

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ  
імені ІГОРЯ СІКОРСЬКОГО»

# СИСТЕМНІ ТЕХНОЛОГІЇ ДЛЯ ЗАСТОСУВАНЬ WINDOWS

## КОМП'ЮТЕРНИЙ ПРАКТИКУМ

### Навчальний посібник

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського  
як навчальний посібник для здобувачів ступеня бакалавра  
за освітніми програмами «Системи, технології та математичні методи кібербезпеки» і  
«Системи технічного захисту інформації» спеціальності 125 Кібербезпека та за освітньою  
програмою «Математичні методи моделювання, розпізнавання образів та комп'ютерного  
зору» спеціальності 113 Прикладна математика

Укладачі: Л. Ю. Гальчинський, В. В. Личик

Електронне мережне навчальне видання

Київ  
КПІ ім. Ігоря Сікорського  
2023

Рецензент *Стьопочкіна Ірина Валеріївна, доцент, к.т.н, кафедра інформаційної безпеки НН ФТІ «КПІ імені Ігоря Сікорського»*

Відповідальний редактор *Смирнов Сергій Анатолійович, доцент, к.ф.-м.н., с.н.с*

*Гриф надано Методичною радою КПІ ім. Ігоря Сікорського  
(протокол № 8 від 02.06.2023 р.)  
за поданням Вченої ради навчально-наукового фізико-технічного інституту  
(протокол № 7 від 15.05.2023 р.)*

Навчальний посібник призначений для здобувачів ступеня «Бакалавр» спеціальностей 125 «Кібербезпека» та 113 «Прикладна математика». Посібник містить теоретичний матеріал та практичні вказівки з прикладами лабораторних робіт, а також вимоги до порядку виконання лабораторних робіт з дисципліни «Системні технології для застосувань Windows». У посібнику приділено увагу теоретичній підготовці для сучасної технології системного програмування, здобуттю навичок створення системних програмних засобів оцінювання та забезпечення необхідного рівня захищеності інформації. Викладено технології спеціальних засобів Windows та їх застосування для кіберзахисту інформаційних систем.

Реєстр. № НП 22/23-705. Обсяг 4,54 авт. арк.

Національний технічний університет України  
«Київський політехнічний інститут імені Ігоря Сікорського»  
проспект Перемоги, 37, м. Київ, 03056  
<https://kpi.ua>

Свідоцтво про внесення до Державного реєстру видавців, виготовлювачів  
і розповсюджувачів видавничої продукції ДК № 5354 від 25.05.2017 р.

© КПІ ім. Ігоря Сікорського, 2023

## ЗМІСТ

|                                                                                                     |            |
|-----------------------------------------------------------------------------------------------------|------------|
| <b>Загальні вимоги до виконання комп'ютерного практикуму .....</b>                                  | <b>4</b>   |
| Обладнання і програмне забезпечення .....                                                           | 4          |
| Вимоги до виконання робіт .....                                                                     | 4          |
| Вимоги до оформлення звіту .....                                                                    | 5          |
| Загальні відомості про системні технології застосувань windows .....                                | 8          |
| <b>Лабораторна робота №1. СТВОРЕННЯ ТА ЗВ'ЯЗУВАННЯ<br/>ДИНАМІЧНИХ БІБЛІОТЕК DLL ДЛЯ WIN32 .....</b> | <b>10</b>  |
| 1.1 Теоретичні відомості .....                                                                      | 10         |
| 1.2. Порядок виконання роботи .....                                                                 | 25         |
| 1.3. Завдання .....                                                                                 | 25         |
| Контрольні питання .....                                                                            | 27         |
| <b>Лабораторна робота №2. СИСТЕМНІ ПРОГРАМНІ ЗАСОБИ<br/>ЗАБЕЗПЕЧЕННЯ БЕЗПЕКИ ОС WINDOWS .....</b>   | <b>28</b>  |
| 2.1 Теоретичні відомості .....                                                                      | 28         |
| 2.2. Порядок виконання роботи .....                                                                 | 59         |
| 2.3. Завдання .....                                                                                 | 60         |
| Контрольні запитання.....                                                                           | 63         |
| <b>Лабораторна робота № 3. ОСНОВИ ТЕХНОЛОГІЇ WINSOCKET .....</b>                                    | <b>64</b>  |
| 3.1. Теоретичні відомості .....                                                                     | 64         |
| 3.2. Порядок виконання роботи .....                                                                 | 81         |
| 3.3. Завдання .....                                                                                 | 89         |
| Контрольні запитання.....                                                                           | 92         |
| <b>Лабораторна робота №4. ОСНОВИ СОМ-ТЕХНОЛОГІЙ .....</b>                                           | <b>93</b>  |
| 4.1. Теоретичні відомості .....                                                                     | 93         |
| 4.2. Порядок виконання лабораторної роботи .....                                                    | 105        |
| 4.3. Завдання .....                                                                                 | 107        |
| Контрольні запитання.....                                                                           | 109        |
| <b>Література .....</b>                                                                             | <b>110</b> |

## **ЗАГАЛЬНІ ВИМОГИ ДО ВИКОНАННЯ КОМП'ЮТЕРНОГО ПРАКТИКУМУ**

### **Обладнання і програмне забезпечення**

Для виконання практикуму вам потрібен комп'ютер, на якому встановлена будь-яка операційна система із сімейства WINDOWS NT. Передбачається виконання робіт в комп'ютерній лабораторії ФТІ на системі WINDOWS. Індивідуальне завдання в рамках самостійної роботи цілком можна виконати на власному ноутбукі.

Якщо на Вашому комп'ютері хоча б операційна система WINDOWS 7, то можете не хвилюватися, що Ваших ресурсів недостатньо для виконання даного комп'ютерного практикуму.

Програмне забезпечення, на якому мають бути виконані завдання комп'ютерного практикуму є продуктом компанії Майкрософт Visual Studio (версії не раніше 2013 року), який містить інтегроване середовище розробки програмного забезпечення і його обов'язковий компонент Visual C++.

### **Вимоги до виконання робіт**

Методичні вказівки до кожної роботи складаються з теоретичної частини та завдання для виконання. Завдання для теоретичної частини може виконуватись без комп'ютера. Воно передбачає ознайомлення з теоретичним матеріалом і завданням до виконання, а також підготовку до виконання практичної частини роботи.

Основні теоретичні відомості містяться безпосередньо в методичних вказівках. Але не слід обмежуватись цією неповною інформацією. Зверніться до рекомендованої літератури. Прочитавши теоретичну частину, зверніть увагу на контрольні запитання, а ще краще задайте собі ці запитання. Ще більше

питань виникне при спробі вирішити поставлену у варіанті задачу. Тоді відповіді по ключовим словам можна знайти в Інтернеті (причому краще, якщо ключові слова сформулювати англійською).

Роботи виконуються і захищаються в комп'ютерній лабораторії в присутності викладача. При виконанні роботи студент повинен фіксувати все, що відбувається на екрані монітора комп'ютера. Вирази в кутових дужках означають, що Ви повинні підставити замість цього виразу разом із дужками значення цього виразу.

Наприклад, якщо Ваше прізвище Ковальчук, то замість студент\_<ваше прізвище> ви повинні написати (набрати на клавіатурі) студент\_Ковальчук.

У деяких випадках вам будуть запропоновані варіанти завдань, обирати з яких необхідно той, що відповідає номеру у списку тієї групи, яка сформована для даної дисципліни. Виконати необхідно усі пункти без виключення (якщо викладач не вказав інше), а в окремих пунктах може бути явно вказано, що слід обрати один з варіантів.

Перед захистом роботи студенти мають продемонструвати викладачу працюючі програми, які є практичним вирішенням визначеного у варіанті завдання. Для захисту необхідно надати оформлений звіт, вимоги до оформлення якого викладено далі. Також студент повинен дати правильні відповіді на додаткові запитання за матеріалами цієї лабораторної роботи. Зверніть увагу на те, що студент повинен володіти всім теоретичним і практичним матеріалом, а не тільки тією його частиною, яка безпосередньо входила до індивідуального завдання. За результатом захисту студент отримує оцінку за шкалою, описаною в силабусі дисципліни.

### **Вимоги до оформлення звіту**

Звіт повинен містити:

- титульну сторінку, назву роботи, тему і мету;

- номер варіанту і зміст індивідуального завдання;
- дату і місце виконання роботи, коротку характеристику системи, на якій було виконано роботу (клас комп'ютера, версію ОС тощо);
- протокол виконання роботи, де чітко позначено код програми, яка має вирішення поставленої задачі згідно варіанту;
- висновки по роботі, які повинні містити саме підсумки, які студент може зробити в результаті виконання цієї роботи. У висновках може бути зроблена оцінка якості реалізації системних механізмів для вирішення поставленої задачі, якості або гнучкості їх реалізації, зручності інтерфейсу тощо;

На кожній сторінці (в верхньому або нижньому колонтитулі) прізвище студента і номер групи.

Звіт з кожної роботи повинен мати титульний лист, оформлений згідно стандартних вимог, які наведено у прикладі.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ  
імені ІГОРЯ СІКОРСЬКОГО»  
Навчально-науковий фізико-технічний інститут  
Кафедра інформаційної безпеки

Дисципліна «Системні технології для застосувань Windows»  
Комп'ютерний практикум  
Робота №

Виконав : студент гр.<№групи><прізвище ініціали>

Перевірів: <посада> <прізвище ініціали викладача>

Київ – <поточний рік>

## **Загальні відомості про системні технології застосувань Windows**

Microsoft Windows, відома також як Windows і ОС Windows - операційна система комп'ютера (ОС), розроблена компанією Microsoft для запуску персональних комп'ютерів (ПК). За допомогою першого графічного інтерфейсу користувача (GUI) для IBM-сумісних ПК, ОС Windows стала домінувати на ринку ПК. Станом на 2022 рік приблизно 90 відсотків комп'ютерів працюють під керуванням операційної системи Windows.

Перша версія Windows, випущена в 1985 році, була просто графічним інтерфейсом, який пропонувався як розширення існуючої дискової операційної системи Microsoft, або MS-DOS. Частково на основі ліцензійних концепцій, які Apple Inc. використовувала для свого системного програмного забезпечення Macintosh, Windows вперше дозволила користувачам DOS візуально переміщатися по віртуальному робочому столу, відкриваючи графічні «вікна», що відображають вміст електронних папок і файлів, та керувати натисканням кнопок миші, а не вводячи команди та шляхи до каталогу у текстовій підказці. Головним завданням ОС Windows при її створенні було забезпечення підтримки офісної технології – зручної та інтуїтивно зрозумілої, яка і була реалізована в пакеті Microsoft Office. Який і зараз продовжує домінувати: як і в комерційному сегменті, так і у домашньому застосуванні. Проте Microsoft на цьому не зупинилась.

За відносно короткий період часу, на протязі життя одного покоління, ОС Windows набула рис універсальної операційної системи для різних комп'ютерних платформ, охопила всі можливі інформаційні технології сучасного світу: офісні, мережеві, обробки даних, сфери Інтернету, безпеки та інші. Такому вражаючому успіху в застосуванні ця операційна система завдячує багатому набору внутрішніх системних технологій, які мало відомі спільнотам прикладних програмістів і ще менше - широкому загалу чисельної армії користувачів.

Такі технології як: безпека об'єктів процесів Windows; асинхронний ввід/вивід Windows; багатопоточне програмування в умовах мереж; сокети; виклик віддаленої процедури (Remote Procedure Call); розподілені обчислення DCOM, грають ключову роль для розширеної функціональності застосувань Windows. Ці застосування, по суті будучи прикладними, для своєї розробки та модифікації потребують застосування методів системного програмування.

Метою даного практикума є набуття студентами практичного досвіду, для отримання навичок у цій сфері, яка належить до рівня програмних продуктів, які не визначають стандарти, але в середовищі експертів програмного забезпечення відомі як шар середнього рівня(middle ware). Програмування цього класу задач досить складне, бо вимагає не тільки досконалого вміння писати код мовою C/C++, але, також, розуміння специфіки прикладної задачі та володіння методами застосування функцій API в середовищі ОС WINDOWS. Кращі шанси успішно пройти цей курс мають студенти, які оволоділи поняттями та навичками системного програмування.

# ЛАБОРАТОРНА РОБОТА № 1

## СТВОРЕННЯ ТА ЗВ'ЯЗУВАННЯ ДИНАМІЧНИХ БІБЛІОТЕК DLL ДЛЯ WIN32

**Мета:** Освоєння методики розробки та використання динамічно зв'язуваних бібліотек в середовищі Win32 із застосуванням Visual Studio.

### 1.1. Теоретичні відомості

З самої своєї появи операційна система Windows використовувала бібліотеки динамічного компонування DLL (Dynamic Link Library), в яких містилися реалізації найбільш часто вживаних функцій. Всі спадкоємці Windows - NT і аж до новітніх версій, а також OS/2 - теж залежать від бібліотек DLL в плані забезпечення значної частини їх функціональності.

Ось деякі важливі файли dll, які користувач повинен знати для програмування:

- COMDLG32.DLL - Керує діалоговими вікнами.
- GDI32.DLL - Містить численні функції для малювання графіки, відображення тексту та керування шрифтами.
- KERNEL32.DLL - Містить сотні функцій для управління пам'яттю та різних процесів.
- USER32.DLL - Містить численні функції інтерфейсу користувача. Ці файли беруть участь у створенні програмних вікон та їх взаємодії між собою.

Нижче наведено кілька переваг наявності файлів DLL:

- Використовують менше ресурсів.
- Файли DLL не завантажуються в оперативну пам'ять разом з основною програмою.
- Вони не займають місця, якщо цього не потрібно.
- Коли потрібен файл DLL, він завантажується та запускається. Наприклад, поки користувач Microsoft Word редагує документ, файл DLL принтера не потрібний в оперативній пам'яті. Якщо користувач

вирішив надрукувати документ, то програма Word викликає завантаження та запуск файлу DLL принтера.

- Сприяє модульній архітектурі DLL і допомагає просувати розробку модульних програм. Це допомагає вам розробляти великі програми, які потребують декількох мовних версій, або програму, яка потребує модульної архітектури. Прикладом модульної програми є облікова програма, що має багато модулів, які можна динамічно завантажувати під час виконання.
- Полегшене розгортання та встановлення.

Коли функція в DLL потребує оновлення або виправлення, розгортання та встановлення, DLL не вимагає, щоб програма була повторно пов'язана з DLL. Крім того, якщо кілька програм використовують одну і ту ж DLL, то всі вони отримують користь від оновлення або виправлення. Ця проблема може виникати частіше, коли ви використовуєте сторонні DLL, які регулярно оновлюються або виправляються.

Програми та DLL можуть автоматично посилатися на інші DLL, якщо зв'язок DLL вказаний у розділі IMPORT файлу визначення модуля як частини компіляції. Крім того, ви можете явно завантажити їх за допомогою API-функції Windows LoadLibrary.

Спочатку ми обговоримо проблеми та вимоги, які слід враховувати під час розробки власних DLL-файлів.

Розглянемо ряд аспектів створення і використання бібліотек DLL:

- як статично підключати бібліотеки DLL;
- як динамічно завантажувати бібліотеки DLL;
- як створювати бібліотеки DLL;

*Динамічна бібліотека DLL* – це одна з можливостей повторного використання вже розроблених кодів. DLL – це спеціального вигляду виконуючий файл, з розширенням .dll, що використовується для зберігання функцій і ресурсів окремо від виконуючого файлу.

Бібліотеки DLL можуть зв'язуватися із застосунком двома шляхами:

- *статичним зв'язуванням;*
- *динамічним зв'язуванням.*

*Статичне зв'язування* означає, що DLL завантажується відразу, як тільки починає виконуватися застосунок, що буде її використовувати. Виклик функцій у застосунку, що використовує подібну DLL, майже не відрізняється від виклику будь-яких інших функцій. Деякими недоліками такого підходу можна вважати збільшення часу завантаження застосунка (адже крім виконуваного модуля застосунка в цей момент вантажаться й модулі відповідних DLL) і неможливість виконання застосунка користувачем, у якого немає відповідного файлу DLL. Ще одним недоліком статичного приєднання є те, що DLL займає пам'ять увесь час, протягом якого виконується застосунок, незалежно від того, чи викликаються в даному сеансі роботи з застосунком якісь функції бібліотеки, чи ні.

Статичне зв'язування потребує, щоб для DLL був створений спеціальний файл описів імпортованих функцій (*import library file*). Цей файл має те ж ім'я, що й відповідна DLL, і розширення *.lib*. Цей файл *.lib* повинен бути пов'язаний з застосунком на етапі компіляції.

*Динамічне зв'язування* відрізняється від статичного тим, що бібліотека DLL завантажується тільки в той момент, коли необхідно виконати якусь функцію, яка зберігається в ній. Потім цю бібліотеку можна вивантажити з пам'яті. Це забезпечує, звичайно, більш ефективне використання пам'яті. Але виклик відповідних функцій бібліотеки істотно ускладнюється, та й час виклику теж збільшується через необхідність завантажувати й вивантажувати бібліотеку.

Для виклику бібліотечної функції з бібліотеки, що зв'язується подібним чином, треба насамперед завантажити бібліотеку API-функцією *LoadLibrary* Windows. Потім за допомогою функції *GetProcAddress* треба одержати вказівник на функцію бібліотеки, що цікавить вас. Тільки після цього можна виконувати функцію. А потім за допомогою функції *FreeLibrary* треба вивантажити бібліотеку з пам'яті.

## Створення проєкту бібліотеки DLL в Visual Studio 2019

У рядку меню виберіть **Файл> Створити> Проєкт**, щоб відкрити діалогове вікно **Створення проєкту**.

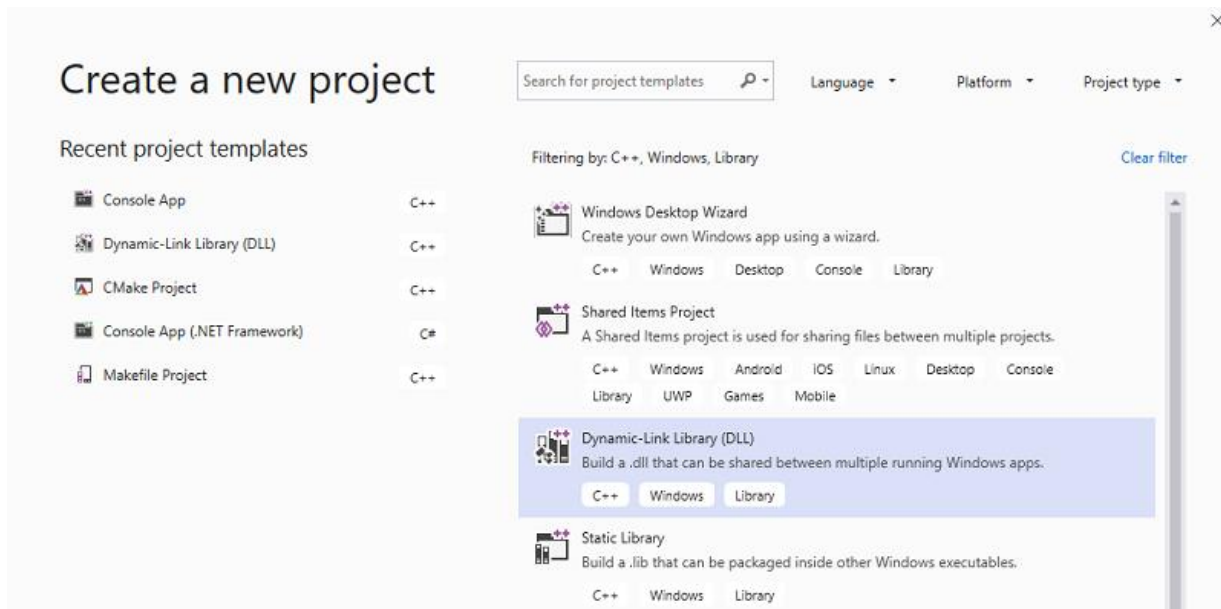


Рисунок 1.1 – Вікно завдання опцій при створенні DLL

1. У верхній частині діалогового вікна для параметра **Мова** встановіть значення C++, для параметра **Платформа** - значення **Windows**, а для параметра Тип **проєкту** - значення **Бібліотека**.
2. У відфільтрованому списку типів проєктів виберіть Бібліотека динамічного компонування (DLL), а потім натисніть кнопку Далі.
3. На сторінці **Налаштування** нового проєкту введіть назву в поле ім'я проєкту, щоб вказати ім'я проєкту. Залиште значення розташування і імені рішення за замовчуванням. Задайте рішення для створення нового рішення. Зніміть прапорець «розміщувати рішення і проєкт в тому ж каталозі», якщо він встановлений.
4. Натисніть кнопку **Створити**, щоб створити проєкт.
5. Після створення рішення можна переглянути створений проєкт і вихідні файли у вікні **Оглядач рішень** в **Visual Studio**.

## Використання DLL

Практично неможливо створити застосунок Windows, в якому не використовувалися б бібліотеки DLL. У DLL містяться всі функції Win32 API і незліченна кількість інших функцій операційних систем Win32. Взагалі кажучи, DLL - це просто набори функцій, зібрані в бібліотеки. Однак, на відміну від своїх статичних родичів (файлів. lib), бібліотеки DLL не приєднуються безпосередньо до виконуваних файлів за допомогою редактора зв'язків. У виконуваний файл занесена тільки інформація про їхнє місцезнаходження. У момент виконання програми завантажується вся бібліотека цілком. Завдяки цьому різні процеси можуть користуватися спільно одними і тими ж бібліотеками, що знаходяться в пам'яті. Такий підхід дозволяє скоротити обсяг пам'яті, необхідний для декількох застосунків, що використовують багато спільних бібліотек, а також контролювати розміри EXE-файлів.

Однак, якщо бібліотека використовується тільки одним застосунком, краще зробити її звичайною, статичною. Звичайно, якщо функції, що входять до її складу будуть використовуватися тільки в одній програмі, можна просто вставити в неї відповідний файл з вихідним текстом. Найчастіше проєкт підключається до DLL статично, або неявно, на етапі компонування.

Завантаженням DLL при виконанні програми керує операційна система. Однак, DLL можна завантажити і явно, або динамічно, в ході роботи програми. За замовчуванням Visual C++ включає бібліотеки імпорту для системних DLL.

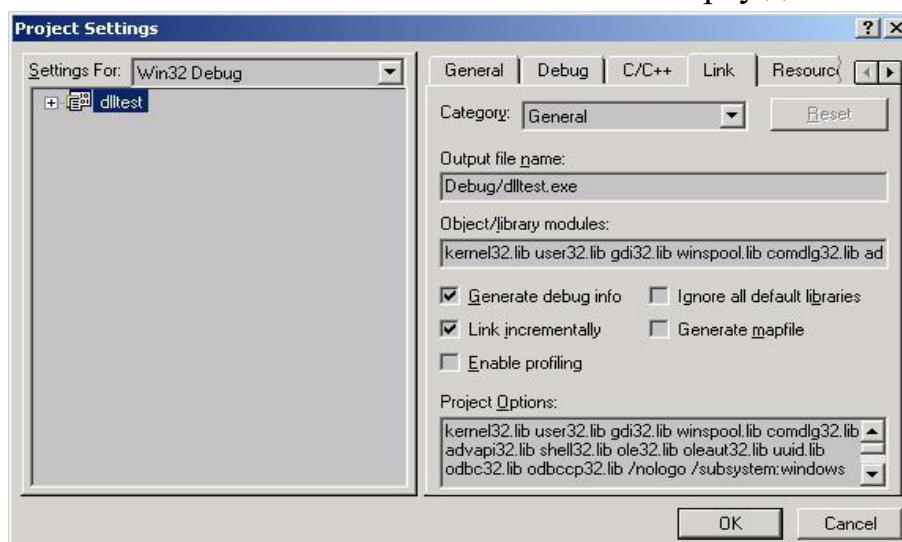


Рисунок 1.2 – Панель імпорту для системних DLL

## Бібліотеки імпортування

При статичному підключенні DLL ім'я .lib-файлу визначається серед інших параметрів редактора зв'язків у командному рядку або на вкладці "Link" діалогового вікна "Project Settings" середовища Developer Studio. Однак .lib-файл, який використовується при неявному підключенні DLL, - це не звичайна статична бібліотека. Такі .lib-файли називаються бібліотеками імпортування (import libraries). У них міститься не сам код бібліотеки, а тільки посилання на всі функції, що експортуються з файлу DLL, в якому все і зберігається. В результаті бібліотеки імпортування, як правило, мають менший розмір, ніж DLL-файли. До способів їх створення повернемося пізніше. А зараз розглянемо інші питання, що стосуються неявного підключення динамічних бібліотек.

