МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ
УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

.

МЕТОДИЧНІ ВКАЗІВКИ

до лабораторних занять з дисципліни
«ОПЕРАЦІЙНІ СИСТЕМИ»
за спеціальністю
121 – ІНЖЕНЕРІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
освітньо-професійна програма «Програмна інженерія»
для здобувачів усіх форм навчання

ЗАТВЕРДЖЕНО
кафедрою ПІ
Протокол № 6
від 9.11.21

ХАРКІВ 2021

Методичні вказівки до лабораторних занять з дисципліни «Операційні системи» за спеціальністю 121 – Інженерія програмного забезпечення, освітньо-професійна програма «Програмна інженерія» для здобувачів усіх форм навчання / Упоряд. Качко О.Г., Мельникова Р.В. − Харків: ХНУРЕ, 2021. −57 с.

Упорядники: О.Г. Качко, Р.В. Мельнікова

Рецензент В.О. Гороховатський, проф. каф. Інформатики ХНУРЕ

ЗМІСТ

ВСТУП

Мета методичних вказівок – оказати допомогу студентам при підготовці, виконанні та оформленні результатів виконання лабораторних робіт по дисципліні, основним призначенням якої є вивчення принципів побудови сучасних операційних систем та найбільш ефективного використання їх функцій при розробці програм та їх застосуванню.

Лабораторні роботи по дисципліні вчать практичному використанню теоретичних положень, які вивчені на лекціях, закріплюють та розширяють практичні навики, що були отримані на практичних заняттях по курсу.

Кожна лабораторна робота потребує попередньої підготовки, а саме:

- вивчення теоретичного матеріалу;
- розробки алгоритмів вирішення задач, які треба виконати під час лабораторної роботи;
- розробка тестів для перевірки програм.

Перед виконанням наступної лабораторної роботи рекомендуємо вивчити відповідну тему в [8].

Студент самостійно виправляють помилки в програмі.. Сама помилка та дії, необхідні для її виправлення, заносяться в звіт по лабораторній роботі. Найбільш характерні помилки обговорюються з групою в ході виконання лабораторних робіт.

Звіт повинен включати в себе:

- назву лабораторної роботи;
- мету лабораторної роботи;
- завдання і тексти програм, тести, помилки, які знайдені при виконанні тестів, результати виконання програм, висновки.

В якості шаблону для підготовки звіту треба використовувати цей документ в електронному вигляді. На титульному листі звіту треба написати автора звіту.

Без наявності електронної копії звіту для поточної лабораторної роботі і всіх попередніх робіт лабораторна робота не приймається. Здача поточної лабораторної роботи може бути виконана в день виконання лабораторної роботи або під час першої половини наступної лабораторної роботи. Лабораторна робота, яка здається невчасно, не може бути

оцінена високою оцінкою. При отриманні заліку по лабораторним роботам в кінці семестру студент повинен представити викладачу повний звіт по усім лабораторним роботам семестру в електронному вигляді.

Всі лабораторні роботи здаються індивідуально.

Бажано застосовувати GitHub для зберігання програмного коду, розробленого під час виконання лабораторних робіт

1 РОЗРОБКА УНІВЕРСАЛЬНИХ ДОДАТКІВ ДЛЯ РІЗНИХ ТИПІВ КОДУВАНЬ СИМВОЛЬНОЇ ІНФОРМАЦІЇ

1.1 Мета роботи

ASCII кодування використовує один байт для завдання одного символу. Цього зовсім не достатньо для завдання текстів, в яких є речення на різних мовах. UNICODE кодування застосовує два байта при кодуванні одного символу. При такому кодуванні максимальна кількість кодів символів 65536 в порівнянні з 256 для ASCII кодування, що забезпечує можливість використання усіх наявних на сьогодні мов. Більшість текстових файлів на сьогодні мають ASCII кодування. Кодування імен файлів залежить від наявної файлової системи, в електронних листах можна використовувати обидві типа кодування. таким чином, обидва способи кодування використовуються. Метою даної лабораторної роботи є навчитися опрацьовувати тексти для обох типів кодування, при чому сама програма не повинна залежати від обраного способу.

1.2 Підготовка до роботи. Теоретичні положення

1.2.1 Створення консольного додатку Windows C++

Після запуску Visual Studio початкова сторінка може мати наступний вид (див. рис. 1.1).

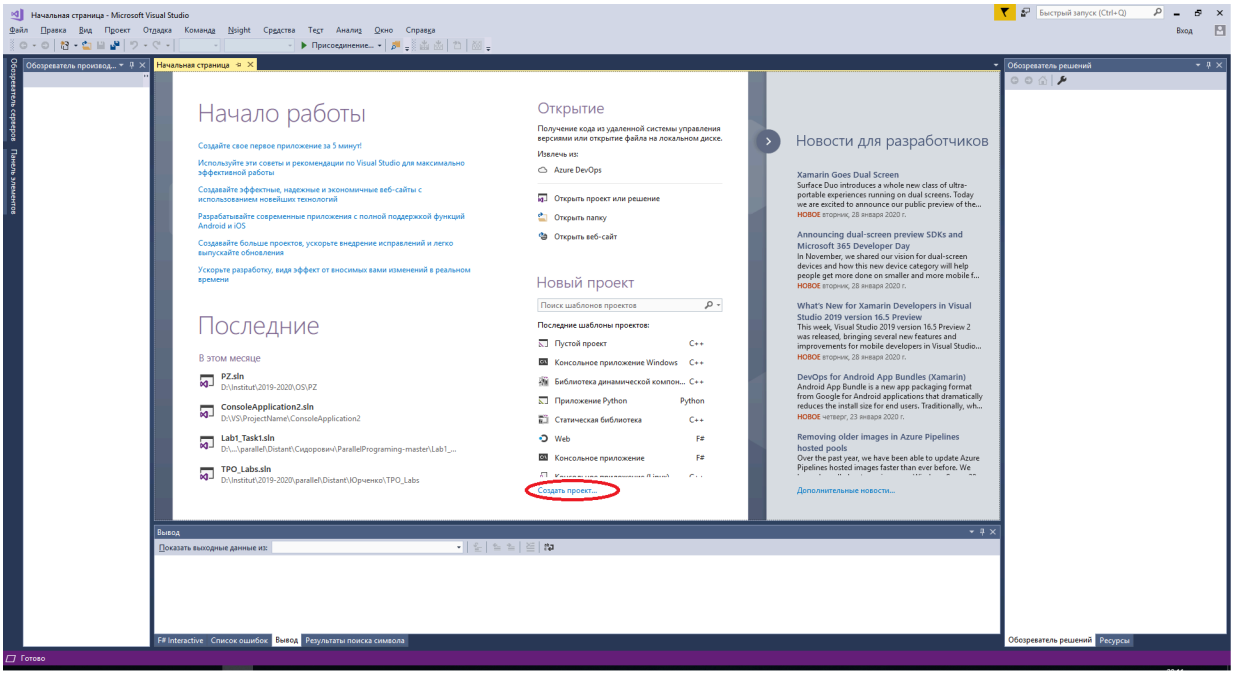

Рисунок 1.1 - Вигляд початкової сторінки після запуску Visual Studio

Початкова сторінка може відрізнятися від наведеної, але на ній присутня можливість створення нового проекту (на рисунку 1.1 відповідний пункт «Создать проект» позначено).

