки операційної системи, яка не є ОС Windows
- Вже існує драйвер типу KMD і його легко можна модифікувати для підтримки нової задачі обробки звукового сигналу

Слід чітко розуміти, яке саме APO потрібно для яких задач, так як задачі у SFX, MFX і EFX досить різні, і мають свої особливості написання. При написанні APO слід модифікувати існуючі приклади APO задля спрощення роботи і уникнення більшості помилок. В залежності від пристроїв, які слід підтримувати (фізичні чи віртуальні пристрої) слід орієнтуватись на спосіб встановлення APO

через INF файл або через ручне встановлення. APO може використовувати різні способи зв’язку з іншими програмами, для зміни параметрів обробки звуку у реальному часі. Способи зв’язку, здебільшого, відрізняються лише складністю написання. Так як програмний файл APO викликається НЕ середою розробки, то для налагодження коду необхідно використовувати «Attach to Process» технологію. Логування необхідне для APO, по-перше це спрощую процес налагодження програмного коду APO на різних машинах та з використанням різних пристроїв, по-друге один й той самий код APO одночасно може бути використаний необмеженою кількістю потоків, в таких умовах ручне налагодження коду є дуже складним. При логуванні APO слід в кожному повідомлені вказувати унікальний ідентифікатор потоку, який надсилає повідомлення, а також унікальний ідентифікатор об’єкту APO (наприклад адреса об’єкту) для простоти виявлення помилок при перегляді журналу логування.

- СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Kernel-Mode Driver Framework [Електронний ресурс]. – 2021. – Режим доступу: https://en.wikipedia.org/wiki/Kernel-Mode_Driver_Framework - Дата доступу: квітень 2021.

2. Прятки по хардкору. Как сделать свой драйвер режима ядра Windows и скрывать процессы. [Електронний ресурс]. – 2020. – Режим доступу: https://xakep.ru/2018/03/14/kmdf-driver/ - Дата доступу: квітень 2021.

3. Виртуальная отладка: отладка Kernel Mode кода с использованием VMware. [Електронний ресурс]. – 2009. – Режим доступу: https://xakep.ru/2009/06/23/48628/ - Дата доступу: квітень 2021.

4. Все драйверы режима ядра для Windows 10 (1607) теперь должны быть подписаны Microsoft. [Електронний ресурс]. – 2016. – Режим доступу: https://habr.com/ru/post/306862/ - Дата доступу: квітень 2021.

5. Что такое звуковая карта и для чего она нужна? [Електронний ресурс]. – 2018. – Режим доступу: http://procomputer.su/sostav-kompyutera/38-chto-takoe-zvukovaya-karta - Дата доступу: квітень 2021.

6. Audio Processing Object. [Електронний ресурс]. – 2020. – Режим доступу: https://en.everybodywiki.com/Audio_Processing_Object - Дата доступу: квітень 2021.

7. Audio Processing Object Architecture. [Електронний ресурс]. – 2019. – Режим доступу: https://docs.microsoft.com/ru-ru/windows-hardware/drivers/audio/audio-processing-object-architecture - Дата доступу: квітень 2021.

8. Audio Signal Processing Modes. [Електронний ресурс]. – 2018. – Режим доступу: https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/audio-signal-processing-modes - Дата доступу: квітень 2021.

9. Implementing Audio Processing Objects. [Електронний ресурс]. – 2020. – Режим доступу: https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/implementing-audio-processing-objects - Дата доступу: квітень 2021.

10. Real-time audio programming 101: time waits for nothing. [Електронний ресурс]. – 2011. – Режим доступу: http://www.rossbencina.com/code/real-time-audio-programming-101-time-waits-for-nothing - Дата доступу: квітень 2021.

11. Overview of INF Files. [Електронний ресурс]. – 2017. – Режим доступу: https://docs.microsoft.com/en-us/windows-hardware/drivers/install/overview-of-inf-files - Дата доступу: квітень 2021.

12. Registry. [Електронний ресурс]. – 2018. – Режим доступу: https://docs.microsoft.com/en-us/windows/win32/sysinfo/registry - Дата доступу: квітень 2021.

13. CRegKey Class. [Електронний ресурс]. – 2016. – Режим доступу: https://docs.microsoft.com/en-us/cpp/atl/reference/cregkey-class?view=msvc-160 - Дата доступу: квітень 2021.

14. CRegKey::NotifyChangeKeyValue. [Електронний ресурс]. – 2016. – Режим доступу: https://docs.microsoft.com/en-us/cpp/atl/reference/cregkey-class?view=msvc-160#notifychangekeyvalue - Дата доступу: квітень 2021.

15. Сервис Windows. [Електронний ресурс]. – 2010. – Режим доступу: https://dic.academic.ru/dic.nsf/ruwiki/1145926 - Дата доступу: квітень 2021.

16. Creating Named Shared Memory. [Електронний ресурс]. – 2018. – Режим доступу: https://docs.microsoft.com/en-us/windows/win32/memory/creating-named-shared-memory - Дата доступу: квітень 2021.

17. Attach to running processes with the Visual Studio debugger. [Електронний ресурс]. – 2020. – Режим доступу: https://docs.microsoft.com/en-us/visualstudio/debugger/attach-to-running-processes-with-the-visual-studio-debugger?view=vs-2019 - Дата доступу: квітень 2021.

18. АНАЛІЗ СПОСОБІВ ПОБУДОВИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ. [Електронний ресурс]. – 2020. – Режим доступу: http://inmad.vntu.edu.ua/portal/static/7BA2FC95-52A4-4FC9-B802-AC99E6755C48.pdf - Дата доступу: квітень 2021.

19. WPP Software Tracing. [Електронний ресурс]. – 2017. – Режим доступу: https://docs.microsoft.com/en-us/windows-hardware/drivers/devtest/wpp-software-tracing - Дата доступу: квітень 2021.

20. TraceView. [Електронний ресурс]. – 2018. – Режим доступу: https://docs.microsoft.com/en-us/windows-hardware/drivers/devtest/traceview - Дата доступу: квітень 2021.

21. Перечисление EDataFlow. [Електронний ресурс]. – 2018. – Режим доступу:https://docs.microsoft.com/en-us/windows/win32/api/mmdeviceapi/ne-mmdeviceapi-edataflow - Дата доступу: травень 2021.

22. IAudioEndpoint :: GetLatency метод. [Електронний ресурс]. – 2018. – Режим доступу: https://docs.microsoft.com/ru-ru/windows/win32/api/audioengineendpoint/nf-audioengineendpoint-iaudioendpoint-getlatency - Дата доступу: травень 2021.

23. IAudioSystemEffects2:: GetEffectsList метод. [Електронний ресурс]. – 2018. – Режим доступу: https://docs.microsoft.com/en-us/windows/win32/api/audioenginebaseapo/nf-audioenginebaseapo-iaudiosystemeffects2-geteffectslist - Дата доступу: травень 2021.

ДОДАТОК А