## Узгодження інтерфейсів

При використанні власних бібліотек або бібліотек незалежних розробників доведеться звернути увагу на узгодження виклику функції з її прототипом.

За замовчуванням в Visual C ++ інтерфейси функцій узгоджуються за правилами C++. Це означає, що параметри заносяться в стек справа наліво, програмою, яка викликає та відповідає за їх видалення з стека при виході з функції і розширення її імені. Розширення імен (name mangling) дозволяє редактору зв'язків розрізняти перевантажені функції, тобто функції з однаковими іменами, але різними списками аргументів. Однак в старій бібліотеці C функції з розширеними іменами відсутні.

Хоча всі інші правила виклику функції в C ідентичні правилам виклику функції в C++, в бібліотеках C імена функцій не розширюються. До них тільки додається попереду символ підкреслення (\_).

Якщо необхідно підключити бібліотеку на C до застосунка на C++, всі функції з цієї бібліотеки доведеться оголосити як зовнішні у форматі C:

```
extern "C" int MyOldCFunction (int myParam);
```

Оголошення функцій бібліотеки зазвичай містяться в файлі заголовка цієї бібліотеки, хоча заголовки більшості бібліотек C не розраховані на

застосування в проєктах на C++. В цьому випадку необхідно створити копію файлу заголовка і включити в неї модифікатор `extern "C"` до оголошенню всіх використовуваних функцій бібліотеки. Модифікатор `extern "C"` можна застосувати і до цілого блоку, до якого за допомогою директиви `#include` підключити файл старого заголовка C. Таким чином, замість модифікації кожної функції окремо можна обійтися всього трьома рядками: `extern "C" {#include "MyCLib.h"}.`

У програмах для старих версій Windows використовувалися також угоди про виклик функцій мови PASCAL для функцій Windows API. У нових програмах слід використовувати модифікатор `winapi`, перетворений в `_stdcall`. Хоча це і не стандартний інтерфейс функцій C або C++, але саме він використовується для звернень до функцій Windows API. Однак зазвичай все це вже враховано в стандартних заголовках Windows.

### **Завантаження DLL, що неявно підключається**

При запуску програма намагається знайти всі файли DLL, неявно підключені до застосунка, і помістити їх в область оперативної пам'яті, яку займає даний процес. Пошук файлів DLL операційною системою здійснюється в наступній послідовності.

- Каталог, в якому знаходиться EXE-файл.
- Поточний каталог процесу.
- Системний каталог Windows.

Якщо бібліотеку DLL не знайдено, застосунок виводить діалогове вікно з повідомленням про її відсутність і шляхи, по яких здійснювався пошук. Потім процес відключається.

Якщо потрібна бібліотека знайдена, вона розміститься в оперативну пам'ять процесу, де і залишається до його закінчення. Тепер застосунок може звертатися до функцій, що містяться в DLL.

## Завантаження звичайної DLL

Перше, що необхідно зробити при динамічному завантаженні DLL, - це помістити модуль бібліотеки в пам'ять процесу. Дана операція виконується за допомогою функції `:: LoadLibrary`, що має єдиний аргумент - ім'я завантаження модуля. Відповідний фрагмент програми повинен виглядати так:

```
HINSTANCE hMyDll;  
::if ((hMyDll = :: LoadLibrary ( "MyDLL"))==NULL) {  
    /* Не вдалося завантажити DLL */  
}  
else { /* Програма має право користуватися функціями DLL через hMyDll */  
}
```

Стандартним розширенням файлу бібліотеки Windows вважає `.dll`, якщо не вказати інше розширення. Якщо в імені файлу зазначений і шлях, то тільки він буде використовуватися для пошуку файлу. В іншому випадку Windows буде шукати файл за тією ж схемою, що і в разі неявно підключених DLL, починаючи з каталогу, з якого завантажується `exe`-файл, і продовжуючи відповідно до значення `PATH`.

Коли Windows виявить файл, його повний шлях буде порівняний зі шляхом бібліотек DLL, вже завантажених даним процесом. Якщо виявиться тотожність, замість завантаження копії застосунку повертається дескриптор вже підключеної бібліотеки.

Якщо файл виявлений і бібліотека успішно завантажилася, функція `:: LoadLibrary` повертає її дескриптор, який використовується для доступу до функцій бібліотеки.

Перед тим, як використовувати функції бібліотеки, необхідно отримати їх адресу. Для цього спочатку слід скористатися директивою `typedef` для визначення типу вказівника на функцію і визначити змінну цього нового типу, наприклад:

```
// тип PFN_MyFunction буде оголошувати вказівник на функцію,  
приймаючи вказівник на символічний буфер і видавати значення типу int  
typedef int (WINAPI * PFN_MyFunction) (char *);
```

```
:: PFN_MyFunction pfnMyFunction;
```

Після цього слід отримати дескриптор бібліотеки, за допомогою якого і визначити адреси функцій, наприклад адреса функції з ім'ям MyFunction:

```
hMyDll = :: LoadLibrary("MyDLL");
```

```
pfnMyFunction= (PFN_MyFunction) :: GetProcAddress (hMyDll,  
"MyFunction");
```

```
:: int iCode = (* pfnMyFunction)( "Hello");
```

Адреса функції визначається за допомогою функції :: GetProcAddress, їй слід передати ім'я бібліотеки та ім'я функції. Останнє має передаватися в тому вигляді, в якому експортується з DLL.

Можна також послатися на функцію за порядковим номером, за яким вона експортується (при цьому для створення бібліотеки має використовуватися def-файл, про це буде розказано далі):

```
pfnMyFunction=(PFN_MyFunction)::GetProcAddress(hMyDll,  
MAKEINTRESOURCE (1));
```

Після завершення роботи з бібліотекою динамічного компонування, її можна вивантажити з пам'яті процесу за допомогою функції :: FreeLibrary:

```
::FreeLibrary(hMyDll);
```

### **Приклад звичайної DLL і способів завантаження**

Наведемо вихідний код, який динамічно підключається бібліотеки, яка називається MyDLL і містить одну функцію MyFunction, яка просто виводить повідомлення.

Спочатку в заголовку визначається макроконстанта EXPORT. Використання цього ключового слова при визначенні певної функції, яка динамічно підключається в бібліотеку і яка дозволяє повідомити компонувальнику, що ця функція доступна для використання іншими програмами, в результаті чого він заносить її в бібліотеку імпорту. Крім цього, така функція, як і віконна процедура, повинна визначатися за допомогою константи CALLBACK:

```
MyDLL.h
```

```
#define EXPORT extern "C" __declspec (dllexport)
```

```
EXPORT int CALLBACK MyFunction (char * str);
```

Файл бібліотеки також дещо відрізняється від звичайних файлів на мові С для Windows. У ньому замість функції WinMain є функція DllMain. Ця функція використовується для виконання ініціалізації, про що буде розказано пізніше. Для того, щоб бібліотека залишилася після її завантаження в пам'яті, і можна було викликати її функції, необхідно, щоб її значенням, яке повертається було TRUE:

```
MyDLL.c
#include
#include "MyDLL.h"
int WINAPI DllMain (HINSTANCE hInstance, DWORD fdReason, PVOID
pvReserved) {return TRUE;}
EXPORT int CALLBACK MyFunction (char * str)
{ MessageBox (NULL, str, "Function from DLL", MB_OK);
return 1;}
```

Після трансляції та компонування цих файлів з'явиться два файли - MyDLL.dll (сама динамічно підключається) і MyDLL.lib (її бібліотека імпорту).

### **Приклад неявного підключення DLL застосунком**

Наведемо тепер вихідний код простого застосунка, який використовує функцію MyFunction з бібліотеки MyDLL.dll:

```
#include
#include "MyDLL.h"
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE
hPrevInstance, LPSTR lpCmdLine, int nCmdShow) {int iCode = MyFunction (
"Hello");
return 0;
}
```

Ця програма виглядає як звичайна програм для Windows, чим вона по суті і є. Проте, слід звернути увагу, що у вихідний її текст крім виклику функції MyFunction з DLL-бібліотеки включені також заголовки цієї бібліотеки MyDLL.h. Також необхідно на етапі компонування застосунку

підключити до нього бібліотеку імпорту MyDLL.lib (процес неявного підключення DLL до виконуваного модулю).

Надзвичайно важливо розуміти, що сам код функції MyFunction не включається в файл MyApp.exe. Замість цього там просто є посилання на файл MyDLL.dll і посилання на функцію MyFunction, яка знаходиться в цьому файлі. Файл MyApp.exe вимагає запуску файлу MyDLL.dll.

Заголовок MyDLL.h включений в файл з вихідним текстом програми MyApp.c як і включений файл windows.h. Включення бібліотеки імпорту MyDLL.lib для компонування аналогічне включенню туди всіх бібліотек імпорту Windows. Коли програма MyApp.exe працює, вона підключається до бібліотеки MyDLL.dll динамічно як і до всіх стандартних бібліотек.

### **Приклад динамічного завантаження DLL застосунком**

Наведемо тепер повністю вихідний код простого застосунка, який використовує функцію MyFunction з бібліотеки MyDLL.dll, використовуючи бібліотеку динамічну завантажену:

```
#include
typedef int (WINAPI * PFN_MyFunction) (char *);
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE
hPrevInstance, LPSTR lpCmdLine, int nCmdShow)
{ HINSTANCE hMyDll;
  if ((hMyDll = LoadLibrary("MyDLL"))==NULL) return 1;
  PFN_MyFunction pfnMyFunction;
  pfnMyFunction=(PFN_MyFunction)GetProcAddress(hMyDll,"MyFunction";
  int iCode = (* pfnMyFunction) ( "Hello");
  FreeLibrary (hMyDll);
  return 0;
}
```

### **Створення DLL**

Тепер, познайомившись з принципами роботи бібліотек DLL в застосунках, розглянемо способи їх створення. При розробці застосунку

функції, до яких звертається кілька процесів, бажано розміщувати в DLL. Це дозволяє більш раціонально використовувати пам'ять в Windows.

Найпростіше створити новий проєкт DLL за допомогою майстра AppWizard, який автоматично виконує багато операцій. Для простих DLL, таких як були розглянуті, необхідно вибрати тип проєкту Win32 Dynamic-Link Library. Новому проєкту будуть присвоєні всі необхідні параметри для створення бібліотеки DLL. Файли вихідних текстів доведеться додавати до проєкту вручну.

### **Функція DllMain**

Більшість бібліотек DLL - просто колекції практично незалежних один від одного функцій, що експортуються в застосунки і використовуються в них. Крім функцій, призначених для експортування, в кожній бібліотеці DLL є функція DllMain. Ця функція призначена для ініціалізації та очищення DLL. Вона прийшла на зміну функцій LibMain і WEP, що застосовувався в попередніх версіях Windows. Структура найпростішої функції DllMain може виглядати, наприклад, так:

```
BOOL WINAPI DllMain (HANDLE hInst, DWORD dwReason, LPVOID  
IpReserved)  
{  
    BOOL bAllWentWell = TRUE;  
    switch (dwReason) {  
        case DLL_PROCESS_ATTACH: // Ініціалізація процесу.  
            break;  
        case DLL_THREAD_ATTACH: // Ініціалізація потоку.  
            break;  
        case DLL_THREAD_DETACH: // Очищення структур потоку.  
            break;  
        case DLL_PROCESS_DETACH: // Очищення структур процесу.  
            break; }  
    if (bAllWentWell) return TRUE;  
    else return FALSE;  
}
```

Функція DllMain викликається в кількох випадках. Причина її виклику визначається параметром dwReason, який може приймати одне з наступних значень.

При першому завантаженні бібліотеки DLL процесом викликається функція DllMain з dwReason, що дорівнює DLL\_PROCESS\_ATTACH. Кожен раз при створенні процесом нового потоку DllMain() викликається з dwReason, що дорівнює DLL\_THREAD\_ATTACH (крім першого потоку, тому що в цьому випадку dwReason дорівнює DLL\_PROCESS\_ATTACH).

Після закінчення роботи процесу з DLL функція DllMain викликається з параметром dwReason, що дорівнює DLL\_PROCESS\_DETACH. При знищенні потоку (крім першого) dwReason буде дорівнювати DLL\_THREAD\_DETACH.

Всі операції з ініціалізації та очищення для процесів і потоків, яких потребує DLL, необхідно виконувати на підставі значення dwReason, як було показано в попередньому прикладі. Ініціалізація процесів зазвичай обмежується виділенням ресурсів, спільно використовуваних потоками, зокрема завантаженням поділюваних файлів і ініціалізацією бібліотек. Ініціалізація потоків застосовується для налагодження режимів, властивих тільки даному потоку, наприклад для ініціалізації локальної пам'яті.

До складу DLL можуть входити ресурси, які не належать застосунку, що викликав цю бібліотеку. Якщо функції DLL працюють з ресурсами DLL, було б, очевидно, корисно зберегти де-небудь в зручному місці дескриптор hInst і використовувати його при завантаженні ресурсів з DLL. Вказівник IpReserved зарезервований для внутрішнього використання Windows. Отже, програма не має претендувати на нього. Можна лише перевірити його значення. Якщо бібліотека DLL була завантажена динамічно, цей вказівник буде дорівнювати NULL. При статичному завантаженні цей вказівник буде ненульовим.

У разі успішного завершення функція DllMain повинна повертати TRUE. У разі виникнення помилки повертається FALSE, і подальші дії припиняються.  
Зауваження: Якщо не написати власної функції DllMain(), компілятор підключить стандартну версію, яка буде просто повертати значення TRUE.

## Файли визначення модуля

Синтаксис файлів з розширенням .def в Visual C++ досить прямолінійний, головним чином тому, що складні параметри, що використовувалися в ранніх версіях Windows, в Win32 більше не застосовуються. Як стане зрозуміло з наступного простого прикладу, .def-файл містить ім'я і опис бібліотеки, а також список експортованих функцій:

```
MyDLL.def
LIBRARY "MyDLL"
DESCRIPTION 'MyDLL - приклад DLL-бібліотеки'
EXPORTS MyFunction @ 1
```

У рядку експорту функції можна вказати її порядковий номер, поставивши перед ним символ @. Цей номер буде потім використовуватися при зверненні до GetProcAddress (). Насправді компілятор присвоює порядкові номери всім експортним об'єктам. Однак спосіб, яким він це робить, частково непередбачуваний, якщо не привласнити ці номери явно. У рядку експорту можна використовувати параметр NONAME. Він забороняє компілятору включати ім'я функції в таблицю експорту DLL:

```
MyFunction @ 1 NONAME
```

Іноді це дозволяє заощадити багато місця у файлі DLL. Програми, що використовують бібліотеку імпортування для неявного підключення DLL, не «помітять» різницю, оскільки при неявному підключенні порядкові номери використовуються автоматично. Застосункам, що завантажують бібліотеки DLL динамічно, треба передавати в GetProcAddress порядковий номер, а не ім'я функції.

При використанні вищенаведеного def-файл опису експортованих функцій DLL-бібліотеки може бути, наприклад, не таким:

```
#define EXPORT extern "C" __declspec (dllexport)
EXPORT int CALLBACK MyFunction (char * str);
```

а таким:

```
extern "C" int CALLBACK MyFunction (char * str);
```

## Пам'ять DLL

На відміну від статичних бібліотек, які, по суті, стають частиною коду програми, бібліотеки динамічного компонування в 16-розрядних версіях Windows працювали з пам'яттю зовсім інакше. Під управлінням Win 16 пам'ять DLL розміщувалася поза адресним простором задачі. Розміщення динамічних бібліотек у глобальній пам'яті забезпечувало можливість спільного використання їх різними завданнями.

У Win32 бібліотека DLL розташовується в області пам'яті процесу, що завантажує її. Кожному процесу надається окрема копія "глобальної" пам'яті DLL, яка реініціалізується кожен раз, коли її завантажує новий процес. Це означає, що динамічна бібліотека не може використовуватися спільно, в загальній пам'яті, як це було в Win16.

І все ж, виконавши ряд хитрих маніпуляцій над сегментом даних DLL, можна створити загальну область пам'яті для всіх процесів, що використовують дану бібліотеку.

Нехай є масив цілих чисел, який повинен використовуватися всіма процесами, що завантажують дану DLL. Це можна запрограмувати таким чином:

```
#pragma data_seg ( ".myseg" )  
int sharedInts [10]; // інші змінні загального користування  
#pragma data_seg ()  
#pragma comment (lib, "msvcrt" "-SECTION: .myseg, rws");
```

Всі змінні, оголошені між директивами #pragma data\_seg (), розміщуються в сегменті .myseg. Директива #pragma comment () - не звичайний коментар. Вона дає вказівку бібліотеці системи C, щоб позначити новий розділ як дозволений для читання, записи і спільного доступу.

## Повна компіляція DLL

Якщо проект динамічної бібліотеки створений за допомогою AppWizard і .def-файл модифікований відповідним чином - цього достатньо. Якщо ж файли проекту створюються вручну або іншими способами без допомоги AppWizard, в командний рядок редактора зв'язків слід включити параметр / DLL. В

результаті замість автономного виконуваного файлу буде створена бібліотека DLL. Якщо в .def-файлі є рядок LIBRARY, вказувати явно параметр / DLL в командному рядку редактора зв'язків не потрібно.

## **1.2. Порядок виконання роботи**

2. Ознайомитись з теоретичними положеннями роботи.
3. Відлагодити і скомпілювати приклади, надані в тексті.
4. Виконати завдання згідно варіанту.
5. Показати викладачу працюючу відлагоджену програму згідно варіанту.
6. Скласти протокол виконаної роботи і представити на захист (Зміст протоколу: тема лабораторної роботи, постановка задачі, обґрунтування рішення, лістинг програми, висновки з роботи).

## **1.3. Завдання**

Суть завдання полягає в багатопотоковій обробці файлів. Розробити дві динамічні бібліотеки і застосунок, що їх використовують. Кожна з бібліотек повинна містити по дві функції обробки. Параметри функцій: ім'я файлу приймача, ім'я файлу джерела. Для виконання функцій першої бібліотеки використовувати явний виклик, для виклику функцій другої бібліотеки використовувати неявний виклик.

У динамічній бібліотеці реалізувати хоча б одну експортовану функцію. Можна використовувати або директиви компілятора `__declspec`, або файли компоувальника для визначення модуля \*.def. У виконуваній програмі потрібно статично (або динамічно) імпортувати функцію-приклад з dll і викликати її. Додаткова вимога: експорт / імпорт функцій має бути незалежним від застосовуваного компілятором декорування імен (name mangling), щоб функції вашої DLL могли використовувати програми, написані на інших мовах (або створені іншими компіляторами).

Таблиця 1.1 – Варіанти завдань

| №  | Суть обробки                                                                                                                                                                                                                                                                                                                                                 | Особливості обробки                                                                                                                                                                                          |
|----|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1. | Зчитати дані з файлу, в якому блоками записані текстові та числові дані. З цих даних вибрати числові дані та сформувати матрицю $M \times N$ .                                                                                                                                                                                                               | Написати код для обчислення суми елементів квадратної матриці, які розташовані нижче головної діагоналі. З її допомогою знайти максимальне значення такої суми в $n$ матрицях.                               |
| 2. | Зчитати дані з файлу, в якому блоками записані текстові та числові дані. З цих даних вибрати числові дані та сформувати матрицю $M \times N$ .                                                                                                                                                                                                               | Написати код, що перевіряє чи є від'ємні елементи у вказаному рядку двовимірного масиву. Видалити з масиву всі рядки з від'ємними елементами. Видалений рядок заповнюється 0 і переноситься в кінець масиву. |
| 3. | Перекодувати вхідний текстовий файл, що містить поблочно англійські, російські і українські літери в кодуванні Windows-1251, в файл, який містить текстове представлення UNICODE-кодування символів. Наприклад символ апострофа ' має бути виведений в новий файл як і той, що має значення 39 UNICODE, а російська буква «а» буде представлена &#39 як «а». | Перекодувати тільки українські букви.                                                                                                                                                                        |

## Продовження таблиці 1.1

|    |                                                                                                                                                                                                                                                                                                                                                              |                                                                                                                            |
|----|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------|
| 4. | Перекодувати вхідний текстовий файл, що містить поблочно англійські, російські і українські літери в кодуванні Windows-1251, в файл, який містить текстове представлення UNICODE-кодування символів. Наприклад символ апострофа ' має бути виведений в новий файл як і той, що має значення 39 UNICODE, а російська буква «а» буде представлена &#39 як «а». | Перекодувати всі символи, а потім в іншому файлі залишити тільки ті, що відображають цифри.                                |
| 5. | Провести потокову обробку вхідного текстового файлу, що містить поблочно англійські, російські і українські літери: вибрати з текстового файлу обсяг перших N кілобайт.                                                                                                                                                                                      | Написати код, при звертанні до якої з першого заданого рядка вилучається кожний символ, що належить ще й до другого рядка. |

## Контрольні питання

1. Що таке DLL?
2. Переваги використання DLL у робочих застосунках?
3. Як створити свою DLL-бібліотеку?
4. У чому полягає статичне зв'язування бібліотеки?
5. Які переваги й недоліки статичного зв'язування?
6. Поясніть принцип динамічного зв'язування бібліотек, із поясненням усіх використовуваних функцій?

## **ЛАБОРАТОРНА РОБОТА №2**

### **СИСТЕМНІ ПРОГРАМНІ ЗАСОБИ ЗАБЕЗПЕЧЕННЯ БЕЗПЕКИ ОС WINDOWS**

**Мета:** вивчення модель безпеки операційної системи Windows та отримання навичок практичного застосування програмних засобів для її безпеки.

#### **2.1. Теоретичні відомості**

##### **Класифікація захисту сімейства ОС Windows**

Захист конфіденційних даних від несанкціонованого доступу є найважливішим фактором успішного функціонування будь-якої багатокористувацької системи. ОС Windows не є винятком і вимоги до захисту об'єктів файлової системи, пам'яті, об'єктів ядра операційної системи внесли істотний внесок у процес її проєктування і реалізації.

Так, наприклад, починаючи з версії Windows NT/2000 була сертифікована система безпеки Windows за класом C2 за критеріями TSSEC («Помаранчева книга»). Для Windows 2000 подібна процедура сертифікації відбулась в 2002 році, але вже не за «Критеріями оцінювання захищених комп'ютерних систем» (TCSEC) Міністерства оборони США, а за міжнародним стандартом ISO 15408 (Common Criteria), що було введено в дію з 1999 року. Усі наступні версії Windows також успішно проходили сертифікацію за Common Criteria. Вимоги до операційної системи, захищеної за класом C2, включають:

- обов'язкову ідентифікацію та аутентифікацію всіх користувачів операційної системи. Під цим розуміється здатність операційної системи ідентифікувати всіх користувачів, які отримують санкціонований доступ до системи, і надання доступу до ресурсів тільки цим користувачам;
- розмежувальний контроль доступу - надання користувачам можливості користуватися захистом їхніх даних;
- системний аудит - здатність системи вести докладний аудит всіх дій, які виконуються користувачами і самою операційною системою;
- захист об'єктів від повторного використання;

- здатність системи запобігти доступ користувача до інформації ресурсів, з якими до цього працював інший користувач.

## Захист об'єктів системи WINDOWS

Система безпеки платформи Win32 ґрунтується на моделі безпеки для кожного користувача або групи користувачів. Кожен користувач, зареєстрований в системі, має власний обліковий запис, який містить персональну інформацію про користувача. Ці дані система використовує для перевірки автентичності користувача та для авторизації його при доступі до ресурсів домена. Після проходження процедури аутентифікації користувачеві присвоюється маркер доступу, що ідентифікує користувача, його групу, а також визначає доступні користувачеві привілеї в системі для доступу до ресурсів. При цьому кожному об'єкту системи, включаючи файли, принтери, мережні служби, контейнери Active Directory та інші, присвоюється дескриптор безпеки. Дескриптор безпеки об'єкта визначає права доступу до об'єкта і містить список контролю доступу (ACL - Access Control List), в якому явно визначається, яким користувачам дозволено виконувати ті чи інші дії з цим об'єктом. Дескриптор безпеки об'єкта також визначає, для яких подій має вестися аудит.