Після обрання цього пункту отримаємо вікно (див. рис. 1.2), заповнюємо назви для імені проекту (Имя), обираємо папку, де буде розташовано проект (Расположение) та обираємо пункт, позначений як Консольное приложение Windows, та натискаємо кнопку ОК.

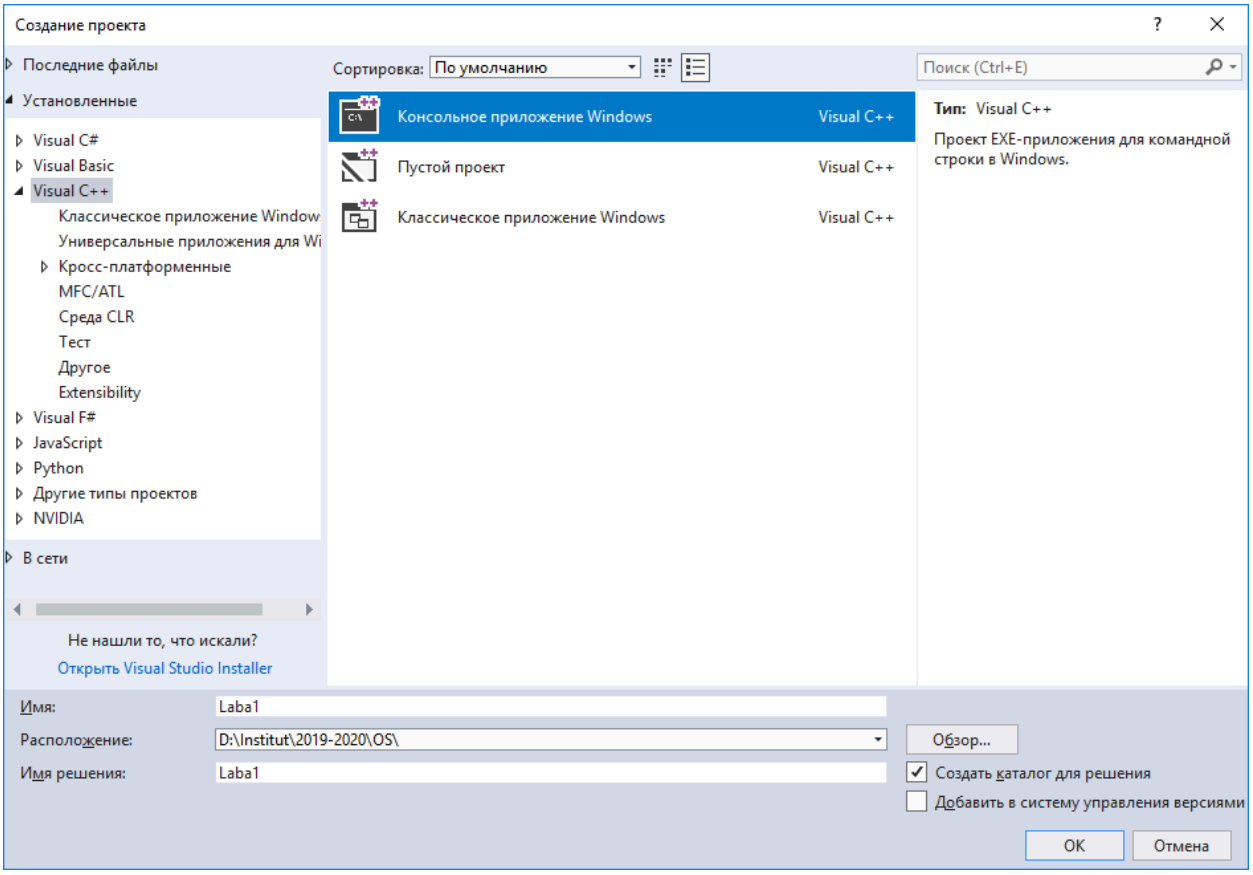

Рисунок 1.2 -– Вікно створення нового проекту

Після створення проекту треба обрати платформу (32- бітна або 64 бітна). По замовченню створюється 32-бітний додаток. При виконанні лабораторних робіт будемо застосовувати 64-бітну платформу, для цього в полі Платформа рішення обираємо 64-бітну платформу.

В папці, обраної для проекту, отримаємо папку з іменем проекту та файл <Ім’я проекту >.sln. Останній файл містить інформацію про створене рішення, в якому знаходиться наш проект і можуть знаходитися інші проекти.

В папці з ім’ям проекту знаходяться службові файли та файли з розширенням cpp та h. В програмах, які складаються для лабораторних робіт, службові файли не

застосовуються, тому ці файли змінюватися не будуть, але видаляти їх не можна.

Файл з іменем проекту та розширенням .cpp містить код[1]:

```
#include <iostream>
int main()
{
   std::cout << "Hello World!\n";
}
```

Цей код виводить рядок Hello World! .

Далі файл, як коментар, містить інформацію про створення проектів.

Саме цей файл буде доповнюватись в першій лабораторній роботі і застосовуватись в інших роботах.

1.2.2 Типи даних char і wchar_t. Універсальне завдання типу

При використанні ASCII кодування використовується тип даних char. Даному типа char виділяється один байт. Приклади використання цього типу:

```
char symbol  = ‘x’;
char array  [] = “This is array”;
```

При використанні UNICODE кодування використовується тип даних wchar_t. Даному типа wchar_t виділяється два байти. Приклади використання цього типу:

```
wchar_t  symbol  = L‘x’;
wchar_t array  [] = L“This is array”;
```

Для створення універсального тексту програми, який можна використовувати для обох типів кодування можна використовувати макроси.

Приклад макросу для універсальної об’яви типу даних і їх ініціалізації:

```
#ifdef UNICODE
        typedef         wchar_t         TCHAR;
        #defineTEXT(a)      L##a
#else
        typedef         char            TCHAR;
```

[1] Код може відрізнятися для різних версій Visual Studio

```
        #define     TEXT(a)     a
#endif
```

В цьому макросі визначено універсальний тип TCHAR, значення якого залежить від того, визначена змінна UNICODE чи ні. Визначено макрос TEXT, за допомогою якого до визначення літералу додається буква L або ні.

Розгляньте цей макрос! Він повинен бути зрозумілим!

Використаємо цей макрос для визначення символу та рядка в універсальному вигляді.

```
TCHAR     symbol  = TEXT('x');
TCHAR     array  [] = TEXT("This is array");
```

Цей код відповідає ASCII кодуванню, якщо не визначено змінної UNICODE, і UNICODE, якщо визначена ця змінна.

Ці макроси можна визначити в своїй програмі. Замість цього можна підключити до програми файл заголовків tchar.h, в якому визначені всі макроси, які забезпечують застосування універсальних типів в програмі (дивись пункт 1.2.3)

1.2.3 Функції для роботи з рядками для ASCII і UNICODE кодувань

Функції для роботи з рядками для ASCII кодування визначено в файлі заголовків string.h. Більшість функцій починається з префіксу str, наприклад, strcpy, strcat, strlen,… .

Функції для роботи з рядками для UNICODE кодування визначено в тому ж файлі заголовків string.h. Більшість функцій починається з префіксу wcs, наприклад, wcscpy, wcscat, wcslen,… .

Визначимо універсальну функцію, наприклад, для визначення довжини рядка (в символах):

```
#ifdef UNICODE
#define_tcslen     wcslen
#else
#define_tcslen     strlen
#endif
```

Аналогічно можна визначити усі функції для роботи з рядками та з потоками даних:

```
using namespace std;
#ifdef UNICODE
#define _tcout wcout
#else
#define _tcout cout
#endif
```

1.2.4 Файл tchar.h. Пошук імен для універсальних функцій

Макроси, які наведені вище, а також багато інших, наведені в файлі tchar.h, якій необхідно підключити для використання універсального кодування. Для застосування універсального кодування необхідно підключити цей файл за допомогою директиви #include < tchar.h >

Для того, щоб знайти ім’я універсальної функції необхідно відкрити файл tchar.h, та знайти в ньому визначення функції для ASCII кодування. Це визначення має такий же формат, який ми використовували для визначення функції strlen. Отримайте універсальне ім’я функції.

1.2.5 Визначення типу тексту і перетворення типу кодування тексту

Нехай задано текст, наприклад, вміст файлу. Необхідно визначити, яке кодування використовується для завдання цього тексту.

Для цього можна використовувати функцію IsTextUnicode:

BOOL IsTextUnicode(CONST VOID* pBuffer, int cb, LPINT lpi);

де:

- pBuffer – буфер з символами;
- cb - кількість символів в буфері;
- lpi - ознаки, по яким визначається тип кодування. Кожна ознака задається одним бітом. На вході зазвичай задають 1 в усіх бітах, тобто число -1. На виході в 1 установлені ті біти, ознаки для яких підтверджені.

Відкрийте допомогу для цієї функції і розгляньте ці ознаки!

Функція повертає TRUE, якщо текст по більшості ознак типу Unicode, FALSE для типу ASCII.

Функція для перетворення ASCII в Unicode MultiByteToWideChar :

```
int MultiByteToWideChar(
UINT CodePage,            // Кодова сторінка, звичайно задається CP_ACP
DWORD dwFlags,            // 0
LPCSTR lpMultiByteStr,    // Рядок, який перетворюється
int cbMultiByte,          // Розмір (в байтах)
LPWSTR lpWideCharStr,     // Рядок - результат
int cchWideChar           // Розмір в wchar_t символах.
);
```

Функція повертає кількість символів в рядку-результату.

Функція для перетворення Unicode в ASCII

```
int WideCharToMultiByte(
  UINT CodePage,            // Кодова сторінка, звичайно задається CP_ACP
  DWORD dwFlags,            // 0
  LPCWSTR lpWideCharStr,    // Рядок, який перетворюється
  int cchWideChar,          // // Розмір
  LPSTR lpMultiByteStr,     // Рядок - результат
  int cbMultiByte,          //  Розмір
  LPCSTR lpDefaultChar,     // Адреса символу, яким заміняється
        //символ, що перетворюється, якщо він не може бути
        // відображений. Дорівнює NULL, якщо використовується
        // символ за  замовчуванням.
  LPBOOL lpUsedDefaultChar // Покажчик на прапорець, який вказує на
        // те, чи використовувався символ за замовчуванням у
        // попередньому  параметрі. Може бути NULL.
);
```

Функція повертає кількість символів в рядку, який є результатом.

1.2.6 Введення – виведення даних за допомогою кіриліци

1. Встановити кодову сторінку для кіриліци

SetConsoleOutputCP (1251); - для виведення даних

SetConsoleCP (1251); для введення даних

2 Визначити макрос для виведення даних типу TCHAR

```
using namespace std;
#ifdef UNICODE
#define _tcout wcout
#else
#define _tcout cout
#endif
```

Або застосовувати для введення виведення _tscanf, _tprintf

3 Спробувати вивести текст

```
TCHAR tstr[] = TEXT("Привіт");
_tcout << tstr << _T("\n");\
```

Якщо текст виводиться невірно відкоректувати реєстр

Win+R і виконати програму regedit

Перйти в розділ реєстру

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Nls\CodePage
```

В кінці розділу знайти параметр ACP. Там скоріше за все стоїть 1252 (тобто встановлювалась ОС для англійської мови). Замінити значення цього параметру на 1251

Перезавантажити ОС

1.3 Завдання до лабораторної роботи. Порядок виконання

1. Створіть консольний додаток для C++ програм;
2. Складіть оператори для програмної перевірки типу кодування, заданого за замовчуванням (визначте довжину в байтах типу TCHAR);
3. Визначте тип кодування по заданих макросах в командному рядку. Для цього визначимо командний рядок для трансляції програми:

Свойства конфигурации→ C/C++→Командная строка

Усі змінні, які задаються при трансляції, задаються параметром /D. Якщо визначено змінну UNICODE, то використовується режим UNICODE, якщо така змінна не задана, то використовується режим ASCII;

4. Перемикніть режим завдання символу на протилежний. Для перемикання режиму з UNICODE в режим ASCII використовують:

Свойства конфигурации→ Общие→Набор символов→Использовать многобайтовую кодировку

Для перемикання з режиму ASCII в режим UNICODE використовують:

Свойства конфигурации→Общие→Набор символов→Использовать набор символов Юникода;

5. Після перемикання режиму знову перевірте тип символу за замовчуванням та командний рядок.

(Після трансляції!!!);

6. Задайте ПІБ членів своєї сім'ї в ASCII та виведіть задані значення. Для виведення букв кирилиці необхідно встановити локальні режими.

Зверніть увагу на особливості виведення рядків з кириліцею

7. Переведіть задані рядки в UNICODE за допомогою функції (MultiByteToWideChar);

8. Виведіть отриманий масив. Перевірте можливість виведення кожним з 3-х способів:

- функція _tprintf, якщо встановлено локальний режим;
- потік wcout, якщо встановлено локальний режим;
- функція MessageBox (незалежно від встановленого режиму)

int MessageBox (0, <Рядок, який виводиться>, <Заголовок вікна>, MB_OK)) ;

9. Виконайте упорядкування масиву рядків, заданих в UNICODE. Для сортування використовувати універсальну стандартну функцію qsort та шаблонну функцію sort (для отримання найвищої оцінки):