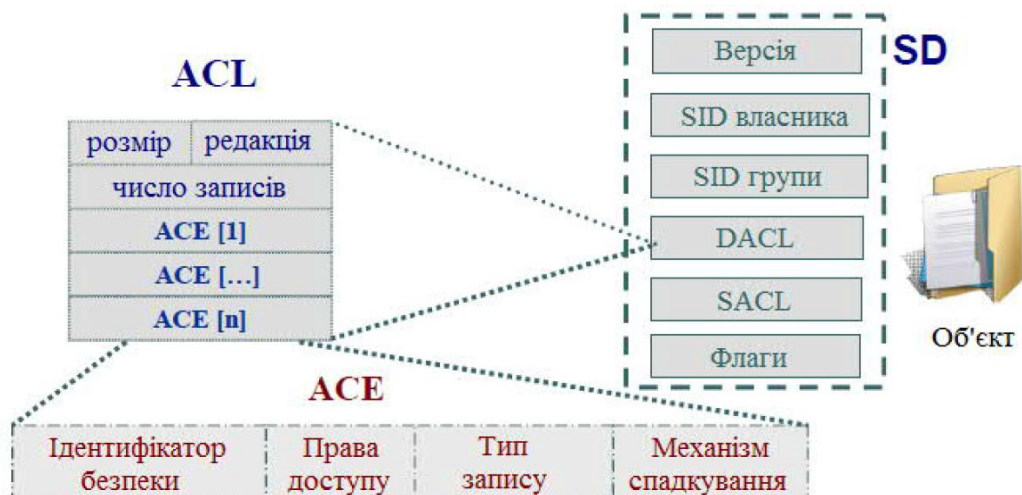


Рисунок 2.1 – Визначення прав доступу до об'єкту

Авторизація Win32 ґрунтується на зіставленні маркера доступу суб'єкта з дескриптором безпеки об'єкта. Керуючи властивостями об'єкта, адміністратори можуть встановлювати дозволи, призначати права володіння і відстежувати

доступ користувачів. Кожен дескриптор безпеки може містити списки двох типів. Системний список управління доступом (SACL - System Access Control List) дозволяє відстежувати права і обмеження, встановлені для об'єкта на системному рівні. У цей список можуть вносити зміни тільки ті користувачі, які володіють правами доступу на рівні системи.

Призначений для користувача список управління доступом (DACL - Discretionary Access Control List) дозволяє відстежувати права і обмеження, встановлені власником даного об'єкта. DACL може бути змінений користувачем, який вказаний як поточний власник об'єкта.

Обидва списки мають однакову структуру. Вони можуть не містити жодного запису або містити один, або кілька записів. Кожен запис (ACE - Access Control Entry) складається з трьох частин: в першій вказуються користувачі або групи, до яких стосується цей запис, в другій - права доступу, а третя інформує про те, чи надаються ці права або відбираються.

Список управління доступом містить набір елементів (Access Control Entries, ACE). У DACL кожен ACE складається з чотирьох частин: у першій вказуються користувачі або групи, до яких стосується цей запис, в другій - права доступу, а третя інформує про те, надаються ці права або відбираються. Четверта частина являє собою набір прапорів, що визначають, як даний запис буде успадковуватися вкладеними об'єктами (актуально, наприклад, для папок файлової системи, розділів реєстру).

Якщо список ACE в DACL порожній, до нього немає доступу ні у одного користувача (тільки у власника на зміну DACL). Якщо відсутній сам DACL в SD об'єкта - повний доступ до нього мають всі користувачі.

Якщо який-небудь потік запросив доступ до об'єкта, підсистема SRM здійснює перевірку прав користувача, що запустив потік, на даний об'єкт, переглядаючи його список DACL. Перевірка здійснюється до появи прав, що дозволяють всі запитані операції. Якщо зустрінеться правило, що забороняє хоча б на одну запитану операцію, доступ не буде надано.

Розглянемо приклад на рис.4. процес намагається отримати доступ до об'єкта з заданим DACL. У маркері процесу вказано SID користувача, який запустив його, а також SID груп, в які він входить. У списку DACL об'єкта присутні правила на читання для користувача з SID = 100, і на запис для групи з

SID = 205. Однак, в доступі користувачеві буде відмовлено, оскільки раніше зустрічається заборона запис для групи з SID = 201.

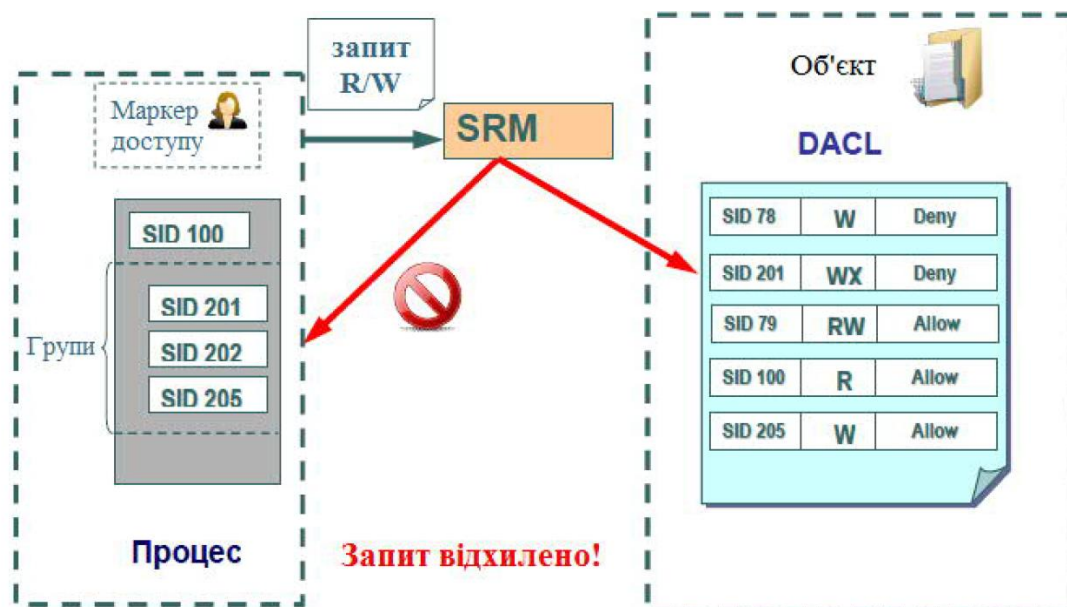


Рисунок 2.2 – Спроба процесу отримати доступ до об'єкта із заданим DACL

Дескриптори безпеки можуть бути представлені або в абсолютному, або у відносному форматі. Абсолютний формат передбачає запис дескриптора безпеки у пам'ять у вигляді структури вказівників і, тому, він більш зручний для оновлення вмісту дескриптора.

З іншого боку, оскільки вказівники містять посилання на області пам'яті, доступ до яких зазвичай забезпечується тільки комп'ютером-власником, абсолютний формат не дуже хороший для передачі дескрипторів по мережі (або запису на диск), тому що при цьому записуються не самі структури, а тільки набір вказівників. Тому існує інший формат дескрипторів безпеки, відповідно до якого структура з поточним вмістом всіх полів записується в єдиному безперервному блоці пам'яті. Такий формат більш зручний для передачі дескриптора. Він називається відносним форматом.

Засобами Windows список прав доступу модифікувати можна безпосередньо у віконному діалозі Windows у вікні властивостей об'єкта (див. рис.5), а можна програмно, використовуючи спеціальні API-функції. У лістингу 1 представлений фрагмент програми, що дозволяє програмно змінити права доступу до файлу для певного користувача.

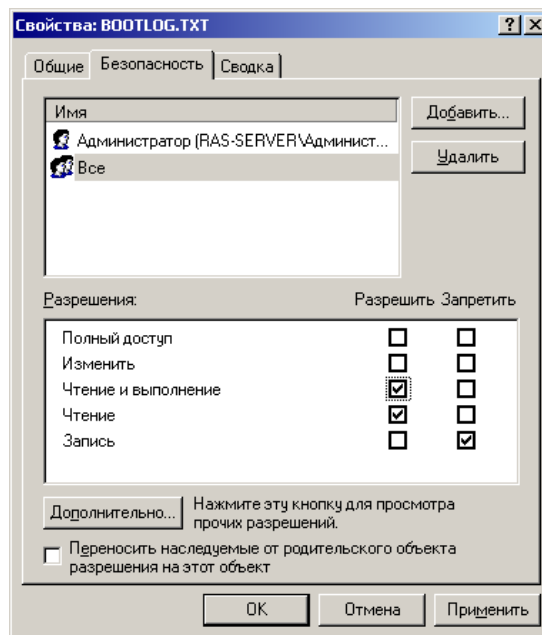


Рисунок 2.3 – Визначення прав доступу до файлу в Windows

Користувачу на програмному рівні дозволено проводити зміни у списку з правами доступу, який реалізується через набір функцій API. Спочатку необхідно ініціалізувати дескриптор безпеки за допомогою функції *InitializeSecurityDescriptor*.

Параметр *pSecurityDescriptor* повинен вказувати адресу структури *SECURITY\_DESCRIPTOR*. Ці структури є непрозорими для користувача і управляються спеціальними функціями.

Для параметра *dwRevision* слід встановлювати значення: *SECURITY\_DESCRIPTOR\_REVISION* *BOOL* *InitializeSecurityDescriptor* (*PSECURITY\_DESCRIPTOR* *pSecurityDescriptor*, *DWORD* *dwRevision*). Дескриптор безпеки, ініціалізований таким чином може бути змінений.

**Як працювати зі списками ACL, пов'язувати ACL з дескриптором безпеки і додавати ACL?**

1. Спочатку необхідно ініціалізувати структуру ACL. Оскільки безпосередній доступ до ACL здійснюватися не повинен, його внутрішня будова для нас мало цікавить.

2. Однак програма має надати буфер, який виступає в ролі ACL. Далі обробка вмісту виконується відповідними функціями

*BOOL InitializeAcl (PACL pAcl, DWORD cbAcl, DWORD dwAclRevision.*  
*pAcl* - адреса надається програмістом буфера розміром *cbAcl* байт. Значення параметра *dwAclRevision* слід встановлювати рівним *ACL\_REVISION*.

3. Далі ми маємо додати ACE в бажаному порядку, використовуючи функції *AddAccessAllowedAce* і *AddAccessDeniedAce*.

*BOOL AddAccessAllowedAce (PACL pAcl, DWORD dwAclRevision DWORD*  
*dwAccessMask, PSID pSid);*

*BOOL AddAccessDeniedAce (PACL pAcl, DWORD dwAclRevision, DWORD*  
*dwAccessMask, PSID pSid).*

Права, які надаються або в яких відмовляють користувачу або групі, які можуть бути ідентифіковані даними SID, визначаються маскою доступу (*dwAccessMask*).

3. Останнє, що потрібно зробити - це зв'язати ACL з дескриптором безпеки. Спочатку необхідно ініціалізувати структуру ACL. Оскільки безпосередній доступ до ACL небажаний, програма має надати буфер, який виступає в ролі ACL. Після чого йде маніпуляція відповідними API-функціями.

*BOOL InitializeAcl (PACL pAcl, DWORD cbAcl, DWORD dwAclRevision.*

*pAcl* - адреса надається програмістом буфера розміром *cbAcl* байт. Як визначити розмір ACL розглянемо пізніше, але для більшості цілей розміру 1 Кбайт буде цілком достатньо. Значення параметра *dwAclRevision* слід встановлювати рівним *ACL\_REVISION*. Далі ми маємо додати ACE в бажаному порядку, використовуючи функції: *AddAccessAllowedAce* і *AddAccessDeniedAce*.

*BOOL AddAccessAllowedAce (PACL pAcl, DWORD dwAclRevision DWORD*  
*dwAccessMask, PSID pSid)*

*BOOL AddAccessDeniedAce (PACL pAcl, DWORD dwAclRevision, DWORD*  
*dwAccessMask, PSID pSid).*

Параметр *pAcl* вказує на ту ж структуру ACL, яка була ініціалізована функцією *InitializeACL*, а параметр *dwAclRevision* також слід встановлювати рівним *ACL\_REVISION*. Параметр *pSid* вказує на SID, наприклад на той, який був отриманий за допомогою функції *LookupAccountName*.

У разі розмежувального ACL для цього використовується функція *SetSecurityDescriptorDacl*.

*BOOL SetSecurityDescriptorDacl (PSECURITY\_DESCRIPTOR pSecurityDescriptor, BOOL bDaclPresent, PACL pAcl, BOOL fDaclDefaulted).*

При значенні параметра *bDaclPresent == TRUE*, в структурі *pAcl* присутній ACL. Якщо цей параметр дорівнює *FALSE*, то наступні два параметри, *pAcl* і *fDaclDefaulted*, ігноруються. Прапор *SE\_DACL\_PRESENT* структури *SECURITY\_DESCRIPTOR\_CONTROL* також встановлюється рівним значенню цього параметра.

Значення *FALSE* параметра *fDaclDefaulted* вказує на те, що ACL був згенерований програмістом. В іншому випадку ACL був отриманий з використанням механізму, прийнятого за замовчуванням, наприклад, шляхом успадкування; разом з тим, для вказівки того, що є ACL, значення параметра має бути рівним *TRUE*. Прапор *SE\_DACL\_PRESENT* структури *SECURITY\_DESCRIPTOR\_CONTROL* також встановлюється рівним значенню цього параметра.

*Лістинг* – Програма управління правами доступу до об'єкту Windows - файлу

```
SID_NAME_USE SIDType;
PSECURITY_DESCRIPTOR pNewFileSD;
PACL pNewFileDACL;
char * filename = "c: \\ testfle.dat"; //ім'я файлу
char * username = "guest"; //Ім'я користувача
char szDomainName [250] = "VPI"; // назва домену
char UserSID [2048]; // ідентифікатор безпеки користувача
DWORD dwSIDLength = sizeof (UserSID);
DWORD dwDomainLength = sizeof (szDomainName);
DWORD dwNewACLSize;
```

```

// отримати SID (ідентифікатор безпеки) користувача (групи)
LookupAccountName ((LPSTR) NULL, username, UserSID, & dwSIDLength,
szDomainName, & dwDomainLength, & SIDType);
// Ініціалізувати новий SD (дескриптор безпеки)
InitializeSecurityDescriptor (pNewFileSD,
SECURITY_DESCRIPTOR_REVISION);
// отримати розмір ACL
dwNewACLSize = sizeof (ACL) + sizeof (ACCESS_ALLOWED_ACE) +
GetLengthSid (UserSID) - sizeof (DWORD);
// створити і проініціалізувати новий ACL
InitializeAcl (pNewFileDACL, dwNewACLSize, ACL_REVISION2);
// помістити новий ACE з дозволом на читання файлу для користувача в
кінець DACL
AddAccessAllowedAce (pNewFileDACL, ACL_REVISION2, GENERIC_READ,
& UserSID);
// записати DACL в новий SD файлу
SetSecurityDescriptorDacl (pNewFileSD, TRUE, pNewFileDACL, FALSE);
// встановити новий SD для файлу з ім'ям в filename
SetFileSecurity (filename, DACL_SECURITY_INFORMATION, pNewFileSD);

```

Розмежування прав доступу користувачів до файлів і папок в Win32 можливо тільки при використанні файлової системи NTFS. Ця файлова система забезпечує підтримку дескрипторів безпеки для елементів файлової системи і дозволяє визначати такі права на доступ: читання, запис, виконання, видалення, зміна прав доступу, отримання права власника, запис / читання атрибутів.

Локальні облікові записи користувачів в Win32 зберігаються в базі даних SAM (Security Account Manager). Вона розташовується в каталозі: \ % Systemroot% \ System32 \ ConfIg - в окремому файлі, який називається SAM. В облікових записах бази даних SAM знаходиться інформація про користувача імена і паролі, яка необхідна для ідентифікації і аутентифікації користувачів при їх інтерактивному вході в систему. Облікові записи створюються з використанням інструменту Local Users and Groups (Локальні користувачі та групи) з програми Computer Management (Керування комп'ютером). Інформація в SAM зберігається в зашифрованому вигляді. Пароль користувача спочатку

хешується алгоритмом MD4, потім результат хешування кодується алгоритмом DES, причому в ролі ключа шифрування використовується так званий відносний ідентифікатор користувача (Relative Identifier, RID), який представляє собою порядковий номер облікового запису даного користувача в базі даних SAM, який автоматично збільшується. При вході користувача в систему його пароль піддається аналогічним перетворенням, і якщо їх результат співпадає з вмістом SAM, користувач проходить аутентифікацію. Для запобігання доступу зломисника до бази даних SAM доступ до неї заблокований для всіх без винятку її користувачів.

### **Ідентифікація користувачів**

Для захисту даних Windows використовує такі основні механізми: аутентифікація і авторизація користувачів, аудит подій в системі, шифрування даних, підтримка інфраструктури відкритих ключів, вбудовані засоби мережевого захисту. Ці механізми підтримуються такими підсистемами Windows як LSASS (Local Security Authority Subsystem Service, підсистема локальної аутентифікації), SAM (Security Account Manager, диспетчер локальних записів безпеки), SRM (Security reference Monitor, монітор стану захисту), Active Directory (служба каталогів), EFS (Encrypting File System, шифруюча файлова система) і ін.

Захист об'єктів і аудит дій з ними в ОС Windows організовані на основі вибіркового (дискреційного) доступу, коли права доступу (читання, запис, видалення, зміна атрибутів) суб'єкта до об'єкта задається явно в спеціальній матриці доступу. Для збільшення матриці користувачі можуть об'єднуватися в групи. При спробі суб'єкта (одного з потоків процесу, запущеного від його імені) отримати доступ до об'єкта вказуються, які операції користувач збирається виконувати з об'єктом. Якщо подібний тип доступу дозволений, потік отримує описувач (дескриптор) об'єкта і всі потоки процесу можуть виконувати операції з ним. Подібна схема доступу, очевидно, вимагає аутентифікації кожного користувача, який отримує доступ до ресурсів і його надійну ідентифікацію в системі, а також механізмів опису прав користувачів і груп користувачів в системі, опису та перевірки дискреційних прав доступу користувачів до об'єктів.

Всі діючі в системі об'єкти (користувачі, групи, локальні комп'ютери, домени) ідентифікуються в Windows не за іменами (унікальність яких не завжди вдається досягти), а за ідентифікаторами захисту (Security Identifiers, SID).

SID являє собою числове значення змінної довжини наступного формату: **S - R - I - S0 - S1 - ... - Sn – RID**, де **S** - незмінний ідентифікатор рядкового подання SID; **R** - рівень ревізії (версія). На сьогодні 1. **I** - (identifier-authority) ідентифікатор повноважень. Являє собою 48-бітний рядок, що ідентифікує комп'ютер або мережу, який (яка) видав SID об'єкту.

Можливі значення:

- 0 (*SECURITY\_NULL\_SID\_AUTHORITY*) - використовуються для порівнянь, коли невідомі повноваження ідентифікатора;
- 1 (*SECURITY\_WORLD\_SID\_AUTHORITY*) - застосовуються для конструювання ідентифікаторів SID, які представляють всіх користувачів. Наприклад, ідентифікатор SID для групи Everyone (Всі користувачі) - це S-1-1-0;
- 2 (*SECURITY\_LOCAL\_SID\_AUTHORITY*) - використовуються для побудови ідентифікаторів SID, які представляють користувачів, які входять на локальний термінал;
- 5 (*SECURITY\_NT\_AUTHORITY*) - сама операційна система. Тобто, даний ідентифікатор завантажений комп'ютером або доменом.

Sn - 32-бітові коди (кількість 0 і більше) субагентів, яким було передано право видати SID. Значення перших підлеглих повноважень загальновідомі.

Вони можуть мати значення: • 5 - ідентифікатори SID присвоюються сеансам реєстрації для видачі прав будь-якого застосунку, що запускається під час певного сеансу реєстрації. У таких ідентифікаторів SID перші підлеглі повноваження встановлені як 5 і приймають форму S-1-5-5-x-y;

• 6 - коли процес реєструється як служба, він отримує спеціальний ідентифікатор SID в свій маркер для позначення даної дії. Цей ідентифікатор SID має підлеглі повноваження 6 і завжди буде S-1-5-6;

• 21 (*SECURITY\_NT\_NON\_UNIQUE*) - позначають ідентифікатор SID користувача і ідентифікатор SID комп'ютера, які не є чимось унікальним в глобальному масштабі;

- 32 (*SECURITY\_BUILTIN\_DOMAIN\_RID*) - позначають вбудовані ідентифікатори SID. Наприклад, відомий ідентифікатор SID для вбудованої групи адміністраторів S-1-5-32-544;
- 80 (*SECURITY\_SERVICE\_ID\_BASE\_RID*) - позначають ідентифікатор SID, який належить службі.

Решта підлеглих повноважень ідентифікатора спільно позначають домен або комп'ютер, який видав ідентифікатор SID. RID - 32-бітний відносний ідентифікатор. Він є ідентифікатором унікального об'єкта безпеки в області, для якої було визначено SID. Наприклад, 500 - позначає вбудований обліковий запис Administrator, 501 - позначає вбудований обліковий запис Guest, а 502 - RID для квитка на отримання квитків протоколу Kerberos.

При генерації SID Windows використовує генератор випадкових чисел, щоб забезпечити унікальність SID для кожного користувача. Для деякого довільного користувача SID може виглядати так: S-1-5-21-789336058-484763869-725345543-1003.

Визначеним користувачам та групам Windows видає характерні SID, що складаються з SID комп'ютера або домену та зумовленого RID. У таблиці 2.1 наведено перелік деяких загальновідомих SID.

Таблиця 2.1 – Загальновідомі SID Windows

| SID             | Назви          | Опис                                                                                                         |
|-----------------|----------------|--------------------------------------------------------------------------------------------------------------|
| S-1-1-0         | Всі            | Група, в яку входять всі користувачі                                                                         |
| S-1-5-2         | Мережа         | Група, в яку входять всі користувачі, що зареєструвались в системі з мережі                                  |
| S-1-5-7         | Анонімний вхід | Група, в яку входять усі користувачі, що ввійшли в систему анонімно                                          |
| S-1-5-домен-500 | Адміністратор  | Обліковий запис адміністратора системи.<br>За умовчанням тільки цей запис забезпечує повний контроль системи |
| S-1-5-домен-501 | Гість          | Обліковий запис користувача-гостя                                                                            |

Повний список загальновідомих SID можна подивитися в документації Platform SDK. Дізнатися SID конкретного користувача в системі, а також SID Груп, в які він включений, можна, використовуючи консольну команду:

*whoami: whoami /user /sid*

Після аутентифікації користувача процесом Winlogon, всі процеси, запущені від імені цього користувача будуть ідентифікуватися спеціальним об'єктом - маркером доступу (access token). Якщо процес користувача запускає дочірній процес, то його маркер успадковується, тому маркер доступу уособлює користувача для системи в кожному запущеному від його імені процесі. Основні елементи маркера представлені на таблиці 2.2

Таблиця 2.2 – Структура маркера доступу

| SID<br>користувача | SID1 ... SIDn<br>Ідентифікатори груп<br>користувача | DACL За<br>умовчанням | Привілеї | Інші<br>параметри |
|--------------------|-----------------------------------------------------|-----------------------|----------|-------------------|
|--------------------|-----------------------------------------------------|-----------------------|----------|-------------------|

Маркер доступу містить ідентифікатор доступу самого користувача і всіх груп, в які він включений. В маркер включений також DACL за замовчуванням - початковий список прав доступу, який приєднується до створюваних користувачем об'єктів. Ще одна важливий аспект для визначення прав користувача в системі частина маркера - список його привілеїв. Привілеї - це права довіреного об'єкта на вчинення будь-яких дій по відношенню до всієї системи. У таблиці 2.3 перераховані деякі привілеї, які можуть бути надані користувачеві.