```
void qsort(
void *base,          // Масив, що упорядковуємо
size_t num,          // Кількість елементів масиву;
size_t width,        // Ширина елементу масиву;
int (__cdecl *compare )(const void *, const void *) );
                     // Функція для порівняння елементів масиву
```

Використання шаблонної функції sort вивчити самостійно.

10. Виконайте зворотне перетворення масиву з Unicode в ASCII;

11. Виведіть отриманий результат;

12. Завдання для отримання найвищої оцінки. Задано текстовий файл. Не

залежно від способу кодування символів в цьому файлі, переставити ці символи в зворотному порядку. Для роботи з файлом можна використовувати довільні засоби роботи з файлами в C++, які треба вивчити самостійно;

1.4 Зміст звіту

Звіт має містити:

- повний опис усіх типів і функцій, які використовуються для забезпеченості універсальності кодування;
- повний опис усіх типів і функцій, які використовуються для виведення інформації в консольному режимі на російській (українській) мові;
- тексти розробленої програми з коментарями;
- тести для перевірки правильності програми;
- Висновки по роботі

1.5 Контрольні запитання і завдання

1. Навіщо створювати універсальні додатки для роботи з ASCII, UNICODE?
2. Як забезпечити універсальність об’явлення символу?
3. Які функції використовуються для введення та виведення універсальних типів даних?
4. Як встановити локальні характеристики для виведення інформації російською(українською) мовами?
5. Що треба змінити в програмі для упорядкування масиву ASCII символів?

2 СТВОРЕННЯ ТА ВИКОРИСТАННЯ БІБЛІОТЕК. ЧАСТИНА 1

2.1 Мета роботи

Вивчити прийоми та методи створення та використання динамічних бібліотек

2.2 Підготовка до роботи

2.2.1 Загальна характеристика динамічних бібліотек

Динамічні бібліотеки (Dynamic Link Library - DLL), файли з розширенням DLL, завантажуються під час завантаження модуля, який використовує бібліотеку, або під час його виконання.

Переваги DLL:

- бібліотеки не залежать від середовища, в якому вони створені. Так бібліотека, яку було створено в середовищі C++ Builder, можна використовувати в середовищі Visual Studio та навпаки. Бібліотеки, створені в середовищі C++, можна застосовувати в програмі на C#, java;
- при зміні коду бібліотеки не потрібна повторна компоновка додатків, які використовують цю бібліотеку, ось чому операційна система використовує цей тип бібліотек для модулів, які можуть змінюватися в залежності від версії та в разі помилок;
- якщо декілька додатків використовують одну і ту ж бібліотеку, копія цієї бібліотеки зберігається в пам’яті тільки один раз.

Недоліки DLL:

- окрім програми, яка виконується необхідно мати додатковий модуль – саму бібліотеку;
- функції DLL використовувати складніше, ніж функції статичної бібліотеки.

Решта переваг та недоліків DLL залежать від режимів використання бібліотеки цього типу.

2.2.2 Створення DLL

Розглянемо формування DLL бібліотеки для Visual Studio, для інших середовищ бібліотеки створюються подібно (рис. 2.1).

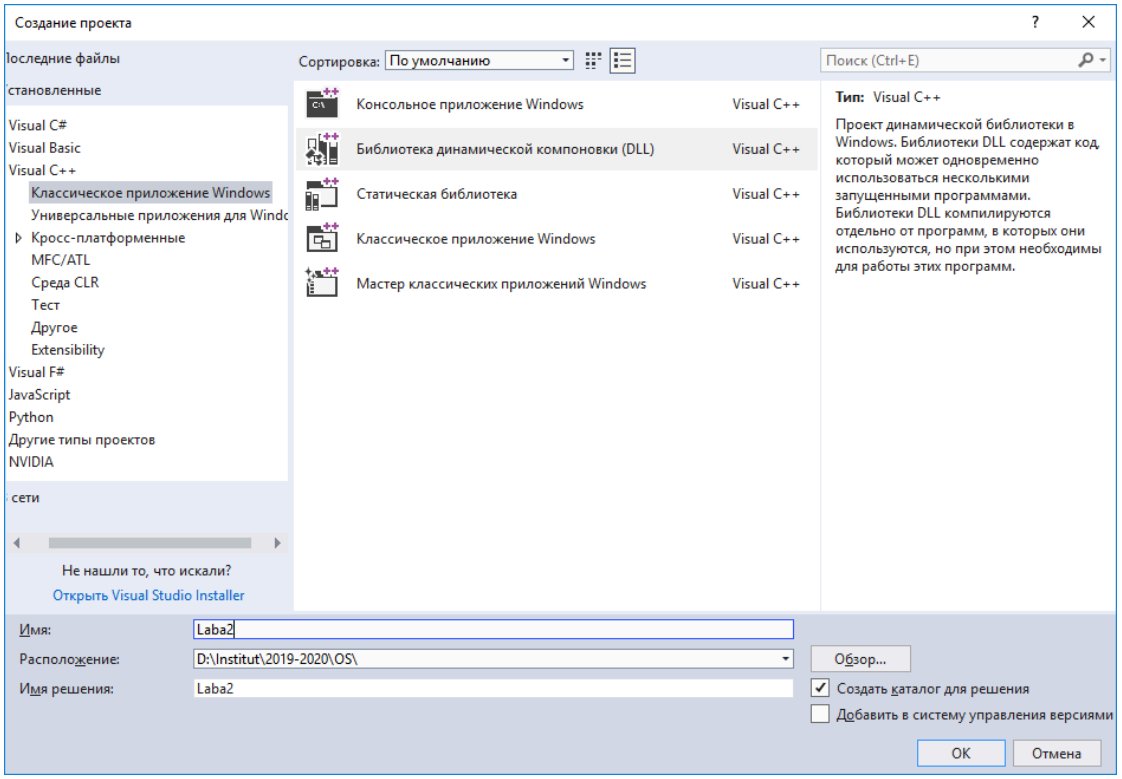

Рисунок 2.1 - Створення динамічної бібліотеки

Як і в разі створення консольного додатку створюється папка з іменем проекту. Не забудьте задати 64-бітну платформу!

Розглянемо файли, які створюються для бібліотеки. Вони розташовані во внутрішній папці з ім’ям бібліотеки:

- dllmain.cpp – файл, призначення якого буде розглянуто далі (дивись лабораторну роботу № 3);
- <Імя DLL>.cpp - файл для зберігання реалізації функцій, що експортуються;
- stdafx.cpp - службовий файл;
- stdafx..h - службовий файл;
- targetver.h - службовий файл.

Набір службових файлів може змінюватись в залежності від версії Visual Studio.

Додамо до проекту необхідні файли з визначенням функцій. Додамо файл з заголовками експортованих функцій. Ці функції повинні в заголовку мати __declspec (dllexport).

Трансляція та компоновка DLL виконується як і для проектів інших типів.

В результаті виконання буде створено один або 2 файли: файл *.dll і, *.lib, якщо є хоч одна експортована функція. Імена файлі по замовченню співпадають з іменем проекту. Ці імена можна замінити. Файл *.lib вміщує довідник функцій, а саме ім’я відповідної

DLL та таблицю функцій. для кожної функції задається її ім’я, номер, та адреса відносно початку DLL. Раніше номер використовувався для виклику відповідної функції, сьогодні для виклику функцій використовується тільки її ім’я.

2.2.3 Використання DLL

Режими використання DLL:

− завантаження DLL під час завантаження додатка, який використовує DLL (якщо попередні додатки не завантажили цю DLL). Вивантаження після завершення цього додатку (якщо другі додатки не використовують цю DLL);

− завантаження та вивантаження DLL виконується за допомогою функцій WINAPI тоді, коли необхідно використовувати функції DLL (коли необхідність в функціях відпадає) (цей режим використання буде розглянуто в лабораторній роботі 3).