Таблиця 2.3 – Привілеї, якими можуть бути наділені користувачі

| Ім'я та ідентифікатор привілею                                                  | Опис привілею                                                                                                                                                                                    |
|---------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Збільшення пріоритету диспетчеризація<br><i>SeIncreaseBasePriorityPrivilege</i> | Користувач, що володіє даним привілеєм може змінювати пріоритет диспетчеризація процесу за допомогою інтерфейсу Диспетчера завдань.                                                              |
| Закріплення сторінок в пам'яті<br><i>SeLockMemoryPrivilege</i>                  | Процес отримує можливість зберігати дані у фізичній пам'яті, не вдаючись до кешування даних у віртуальній пам'яті на диску.                                                                      |
| Управління аудитом і журналом безпеки<br><i>SeAuditPrivilege</i>                | Користувач отримує можливість вказувати параметри аудиту доступу до об'єкту для окремих ресурсів, таких як файли, об'єкти Active Directory і розділи реєстру.                                    |
| Оволодіння файлами або іншими об'єктами<br><i>SeTakeOwnershipPrivilege</i>      | Користувач отримує можливість ставати власником будь-яких об'єктів безпеки системи, включаючи об'єкти Active Directory, файли і папки NTFS, принтери, розділи реєстру, служби, процеси і потоки. |
| Завершення роботи системи<br><i>SeShutdownPrivilege</i>                         | Завершення роботи системи.                                                                                                                                                                       |
| Обхід перехресної перевірки<br><i>SeChangeNotifyPrivilege</i>                   | Використовується для обходу перевірки дозволів для проміжних каталогів при проході багаторівневих каталогів.                                                                                     |

Управління привілеями користувачів здійснюється в сегменті «Групова політика», розділ Конфігурація Windows / Локальні політики / Призначення прав користувача. Щоб подивитися привілеї користувача, можна також використовувати команду: *whoami / all*.

Інші параметри маркера носять інформаційний характер і визначають, наприклад, яка підсистема створила маркер, унікальний ідентифікатор маркера, час його дії. Необхідно також відзначити можливість створення обмежених маркерів (*restricted token*), які відрізняються від звичайних тим, що з них видаляються деякі привілеї і його SID-ідентифікатори перевіряються тільки на заборонні правила. Створити обмежений маркер можна програмно, використовуючи API-функцію *CreateRestrictedToken*, а можна запустити процес з обмеженим маркером, використовуючи пункт контекстного меню Windows "Запуск від імені..." і обравши пункт "Захистити комп'ютер від несанкціонованих дій цієї програми" (дивись рисунок 2.4).

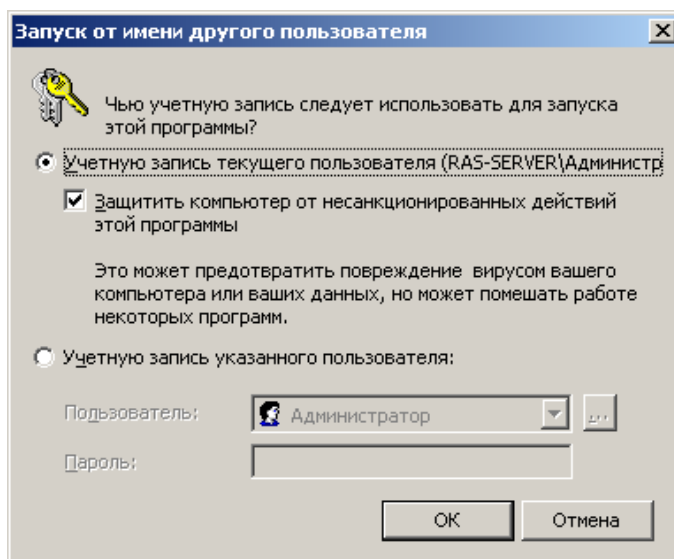


Рисунок 2.4 – Запуск процесу з обмеженим маркером

Обмежені маркери використовуються для процесів, що підміняють клієнта і виконують небезпечний код. Маркер доступу може бути створений не тільки при первинному вході користувача в систему. Windows надає можливість запуску процесів від імені інших користувачів, створюючи для цих процесів відповідний маркер. Для цих цілей можна використовувати:

1. Віконний інтерфейс, при виборі пункту контекстного меню: "Запуск від імені ...";

2. Консольну команду runas: *runas/user: ім'я користувача program*, де ім'я\_користувача - ім'я облікового запису користувача, який буде використаний для запуску програми в форматі користувач@домен або домен / користувач;

*program* - команда або програма, яка буде запущена за допомогою облікового запису, зазначеного в параметрі */user*. У будь-якому варіанті запуску процесу від імені іншого облікового запису необхідно задати її пароль.

3. API-функції *CreateProcessAsUser*, *CreateProcessWithLogon*;

Приклад коду використання *CreateProcessAsUser* для Windows 7:

```
PROCESS_INFORMATION pi;
STARTUPINFO si;
BOOL bResult = FALSE;
DWORD dwSessionId;
HANDLE hUserToken;

// Log the client on to the local computer.
dwSessionId = WTSGetActiveConsoleSessionId();
WTSQueryUserToken(dwSessionId,&hUserToken);
ZeroMemory(&si, sizeof(STARTUPINFO));
si.cb= sizeof(STARTUPINFO);
si.lpDesktop = L"winsta0\\default";
ZeroMemory(&pi, sizeof(pi));
LPVOID pEnv =NULL;
if(CreateEnvironmentBlock(&pEnv,hUserToken,TRUE)){
}
else
pEnv=NULL;
bResult = CreateProcessAsUser(
hUserToken, // client's access token
L"C:\\Windows\\System32\\cmd.exe", // file to execute
L"", // command line
NULL, // pointer to process SECURITY_ATTRIBUTES
NULL, // pointer to thread SECURITY_ATTRIBUTES
```

```

FALSE, // handles are not inheritable
CREATE_UNICODE_ENVIRONMENT|HIGH_PRIORITY_CLASS, // creation flags
pEnv, // pointer to new environment block
NULL, // name of current directory
&si, // pointer to STARTUPINFO structure
&pi // receives information about new process
);
//Perform All the Close Handles tasks
DestroyEnvironmentBlock(pEnv);
CloseHandle(pi.hThread);
CloseHandle(pi.hProcess);
CloseHandle(hUserToken);

```

Функція *CreateProcessWithLogonW* створює новий процес і його первинний (головний) потік. Новий процес потім запускає заданий виконуваний файл в контексті системи безпеки, при цьому визначаються повноваження (користувач, домен і пароль). Вона може необов'язково завантажувати профіль користувача, наданий йому.

Функція *CreateProcessWithLogonW* походить на функцію *CreateProcessAsUser*, за винятком того, що програмі яка викликається не потрібно викликати функцію *LogonUser*, щоб підтвердити справжність користувача і отримати маркер доступу.

Синтаксис функції:

```

BOOL CreateProcessWithLogonW (
LPCWSTR lpUsername, // ім'я користувача
LPCWSTR lpDomain, // домен користувача
LPCWSTR lpPassword, // пароль користувача
DWORD dwLogonFlags, // опція входу в мережу
LPCWSTR lpApplicationName, // ім'я виконуваного модуля
LPWSTR lpCommandLine, // командний рядок
DWORD dwCreationFlags, // прапорці створення
LPVOID lpEnvironment, // блок нової конфігурації

```

*LPCWSTR lpCurrentDirectory, // ім'я поточної папки*

*LPSTARTUPINFOW lpStartupInfo, // інформація передумовки*

*LPPROCESS \_ INFORMATION lpProcessInfo // інформація про процес;*

## **Привілеї**

Привілей - це право облікового запису, такого як обліковий запис користувача або групи, виконувати різні локальні комп'ютерні операції, такі як вимкнення системи, завантаження драйверів пристроїв або зміна системного часу. Привілеї відрізняються від прав доступу двома особливостями:

1. Привілеї контролюють доступ до системних ресурсів та пов'язаних із системою завдань, тоді як права доступу контролюють доступ до захищених об'єктів.
2. Системний адміністратор призначає привілеї обліковим записам користувачів та груп, тоді як система надає або забороняє доступ до захищеного об'єкта на основі прав доступу, наданих в ACE, DACL об'єкта.

Кожна система має базу даних облікових записів, яка зберігає привілеї, що зберігаються в облікових записах користувачів та груп. Коли користувач входить у систему, система виробляє маркер доступу, який містить перелік привілеїв користувача, включаючи ті, які надані користувачеві або групам, до яких належить користувач. Зауважте, що привілеї поширюються лише на локальний комп'ютер. Обліковий запис домену може мати різні привілеї на різних комп'ютерах.

Коли користувач намагається виконати привілейовану операцію, система перевіряє маркер доступу користувача, щоб визначити, чи володіє користувач необхідними привілеями, і якщо так, то перевіряється, чи привілеї включені. Якщо користувач не виконає цих тестів, система не виконає операцію. Для визначення привілеїв, що містяться в маркері доступу, викличте функцію *GetTokenInformation()*, яка також вказує, які привілеї включені.

**Більшість привілеїв вимкнено за замовчуванням.** API Windows визначає набір рядкових констант, таких як *SE\_ASSIGNPRIMARYTOKEN\_NAME*, для ідентифікації різних привілеїв. Ці константи однакові у всіх системах і визначаються у *winnt.h*. Однак функції,

які отримують і коригують привілеї в маркері доступу, використовують тип LUID для ідентифікації привілеїв. Значення LUID для привілею можуть відрізнятися від одного комп'ютера до іншого та від одного завантаження до іншого, на тому ж комп'ютері.

Щоб отримати поточний LUID, який відповідає одній з констант рядків, використовуйте функцію *LookupPrivilegeValue ()*. Використовуйте функцію *LookupPrivilegeName ()* для перетворення LUID у відповідну йому постійну константу. Система надає набір відображуваних імен, які описують кожен із привілеїв. Вони корисні, коли потрібно відобразити опис привілею користувачеві. Можна скористатися функцією *LookupPrivilegeDisplayName ()*, щоб отримати рядок опису, що відповідає постійній рядку для привілею. Можна використовувати функцію *PrivilegeCheck ()*, щоб визначити, чи містить маркер доступу визначений набір привілеїв. Це корисно насамперед для серверних застосунків, що уособлюють себе за клієнта.

Системний адміністратор може використовувати адміністративні інструменти, такі як Диспетчер користувачів, для додавання або видалення привілеїв для облікових записів користувачів та груп. Адміністратори можуть програмно використовувати функції LSA для роботи з привілеями. Функції *LsaAddAccountRights ()* та *LsaRemoveAccountRights ()* додають або видаляють привілеї з облікового запису. Функція *LsaEnumerateAccountRights ()* перераховує привілеї, що зберігаються у визначеному обліковому записі. Функція *LsaEnumerateAccountsWithUserRight ()* перераховує облікові записи, які мають вказаний привілей. Інформація про структуру LUID наведена в наступній таблиці. Не маніпулюйте LUID безпосередньо. Програми повинні використовувати функції та структури для маніпулювання значеннями LUID.

### **Робота зі спеціальними привілеями**

Для коректного виконання деяких функцій потрібні спеціальні привілеї. У деяких випадках функцію можуть виконувати лише певні користувачі або члени певних груп. Найпоширеніша вимога - користувач є локальним адміністратором. Інші функції вимагають, щоб у акаунта користувача були включені певні привілеї. Щоб зменшити можливість несанкціонованого коду отримати контроль, система має працювати з найменшими необхідними

привілеями. Програми, які потребують виклику функцій, для яких потрібні спеціальні привілеї, можуть залишати систему відкритою для атаки з боку хакерів. Такі програми мають бути розроблені для запуску на короткий проміжок часу і повинні інформувати користувача про наслідки для безпеки.

## Робота з правами адміністратора

Перший крок у встановленні того, який тип облікового запису має працювати у вашій програмі, - це вивчити, якими ресурсами користуватиметься програма та які привілейовані API буде викликати.

У випадку, коли користувачеві справді необхідно надати захищений ресурс, можна піти двома шляхами:

1. Змінити рівень доступу відповідно рівню доступу клієнта.
2. Надати клієнту необхідний привілей.

Перший варіант значно підвищує рівень загроз від зловмисного проникнення. За другим варіантом можна надати привілеї, які потрібні вашій програмі з меншим впливом зловмисної атаки, скориставшись одним із наступних підходів:

1. Увійти під обліковим записом з меншими привілеями. Один із способів зробити це - за допомогою *PrivilegeCheck ()* визначити, які привілеї включені в маркер. Якщо доступні привілеї не є адекватними для поточної операції, можна вимкнути цей код і надати користувачу увійти в обліковий запис з більшими правами.
2. Ідентифікувати користувача, викликом *CredUIPromptForCredentials ()* (GUI) або *CredUICmdLinePromptForCredentials ()* (командний рядок), щоб отримати ім'я користувача та пароль .
3. Уособлювати (Impersonate) себе за користувача. Процес, який починається в умовах привілейованого облікового запису, як-от System, може себе уособлювати обліковий запис користувача, викликаючи *ImpersonateLoggedOnUser ()* або подібні імперсональні функції, тим самим знижуючи рівень привілеїв. Однак якщо виклик *RevertToSelf ()* вводиться в потік, процес повертається до початкових системних привілеїв.

4. Розбити на окремі модулі функції, які потребують дозволу адміністратора. Можна надати користувачеві ярлик, який виконує команду *RunAs*. Програмно можна налаштувати команду *RunAs* під ключем реєстру *AppId* для вашої програми.

Виклик API C++ для *LogonAsUser* виглядає так:

```
BOOL LogonUser (  
    LPTSTR lpszUsername,  
    LPTSTR lpszDomain,  
    LPTSTR lpszPassword,  
    DWORD dwLogonType,  
    DWORD dwLogonProvider,  
    PHANDLE phToken  
);
```

Наступний приклад показує частину коду, яка дозволяє увімкнути або вимкнути привілей у маркері доступу. Приклад викликає функцію *LookupPrivilegeValue ()*, щоб отримати LUID, який використовує локальна система для ідентифікації привілею. Потім приклад викликає функцію *AdjustTokenPrivileges ()*, яка або надає або відключає привілей, що залежить від значення параметра *bEnablePrivilege*.

```
BOOL SetPrivilege (  
    HANDLE hToken, // хендл маркера доступу  
    LPCTSTR lpszPrivilege, // ім'я привілею для включення / відключення  
    BOOL bEnablePrivilege // щоб увімкнути або вимкнути привілей  
)  
{  
    TOKEN_PRIVILEGES tp;  
    LUID luid;  
    lpszPrivilege = "Створення аудиту безпеки";  
    if (! LookupPrivilegeValue (  
        NULL, // привілей пошуку в локальній системі  
        lpszPrivilege, // привілей до пошуку
```

```

& luid)) // отримуює LUID привілей
{
    printf("LookupPrivilegeValue error: %u\n", GetLastError());
    return FALSE;
}
tp.PrivilegeCount = 1;
tp.Privileges[0].Luid = luid;
if(bEnablePrivilege)
    tp.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;
else
    tp.Privileges[0].Attributes = 0;

// Увімкнути привілей або відключити всі привілеї.
if(!AdjustTokenPrivileges(
    hToken,
    FALSE,
    &tp,
    sizeof(TOKEN_PRIVILEGES),
    (PTOKEN_PRIVILEGES) NULL,
    (PDWORD) NULL))
{
    printf("AdjustTokenPrivileges error: %u\n", GetLastError());
    return FALSE;
}

if(GetLastError() == ERROR_NOT_ALL_ASSIGNED)

{
    printf("The token does not have the specified privilege. \n");
    return FALSE;
}

return TRUE;
}

```

## Уособлення( Impersonate)

Важливим аспектом спілкування клієнт / сервер, крім аутентифікації та обміну повідомленнями, є здатність сервера визначати, чи повинен він обслуговувати запит клієнта. Велика кількість серверів працює в контексті системи і тому має набагато більше привілеїв та можливостей, ніж типовий клієнт, який вимагає послуги.

Windows надає API, *ImpersonateSecurityContext*, яка дозволяє серверу уособлювати собою контекст безпеки клієнта, а також повертатися до власного контексту безпеки (*RevertSecurityContext*) при завершенні обслуговування.

Наведений нижче приклад показує, як використовувати *ImpersonateSecurityContext* та *RevertSecurityContext* API:

```
//  
// Під час доступу до ресурсу від імені клієнта, нам це потрібно // видавати  
себе за клієнта, щоб зробити відповідну перевірку доступу.  
//  
SecurityStatus = (* SecurityInterface-> ImpersonateSecurityContext) (&  
SecurityContext);  
якщо (SecurityStatus! = SEC_E_OK)  
{  
    // У нас проблема...  
    // Цей контекст безпеки не є принаймні рівнем імперсонації.  
    // помилка повернення;  
}  
//  
// У цей момент викликаний потік знаходиться під маркером неможливості з  
обліковими даними клієнта  
//  
// Обробити запит ..  
//  
. . .  
//  
// Повернення до основного маркера, як тільки ми закінчимо.
```

```
//
SecurityStatus = (* SecurityInterface-> RevertSecurityContext) (& SecurityContext);
якщо (SecurityStatus! = SEC_E_OK)
{
    //
    // перевірити наявність помилок ...
    //
    помилка повернення;
}
//
// Потік сервера повернувся до початкового контексту.
//
```

## **Права доступу до файлів і директорій в операційній системі Windows**

Файли та директорії є об'єктами ядра, які мають специфіку постійного зберігання, тобто такі, що не руйнуються після закінчення роботи з ними. Надалі розглянемо програмні аспекти дискреційної політики Windows на прикладі роботи з файлами та директоріями. Дозволи NTFS дозволяють явно вказати, які користувачі і групи мають доступ до файлів і директорій, і які операції з вмістом цих файлів або директорій їм дозволено виконувати. Дозволи NTFS застосовні тільки до томів, які відформатовані з використанням файлової системи NTFS. Система безпеки NTFS ефективна незалежно від того, чи звертається користувач до файлу або до директорії на локальному комп'ютері або в мережі.

Дозволи, що встановлюються для директорій, відрізняються від дозволів, що встановлюються для файлів. Адміністратори, власники файлів або директорій і користувачі з дозволом «Повний доступ» (Full Control) мають право призначати дозволи NTFS користувачам і групам для управління доступом до цих файлів і директорій.

## Дозволи для директорій в NTFS

Нижче перераховані стандартні дозволи для директорій в NTFS, які можна встановити, і які типи доступу забезпечуються. Можна заборонити дозвіл для користувача або групи. Для повної заборони доступу користувача, або групи до директорії, відмовте в дозволі «Повний доступ» (Full Control).

Таблиця 2.4 – Стандартні розширення для директорій в NTFS

| Дозволи для директорій в NTFS                   | Права користувача                                                                                                                                                                                                                                                                 |
|-------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Читання (Read)                                  | Переглядати файли і директорії, а також список власників, дозволи і атрибути директорії [такі, як Тільки читання (Read-Only), Прихований (Hidden), Архівний (Archive) і Системний (System)].                                                                                      |
| Запис (Write)                                   | Створювати нові файли і директорії всередині директорії, змінювати атрибути директорії і переглядати власників та дозволи для директорії.                                                                                                                                         |
| Список вмісту директорії (List Folder Contents) | Переглядати імена файлів і директорій.                                                                                                                                                                                                                                            |
| Читання і виконання (Read & Execute)            | Переміщатися по структурі директорій в пошуках інших файлів або директорій, навіть якщо користувач не має роздільну здатність на доступ до перегляду директорій. Виконувати всі дії, право на які дають дозволи Читання (Read) і Список вмісту директорії (List Folder Contents). |
| Змінити (Modify)                                | Видаляти директорії і виконувати всі дії, право на які дають дозволи Запис (Write) і Читання і виконання (Read & Execute)                                                                                                                                                         |
| Повний доступ (Full Control)                    | Змінювати дозволи, змінювати власника, видаляти директорії та файли і виконувати дії, право на які дають всі інші дозволи NTFS для директорій.                                                                                                                                    |

Дозволи для файлів в NTFS керують доступом користувачів до файлів. Нижче перераховані стандартні дозволи для файлів в NTFS, які можна встановити, і які ними забезпечуються типи доступу.

В NTFS зберігається список управління доступом (access control list, ACL) для кожного файлу і директорії на томі NTFS. У цьому списку перераховані користувачі і групи, для яких встановлені дозволи для файлу або директорії, а також самі призначені дозволи. Щоб користувач отримав доступ до ресурсу, в ACL має бути запис - елемент списку управління доступом (access control entry, ACE), для цього користувача або групи, до якої він належить. Цей запис призначить запитуваний тип доступу [наприклад, Читання (Read)] користувачеві. Якщо в ACL немає відповідного ACE, то користувач не отримає доступ до ресурсу.

Таблиця 2.5 – Стандартні дозволи для файлів в NTFS

| Дозволи для файлів в NTFS            | Права користувача                                                                                                                       |
|--------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| Читання (Read)                       | Переглядати файли, власників, дозволи і атрибути.                                                                                       |
| Запис (Write)                        | Перезаписувати файл, змінювати атрибути файлів і переглядати власників та дозволи.                                                      |
| Читання і виконання (Read & Execute) | Запускати застосунки і виконувати (Read & Execute) всі дії, право на які дає дозвіл Читання (Read).                                     |
| Змінити (Modify)                     | Змінювати і видаляти файл, а також виконувати всі дії, право на які дають дозволи Запис (Write) і Читання і виконання (Read & Execute). |
| Повний доступ (Full Control)         | Змінювати дозволи, змінювати власника і виконувати дії, право на які дають всі інші дозволи NTFS для файлів.                            |

## **Множинні дозволи NTFS**

Можна встановити кілька дозволів користувачеві і всім групам, членом яких він є. Для цього ви повинні мати уявлення про правила і пріоритети, за якими в NTFS призначаються і об'єднуються множинні дозволи і про успадкування дозволів NTFS.

Ефективні дозволи користувача для ресурсу - це сукупність дозволів NTFS, які ви призначаєте окремому користувачеві і всім групам, до яких він належить. Якщо у користувача є дозвіл Читання (Read) для директорії і він входить до групи, у якій є дозвіл Запис (Write) для тієї ж директорії, значить, у цього користувача є обидва дозволи.

## **Пріоритет дозволів для файлів над дозволами для директорій**

В NTFS дозволи для файлів мають більший пріоритет, ніж дозволи для директорій. Якщо у вас є дозвіл на доступ до файлу і право Обхід перехресної перевірки (Bypass Traverse Checking), то ви зможете скористатися доступом до цього файлу, навіть якщо у вас немає доступу до директорії, в якій міститься файл. Можна отримати доступ до тих файлів, для яких у вас є дозволи, скориставшись UNC-ім'ям файлу або локальним шляхом для відкриття файлу у відповідній програмі. Це можливо, навіть якщо директорія, в якій знаходиться файл, невидима і у вас немає відповідного дозволу для даної директорії. Іншими словами, якщо у вас немає дозволу на доступ до директорії, яка містить потрібний вам файл, для доступу до файлу вам необхідно мати право на Обхід перехресної перевірки (Bypass Traverse Checking) і знати повний шлях до файлу.

Можна заборонити доступ до файлу користувачеві або групі, хоча цей метод контролю ресурсів не можна назвати кращим. Заборона має більший пріоритет, ніж дозвіл на всіх рівнях, на які він поширюється. Навіть якщо у групи, в яку входить користувач є дозвіл на доступ до файлу або папки, заборона на доступ для користувача блокує всі наявні дозволи.

- Дозволи NTFS підсумовуються.
- Дозволи для файлів перезаписують дозволи для директорій.
- Заборона має більш високий пріоритет.

Можна скасувати успадкування дозволів, призначених для батьківської директорії , піддиректорії і файлами, що містяться в ній. Це означає, що директорії та файли залишаються без успадкованих дозволів, встановлених для батьківської директорії . Директорія , для якої ви скасували успадкування, стає новою батьківською папкою. Директорії і файли, що містяться в цій новій батьківській директорії, успадкують дозволи, встановлені для неї.

### **Встановлення дозволів NTFS**

За замовчуванням при форматуванні томи з використанням файлової системи NTFS дозвіл Повний доступ (FullControl) встановлюється для групи Bci (Everyone). У Windows обліковий запис Анонімний вхід (Anonymous Logon) не включена в групу Bci (Everyone). Адміністратори, користувачі з дозволом Повний доступ (Full Control) і власники файлів і директорій можуть встановлювати дозвіл для окремих користувачів і груп. Для встановлення або зміни дозволу NTFS для файлу або директорії налаштуйте параметри на вкладці Безпека (Security) діалогового вікна властивості файлу або директорії.