Для першого режиму використання необхідно підключити до проекту, який використовує функції з DLL, файл з розширенням lib, який сформовано при виконанні 2.2.4. Для цього в Solution Explorer обрати проект з програмою, яка використовує DLL, правою кнопкою мишки обрати Reference, та обрати необхідну бібліотеку. В заголовках функцій з DLL необхідно вказати __declspec (dllimport). Далі функції викликаються як для функцій користувача.

Переваги першого режиму:

- якщо немає необхідної DLL, програма не буде завантажена;
- використання по складності однаково з використанням статичних бібліотек.

Недоліки першого режиму:

- файл з розширенням .lib є платформенно залежним, тобто Visual Studio, C++ Builder використовують різні файли;
- бібліотека завантажується при завантаженні програми та залишається завантаженою до тих пір, поки програма не завершиться. А можливо, що бібліотека зовсім не буде потрібна (виконується частина програми, яка не викликає функцій бібліотеки), або використовується тільки на початку програми;
- не може використовуватися для DLL, які вміщують тільки ресурси і не вміщують експортованих функцій.

2.2.4 Алгоритм шифрування RSA

Алгоритм шифрування RSA:

- формується 2 простих числа (p, q, p!=q);

- формується добуток для цих чисел, будемо далі називати цей добуток модулем n = p * q;

- формується функція Ейлера для цих чисел по формулі:

ϕ = (p - 1) * (q - 1).

- формуються ключі для крипто перетворень. Ключ для зашифрування даних E – випадкове число, яке взаємно просте з ϕ, ключ для розшифрування даних D задовольняє формулі D * E = 1 mod (ϕ). Остання формула означає, що:

(D * E) % ϕ = 1;

- зашифрувати можна довільні дані, значення яких менше ніж n^2. Для зашифрування даних використовується ключ E, який в криптографії називають відкритим ключем. Для шифрування даного M використовується формула:

$M' = M^E \bmod n$.

Отримане значення M' – це зашифроване значення;

- для розшифрування використовується ключ D, який в криптографії називають особистим ключем. В даному випадку розшифрувати дане зможе тільки власник особистого ключа. Зашифрувати дане для нього може любий користувач, якому передали відкритий ключ даного користувача. Для розшифрування використовується формула $M = M'^D \bmod n$.

2.3 Порядок виконання лабораторної роботи

1. Визначите необхідні функції для реалізації алгоритму RSA. Визначите серед цих функцій ті, які повинні використовуватись в зовнішніх програмах;
2. Складіть файл заголовків таким чином, щоб цей файл заголовків можна було б використовувати для динамічної бібліотек та програм, які використовують бібліотеку;
3. Реалізуйте функції бібліотек;
4. Створіть динамічну бібліотеку;
5. Реалізуйте головну програму для динамічної бібліотеки для першого способу її використання, для цього:

- сформувати 2 пари ключів (d0, e0, n0, d1, e1, n1);

---

[2] Для здолання цього обмеження дані діляться на блоки, розмір кожного блоку задовольняє обмеженню.

Виконати цикл для різних даних, в якому:

- виконати зашифрування відкритого даного (позначимо його t[i]) за допомогою ключа {e1, n1}. Результат зашифрування позначимо e1t;
- виконати розшифрування даного e1t за допомогою ключа {d1, n1}. Результат розшифрування позначимо d1e1t. Перевірити d1e1t = t [i]. Якщо рівність не виконується – помилка;
- виконати зашифрування даного отриманого після розшифрування d1e1t за допомогою ключа {e0, n0}. Позначимо результат e0d1e1t;
- виконати розшифрування даного за допомогою ключа {d0, n0}. Позначимо результат d0e0d1e1t. Перевірити d0e0d1e1t = ot [i]. Якщо рівність не виконується – помилка.

6. Для отримання найвищої оцінки для обчислення ключів використовувати 64-бітні дані, тобто значення p, q – цілі числа в діапазоні [3 - 2^{32}-1], для обчислення степені – двійковий алгоритм.

2.4 Зміст звіту

Звіт має містити:

- правила формування проекту для динамічної бібліотеки та головної програми;
- текст функцій для бібліотеки;
- текст головної програми;
- результати роботи програми.

2.5 Контрольні запитання та завдання

1. Що таке динамічна бібліотека?
2. Якого типу файли можна підключати до динамічних бібліотек?
3. Які режими використання динамічних бібліотек ви знаєте? Дайте характеристику кожного режиму?

4. Чим обмежується інтервал чисел, які можна шифрувати за допомогою реалізованої функції шифрування, розробленої Вами?

5. Чому значення p, q не можуть бути близькими?

6. Як зробити бібліотеку, яку можна використовувати в C та C++ програмах?

3 СТВОРЕННЯ ТА ВИКОРИСТАННЯ БІБЛІОТЕК. ЧАСТИНА 2

3.1 Мета роботи

Вивчити розширені прийоми та методи створення та використання різних типів динамічних бібліотек.

3.2 Підготовка до роботи

3.2.1 Типи динамічних бібліотек та способи їх застосування

При виконанні попередньої лабораторної роботи Ви навчилися створювати динамічні бібліотеки з функціями, які експортуються, та використовувати їх в режимі статичного застосування. По-перше необхідно навчитися використовувати такі бібліотеки в режимі динамічного застосування, який має переваги перед статичним режимом і може застосовуватись не тільки для бібліотек з функціями, які експортуються. Саме до таких бібліотек відноситься бібліотека ресурсів.