Таблиця 2.6 – Особливі дозволи

| Дозвіл                                                               | Призначення                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|----------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Повний доступ (Full Control)                                         | Повний доступ (Full Control) встановлює всі дозволи для користувача або групи.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| Огляд директорій / Виконання файлів (Traverse Folder / Execute File) | Дозвіл Огляд директорій (Traverse Folder) дозволяє або забороняє пересування по структурі директорій в пошуках інших файлів або директорій, навіть якщо користувач не має роздільної здатності на доступ до директорії, що переглядається. Дозвіл на огляд директорій не застосовується в тому випадку, якщо група або користувач має право Обхід перехресної перевірки (Bypass Traverse Checking), який встановлюється в оснащенні Групова політика (Group Policy). За замовчуванням група Всі (Everyone) наділена правом Обхід перехресної перевірки (Bypass Traverse Checking), тому, якщо ви хочете скористатися дозволом Огляд директорій (Traverse Folder), вам доведеться змінити групову політику. Дозвіл Огляд директорій (Traverse Folder) застосовується лише до директорій. Виконання файлів (Execute File) дозволяє або забороняє запуск виконуваних файлів (файлів застосунків). Дозвіл виконання файлів (Execute File) може бути застосовано тільки до файлів. |
| Зміст директорії / Читання даних (List Folder / Read Data)           | Зміст директорії (List Folder) дозволяє або забороняє перегляд імен файлів і піддиректорій, що містяться в папці. Цей дозвіл може бути застосовано лише до директорій. Дозвіл Читання даних (Read Data) дозволяє або забороняє перегляд вмісту файлів. Дозвіл Читання даних (Read Data) може бути застосовано тільки до файлів.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| Читання атрибутів (Read Attributes)                                  | Читання атрибутів (Read Attributes) дозволяє або забороняє перегляд атрибутів файлу або директорії. Атрибути визначаються файловою системою NTFS.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |

Продовження таблиці 2.6

| Дозвіл                                                              | Призначення                                                                                                                                                                                                                                                                                                                                                   |
|---------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Читання додаткових атрибутів (Read Extended Attributes)             | Читання додаткових атрибутів (Read Extended Attributes) дозволяє або забороняє перегляд додаткових атрибутів файлу або директорії. Додаткові атрибути визначаються програмами.                                                                                                                                                                                |
| Створення файлів / Запис даних (Create Files / Write Data)          | Створення файлів (Create Files) дозволяє або забороняє створення файлів у папці. Цей дозвіл може бути застосовано лише до директорій. Запис даних (Write Data) дозволяє або забороняє внесення змін до файлу і запис поверх наявного вмісту. Цей дозвіл може бути застосовано тільки до файлів.                                                               |
| Створення директорій / Дозапис даних (Create Folders / Append Data) | Створення директорій (Create Folders) дозволяє або забороняє створення директорій всередині директорії. Цей дозвіл може бути застосовано лише до директорій. Дозапис даних (Append Data) дозволяє або забороняє додавання даних в корінь файлу, але не дозволяє зміну, видалення або заміну наявних даних. Цей дозвіл може бути застосовано тільки до файлів. |
| Запис атрибутів (Write Attributes)                                  | Запис атрибутів (Write Attributes) дозволяє або забороняє зміну атрибутів файлу або директорії. Атрибути визначаються файловою системою NTFS.                                                                                                                                                                                                                 |
| Запис додаткових атрибутів (Write Extended Attributes)              | Запис додаткових атрибутів (Write Extended Attributes) дозволяє або забороняє зміну додаткових атрибутів файлу або директорії. Додаткові атрибути визначаються програмами.                                                                                                                                                                                    |
| Видалення піддиректорій і файлів (Delete Subfolders and Files)      | Видалення під-директорій і файлів (Delete Subfolders and Files) дозволяє або забороняє видалення під-директорій і файлів навіть при відсутності дозволу Видалення для даної директорії або файлу.                                                                                                                                                             |

Продовження таблиці 2.6

| Дозвіл                              | Призначення                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|-------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Видалення (Delete)                  | Видалення (Delete) дозволяє або забороняє видалення файлу або директорії. Якщо для файлу або директорії відсутній дозвіл Видалення (Delete), об'єкт все ж можна видалити при наявності дозволу Видалення піддиректорій і файлів (Delete Subfolders and Files) для батьківської директорії.                                                                                                                                                                            |
| Читання дозволів (Read Permissions) | Читання дозволів (Read Permissions) дозволяє або забороняє читання дозволів на доступ до файлу або папки.                                                                                                                                                                                                                                                                                                                                                             |
| Зміна дозволів (Change Permissions) | Зміна дозволів (Change Permissions) дозволяє або забороняє зміну дозволів на доступ до файлу або папки. Ви можете надати адміністраторам та користувачам можливість зміни дозволів для файлу або директорії без встановлення дозволу Повний доступ (Full Control) для даної директорії або файлу. Таким чином, адміністратор або користувач не зможуть видалити, змінити або записати дані в файл або папку, але зможуть встановити дозволи для файлу або директорії. |
| Зміна власника (Take Ownership)     | Зміна власника (Take Ownership) дозволяє або забороняє вступати у володіння файлом або папкою. Власник файлу або директорії завжди може змінювати дозволи на доступ до них незалежно від будь-яких дозволів, які захищають цей файл або папку.                                                                                                                                                                                                                        |
| Синхронізація (Synchronize)         | Синхронізація (Synchronize) дозволяє або забороняє очікування різними потоками файлів або директорій і синхронізацію їх з іншими потоками, які можуть займати їх. Цей дозвіл може бути застосовано тільки до програм, що виконуються в багатопотоковому режимі з декількома процесами                                                                                                                                                                                 |

## 2.2. Порядок виконання роботи

### Підготовче лабораторне завдання

На своїй робочій станції засобами утіліт Windows утворіть трьох користувачів один з яких має права адміністратора, два інші повинні мати права Everyone та Anonimus відповідно, причому вхід в систему Everyone та Anonimus має відбуватися через паролі.

### Основне лабораторне завдання

Планування і установка дозволів NTFS. Визначення встановлюються за замовчуванням дозволів NTFS для директорії 1.

1. Для початку роботи, використовуючи обліковий запис члена групи Адміністратори (Administrators), створіть облікові записи користувачів за наступними даними.

Таблиця 2.7 – Обліковий запис

| Обліковий запис користувача | Тип                       |
|-----------------------------|---------------------------|
| User                        | Обмежений обліковий запис |

2. В режимі Адміністратора створіть наступні директорії:

*C:\DisrPolitic; C:\DisrPolitic\Library.* з правами згідно варіанту *List Folder*.

3. В режимі Адміністратора напишіть програмний модуль, який дозволить визначити:

А) дозволи встановлених для директорії *C:\DisrPolitic; C:\DisrPolitic*.

Б) для Груп користувачів та окремих користувачів встановити загальні та особливі дозволи згідно варіанту.

4. Створити файл User1 в директорії *C:\DisrPolitic* дозволами згідно варіанту та User2 в директорії дозволами згідно варіанту *C:\DisrPolitic\Library*.
5. Увійдіть в систему, використовуючи обліковий запис User (Everyone або Anonimus) і напишіть код для роботи з файлами та директоріями згідно завдання варіанту.

## 2.3. Завдання

Таблиця 2.8 – Варіанти завдань

| № | 1                                            | 2                                                                             |                                                                                                              | 3                                                                                                                                            |                                                                                                                                                        | 4                                                                                                                                                                                                       |                                                                                                                                                            | 5                                                                     |
|---|----------------------------------------------|-------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------|
|   | П.2<br>Створення<br>початкових<br>директорій | П.3<br>Дозволи для груп та користувачів заданих<br>адміністратором(початкові) |                                                                                                              | П.4<br>Рівень дозволів доступу до об'єктів для<br>заданих адміністратором груп та<br>користувачів у програмі клієнта<br>(запит користувачів) |                                                                                                                                                        | П.6<br>Зміна дозволів(прав) для директорій та файлів,<br>наданих адміністратором на запит<br>користувачів                                                                                               |                                                                                                                                                            | П.8<br>Надання<br>привілеїв<br>(номер методу<br>надання<br>привілеїв) |
|   |                                              | файли                                                                         | директорії                                                                                                   | файли                                                                                                                                        | директорії                                                                                                                                             | файли                                                                                                                                                                                                   | директорії                                                                                                                                                 |                                                                       |
| 1 | Повний<br>доступ<br>2.1                      | Всі<br>користувачі<br>можуть читати<br>імена файлів<br>3.1.1                  | Всі користувачі повинні<br>мати можливість читати<br>імена файлів, власників, і<br>атрибути в папці<br>3.2.1 | Всі користувачі<br>можуть читати<br>імена файлів в<br>папці і зміст<br>файлів<br>DisrPolitic;<br>4.11                                        | Всі користувачі<br>повинні мати<br>можливість читати<br>імена файлів в папці<br>DisrPolitic, але не в<br>папці C:\DisrPolitic\<br>Library;<br>4.1.12   | Деякі користувачі<br>можуть читати імена<br>файлів в папці і зміст<br>файлів DisrPolitic,<br>решта користувачів<br>можуть читати імена<br>файлів в папці і читати<br>зміст файлів DisrPolitic<br>6.1.11 | Деякі користувачі<br>можуть читати<br>імена файлів в<br>папці і читати зміст<br>файлів DisrPolitic,<br>інші користувачі<br>читати зміст файлів<br>6.1.12   | 1                                                                     |
|   |                                              |                                                                               | Всі користувачі повинні<br>мати можливість читати<br>імена файлів, власників, і<br>атрибути в папці<br>3.2.1 | Всі користувачі<br>читати імена<br>файлів в папці<br>і читати зміст<br>файлів:<br>C:\DisrPolitic\<br>Library;<br>4.1.21                      | Всі користувачі<br>повинні мати<br>можливість читати<br>імена файлів і<br>атрибути в папці<br>DisrPolitic, але не в<br>папці C:\DisrPolitic;<br>4.1.22 | Деякі користувачі<br>можуть читати<br>імена файлів в<br>папці і читати зміст<br>файлів:<br>C:\DisrPolitic\<br>Library;<br>6.1.21                                                                        | Всі користувачі<br>повинні мати<br>можливість читати<br>імена файлів і<br>атрибути в папці<br>DisrPolitic, але не в<br>папці:<br>C:\DisrPolitic;<br>6.1.22 |                                                                       |

Продовження таблиці 2.8

|   |                                 |                                                                                        |                                                                         |                                                                                         |                                                                                                     |                                                                                                                   |                                                                                                                    |   |
|---|---------------------------------|----------------------------------------------------------------------------------------|-------------------------------------------------------------------------|-----------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------|---|
| 2 | Читання<br>2.2                  | Всі користувачі мають можливість переглядати та писати файли<br>3.2.1                  | Всі користувачі мають можливість переглядати та видаляти файли<br>3.2.2 | Всі користувачі мають можливість переглядати та писати файли<br>4.2.11                  | Всі користувачі мають можливість переглядати файли<br>4.2.21                                        | Деякі користувачі мають можливість переглядати та писати файли, всі інші користувачі можуть переглядати<br>6.2.11 | Всі користувачі мають можливість переглядати файли. Один з користувачів може переглядати та писати файли<br>6.2.21 | 2 |
|   |                                 |                                                                                        | Всі користувачі мають можливість переглядати та видаляти файли<br>3.2.2 | Всі користувачі мають можливість переглядати та писати файли в папці<br>4.2.12          | Всі користувачі мають можливість переглядати та видаляти файли в папці<br>4.2.22                    | Деякі користувачі мають можливість переглядати та писати файли в папці<br>6.2.12                                  | Деякі користувачі мають можливість переглядати та видаляти файли в папці<br>3.2.22                                 |   |
| 3 | Список вмісту директорії<br>2.3 | Всі користувачі мають можливість переглядати файли та писати, читати атрибути<br>3.3.1 | Переглядати файли, власників, дозволи і атрибути<br>3.3.2               | Всі користувачі мають можливість переглядати файли та читати атрибути в папці<br>4.3.11 | Всі користувачі мають можливість переглядати файли, власників, дозволи і атрибути в папці<br>3.3.21 | Деякі користувачі мають можливість переглядати файли та читати в папці<br>6.3.11                                  | Всі користувачі мають можливість переглядати файли, власників в папці<br>6.3.21                                    | 3 |
|   |                                 |                                                                                        | Переглядати файли, власників, дозволи і атрибути<br>3.3.2               | Всі користувачі мають можливість переглядати файли та писати в папці<br>4.3.12          | Всі користувачі мають можливість переглядати файли і атрибути в папці<br>4.3.22                     | Деякі користувачі мають можливість переглядати файли та писати в папці<br>6.3.12                                  | Всі користувачі мають можливість переглядати файли, і атрибути в папці<br>6.3.22                                   |   |

Продовження таблиці 2.8

|   |                |                                                                                                                             |                                                                                                                                                                                                                    |                                                                                                         |                                                                                                                                      |                                                                                                                                                    |                                                                                                                                                  |  |
|---|----------------|-----------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------|--|
| 4 | Змінити<br>2.4 | Всі користувачі мають можливість перезаписувати файл, змінювати атрибути файлів і переглядати власників та дозволи<br>3.4.1 | Видаляти директорії і виконувати всі дії, право на які дають дозволи запис<br>3.4.2                                                                                                                                | Всі користувачі мають можливість перезаписувати файл<br>4.4.11                                          | Всі користувачі мають можливість видаляти директорії і виконувати всі дії<br>4.4.21                                                  | Деякі користувачі мають можливість перезаписувати файл, один користувач має можливість змінювати атрибути файлів і переглядати власників<br>6.4.11 | Всі користувачі мають можливість видаляти директорії, один користувач має можливість виконувати всі дії, право на які дають дозволи запис 6.4.21 |  |
|   |                |                                                                                                                             | Видаляти директорії і виконувати всі дії, право на які дають дозволи запис<br>3.4.2                                                                                                                                | Всі користувачі мають можливість змінювати атрибути файлів і переглядати власників та дозволи<br>4.4.12 | Всі користувачі мають можливість виконувати всі дії, право на які дають дозволи запис<br>4.4.22                                      | Деякі користувачі мають можливість змінювати атрибути файлів і переглядати власників та дозволи<br>6.4.12                                          | Всі користувачі мають можливість виконувати всі дії, перезаписувати файл<br>6.4.22                                                               |  |
| 5 | Запис<br>2.5   | Всі користувачі мають можливість перезаписувати файл, змінювати атрибути файлів і переглядати власників та дозволи          | Всі користувачі мають можливість читати імена файлів, власників, створювати нові файли і директорії всередині директорії, змінювати атрибути директорії і переглядати власників та дозволи для директорії<br>3.5.2 | Всі користувачі мають можливість перезаписувати файл<br>4.5.11                                          | Деякі користувачі мають можливість читати імена файлів, власників, створювати нові файли і директорії всередині директорії<br>4.5.21 | Деякі користувачі мають можливість перезаписувати файл та копіювати файли<br>6.5.11                                                                | Деякі користувачі мають можливість читати імена файлів, власників, створювати нові файли і директорії всередині директорії<br>6.5.21             |  |

## Контрольні запитання

1. На чому ґрунтується реалізація політики доступу для Windows?
2. Як влаштована структура елементів Access Control Entries?
3. В чому різниця абсолютного і відносного формату дескрипторів безпеки?
4. Які є програмні засоби для змін у списку з правами доступу на програмному рівні?
5. Як працювати зі списками ACL, пов'язувати ACL з дескриптором безпеки і додавати ACL?
6. Як зв'язати ACL з дескриптором безпеки?
7. Які програмні засоби розмежування прав доступу користувачів до файлів і папок в Win32?
8. Які програмні механізми Windows використовує для захисту даних?
9. Опишіть структура маркера доступу.
10. Яким чином програмно можна встановити відповідний маркер доступу?
11. Опишіть особливості API-функцій *CreateProcessAsUser*, *CreateProcessWithLogon*.
12. Чим відрізняються привілеї від прав доступу?
13. Опишіть особливості використання *LookupPrivilegeValue ()*?
14. Опишіть програмні засоби Уособлення (Імперсоналізації)?

## ЛАБОРАТОРНА РОБОТА № 3

### ОСНОВИ ТЕХНОЛОГІЇ WINSOCKET

#### **Мета роботи:**

а) вивчити:

- можливості реалізації архітектури клієнт-сервер на основі інтерфейсу сокетів Windows Sockets API;
- типи сокетів TCP / IP;
- основні методики і API-функції для розробки мережеских застосунків з використанням Winsock.

б) Розробити консольний клієнт-серверний застосунок, на основі взаємодії потокових та дейтаграмних вінсокетів.

#### **3.1. Теоретичні відомості**

**Поняття сокета** Сокет (Socket - гніздо, роз'єм) - абстрактне програмне поняття, що використовується для позначення в прикладній програмі кінцевої точки каналу зв'язку з комунікаційним середовищем, утвореної обчислювальною мережею.

Поеднуючи разом два сокета, можна передавати дані між різними процесами (локальними і віддаленими). Реалізація сокетів забезпечує інкапсуляцію протоколів мережевого і транспортного рівнів.

Інтерфейс, який використовується застосунком при взаємодії з програмним забезпеченням транспортного протоколу, називається інтерфейсом прикладного програмування (Application Programming Interface - API). API інтерфейс визначає набір операцій, які можуть бути виконані застосунком при взаємодії з програмним забезпеченням протоколу.

Функції прикладного програмного інтерфейсу сокетів (Sockets API) забезпечують ідентифікацію кінцевих точок з'єднання, установку з'єднання, відправку повідомлень, очікування вхідних повідомлень, призвести до втрати з'єднання і обробку помилок.

## Протоколи і сімейства адрес

Найважливішою перевагою сокетів Windows є надання єдиного незалежного інтерфейсу мережевого програмування (Sockets API) для різних мережевих протоколів.

Платформи Win32 підтримують різноманітні мережеві стеки протоколів: TCP / IP, IPX / SPX, NetBIOS / SMB, AppleTalk, ATM, Infrared Sockets. Кожному з них відповідає своє сімейство адрес сокетів. Наприклад, TCP / IP відповідає сімейство адрес AF\_INET, IPX / SPX - AF\_NS, ATM - AF\_ATM і т.д.

Сімейство адрес - найважливіший параметр сокета. Він вказує на використовуваний в даний час протокол і обмежує застосування інших параметрів сокета. В цій роботі ми розглянемо адресацію сокетів тільки для стека протоколів TCP / IP, як найпоширенішого на сьогоднішній день.

Адреса сокету при використанні протоколів TCP / IP задає такий набір значень:

- номер мережі;
- номер кінцевого вузла;
- номер порту прикладної служби.

## Ініціалізація Winsock

Winsock – це бібліотека, що являє собою інтерфейс мережевого програмування, незалежного від транспортного протоколу. Вона реалізована версіями Winsock 1.1 та Winsock 2.2. Програма, написана для Winsock 1.1, зможе виконуватись і з бібліотекою Winsock 2.2.

Імена нових або оновлених API-функцій, додані в Winsock 2, починаються з префікса WSA. Ініціалізацію Winsock виконує функція WSASStartup:

*int WSASStartup( WORD VersionRequested, LPWSADATA lpWSADATA).*

## Параметри функції

VersionRequested – версія бібліотеки Winsock, яку необхідно завантажити. На сучасних платформах Windows використовується версія 2.2. Для

завантаження версії Winsock 2.2 потрібно вказати значення 0x0202, або макрос MAKEWORD(2,2). Верхній байт визначає застосунковий номер версії, нижній – основний.

lpWSAData – структура WSADATA, яка повертається після завершення виклику. Вона містить інформацію про версію Winsock, завантажену функцією WSASStartup. Структура WSADATA описується так:

```
typedef struct WSAData
{
    WORD wVersion;
    WORD wHighVersion;
    char szDescription[WSADESCRIPTION_LEN + 1];
    char szSystemStatus[WSASYS_STATUS_LEN + 1];
    unsigned short iMaxSockets;
    unsigned short iMaxUdpDg;
    char * lpVendorInfo;
} WSADATA, * LPWSADATA;
```

Єдина корисна інформація, що повертається у структурі WSADATA, – поля wVersion і wHighVersion. Після завершення роботи з бібліотекою Winsock потрібно викликати функцію WSACleanup для вивантаження бібліотеки і звільнення ресурсів. Інтерфейс функції: *int WSACleanup (void)*.

Сокет – це описувач постачальника транспорту. Сокет створюється однією з двох функцій:

```
SOCKET socket (int af, int type, int protocol);
або
SOCKET WSASocket (
    int af,
    int type,
    int protocol,
    LPWSAPROTOCOL_INFO lpProtocolInfo,
    GROUP g,
    DWORD dwFlags);
```

Параметри функції такі:

- `af` – визначає сім'ю адрес протоколу;
- `type` – тип сокета для даного протоколу;
- `protocol` – вказує конкретний транспорт.

У таблиці 3.1. перераховані значення, які використовуються у полях сім'ї адрес, типу сокета і протоколу для даного мережевого транспорту.

Найважливішим параметром сокета є сім'я адрес `af`. Цей параметр вказує на протокол та обмежує значення інших параметрів. Сім'я адрес і тип сокета обмежують вибір транспортного протоколу. У такому разі можна встановити у параметрі `protocol` значення нуль, а система сама обирає постачальника транспорту, виходячи з параметрів `af` і `type`. Але це можливо лише для тих протоколів, для яких після виклику функції `IpProtocolInfo` – вказівник на структуру `WSAPROTOCOL_INFO`, яка повертається функцією `WSAEnumProtocols`.

`g` – завжди дорівнює нулю, оскільки Windows не підтримує групи сокетів.

`dwFlags` – може задаватись одним або декількома прапорами:

`WSA_FLAG_OVERLAPPED`, `WSA_MULTIPOINT_C_ROOT`,

`WSA_MULTIPOINT_C_LEAF`, `WSA_MULTIPOINT_D_ROOT`,

`WSA_MULTIPOINT_D_LEAF`.

Таблиця 3.1 – Основні параметри сокетів

| Протокол  |               | Сокет            |                   |                             |
|-----------|---------------|------------------|-------------------|-----------------------------|
| Мережевий | Транспортний  | Сім'я адрес (af) | Тип сокета (type) | Тип протоколу (protocol)    |
| IP        | TCP           | AF_INET          | SOCK_STREAM       | IPPROTO_TCP                 |
|           | UDP           |                  | SOCK_DGRAM        | IPPROTO_UDP                 |
|           | Прості сокети |                  | SOCK_RAW          | IPPROTO_RAW<br>IPPROTO_IGMP |

`SOCK_STREAM IPPROTO_TCP` – потоковий сокет;

`UDP SOCK_DGRAM IPPROTO_UDP` – дейтаграмний сокет ;

Прості (сірі) сокети `SOCK_RAW IPPROTO_RAW` та `IPPROTO_IGMP`,

використання яких можливе без участі транспортних протоколів TCP та UDP.

Перший прапорець – `WSA_FLAG_OVERLAPPED`, який встановлює ввід/вивід з перекриттям, рекомендується встановлювати завжди при виклику функції `WSASocket`. Решта прапорців відносяться до багатоадресного передавання. Основні виклики функцій, що мають зробити клієнт і сервер для встановлення каналу зв'язку між комп'ютерами у мережі, зображені на рисунку 3.1.

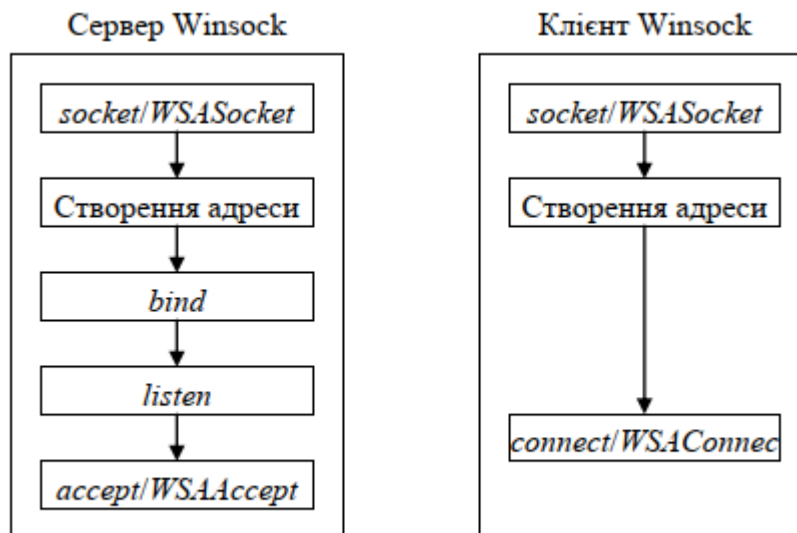


Рисунок 3.1 – Встановлення з'єднання

Після завершення роботи з бібліотекою Winsock необхідно викликати функцію `WSACleanup` для вивантаження бібліотеки і звільнення ресурсів: `int WSACleanup (void);`

Сервер Winsock:

*Створення адреси*

*socket/WSASocket*

*bind*

*listen*

*accept/WSAAccept*

Клієнт Winsock:

*Створення адреси*

*socket/WSASocket*

*connect/WSAConnect*

Розглянемо ці функції. Створення IP-сокета дозволяє програмам здійснювати під'єднання через TCP, UDP і протоколи IP.

## **Сервер Winsock**

Для створення IP-сокета з використанням протоколу TCP викликаються функції `socket` чи `WSASocket` із сім'єю адрес `AF_INET`, типом сокета `SOCK_STREAM`, та значенням нуль поля протоколу, наприклад:

*SOCKET s;*

*s=socket(AF\_INET, SOCK\_STREAM, 0);*

*s=WSASocket(AF\_INET, SOCK\_STREAM, 0, NULL, 0,  
WSA\_FLAG\_OVERLAPPED);*

Для взаємодії із сервером через TCP чи UDP клієнт повинен вказати IP-адресу сервера і номер порту служби. Щоб прослуховувати вхідні запити клієнта, сервер також повинен вказати IP-адресу і номер порту. У Winsock IP-адреса і порт служби задають у структурі `sockaddr_in`.

*struct sockaddr\_in*

*{*

*short sin\_family;*

*u\_short sin\_port;*

*in\_addr sin\_addr;*

*char sin\_zero[8];*

*};*

*Поля структури:*

*sin\_family – повинно дорівнювати AF\_INET;*

*sin\_port* – задає порт;

*sin\_addr* – зберігає IP-адресу і є типом *unsigned long int*;

*sin\_zero* – заповнювач, не задається.

IP-адреси звичайно задають у крапковій нотації a.b.c.d у вигляді рядка символів. Функція `inet_addr` перетворює IP-адресу з крапкової нотації у 32-бітове довге ціле без знака:

*unsigned long inet\_addr(const char \*cp );*

Параметр `cp` є рядком, що закінчується нульовим символом. Ця функція як результат повертає IP-адресу, подану 32-бітовим числом з мережевим порядком проходження байтів (*network-byte order*).

## Спеціальні адреси

`INADDR_ANY` – дозволяє серверній програмі слухати з'єднання від будь-яких клієнтів.

`INADDR_BROADCAST` – дозволяє широкомовно розсилати UDP-дейтаграми. Для її використання необхідно у програмі встановити параметр сокета `==SO_BROADCAST`.

`INADDR_LOOPBACK` – адреса локального комп'ютера.

При використанні TCP і UDP програма вказує, через який порт зв'язатися. Номери портів поділяють на три категорії:

- 0–1023 – стандартні, контролюються IANA і зарезервовані для стандартних служб;
- 1024–49151 – зареєстровані IANA і можуть використовуватися процесами і програмами;
- 49152–65535 – динамічні (приватні).

У пам'яті комп'ютера IP-адреса і номер порту представляються в системному порядку (*host-byte-order*). Для процесорів Intel Pentium це порядок від менш значущого до більш значущого байту (зворотній). Стандарти Internet вимагають, щоб багатобайтові значення передавалися від старшого байта до молодшого, що називається мережевим порядком (*network-byte order*) або прямим порядком проходження байтів. Тому існує необхідність перетворення чисел з однієї форми в іншу. Для цього призначений цілий ряд функцій.

Наприклад, функції `htonl` (Host TO Network Long), `WSAhtonl`, `htons` (Host TO Network Short), `WSAhtons` перетворюють числа з системного порядку в мережевий.

Функції `ntohl`, `WSAntohl`, `ntohs`, `WSAntohs` вирішують зворотну задачу (з мережевого в системний). Корисна допоміжна функція `inet_addr` перетворює IP-адресу з точково-десятькової нотації в 32-бітове довге ціле без знака з мережевим порядком проходження байтів:

```
unsigned long inet_addr (const char FAR * cp);
```

Поле `cp` - рядок, що закінчується нульовим символом, в якій задається IP-адреса в точково-десятьковій нотації. Чотири функції перетворюють числа із системного порядку у мережевий:

```
u_long htonl(u_long hostlong);
```

```
int WSAhtonl ( SOCKET s, u_longl hostlong, u_long * lpnetlong);
```

```
u_short htons(u_short hostshort);
```

```
int WSAhtons(SOCKET s, u_short hostshort, u_short * lpnetshort );
```

де `hostlong` – числа типу `long` із системним порядком;

`hostshort` – числа типу `short` із системним порядком.

Функція `bind` зв'язує сокет із заданою адресою. Інтерфейс функції:

```
int bind ( SOCKET s, const sockaddr *name, int namelen);
```

де `s` – сокет, на якому будуть очікуватись запити клієнтів на з'єднання;

`name` – вказівник на структуру, у якій знаходиться локальна адреса.

При виклику `bind` потрібно привести цю структуру до типу `sockaddr*`;

`namelen` – розмір передаваної структури адреси.

Функція `listen` переводить сокет у режим прослуховування. Інтерфейс функції:

```
int listen(SOCKET s, int backlog);
```

Параметри функції:

`s` – зв'язаний сокет;

`backlog` – максимальна довжина черги з'єднань, що очікують обробку.

Приєднання нового сокета до прослуховувального сокета виконується за допомогою функцій `accept` і `WSAAccept`.

**Функція listen.** Для переведення сокета в стан очікування вхідних з'єднань використовується API-функція `listen` ():  
*int listen (SOCKET s, int backlog);*

Перший параметр `s` - дескриптор сокета. Параметр `backlog` - максимальне число вхідних запитів на встановлення з'єднання, які можуть чекати в черзі на обробку поки сервер зайнятий. Значення `backlog` залежить від постачальника протоколу. Неприпустиме значення замінюється найближчим дозволенням.

**Функція accept:** *SOCKET accept( SOCKET s, sockaddr \*addr, int \*addrlen);*  
Параметри функції:

`s` – зв'язаний сокет у стані прослуховування;

`addr` – вказівник на адресу існуючої структури `sockaddr_in`;

`addrlen` – вказівник на довжину структури `sockaddr_in`.

**Функція WSAAccept** здатна встановлювати з'єднання залежно від результату обчислення умови:

```
SOCKET WSAAccept(  
SOCKET s,  
sockaddr * addr,  
LPINT addrlen,  
LPCONDITIONPROC lpfnCondition,  
DWORD dwCallbackData );
```

Перші три параметри – ті ж, що і в `accept` для Winsock 1. `lpfnCondition` – вказівник на функцію, що викликається при запиті клієнта. Вона визначає можливість прийому з'єднання і має такий прототип:

```
int CALLBACK ConditionFunc(  
LPWSABUF lpCallerId.  
LPWSABUF lpCallerData,  
LPQOS lpSQOS.  
LPQOS lpGQOS,  
LPWSABUF lpCalleeId,  
LPWSABUF lpCalleeData,  
GROUP * g.  
DWORD dwCallbackData );
```

lpCallerId – містить адресу об'єкта, що з'єднується. Структура WSABUF використовується багатьма функціями Wipsock 2 і визначена так:

```
typedef struct __WSABUF
```

```
{
```

```
u_long len;
```

```
char * buf;
```

```
} WSABUF, * LPWSABUF;
```

де len – розмір буфера buf або кількість даних у ньому.

- lpCallerId – вказівник на структуру WSABUF, поле buf якої вказує на структуру sockaddr з адресою віддаленого клієнта. Для доступу до інформації потрібно привести вказівник buf до типу sockaddr\_in.
- lpCallerData – не підтримується і дорівнює NULL.
- lpSQOS – задає рівень якості обслуговування, запитуваний клієнтом.
- Посилається на структуру, що містить інформацію про вимоги пропускну здатності для прийому і передавання. Якщо клієнт не запитує параметри якості обслуговування (quality of service, QoS), то параметру задають значення NULL.
- lpGQOS – не підтримується і дорівнює NULL.
- lpCalleeId – вказівник на структуру WSABUF, поле buf якої вказує на структуру sockaddr з локальною адресою сервера.
- lpCalleeData – не підтримується і дорівнює NULL.

Якщо з'єднання приймається, функція ConditionFunc поверне значення CF\_ACCEPT, якщо відхиляється – CF\_REJECT. Якщо на даний момент рішення не може бути прийнято, повертається CF\_DEFER. Коли сервер готовий обробити запит, він викликає функцію WSAAccept.

Клієнтська частина значно простіша і для встановлення з'єднання потрібно всього три етапи: створити сокет функцією socket чи WSASocket, перетворити ім'я сервера (залежить від використовуваного протоколу) та ініціювати з'єднання функцією connect чи WSAGconnect.

### **Функція connect:**

Параметри:

s – існуючий TCP-сокет для встановлення з'єднання;

name – структура, що містить адресу і порт сокета віддаленого сервера;

namelen – довжина змінної name.

### **Функція WSAConnect:**

```
int WSAConnect(  
SOCKET s,  
const sockaddr * name,  
int namelen,  
LPWSABUF lpCallerData,  
LPWSABUF lpCalleeData,  
LPQOS lpSQOS,  
LPQOS lpGQOS);
```

Перші три параметри такі ж, як і у функції connect.

lpCallerData – не підтримується і дорівнює NULL.;

lpCalleeData – не підтримується і дорівнює NULL..

Обидві змінні є структурами WSABUF, і для lpCallerData поле len має вказувати довжину буфера buf для передавання даних. У випадку lpCalleeData поле len визначає розмір буфера buf для прийому даних.

lpSQOS – вказівник на структуру QoS, що визначає вимоги до якості обслуговування, тобто до пропускну́ї спроможності відправлення і прийому даних. Нульове значення параметра lpSQOS означає, що програма не висуває вимог до якості обслуговування.

lpGQOS – не підтримується і дорівнює NULL.

### **Передача і отримання даних сокету (потоківі протоколи)**

Для пересилання даних по сокету використовуються функції send і WSASend. Аналогічно, для прийому даних існують функції recv і WSAREcv.

Всі буфери, що використовуються при відправці і прийомі даних, складаються з елементів типу char.

**Функція send:** API-функція send для відправки даних по сокету визначена так:

```
int send (  
    SOCKET s,  
    const char FAR * buf,  
    int len,  
    int flags);
```

Параметр s визначає сокет для відправки даних. Другий параметр - buf, вказує на символьний буфер, що містить дані для відправки. Третій - len, задає число відправляються з буфера символів. Параметр - flags, може приймати значення 0, MSG\_DONTROUTE, MSG\_OOB, або результат логічного АБО над будь-якими з цих параметрів. При вказуванні прапора MSG\_DONTROUTE не будуть маршрутизовуватися пакети, які відправляються. Прапор MSG\_OOB вказує, що дані повинні бути відправлені поза смугою (out of band), тобто терміново. При успішному виконанні функція send поверне кількість переданих байт, інакше - помилку SOCKET\_ERROR. Число, що повертається функцією send може бути менше зазначеного розміру буфера. Одна з типових помилок - WSAE\_CONNABORTED, відбувається при розриві віртуального з'єднання через помилку протоколу або закінчення часу очікування. В цьому випадку сокет повинен бути закритий, так як він не може використовуватися. Число, що повертається функцією send може бути менше зазначеного розміру буфера. У Winsock версії 2 визначена функція WSASend - аналог send.

**Функція recv** визначена так:

```
int recv (  
    SOCKET s,  
    char FAR * buf,
```

```
int len,  
int flags);
```

Параметр `s` визначає сокет для прийому даних. Другий параметр - `buf`, є символьним буфером і призначений для отриманих даних. Параметр `len` вказує на число прийнятих байт або розмір буфера `buf`. Останній параметр - `flags`, може приймати значення 0, `MSG_PEEK`, `MSG_OOB` або результат логічного АБО над будь-якими з цих параметрів. Зрозуміло, 0 означає відсутність особливих дій. Прапор `MSG_PEEK` вказує, що доступні дані повинні копіюватися в приймаючий буфер і при цьому залишатися в системному буфері. Його використовувати не рекомендується. Прапор `MSG_OOB` вже обговорювався під час розгляду відправки даних. Використання `recv` в сокетах, орієнтованих на передачу повідомлень або дейтаграм, має кілька особливостей. Якщо при виклику `recv` розмір очікують обробки даних більше наданого буфера, то після його повного заповнення виникає помилка `WSAEMSGSIZE`. Зауважте: помилка перевищення розміру повідомлення відбувається тільки при використанні протоколів, орієнтованих на передачу повідомлень. Поточкові протоколи буферизують дані, що надходять і при запиті додатком надають їх в повному обсязі, навіть якщо розмір даних, що очікують обробки більше розміру буфера. Таким чином, помилка `WSAEMSGSIZE` не може відбутися при роботі з поточковими протоколами. Функція `WSARecv` має додаткові порівняно з `recv` можливостями: підтримує перекритий вхід-вихід і фрагментарні дейтаграмні повідомлення.

Після закінчення роботи із сокетом потрібно коректно завершити сеанс за допомогою функції `shutdown`:

```
int shutdown( SOCKET s, int how);
```

Параметр `how` може набувати значення `SD_RECEIVE`, `SD_SEND` чи `SD_BOTH`. Значення `SD_RECEIVE` забороняє виклики функцій прийому даних. `SD_SEND` забороняє всі виклики функції відправлення даних. Нарешті, `SD_BOTH` забороняє як прийому, так і відправлення. Після цього необхідно викликати функцію `closesocket` для закриття з'єднання і звільнення всіх ресурсів:

```
int closesocket( SOCKET s);
```

У протоколах(зокрема UDP), що не потребують з'єднання, архітектурою обміну є не клієнт-сервер, а приймач-передавач. Різниця між приймачем і передавачем полягає тільки у тому, хто викликає функції прийому, а хто – передає інформацію. Функція прийому завжди має викликатись першою. Будь-який з варіантів функції прийому виконується у блокувальному режимі і очікує отримання даних.

При побудові програми-приймача спочатку створюють сокет функцією `socket` чи `WSASocket`. Потім виконують прив'язку сокета з локальною адресою функцією `bind` (як і у випадку протоколів, орієнтованих на сеанси).

Різниця у тому, що не викликаються функції `listen` та `accept`. Замість цього просто передають або отримують дані. Оскільки у цьому випадку з'єднання немає, приймальний сокет може одержувати дейтаграми від будь-якої машини у мережі.

Найпростіша функція прийому – `recvfrom`:

```
int recvfrom(  
    SOCKET s,  
    char * buf,  
    int len,  
    int flags,  
    sockaddr * from,  
    int * fromlen );
```

Перші чотири параметри такі ж, як і для функції `recv`, включно з допустимими значеннями для `flags`: 0, `MSG_OOB` і `MSG_PEEK`;

`from` – структура `sockaddr` передавального сокета, на розмір якої посилається `fromlen`. Після повернення виклику структура `sockaddr` буде містити адресу робочої станції, що відправляє дані.

У Winsock 2 для прийому інформації без встановлення з'єднання застосовується `WSARecvFrom`:

```
int WSARecvFrom(  
    SOCKET s,  
    LPWSABUF lpBuffers,  
    DWORD dwBufferCount,  
    LPDWORD lpNumberOfBytesRecvd,
```

*LPDWORD lpFlags,*  
*sockaddr \* lpFrom,*  
*LPINT lpFromlen,*  
*LPWSAOVERLAPPED lpOverlapped,*  
*LPWSAOVERLAPPED\_COMPLETION\_ROUTINE lpCompletionRoutine );*

Функції можна надати один чи кілька буферів WSABUF, вказавши їх кількість у dwBufferCount, – у цьому випадку можливе комплексне забезпечення вводу/виводу. Сумарна кількість зчитаних байтів передається у lpNumberOfBytesRecv. При виклику функції WSARecvFrom lpFlags може приймати такі значення: 0 (при відсутності параметрів), MSG\_OOB, MSG\_PEEK чи MSG\_PARTIAL. Дані прапорці можна комбінувати логічною операцією АБО. Якщо при виклику функції задано прапорець MSG\_PARTIAL, постачальник перешле дані, навіть у випадку прийому, лише частини повідомлення. Після повернення прапорець встановлюється у MSG\_PARTIAL, тільки якщо повідомлення прийняте частково. Після повернення WSARecvFrom присвоїть параметру lpFrom (вказівник на структуру sockaddr) адресу комп'ютера-відправника. Знову ж, параметр lpFromLen вказує на розмір структури SOCKADDR, однак у даній функції він є вказівником на DWORD. Два останніх параметри – lpOverlapped і lpCompletionRoutine – використовуються для вводу/виводу з перекриттям.

Інший спосіб прийом/відправлення даних у сокетах, як не дивно – це з'єднання, що не потребують встановлення з'єднання. Дейтаграмні сокети працюють без установки з'єднання, тому, як правило, не звертаються до функції connect.

Проте за "як правило" криється один хитрий прийом програмування - виклик connect дозволяє дейтаграмному сокету обмінюватися даними з вузлом не тільки функціями sendto, recvfrom, але і більш зручними і компактними send і recv. Ця тонкість описана в Winsocket SDK і широко використовується.

Після створення сокета і зв'язування з локальною адресою можна викликати connect чи WSAConnect, присвоївши параметру sockaddr адресу комп'ютера з мережі, з яким необхідно зв'язатися. Фактично ніякого з'єднання

не відбувається. Адреса сокета, переданого у функцію з'єднання, асоціюється із сокетом, щоб було можна використовувати функції `recv` і `WSARecv` замість `recvfrom` чи `WSARecvFrom` (оскільки джерело даних відоме). Якщо програмі потрібно одночасно зв'язуватися лише з однією кінцевою точкою, використовується можливість під'єднання сокета дейтаграм.

При побудові передавача є два способи відправлення даних через сокет, для яких не потрібне з'єднання. Перший і найпростіший – створити сокет і викликати функцію `sendto` чи `WSASendTo`.

Розглянемо функцію **`sendto`**:

```
int sendto(
    SOCKET s,
    const char * buf,
    int len,
    int flags,
    const sockaddr * to,
    int tolen );
```

Параметри цієї функції такі самі, як і у `recvfrom`, за винятком `buf` – буфера даних для відправлення і `tolen` – розміра структури `sockaddr`. Параметр `to` – вказівник на структуру `sockaddr` з адресою приймальної робочої станції.

Також можна використовувати функцію `WSASendTo` з Winsock 2:

```
int WSASendTo(
    SOCKET s,
    LPWSABUF lpBuffers,
    DWORD dwBufferCount,
    LPDWORD lpNumberOfBytesSent,
    DWORD dwFlags,
    const sockaddr * lpTo,
    int iToLen,
    LPWSAOVERLAPPED lpOverlapped,
    LPWSAOVERLAPPED_COMPLETION_ROUTINE
    lpCompletionRoutine);
```

Функція `WSASendTo` аналогічна своїй попередниці. Вона приймає вказівник на одну чи кілька структур `WSABUF` з даними для відправлення одержувачу у вигляді параметра `lpBuffers`, а `dwBufferCount` задає кількість структур. Для комплексного вводу/виводу можна відправити кілька структур `WSABUF`. Перед виходом `WSASendTo` присвоює четвертому параметру - `lpNumberOfBytesSent` - кількість реально відправлених одержувачу байтів. Параметр `lpTo` - структура `sockaddr` для даного протоколу з адресою приймача. Параметр `iToLen` - довжина структури `sockaddr`. Два останніх параметри - `lpOverlapped` і `lpCompletionRoutine` - використовуються для вводу/виводу з перекриттям.

Як і при отриманні даних, сокет, що не потребує з'єднання, можна підключати до адреси кінцевої точки і відправляти дані функціями `send` і `WSASend`. Після створення цього прив'язування не можна використовувати `sendto` і `WSASendTo` з іншою адресою - буде видано помилку `WSAEISCONN`.

Протоколи, що потребують з'єднання, орієнтовані на передавання повідомлень. Тому потрібно враховувати такі фактори.

По-перше, оскільки протоколи, орієнтовані на передачу повідомлень зберігають межі повідомлень, то дані, поставлені у чергу для відправлення, блокуються до завершення виконання функції відправлення. Якщо відправлення не може бути завершене, то при асинхронному або неблокувальному режимі вводу/виводу функція відправлення поверне помилку `WSAEWOULDBLOCK`. Це означає, що операційна система не змогла обробити дані і потрібно викликати функцію відправлення повторно. Важливим є те, що в орієнтованих на повідомлення протоколах запис відбувається тільки внаслідок самостійної дії.

По-друге, при виклику функції прийому потрібно надати місткий буфер, інакше функція видасть помилку `WSAEMSGSIZE`: буфер заповнений, і дані, що залишилися, відкидаються. Винятком є протоколи, що підтримують обмін фрагментарними повідомленнями, наприклад `AppleTalk PAP`. Якщо була прийнята лише частина повідомлення, функція `WSARecvEx` присвоює параметру `flag` значення `MSG_PARTIAL`.

Для передавання дейтаграм на основі протоколів, що підтримують фрагментарні повідомлення, використовують одну з функцій `WSARecv` чи `recv`. Після чергового виклику `recv` може бути отримана лише частина дейтаграми.

Проте при виклику `recv` не можна відстежити повноту зчитування повідомлення (це задача програміста). Через дане обмеження зручніше використовувати функцію `WSARecvEx`, що дозволяє задавати і зчитувати прапорець `MSG_PARTIAL`. Функції Winsock 2 `WSARecv` і `WSARecvFrom` також підтримують роботу з даним прапорцем.

Якщо замість `bind` викликається `sendto` чи `WSASendTo`, чи спочатку встановлюється з'єднання, мережевий стек автоматично вибирає локальну IP-адресу з таблиці маршрутизації. Таким чином, якщо спочатку була виконана прив'язка, початкова IP-адреса може бути неправильною і не відповідати інтерфейсу, з якого фактично була відправлена дейтаграма.

### **3.2. Порядок виконання роботи**

Релевантним середовищем для виконання даної лабораторної роботи є мережа комп'ютерів, поєднаних за протоколами TCP/IP, UDP/IP. Однак її можна реалізувати і на одній робочій станції, де встановлена операційна система Microsoft версії Windows NT і вище. Завдання передбачає розробку програм сервера та клієнтів. При реалізації завдання лабораторної роботи необхідно врахувати визначення адрес сервера та клієнта. Рекомендації для виконання лабораторної роботи на одній робочій станції наступні: треба скористатися інтерфейсом внутрішньої петлі (loopback interface)

#### **Інтерфейс внутрішньої петлі**

Більшість реалізацій TCP / IP підтримує так званий інтерфейс внутрішньої петлі (loopback interface), який дозволяє двом прикладним процесам, які перебувають на одному хості, обмінюватися даними за допомогою протоколу TCP/IP.

При цьому, як правило, формуються дейтаграми, але вони не покидають межі одного хоста. Для інтерфейсу внутрішньої петлі, за угодою, зарезервовано мережу 127.0.0.0. Відповідно до загальноприйнятих угод більшість операційних систем призначають для інтерфейсу внутрішньої петлі адресу 127.0.0.1 і надають символічне ім'я `localhost`.

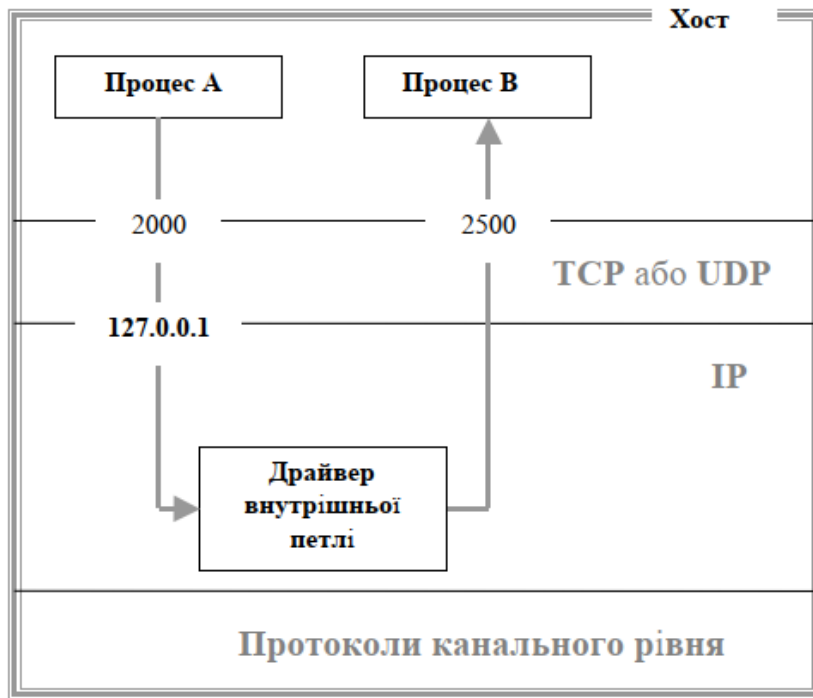


Рисунок 3.2 – Спрощена схема роботи інтерфейсу внутрішньої петлі

На рисунку 3.2 приведена спрощена схема обробки даних інтерфейсом внутрішньої петлі. Прикладний процес А, використовуючи номер порту, наприклад, 2000, відправляє дані процесу В. Вказавши в параметрах сокета процесу В мережеву адресу 127.0.0.1, процес А забезпечив обробку надісланих дейтаграм на міжмережевому рівні драйвером внутрішньої петлі, який направляє ці дейтаграми до вхідної черги модуля ІР. ІР-модуль, слідує звичайної логіки своєї роботи, доставляє дані на транспортний рівень. Далі протокол транспортного рівня відповідно до номером порту, наприклад, 2500 в заголовку сегмента (або пакета) направляє дані процесу В. Слід звернути увагу на наступне: всі дані, що пересилаються по інтерфейсу внутрішньої петлі, не тільки не залишають меж хоста, але і не зачіпають ніяких зовнішніх механізмів за межами стека TCP(UGP)/ IP.

## Реалізації проєкту

**Варіант 1:** Потрібно попередньо написати вихідні тексти, що реалізують прості TCP або UDP сервери і TCP, та UDP клієнти. Далі для їх компіляції за допомогою Microsoft Visual C++ досить віддати команду: "cl.exe ім'я\_файла.cpp ws2\_32.lib".

**Варіант 2:** Скористатися можливостями середовища Microsoft Visual C++. Далі йде досить детальний опис цієї методики (з можливими необхідними корективами для нових версій Microsoft Visual Studio).

Відкриємо Visual Studio:

1. File-> New-> project.
2. В вікні New project виберемо Other Project Types-> Visual Studio Solutions-> Blank Solution.
3. В полі Name задамо ім'я Solution, наприклад Lab1.
4. Кнопкою Browse, задамо папку для Solution.
5. Натиснемо Ok. Solution Lab1 з'явиться в Solution Explorer.
6. Натиснемо праву кнопку миші на імені нашого Solution Lab1 і виберемо Add-> New Project.
7. Оберемо Visual C ++ -> Win32-> Console Application.
8. У полі Name задамо ім'я нашого проєкту, наприклад TcpServer.
9. Задамо в полі Location за допомогою кнопки Browse, щоб проєкт був у папці нашого Solution Lab1.
10. Натиснувши Ok, потрапимо в Win32 Application Wizard. Натиснемо в ньому Next.
11. Перевіримо, щоб радіокнопка Application Type, вказувала на Console Application.
12. Приберемо галочку в Precompiled headers.
13. Встановимо галочку в Empty Project і натиснемо Finish.
14. Проєкт TcpServer з'явиться в Solution Explorer.
15. Натиснемо праву кнопку миші на імені нашого проєкту TcpServer -> Source Files та оберемо Add-> New Item.
16. У вікні Add New Item виберіть Visual C ++ -> Code-> C ++ file.
17. У полі Name введемо ім'я файлу з розширенням .c (НЕ .cpp), наприклад

TcpServer.c.

18. Проконтролюємо в полі Location сза допомогою кнопки Browse, щоб TcpServer.c знаходився в папці нашого проєкту TcpServer.

19. Введемо у вікно TcpServer.c текст програми.

20. Натиснемо праву кнопку миші на імені нашого проєкту TcpServer та оберемо Property.

21. У вікні Property Pages оберемо Configuration Properties-> General-> Caracter Set-> Use Multi-Byte Character Set.

22. Якщо додаток вимагає аргументів командного рядка, як, наприклад TcpClient або UDPReceiver, то у вікні Property Pages виберіть Configuration Properties-> Debugging-> Command Arguments і введіть аргументи командного рядка, наприклад 127.0.0.1.

23. Задайте проєкту режим компонування з бібліотеками Winsock: у вікні Property Pages виберіть Configuration Properties-> Linker-> Input-> Additional Dependencies і введіть імена Winsock бібліотек: ws2\_32.lib mswsock.lib.

24. За умовчанням встановлений режим налагодження Debug Win32, перевірте.

25. Натиснемо праву кнопку миші по імені нашого проєкту TcpServer та оберемо Build.

26. виправте помилки. Має бути повідомлення про успішну побудову проєкту.

27. Повторимо кроки 6 - 26 для інших проєктів поточного Solution. Так для Lab1 додамо проєкти TcpClient, UDPReceiver.

28. Можна побудувати або перебудувати всі проєкти Solution Lab1, натиснувши правою кнопкою на імені Solution Lab1 і вибравши build Solution або rebuild Solution налагодження проєктів.

Перед початком роботи з допомогою команди netstat -an переконайтеся, чи не зайнятий використовуваний порт, наприклад, 5150. На початку роботи проєкт треба зробити активним в Solution, натиснувши по його імені правою кнопкою миші і вибрати Set as StartUp project. Клавiша F10 задає послiдовне виконання без заходу у функції, які викликаються, а клавiша F11 задає покрокове виконання із заходом у функції, які викликаються. Клік миші лiвiше оператора встановлює/знімає контрольну точку. Клавiша F5 виконує програму до наступної контрольної точки, або до кінця. Підвівши курсор миші до змінної, можна побачити її значення. Це можна зробити і за

допомогою пунктів контекстного меню add watch і quick watch. Налагодження зупиняється вибором пункту меню debugging-> stop debugging.

Запустити програму (.exe) з командного рядка можна або набравши її ім'я з аргументами через start> пуск, або запустивши командний процесор cmd.exe, або запустивши з менеджера файлів типу Far. Виконувані файли знаходяться в папці debug або release папки Solution Lab1. Важливо наголосити на необхідності наявності в кінці програм макросу getchar, без нього консольне вікно програми після завершення програми просто зникне. Програми працюють тільки в парах: пара TcpClient, TcpServer і пара UDPReceiver, UDPSender. Тому доцільно запустити два примірники програм з пари. Можна обидві програми запускати з командного рядка, можна одну запускати в командному рядку, а другу в режимі налагодження в Visual Studio і можна обидві програми запускати в режимі налагодження в Visual Studio.

Розглянемо останній варіант більш докладно для пари TcpClient, TcpServer.

**Клієнт і сервер TCP.** Запустимо Visual Studio з активним проєктом TcpServer (не забудьте Set as StartUp project) і встановимо контрольні точки на викликах функцій accept і recv. Запустимо другий екземпляр Visual Studio з активним проєктом TcpClient. Ігноруємо попередження. Встановимо контрольні точки на викликах функцій connect і send. Не забудьте вказати аргумент командного рядка 127.0.0.1. Перейдемо до примірника Visual Studio з активним проєктом TcpClient і натиснемо F5. Програма зупиниться на встановленій контрольній точці на функції connect. Підвівши курсор до змінних s, msg, Port, Ret, lmsg, caSize, argv, подивіться їх значення. Зверніть увагу на значення полів структури ServerAddr. У командному вікні з'явиться: *Спроба з'єднання до 127.0.0.1:5150*. Натискаючи F11 або F10, дійдіть до функції closesocket (s). У командному вікні з'явиться помилка connect WSAECONNREFUSED: У з'єднанні відмовлено (Передбачається, що ви завершили написання функції розшифровки помилок encodeWSAGetLastError). Завершіть налагодження. Все логічно. Ми спробували з'єднатися до незапущеного сервера.

Перейдемо до примірника Visual Studio з активним проєктом TcpServer і натиснемо F5. Програма зупиниться на встановленій контрольній точці на функції accept. Підвівши курсор до різних змінних, подивіться їх значення. У

командному вікні з'явиться: *Очікуємо з'єднання на порту 5150*. Натиснемо F10. Додаток TcpServer заблоковано на функції асепт, чекаючи запиту від клієнта на з'єднання. Заголовок вікна Visual Studio змінився від Debugging до Running. Команда netstat -an, видасть інформацію, в якій є рядок: *TCP 0.0.0.0:5150 0.0.0.0:0 LISTENING*

Перейдемо до примірника Visual Studio з активним проєктом TcpClient і натиснемо F5. Програма зупиниться на встановленій контрольній точці на функції connect. Натиснемо F5. Програма зупиниться на встановленій контрольній точці на функції send. Підвівши курсор до змінної клієнтської адреси сокета s, подивіться її значення. У командному вікні перейдемо до примірника Visual Studio з активним проєктом TcpClient і натиснемо F5. Програма зупиниться на встановленій контрольній точці на функції connect. Натиснемо F5. Програма зупиниться на встановленій контрольній точці на функції send. Підвівши курсор до змінної sa клієнтського адреси сокета s, подивіться її значення.

У командному вікні з'являються додаткові рядки: *Наше з'єднання успішно. Локальна адреса: 127.0.0.1 і порт: 2215 сокета клієнта. Спроба передати повідомлення Привіт.*

Природно, що у вас буде номер порту відмінний від 2215. Команда netstat -an, видасть інформацію, в якій є рядки

|     |                |                |             |
|-----|----------------|----------------|-------------|
| TCP | 127.0.0.1:2215 | 127.0.0.1:5150 | ESTABLISHED |
| TCP | 127.0.0.1:5150 | 127.0.0.1:2215 | ESTABLISHED |

Тобто встановлено двостороннє повнодуплексне TCP з'єднання. Перший рядок відповідає клієнту, а другий серверу. Перейдемо до примірника Visual Studio з активним проєктом TcpServer. Програма Розблокувати на виклик асепт. Заголовок вікна Visual Studio змінився від Running до Debugging. Обчислення зупинилися на операторі myprintf. Натиснувши F5 ми переїдемо до наступної контрольної точки функції recv. Підвівши курсор до змінної NewConnection нового сокета і до змінної адреси під'єднання сокета ClientAddr, подивіться їх значення. У командному вікні з'являються додаткові рядки: *Успішно з'єдналися з 127.0.0.1:2215. Чекаємо дані для отримання.* Зверніть увагу на порт 2215 - він однаковий і для клієнта і для сервера. Цей порт система виділила динамічно. У вас буде інший порт. TCP з'єднання характеризується четвіркою

IP-адреса джерела: порт джерела, IP-адреса приймача, порт приймача, та інформація про якість обслуговування QoS (quality of service), яке тут не розглядається. У нашому випадку четвірка, яка визначає з'єднання має вигляд 127.0.0.1:2215, 127.0.0.1:5150.

Далі можна діяти двома способами, за двома сценаріями.

**Сценарій 1.** Перейдемо до примірника Visual Studio з активним проєктом TcpClient. Програма зупинена на встановленій контрольній точці на функції send. Натиснемо F5. Програма за допомогою функції send передасть дані по напрямку до сервера і заблокується на функції getchar, чекаючи від користувача натискання клавіші Enter. Зауважимо, що сервер ще не викликав функцію прийому recv, тобто дані вирушили не в функцію recv сервера безпосередньо, а в мережу по шляху в цю функцію. Зауважимо, що обсяг таких даних, які нікому безпосередньо прийняти в даний конкретний момент теоретично необмежений і визначається лише ресурсами комп'ютерів, які беруть участь в обміні даними. У командному вікні з'являться додаткові рядки: *Успішно передано 6 байтів повідомлення Привіт*. Закриваємо з'єднання з сервером. Натисніть Enter для завершення. Дані передані, вони в мережі. Клієнт можна навіть зупинити. Зробіть це, завершивши налагодження, натиснувши Enter Команда netstat-an, видасть інформацію, в якій є рядки:

|     |                |                |            |
|-----|----------------|----------------|------------|
| TCP | 127.0.0.1:2215 | 127.0.0.1:5150 | FIN_WAIT_2 |
| TCP | 127.0.0.1:5150 | 127.0.0.1:2215 | CLOSE_WAIT |

Тобто з'єднання починають закриватися.

Через кілька хвилин netstat-an, видасть інформацію:

|     |                |                |            |
|-----|----------------|----------------|------------|
| TCP | 127.0.0.1:5150 | 127.0.0.1:2215 | CLOSE_WAIT |
|-----|----------------|----------------|------------|

Бачимо, що клієнтський сокет закрився. Перейдемо до примірника Visual Studio з активним проєктом TcpServer. Програма зупинена на встановленій контрольній точці на функції recv. Натиснемо F5. На цей момент клієнта не існує і його сокет закритий. Програма за допомогою функції recv забере дані з мережі і заблокується на функції getchar, чекаючи від користувача натискання клавіші Enter. Зауважимо, що дані сервер може забрати з мережі через досить тривалий час після смерті клієнта. У командному вікні з'являться додаткові рядки: *Успішно отримано 6 байтів в повідомленні Привіт*. Закриваємо з'єднання з клієнтом. Натисніть Enter для завершення. Команда netstat-an, не видасть інформації про наших сокетах - вони закрилися.

**Сценарій 2.** Перейдемо до примірника Visual Studio з активним проєктом TcpServer. Програма зупинена на встановленій контрольній точці на функції recv. Натиснемо F5. Програма блокується на виклик recv, чекаючи дані з мережі. Перейдемо до примірника Visual Studio з активним проєктом TcpClient. Програма зупинена на встановленій контрольній точці на функції send. Натиснемо F5. Програма за допомогою функції send передасть дані сервера і заблокується на функції getchar, чекаючи від користувача натискання клавіші Enter. У командному вікні з'являться додаткові рядки: *Успішно передано 6 байтів повідомлення Привіт*. Закриваємо з'єднання з сервером. Натисніть Enter для завершення. Перейдемо до примірника Visual Studio з активним проєктом TcpServer. Програма прийняла свої дані з мережі, розблокувалась і зупинилася на наступному операторі. Натиснемо F5. Програма заблокується на функції getchar, чекаючи від користувача натискання клавіші Enter. У командному вікні з'являться додаткові рядки: *Успішно отримано 6 байтів в повідомленні Привіт*. Закриваємо з'єднання з клієнтом. Натисніть Enter для завершення. Закриємо всі примірники Visual Studio.

## UDP

Розглянемо тепер пару проєктів для UDP: UDPsender і UDPreceiver. Відкриємо кожен з них в окремому примірнику Visual Studio. Встановіть в UDPreceiver контрольну точку на виклик recvfrom, а в UDP sender на виклик sendto. Запустіть UDPsender. Він зупиниться на встановленій контрольній точці на функції sendto. Натисніть F5. UDPsender успішно відправить дейтаграму за допомогою виклику sendto і заблокується на функції getchar, чекаючи введення з клавіатури Enter. У командному вікні побачимо: *Успішно відіслано 6 байт повідомлення Привіт до 127.0.0.1:5151*. Натисніть Enter для завершення. Запустіть UDPreceiver. Він зупиниться на встановленій контрольній точці на функції recvfrom. Натисніть F5. UDPreceiver заблокується на функції recvfrom. Тобто дейтаграма, яка була надіслана UDPsender загубиться. Як ви пам'ятаєте, що для випадку TCP після установки з'єднання дані не губилися. У командному вікні побачимо: *Ми готові отримати 1 діаграму з будь-якого інтерфейсу на порту 5151...*

Netstat-an покаже, що порт UDP 5150 відкритий UDP 0.0.0.0:5150 \*: \*. Відкриємо екземпляр Visual Studio з проєктом UDPsender. Приберемо в ньому контрольні точки і запустимо на виконання. Відкриємо екземпляр Visual Studio з проєктом UDPreceiver. Він розблокується з функції recfrom і заблокується на функції getchar, чекаючи введення з клавіатури Enter. У командному вікні побачимо додаткові рядки: *Ми успішно отримали 6 байт повідомлення Привіт від адреси 127.0.0.1:2605*. Натисніть Enter для завершення.

### 3.3. Завдання

Розробити клієнтську і серверну частину технології Winsocket в кодї C++ згідно варіанту. Передбачити розробку одного сервера та декільких клієнтів, які обмінюються даними через сокет з'єднання, причому клієнти можуть подавати запити на з'єднання та передавати дані серверу та приймати від сервера його дані, а сервер повинен приймати дані, обробляти і надсилати клієнтам. Дані сервера та клієнтів зберігаються у відповідних файлах.

#### 1) Протокол обміну

1.TCP

2.UDP

#### 2) Моделі вводу/виводу для Winsock 2.0

1. "Звичайні" блокуючі сокети - "рідний" стан будь-якого сокета - блокуючий, і виклик операції на ньому не поверне управління, доки вона не закінчиться.
2. "Чисто" неблокуючі сокети - виклик на таких сокетах повертає управління негайно, навіть якщо операція ще триватиме. Хоча принципово до моменту повернення програма може проводити деякі дії, цей метод вимагає постійного опитування для визначення моменту завершення операції. Зазвичай за допомогою: `select` `()`.
3. Мультиплексування зf допомогою `select` `()`. Функція `select()` блокує батьківський потік до тих пір, поки на одному з сокетів не відбудеться якась мережева подія. Звичайна практика - використання `select` `()` на неблокуючих сокетах, щоб уникнути опитування.

4. Асинхронні сокети - це теж неблокуючі сокети (`WSAAsyncSelect ()`), за винятком того, що не потрібно проводити опитування - модуль стекового протоколу посилає спеціальне віконне повідомлення, коли на сокеті станеться те, що буде цікавити програму.
5. Об'єкти-події - використовується `WSAEventSelect ()`, механізм схожий на метод мультиплексування з допомогою `select ()`, але більш ефективний.
6. I/O з перекриттям - одна з найбільш примітних особливостей WinSock 2, яка дає можливість узагальнити на сокетних операціях уніфікований I/O-механізм Win32 API і дає суттєву перевагу в продуктивності в порівнянні з попередніми моделями.

### **3) Суть обробки для сервера**

Спеціально виділений потік сервера читає файл, що знаходиться у просторі сервера і обробляє за запитом клієнта.

Робота з рядками та символами:

1. перетворити символи у верхній регістр;
2. перетворити в символи нижній регістр;
3. перевірити, чи є перший символ заданого рядка цифрою;
4. перевірити чи є символ заданого рядка буквою;
5. отримати значення елемента масиву заданого рядка;
6. перетворити символи останнього рядка верхній регістр.

Робота з масивом чисел:

1. Знайти найбільше число заданого числового рядка.
2. Знайти середнє значення числового рядка.
3. Знайти кількість елементів у масиві числового рядка.
4. Отримати значення елемента масиву за номером числового рядка.
5. Знайти чи є від'ємні числа в числовому рядку.
6. Збільшити значення елементів числового рядка на задане число.

Основний потік повертає результат клієнту.

Спеціально виділений потік сервера обробляє запити клієнтів і пише у файл результат обробки рядка від клієнта.

Робота з рядком(букви):

1. Чи є символи в рядку, що знаходяться у верхньому регістрі.
2. Чи є перший символ заданого рядка цифрою.
3. Вставити заданий символ у визначену позицію.
4. Чи є останній символ заданого рядка буквою.
5. Вирахувати довжину рядка.

Робота з масивом чисел:

1. Знайти найбільше число заданого числового рядка.
2. Знайти середнє значення числового рядка.
3. Знайти кількість елементів у масиві числового рядка.
4. Отримати значення елемента масиву за номером числового рядка.
5. Знайти чи є від'ємні числа в числовому рядку.
6. Збільшити значення елементів числового рядка на задане число.

Основний потік повертає результат клієнту.

### Обробка даних для клієнта

Клієнт робить:

1. запит,
2. може надсилати дані у вигляді рядка,
3. отримує результат запиту
4. перевіряє результа запиту.

Якщо відповідь невірна повторює запит

Таблиця 3.2 – Варіанти завдань

| №  | Протокол | Модель вводу/виводу для Winsock 2.0 | Суть обробки для сервера за запитом клієнтів | Обробка даних для клієнтів |
|----|----------|-------------------------------------|----------------------------------------------|----------------------------|
|    | 1        | 2                                   | 3                                            | 4                          |
| 1. | 1        | 1                                   | 1.1.1, 1.2.1                                 | 1,2,3                      |
| 2. | 1        | 2                                   | 1.1.2, 1.2.2                                 | 1,2,3                      |
| 3. | 1        | 3                                   | 1.1.3, 1.2.3                                 | 1,2,3                      |
| 4. | 2        | ??                                  | 2.2.1, 2.2.2, 2.2.3                          | 1,2,3,4,5                  |
| 5. | 1        | 4                                   | 1.1.5, 1.2.5                                 | 1,2,3                      |

## **Контрольні запитання**

1. В чому полягає основна перевага сокетів Windows для програмування мережевого обміну?
2. Що собою являє сімейство адрес сокетів?
3. Опишіть основну структуру при ініціалізації сокета у Windows.
4. Опишіть послідовність використання функцій API для забезпечення з'єднання клієнта та сервера.
5. Опишіть поля структури `sockaddr_in`.
6. Опишіть призначення та параметри функції `bind`.
7. Опишіть призначення та параметри функції `listen`.
8. Опишіть призначення та параметри функції `accept`.
9. Опишіть призначення та параметри функції `recv`.
10. Опишіть призначення та параметри функції `sendto`.
11. Опишіть призначення та параметри функції `connect`:
12. Опишіть програмні засоби передачі і отримання даних сокету.
13. Опишіть особливості прийому-передачі у протоколі UDP.
14. Поясніть способи обробки запитів клієнтів на боці сервера.

## **ЛАБОРАТОРНА РОБОТА №4**

### **ОСНОВИ СОМ-ТЕХНОЛОГІЙ**

**Мета:** Вивчити механізм дії СОМ-технології та оволодіти навичками системного програмування при створенні компонент

#### **4.1. Теоретичні відомості**

СОМ-технології сьогодні є однією із найбільш використовуваних систем побудови розподілених прикладних програм для платформ Microsoft. Практично кожна програма для Windows тією чи іншою мірою використовує цю технологію, багато системних об'єктів і API реалізовані у вигляді СОМ-компонентів. Більш того – деякі прикладні програми (наприклад MS Office) використовують СОМ як основу внутрішньої об'єктної моделі, що дозволяє легко розширювати їхній і взаємодіяти з використанням будь-якої мови. Розширення бібліотеки одержало назву СОМ+, а після вийшло ще одне розширення для створення розподілених прикладних програм – DCOM. Платформа, що зараз просувається .NET також є продовженням СОМ-технології. Справа дійшла навіть до того, що з'явилися крос-платформні реалізації СОМ, що працюють у Mac OS, Linux і навіть FreeBSD. Крім СОМ існують і інші компонентні моделі (наприклад дійсно крос-платформна CORBA), однак вони не одержали настільки тотального поширення, як СОМ.

У цій лабораторній роботі ми спробуємо розглянути базові поняття СОМ-моделі і створити власний компонент.

#### **Поняття інтерфейсу**

Основним поняттям СОМ є інтерфейс. Інтерфейс це зовнішня оболонка об'єкта, клавіатура керування, за допомогою якої ми можемо працювати з ним. При цьому працюючи з інтерфейсом абсолютно не важливо, як об'єкт улаштований усередині. Візьмемо, наприклад, звичайний музичний центр. Їм можна керувати на панелі керування центром або пультом ДУ, що підходить ще і до телевізора і відеомагнітофона. При цьому ми абсолютно не

здумуємося, які мікросхеми стоять усередині цих пристроїв. Тобто пристрою є компоненти, а пульти керування – інтерфейси.

Згадаємо визначення віртуальної функції. Це функція, що у класі нащадка має ідентичний опис (ім'я, тип, параметри), але виконує вже іншу функцію. Далі: успадкований клас є клас-нащадок, тобто при виклику віртуальної функції по вказівнику на об'єкт предок, буде викликана функція в залежності від значень вказівника. Віртуальні функції можуть не мати реалізації (абстрактні віртуальні функції). У цьому випадку при успадкуванні вони обов'язково повинні бути перевантажені. При цьому до об'єктів нащадків можна звертатися по вказівнику типу об'єкта предка. І останнє поняття – абстрактний клас – це клас, у якому є абстрактні віртуальні функції. А тепер ми впритул підійшли до визначення інтерфейсу для C++: інтерфейс це абстрактний клас, у якому є тільки абстрактні віртуальні функції.

Розглянемо приклад:

*// Визначимо клас-інтерфейс*

```
class Interface
```

```
{
```

```
    public:
```

```
        virtual int function() = 0; // Абстрактна функція
```

```
};
```

*// Клас, що підтримує інтерфейс*

```
class Implementation : public Interface
```

```
{
```

```
    public:
```

```
        virtual int function() { return a; } // Функція реалізована
```

```
    private:
```

```
        int a;
```

```
};
```

*// Тепер клас Implementation є Interface*

*// Робота з інтерфейсом*

```
int f(Interface *interf)
```

```
{
```

```

    return interf->function();
}
// Спробуємо
void main()
{
    Implementation impl;
    Interface pObject = &impl;
    cout<<f(pObject); // Увага: передаємо вказівник на інтерфейс
}

```

## Інтерфейс IUnknown

Спочатку розглянемо поняття GUID. GUID (Global Unique Identifier – Глобальний унікальний ідентифікатор) – це 128-бітове число, що є унікальним ім'ям об'єкта в системі. Кожен об'єкт, компонент, інтерфейс і т.п. мають свої ідентифікатори. Для генерації унікальних ідентифікаторів у вашій програмі можна використати програму guidgen, що поставляється разом з MS Visual C++ (Каталог <Ms VS dir>\Common\Tools\ )

Основою всієї COM-моделі є інтерфейс IUnknown, від якого успадковані всі інтерфейси, що означає, що ці три функції є у всіх об'єктах COM-моделі. Приведемо його опис:

```

class IUnknown
{
public:
    HRESULT QueryInterface(REFIID riid, LPVOID* obj);
    ULONG AddRef();
    ULONG Release();
}

```

Функція AddRef збільшує на одиницю внутрішній лічильник використання об'єкта і повертає його.

Функція Release зменшує лічильник об'єкта на одиницю і знищує об'єкт із пам'яті. Фізичне видалення об'єкта з пам'яті відбувається тільки при лічильнику рівному нулю і приховано від вас. Виклик же цієї функції означає, що ви більше не будете використовувати даний інтерфейс.

Функція `QueryInterface` є ключем до потужності COM. Вона дозволяє по заданому ідентифікатору запросити в інтерфейсу посилання на інший інтерфейс, підтримуваний об'єктом. Якщо інтерфейс не підтримується, буде повернута помилка. Інакше ви отримаєте вказівник на запитаний інтерфейс, а внутрішній лічильник об'єкта збільшиться на одиницю. Щоб звільнити інтерфейс, необхідно викликати функцію `Release`.

Кожен компонент у моделі COM зобов'язаний підтримувати інтерфейс `IUnknown`, більш того – всі інтерфейси COM від нього успадковуються.

Давайте тепер спробуємо розібратися – а що значить узагалі підтримка множини інтерфейсів і як вона реалізується. Перший спосіб – це множинне спадкування класу-реалізації від множини інтерфейсів, тоді функція `QueryInterface` буде повертати вказівник на сам об'єкт при кожному її виклику для підтримуваного інтерфейсу.

Розглянемо приклад:

```
class Interface1 : virtual public IUnknown  
[...]
```

```
class Interface2 : virtual public IUnknown  
[...]
```

```
class Implementation : public Interface1, public Interface2  
{  
    [...]  
    QueryInterface(REFIID riid, LPVOID* obj)  
    {  
        if(riid==IID_IUnknown) {  
            *obj = this;  
            AddRef();  
            return S_OK;  
        }  
        if(riid==IID_Interface1) {  
            // Зверніть увагу на перетворення  
            // типу вказівника – це ключовий момент  
            *obj = (Interface1 *)this;  
        }  
    }
```

```

    AddRef();
    return S_OK;
}
if(riid==IID_Interface2) {
    *obj = (Interface2 *)this;
    AddRef();
    return S_OK;
}
return E_NOINTERFACE;
}
[...];
};

```

Наступний спосіб називається агрегацією. У цьому випадку компоненти начебто складаються з декількох окремих класів, кожний з яких реалізує один з підтримуваних інтерфейсів і видає відповідний вказівник. При цьому повинен бути клас, що містить у собі всі реалізації і відповідальний за "дочірні" інтерфейси.

Приклад:

```

class Implementation1 : public Interface1
{
    IUnknown *parent;    // Посилання на клас, що вміщує цей

    QueryInterface(REFIID riid, LPVOID* obj)
    {
        if(riid==IID_Interface1) {
            *obj = this;
            AddRef();
            return S_OK;
        }
        else return parent->QueryInterface(riid, obj);
    }
    [...]
};

```

```

class Implementation2 : public Interface2
{
    IUnknown *parent;
    [...]
};

```

```

class Component : public IUnknown
{
    Implementation1 impl1;
    Implementation1 impl1;

```

```

    Component()
    {
        impl1.parent = this;
        impl2.parent = this;
    }

```

```

QueryInterface(REFIID riid, LPVOID* obj)
{
    if(riid==IID_IUnknown) {
        *obj = this;
        AddRef();
        return S_OK;
    }
    if(riid==IID_Interface1) {
        *obj = &impl1;
        AddRef();
        return S_OK;
    }
    if(riid==IID_Interface2) {
        *obj = impl2;
        AddRef();
        return S_OK;
    }
}

```

```

        return E_NOINTERFACE;
    }
    [...]
};

```

У прикладі спеціально описані класи для кожного інтерфейсу, однак у реальності і вже реалізовані класи з інтерфейсами можуть бути частиною вашого інтерфейсу (наприклад файлові потоки). При цьому вказівник на об'єкт-контейнер передається при створенні дочірнього. Це ми розглянемо нижче. Отже, ми вже знаємо, як влаштовані COM-об'єкти. Тепер прийшов час перейти до технічної реалізації COM-бібліотеки і клієнта, що використовує її.

### Створення COM-клієнта

Для використання COM-компонентів нам знадобляться наступні функції:

```

HRESULT CoInitialize(
    LPVOID pvReserved // Запозичено, повинне бути NULL
);

```

```

void CoUninitialize( );

```

Функція CoInitialize проводить ініціалізацію COM-системи для користувальницької програми. Ви не можете створювати COM-об'єкти стандартними методами ОС без виклику цієї функції. Функція CoUninitialize звільняє усі використовувані COM-бібліотеки і проводить дії , зворотні CoInitialize.

Розглянемо процедуру створення COM-компонента. Кожен COM-компонент має деяку внутрішню реалізацію, недоступну ззовні і надає набір інтерфейсів, що дозволяють з ним працювати. Більш того, не маючи посилань на інтерфейси одержати доступ до COM-компонента неможливо. Це як пральна машина – ви можете натискати кнопки інтерфейсу на передній панелі (інтерфейс керування) і завантажувати білизну і закривати дверця (інтерфейс вводу-виводу), але відкрити машину і довідатися, як це все працює ви не маєте права.

Отже, для роботи з пральною машиною вам необхідно: сама пральна машина (це компонент, він має ідентифікатор класу, інакше CLSID) і два інтерфейси (вони мають ідентифікатори інтерфейсів – IID). Для роботи з машиною вам необхідно мати доступ як мінімум до одного з інтерфейсів. У випадку COM-об'єктів ситуація майже аналогічна – створюючи об'єкт, ви вказуєте два ідентифікатори – CLSID для створення саме цього об'єкта і IID для роботи з ним.

Розглянемо API для створення інтерфейсів:

```
STDAPI CoCreateInstance(  
    REFCLSID rclsid, // Ідентифікатор класу (CLSID) компоненти  
    LPUNKNOWN pUnkOuter, // Вказівник на батьківський об'єкт при  
        // агрегації (див вище)  
    DWORD dwClsContext, // Оточення виконання компоненти  
    REFIID riid, // Ідентифікатор необхідного нам інтерфейсу  
    LPVOID *ppv // Адреса змінної, що одержує вказівник на  
        // необхідний інтерфейс  
);
```

Функція CoCreateInstance створює об'єкт із заданим ідентифікатором rclsid і запитує в нього інтерфейс з ідентифікатором riid. Якщо об'єкт не надає запитаний інтерфейс, він не створюється і функція повертає помилку E\_NOINTERFACE. Оточення виконання це прапор, що вказує на місце розташування коду об'єкта: на локальній чи машині на вилученій.

Приклад створення об'єкта пральної машини:

```
ISiemensWashControl *wash;  
HRESULT r;  
CoInitialize(NULL);  
r = CoCreateInstance(CLSID_SiemensMarherita2m, NULL,  
    CLSCTX_INPROC_SERVER, IID_ISiemensWashControl, &wash);  
if(r!=S_OK) { cout<<"Error!\n"; return 0; }  
[...]
```

## Створення COM-сервера

Як же це все влаштовано? COM-сервер представляє собою бібліотеку, яка завантажується і надає спеціальні функції. Компоненти, що знаходяться в бібліотеці, спеціальним чином реєструються в системному реєстрі для того, щоб при виклику CoCreateInstance можна було визначити їхнє місцезнаходження. Це механізм дозволяє уникнути твердої прив'язки до конкретної бібліотеки і дозволяє в остаточному підсумку зберігати компоненти де завгодно, аж до віддаленої машини в Інтернеті. Розглянемо тепер це окремо.

### Інтерфейс IClassFactory

Цей інтерфейс призначений для стандартизації створення COM-компонентів і використовується з цією метою менеджером COM ОС. Інтерфейс успадкований від інтерфейсу IUnknown і крім стандартних трьох, включає наступні функції:

```
HRESULT CreateInstance(  
IUnknown * pUnkOuter, // Вказівник на батька в агрегації  
REFIID riid,          // Посилання на інтерфейс (GUID)  
void **ppvObject      // Адреса змінної, для одержання інтерфейсу  
);
```

```
HRESULT LockServer(  
BOOL fLock //Increments or decrements the lock count  
);
```

Функція CreateInstance безпосередньо створює екземпляр компонента і передає вказівник на один з його інтерфейсів. Її параметри аналогічні по призначенню параметрам CoCreateInstance. Функція LockServer чи збільшує зменшує в залежності від параметра внутрішній лічильник бібліотеки, для заборони передчасного вивантаження бібліотеки з пам'яті.

Розглянемо приклад типової фабрики класів, заодно і приклад COM-об'єкта:

```
int InternalLockCounter = 0;
```

```

class MyObjectFactory : public IClassFactory
{
public:
    HRESULT QueryInterface(REFIID riid, LPVOID* obj)
    {
        if(riid==IID_IUnknown || riid==IID_IClassFactory) {
            *obj = this;
            AddRef();
            return S_OK;
        }
        else return E_NOINTERFACE;
    }
    ULONG AddRef()
    { return ++m_RefCount; }

    ULONG Release()
    {
        if(--m_RefCoun==0) delete this;
        return m_RefCount;
    }

    HRESULT CreateInstance(IUnknown * pUnkOuter, REFIID riid,
                          void ** ppvObject)
    {
        // Агрегації в нас немає
        if(pUnkOuter!=NULL) return E_NOAGGREGATION;
        MyObject *mo = new MyObject;
        // Самі компоненти прийнятий успадковувати від інтерфейсів,
        // тому робимо так:
        if(mo->QueryInterface(riid, ppvObject)!=S_OK) {
            delete mo;
            return E_NOINTERFACE;
        }
        return S_OK;
    }
}

```

```

    }

    HRESULT LockServer(BOOL fLock)
    {
        if(fLock) InternalLockCounter++;
        else InternalLockCounter--;
        return S_OK;
    }

private:
    int m_RefCount;
};

```

### **Імпортовані бібліотекою підпрограми**

Динамічна бібліотека з COM-компонентами звичайно імпортує чотири функції:

```

STDAPI DllGetClassObject(REFCLSID rclsid, REFIID riid,
                        void** ppObject);
HRESULT DllCanUnloadNow();
STDAPI DllRegisterServer();
STDAPI DllUnregisterServer();

```

Нам особливо необхідна DllGetClassObject. Ця функція створює для заданого параметром rclsid фабрику об'єктів, описану вище.

Приклад реалізації цієї функції:

```

STDAPI DllGetClassObject(REFCLSID rclsid, REFIID riid,
                        LPVOID FAR* ppv)
{
    if(rclsid==CLSID_Lab5Problem) {
        MyObjectFactory *factory = new MyObjectFactory;
        if(factory->QueryInterface(riid, ppv)!=S_OK) {
            delete factory;
            return E_NOINTERFACE;
        }
    }
}

```

```

        else return S_OK;
    }
    return E_NOINTERFACE;
}

```

## Правила реєстрації COM-сервера

Для реєстрації COM-сервера необхідно занести в розділ реєстру такі ключі:

`\HKEY_CLASSES_ROOT\CLSID\{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxx}`.

Де у фігурних дужках записаний ідентифікатор вашого компонента (CLSID) `..\{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxx}\InprocServer32`. У полі за замовчуванням повинен знаходитися адреса бібліотеки, що містить компонент.

Приклад файла реєстру для реєстрації компонента:

*REGEDIT4*

*[HKEY\_CLASSES\_ROOT\CLSID\{AD6F0E80-229F-11D7-8C7B-C83DC4D9034f}]*

*[HKEY\_CLASSES\_ROOT\CLSID\{AD6F0E80-229F-11D7-8C7B-C83DC4D9034F}\InprocServer32]*

*@="з:\windows\system\lab5lib.dll"*

## Як все працює

Коротко процес роботи з COM-компонентом можна описати так. Спочатку клієнт викликає `CoInitialize`, що ініціалізує COM-підсистему для процесу клієнта. Далі клієнт викликає `CoCreateInstance`. Система знаходить у системному реєстрі відповідний запис і намагається завантажити бібліотеку з компонентом. Якщо завантаження пройшло успішно, система викликає функцію `DllGetClassObject` для створення фабрики класів, за допомогою якої у свою чергу створюються всі необхідні екземпляри. Навіщо потрібна фабрика класів? Вона стандартизує метод створення компонентів, роблячи

саме створення СОМ-викликом, що дозволяє однаково легко створювати компоненти як локально, так і на віддаленій машині, в іншому потоці і т.п.

## 4.2. Порядок виконання лабораторної роботи

По легенді Ви працюєте в команді розробників, що створює математичний пакет, здатний вирішувати різні математичні задачі. Кожен Ваш компонент реалізує рішення однієї з задач і надає деякі стандартні інтерфейси для взаємодії з користувачем і власне рішенням задачі. Для здачі роботи Вам необхідно продемонструвати роботи Ваших компонентів на Вашому клієнті, після чого викладач перевірить їх на своєму власному клієнті, що буде прототипом математичної системи майбутнього.

Ідентифікатори й описи інтерфейсів дивіться у файлах, прикладених до системи. Приклад реалізації рішення квадратного рівняння також присутній – настійно рекомендується його переглянути. Зверніть увагу на файл StdAfx.h – у ньому знаходиться список файлів, що включаються, для роботи з СОМ-моделлю.

### Опис інтерфейсів роботи

```
#ifndef __lab5lib_h__included  
#define __lab5lib_h__included
```

```
#include "StdAfx.h"
```

```
static const GUID IID_IInOut =  
{ 0xad6f0e81, 0x229f, 0x11d7, { 0x8c, 0x7b, 0xc8, 0x3d, 0xc4, 0xd9, 0x3,  
0x4f } };
```

```
static const GUID IID_IProblem =  
{ 0xad6f0e82, 0x229f, 0x11d7, { 0x8c, 0x7b, 0xc8, 0x3d, 0xc4, 0xd9, 0x3,  
0x4f } };
```

```
// Інтерфейс взаємодії з користувачем  
class IInOut : public IUnknown
```

```

{
    public:
        // Вводить умови задачі з консолі
        virtual HRESULT STDMETHODCALLTYPE Input() = 0;

        // Виводить опис задачі
        virtual HRESULT STDMETHODCALLTYPE OutProblem() = 0;

        // Виводить рішення задачі на екран
        virtual HRESULT STDMETHODCALLTYPE OutSolution() = 0;
};

class IProblem : public IUnknown
{
    public:
        // Вирішує задачу
        virtual HRESULT STDMETHODCALLTYPE Solve() = 0;
        // Встановлює параметр задачі з номером nIndex
        // Наприклад це номер коефіцієнта многочлена
        // S_FALSE якщо неправильний чи індекс значення
        virtual HRESULT STDMETHODCALLTYPE SetParam(int nIndex, float
Value) = 0;

        // Повертає S_OK і параметр задачі з номером nIndex у pValue
        // S_FALSE якщо неправильний індекс
        virtual HRESULT STDMETHODCALLTYPE GetParam(int nIndex,
float *pValue) = 0;

        // Повертає S_OK і параметр задачі з номером nIndex у pValue
        // Наприклад номер кореня багаточлена
        // S_FALSE якщо неправильний індекс
        virtual HRESULT STDMETHODCALLTYPE GetAnswer(int nIndex,
float *pValue) = 0;
};#endif

```

## Вимоги до реалізації

Якщо в умові сказано «вирішити  $a \cdot x^2 + b \cdot x + c = 0$ » це означає, що потрібно написати COM-сервер з інтерфейсами `InOut` і `IProblem`, що при виклику `InOut::Input()` запитує коефіцієнти  $a$ ,  $b$ ,  $c$ , при виклику `IProblem::Solve()` вирішує рівняння, при виклику `InOut::OutSolution()` виводить його рішення і т.д. (див. приклад).

Те ж відноситься і до: «для даного CLSID визначити, чи зареєстрований даний COM-об'єкт у системі?». `InOut::Input()` повинен запитувати CLSID і т.д. Крім цього необхідно написати клієнт, що створює instance об'єкта і робить відповідні виклики. Вам фактично потрібно скопіювати приклад і вставити туди розв'язок своєї задачі... Але є маленький нюанс: компоненти кожного варіанту повинні лежати в одній DLL.

## 4.3 Завдання

Якщо все написане вище здається Вам занадто складним, то розберіть прикладений до документації приклад дуже простого робочого COM-компонента, а потім ЗНОВУ прочитайте теорію.

Таблиця 4.1 – Варіанти

| № | 1                                                               | 2                                | 3                                                                                        | 4                                                                | 5                                                                                                                                                                                                                                                                                            |
|---|-----------------------------------------------------------------|----------------------------------|------------------------------------------------------------------------------------------|------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|   | В середовищі Visual Studio C++<br>Створити клієнт автоматизації | Створити текстовий файл з даними | По запиту з клієнта створити робочу книгу Excel, використовуючи створений текстовий файл | По запиту з клієнта в робочій книзі Excel провести обробку даних | Дані(матриця, рядки тощо) та результат обчислень в робочій книзі Excel за запитом клієнта відобразити у створеному файлі Word як звіт з «шапкою» на зразок<br>Результат обчислень<br><Перекодування того – то> або <Обчислення чогось> або<br><Видалення чогось > і так далі згідно варіанту |

Таблиця 4.2 – Сценарій роботи

| №  | Суть обробки                                                                                                                                                                                                                                                                                                                                                | Особливості обробки                                                                            |
|----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------|
|    | Перекодувати вхідний текстовий файл, що містить поблочно англійські, російські і українські літери в кодуванні Windows-1251, в файл, який містить текстове представлення UNICODE-кодування символів. Наприклад символ апострофа 'має бути виведений в новий файл ' тому що його UNICODE значення 39, а українська буква а буде представлена &#39 як а.      | Перекодувати тільки англійські букви                                                           |
| 2. | Перекодувати вхідний текстовий файл, що містить поблочно англійські, російські та українські літери в кодуванні Windows-1251, в файл, який містить текстове представлення UNICODE-кодування символів. Наприклад символ апострофа 'має бути виведений в новий файл як' тому що його UNICODE значення 39, а російська буква а буде представлена &#39 як а.    | Перекодувати тільки українські букви                                                           |
| 3. | Перекодувати вхідний текстовий файл, що містить поблочно англійські, російські і українські літери в кодуванні Windows-1251, в файл, який містить текстове представлення UNICODE-кодування символів. Наприклад символ апострофа 'повинен бути виведений в новий файл як' тому що його UNICODE значення 39, а російська буква а буде представлена &#39 як а. | Перекодувати англійські та українські букви, а потім в іншому файлі залишити тільки англійські |
| 4. | Перекодувати вхідний текстовий файл, що містить поблочно англійські, російські і українські літери в кодуванні Windows-1251, в файл, який містить текстове представлення UNICODE-кодування символів. Наприклад символ апострофа 'повинен бути виведений в новий файл як' тому що його UNICODE значення 39, а російська буква а буде представлена &#39 як а. | Перекодувати всі символи, а потім в іншому файлі залишити тільки російські                     |
| 5. | Перекодувати вхідний текстовий файл, що містить поблочно англійські, російські і українські літери в кодуванні Windows-1251, в файл, який містить текстове представлення UNICODE-кодування символів. Наприклад символ апострофа 'повинен бути виведений в новий файл як' тому що його UNICODE значення 39, а російська буква а буде представлена &#39 як а. | Перекодувати всі символи, а потім в іншому файлі залишити тільки заголовні                     |

## **Контрольні запитання**

1. Що таке СОМ інтерфейс?
2. Як визначається клас-інтерфейс?
3. Що собою являє інтерфейс IUnknown?
4. Який компонент у моделі СОМ використовує IUnknown?
5. Опишіть API функції створення СОМ-клієнта.
6. Яку роль виконує інтерфейс IClassFactory?
7. Опишіть засоби імпорту бібліотеки з СОМ-компонентами.
8. Опишіть програмні механізми реєстрації СОМ-сервера.

### **Рекомендована література**

1. Sdk Win32 Api Programming <https://pdfprodocs.vip/download/4677773-sdk-win32-api-programming> Uploaded: 2022 Jul 03, 02:12
2. Системне програмування: конспект лекцій. Частина 1: Використання командного інтерпретатора CMD та вбудованих системних утиліт ОС Windows (Ukrainian Edition) Paperback – July 23, 2019 Ukrainian Edition by Артем Соколов (Author).
3. Pavel Yosifovich Windows 10 System Programming, Part 2 (2021) Режим доступу: <https://p302.zlibcdn.com/dtoken/fb4b7ff4e5bfb6236e22136c0199877d>.

### **Додаткова література**

4. STRUCTURED COMPUTER ORGANIZATION ANDREW S. TANENBAUM Vrije Universiteit Amsterdam, The Netherlands TODD AUSTIN University of Michigan Ann Arbor, Michigan, United States 801 p. SIXTH EDITION 2013.
5. Win32 Api Reference, Режим доступу: <https://pdfprodocs.vip/download/4677773-win32-api-reference> Uploaded: 2022 Jul 03, 01:39.
6. Johnson M. Hart Windows System Programming Fourth Edition, Режим доступу: <https://fr.ua1lib.org/book/834488/1b7630?dsourc=recommend>
7. Bob Quinn, David Shute, Windows sockets network programming Режим доступу: <https://fr.ua1lib.org/book/490687/f2e413?dsourc=recommend>
8. Pavel Yosifovich Windows 10 System Programming, Part 1 (2020) Режим доступу: <https://edu.anarchocopy.org/other/Windows/Windows%2010%20System%20Programming.pdf>
9. Pavel Yosifovich Windows Kernel Programmin 2019 g Режим доступу: <https://p303.zlibcdn.com/dtoken/25b9ac5a56653852add3f8b421822a75>
10. Andrea Allievi, Alex Ionescu, Mark Russinovich, David Solomon Windows Internals, Part 2, 7th Edition (2021) Режим доступу: <https://fr.ua1lib.org/book/17356100/092dac?dsourc=recommend>
11. Jeffrey Richter Programming Applications for Microsoft Windows ITPro collection Microsoft programming series, Seven Edition 1104 p.