В разі динамічного способу застосування бібліотек завантаження та вивантаження DLL виконується за допомогою функцій WINAPI тоді, коли необхідно використовувати функції DLL (коли необхідність в функціях відпадає).

Для цього режиму використання необхідно виконати такі кроки:

- підключити файл заголовків Windows.h;
- перед використанням першої функції з DLL завантажити цю DLL. Для цього використовується функція LoadLibrary:
  HMODULE WINAPI LoadLibrary(LPCTSTR lpFileName);
- після використання цієї функції необхідно перевірити успішність завантаження (результат не дорівнює 0);
- для кожної функції із бібліотеки, яку потрібно використовувати, треба визначити адресу цієї функції. Для цього використовується функція FARPROC WINAPI GetProcAddress(HMODULE hModule, LPCSTR lpProcName);
  де:
  hModule – дескриптор бібліотеки;
  lpProcName – імя функції (внутрішнє).

- внутрішнє ім’я функції можна знайти в самій DLL. Для того, щоб крім внутрішнього імені можна було б використовувати звичайне ім’я, ці імена необхідно задати в файлі з розширенням .def (див. п. 3.2.2). Після використання функції визначення адреси, необхідно перевірити успішність цієї функції (адреса не дорівнює 0);
- далі функції викликаються як звичайні функції;
- після виклику останньої функції бібліотеку можна вигрузити DLL з пам’яті . Для цього використовується функція FreeLibrary:
  BOOL WINAPI FreeLibrary(HMODULE hModule);
- якщо функції DLL використовуються до кінця програми, функцію FreeLibrary можна не використовувати, операційна система визволить усі ресурси при завершенні програми. DLL відноситься до ресурсів.

3.2.2 Додавання .def файлу

Для додавання def файлу треба обрати «Добавление нового элемента», отримаємо вікно (див. рис. 3.1):

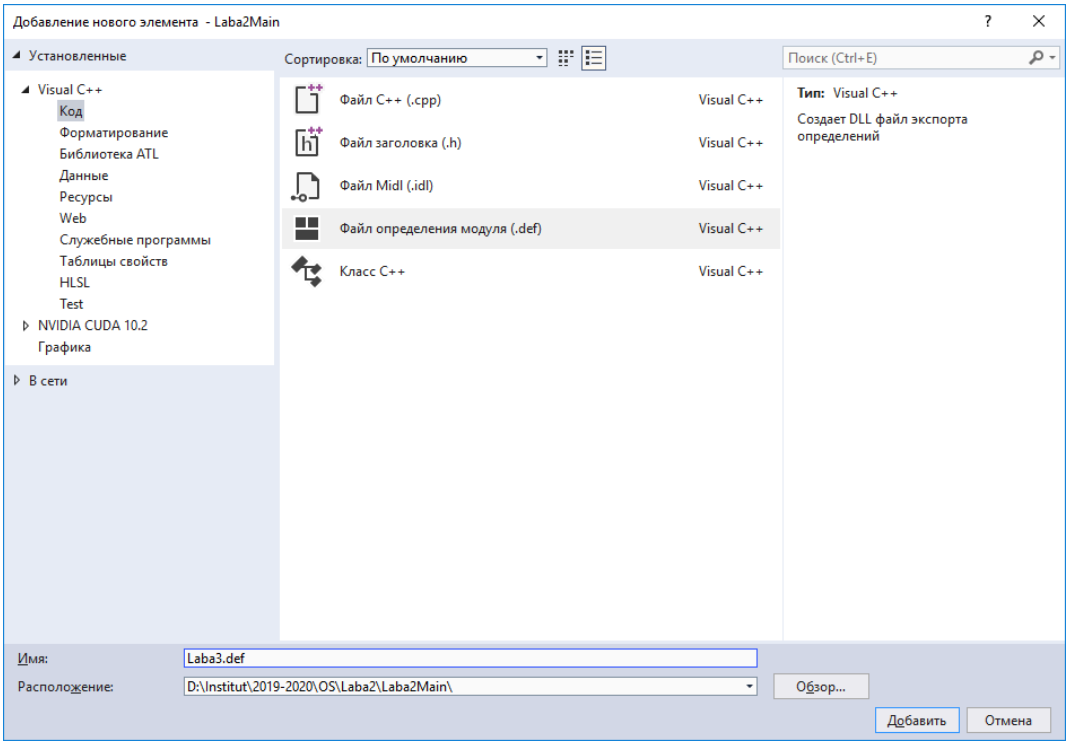

Рисунок 3.1 – Діалогове вікно «Добавление нового элемента»

Обираємо пункт «Файл определения модуля» додаємо файл, наприклад, з ім’ям Laba3.def.

В першому рядку LIBRARY можна задати ім’я бібліотеки, можна нічого не задавати. В другому рядку задаємо EXPORTS і далі вказуємо імена функцій, що експортуються по одній в рядку. В результаті вміст файлу може бути таким:

LIBRARY

EXPORTS

Crypt

DeCrypt

GenKey

3.2.3 Особливості створення та застосування ресурсних бібліотек

Бібліотеки створюються, якщо необхідно в проекті змінювати ресурси без зміни самої програми, наприклад, змінювати мову для усіх повідомлень, відповідні картинки і т.д.

3.2.3.1 Створення ресурсної бібліотеки

Бібліотека створюється як звичайна динамічна бібліотека, але додавати до неї треба

файли ресурсів (див. рис. 3.2):

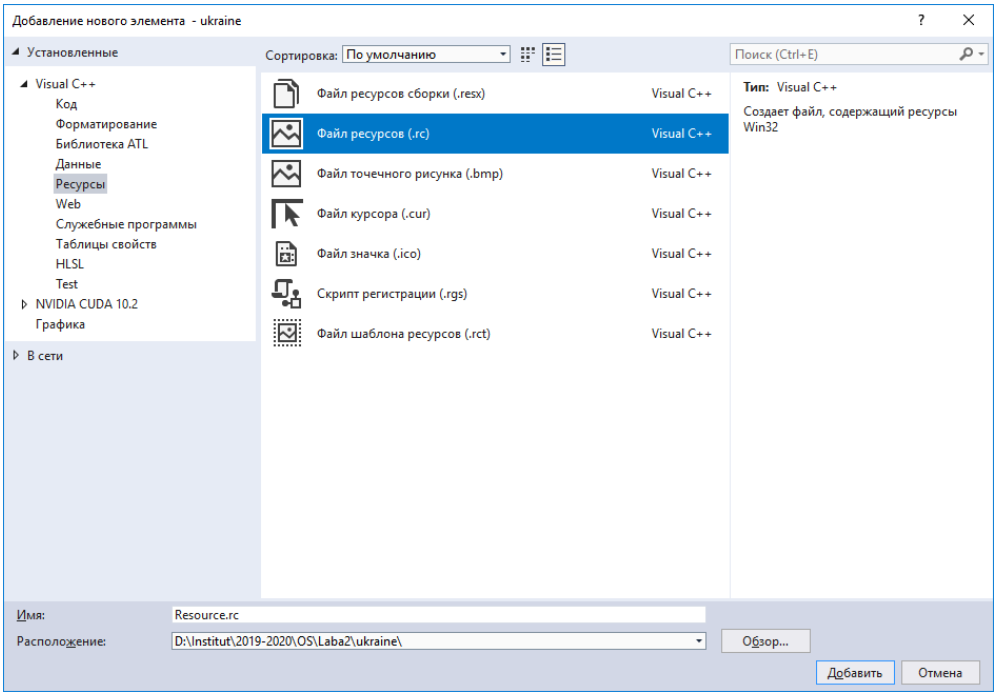

Рисунок 3.2 - Діалогове вікно додавання файлу ресурсів

До створеного файлу ресурсів додаємо окремі ресурси. Для цього обираємо створений файл, обираємо тип ресурсу, права кнопка мишки, додати:

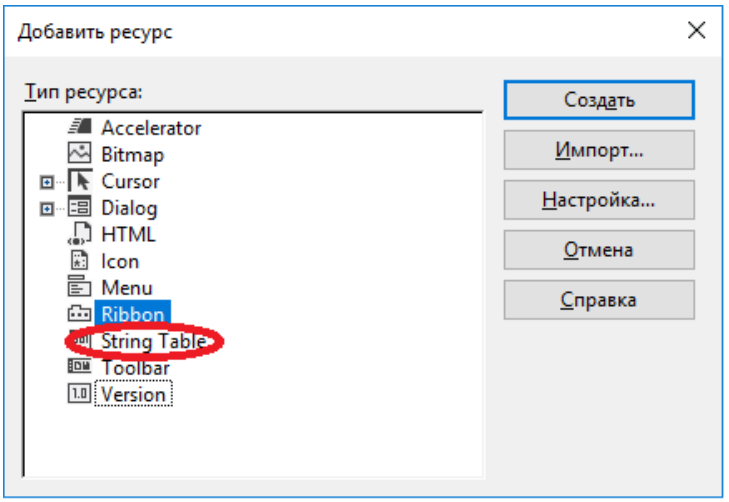

Рисунок 3.3 Діалогове вікно додавання ресурсу, наприклад, String Table

Далі розглянуто шаблон заповнення ресурсу «String Table», який використовується для завдання рядків на різних мовах. Після обрання типу ресурсу «String Table» з’являється таблиця (див. рис. 3.4):

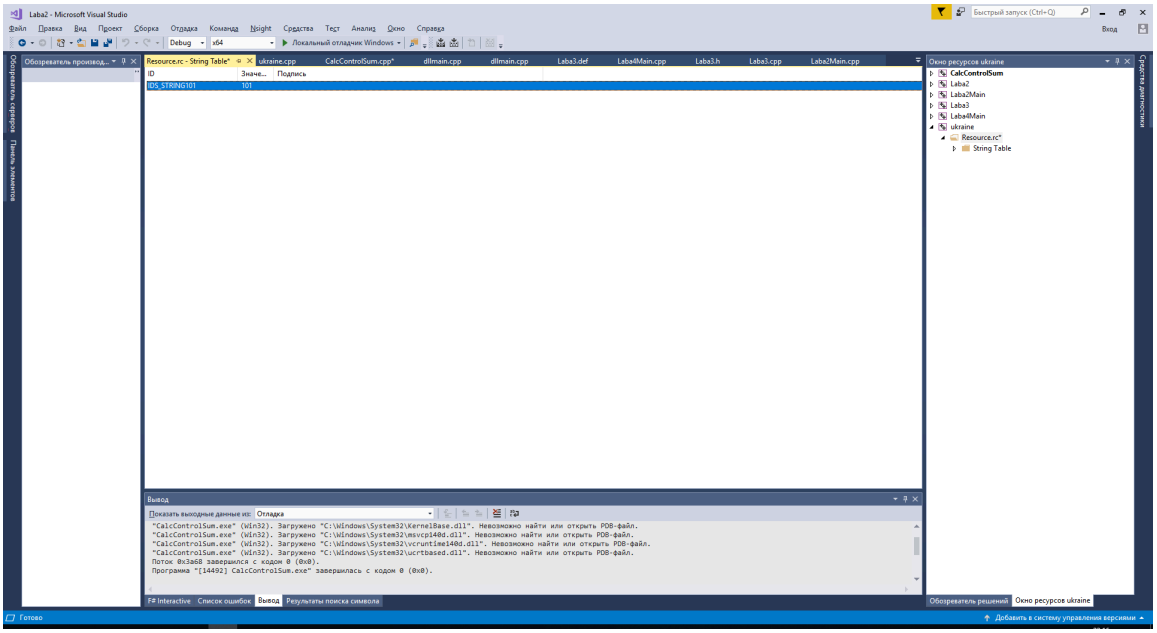

Рисунок 3.4 – Вікно заповнення ресурсу «String Table»

Ця таблиця має 3 колонки:

- ID – це ідентифікатор рядка, по замовченню IDS_STRING101, IDS_STRING102,…. Ідентифікатори рядків можна змінювати;
- значення – це числове значення ідентифікатору (101, 102, …);
- подпись – це рядок, який відповідає даному ідентифікатору.

Зазвичай в таблицю записують усі рядки, які треба виводити в програмі з використанням однієї мови.

Одночасно при створенні DLL створюється файл заголовків resource.h, в якому задаються усі ідентифікатори і відповідні їм числові значення.

Якщо серед рядків є рядки, мова яких не співпадає з мовою по замовченню, необхідно встановити цю мову. Для цього

Відкриваємо файл ресурсів (файл з розширенням rc, який автоматично створено при додаванні ресурсів);

Встановлюємо для його компоненту (String Table) додаткові властивості для тієї мови що потрібно.

3.2.3.2 Використання ресурсної бібліотеки

Для використання ресурсної бібліотеки необхідно:

- завантажити бібліотеку (функція LoadLibrary);
- по значенню ідентифікатора завантажити потрібний рядок за допомогою функції LoadString: int LoadString(HINSTANCE hInstance, UINT uID, LPTSTR lpBuffer, int nBufferMax);
- вивести цей рядок.

Звертаємо увагу, що рядок задається в форматі, який встановлено по замовченню (див. лабораторну роботу № 1).

3.2.4 Завдання параметрів програмі

Для передачі параметрів програмі необхідно:

- обрати для проекту пункт меню Свойства→Отладка→Аргументы команды (див. рис. 3.5);
- задати необхідний список параметрів.

В заголовку головної функції вказати на наявність параметрів:

```
int main(int argc, char **argv)
```

Перевірити, що параметри задані if (argc > 1)…

Для доступу до першого параметру використовується argv[1], другого argv[2], …

Якщо в списку параметрів передається файл, який знаходиться в вихідному каталогу проекту, то треба в рядку Рабочий каталог задати каталог $(OutDir)

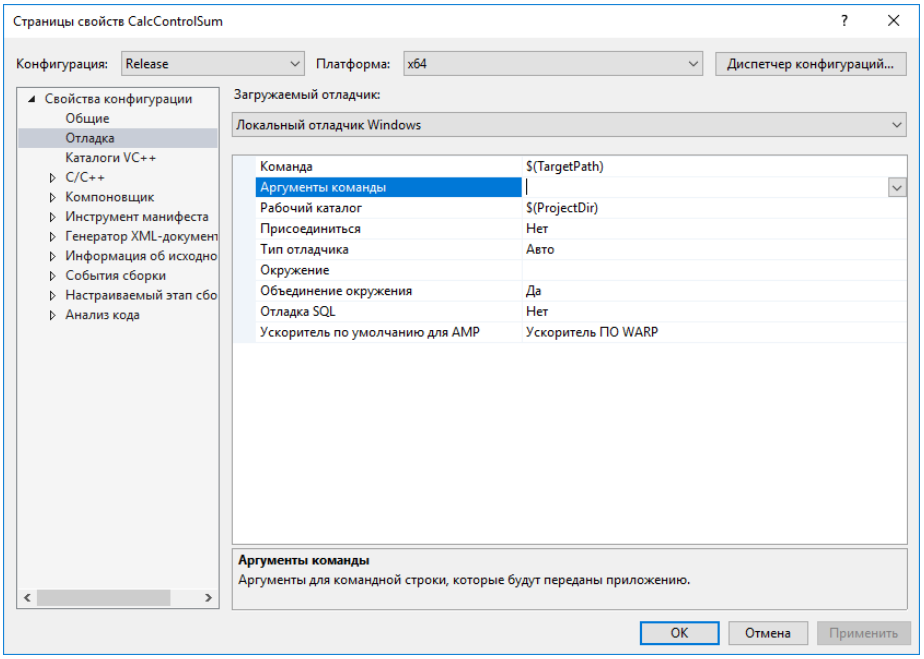

Рисунок 3.5 - Діалогове вікно з можливістю передачі параметрів програмі

3.2.5 Особливості використання головної функції DLL (dllmain)

3.2.5.1 Структура головної функції

Як було визначено при створенні динамічної бібліотеки, компілятор створює файл dllmain.cpp:

```
BOOL APIENTRY DllMain( HMODULE hModule,  DWORD  ul_reason_for_call,
                       LPVOID lpReserved ){

 switch (ul_reason_for_call)
   {
         case DLL_PROCESS_ATTACH:
         case DLL_THREAD_ATTACH:
         case DLL_THREAD_DETACH:
         case DLL_PROCESS_DETACH:
         break;
   }
   return TRUE;
}
```

APIENTRY означає, що це не звичайна функція, а функція зворотнього виклику, яку викликає не додаток, а операційна система в разі наступних подій:

- DLL_PROCESS_ATTACH означає, що DLL завантажується, тобто для неї виконується функція LoadLibrary;
- DLL_PROCESS_DETACH означає, що DLL вивантажується, тобто для неї

виконується функція FreLibrary;

- DLL_THREAD_ATTACH означає, що створюється додатковий потік (функція CreateThread, буде розглянуто при вивченні наступних тем);
- DLL_THREAD_DETACH означає, що потік знищується (функція CloseHandle для дескриптору потоку, розглянуто при вивченні наступних тем).

Призначення параметрів функції DllMain:

- hModule – дескриптор бібліотеки;
- ul_reason_for_call – причина виклику функції DllMain, дивись визначення APIENTRY;
- lpReserved – зарезервоване, можливе застосування буде розглянуто нижче.

Функція DllMain застосовується:

- для визначення режиму використання DLL (див. п. 3.2.5.2);
- для контролю цілісності DLL (див. п. 3.2.5.3);
- для виділення локальної пам’яті потоку (буде розглянуто у темі «потоки»).

3.2.5.2 Застосування DllMain для визначення допустимого режиму використання (статичне або динамічне завантаження)

Значення параметру lpReserved визначає, який режим застосовується для бібліотеки. Це значення дорівнює 0 або значенню, яке є відмінним від 0. Експериментально можна визначити значення цього параметру для кожного способу завантаження.

3.2.5.3 Застосування DllMain для контролю цілісності

Перевірка цілісності Dll дозволяє зменшити імовірність внесення несанкціонованих змін в Dll з метою отримання доступу до персональних даних або виключення функцій перевірки, наприклад, перевірки паролю.

Для перевірки цілісності необхідно:

- записати в Dll контрольну інформацію, яку можна використовувати для перевірки цілісності, наприклад, контрольну суму, CRC, геш значення, цифровий підпис. Це зазвичай виконує додаткова програма;
- під час завантаження Dll (гілка DLL_PROCESS_ATTACH головної функції)

перевірити, що контрольна інформація відповідає файлу з Dll, якщо це не так, головна функція Dll повинна повернути FALSE, це приведе до помилки завантаження Dll;

- функція для перевірки контрольного значення повинна бути частиною бібліотеки.

Таким чином, технологія створення Dll з контролем цілісності потребує виконання наступних кроків та перевірок:

1) створити Dll;

2) до Dll додати функцію контролю цілісності;

3) до гілки DLL_PROCESS_ATTACH додати визначення імені Dll і перевірку цілісності (функція GetModuleFileName);

4) виконати створення проекту з Dll;

5) за допомогою додаткової програми додати до файлу з Dll контрольне значення;

6) перевірити, коректність функціонування Dll після додавання контролю цілісності.

Для того щоб крок 5 виконувався автоматично після створення нової версії Dll, треба відкрити властивості проекту з Dll, обрати опцію «События сборки→События после сборки» та задати там командний рядок для виклику програми підрахунку контрольного елементу.

3.3 Порядок виконання лабораторної роботи

1) до бібліотеки, яка була розроблена при виконанні лабораторної роботи №2, додати .def файл з визначенням функцій, які експортуються. Створити новий варіант бібліотеки;

2) додати новий проект, в якому застосовувати нову бібліотеку в динамічному режимі. Перевірити роботу бібліотеки;

3) у головній програмі визначити типи функцій, які будуть використовуватись, наприклад:

typedef unsigned (*pGenKey)(unsigned *E, unsigned *D);

4) у головній програмі загрузити бібліотеку та перевірити успішність завантаження;

5) визначити адреси функцій, які використовуються;

6) перевірити роботу функцій за допомогою коду як в попередній лабораторній роботі;

7) додати два проекти для створення динамічних бібліотек для мов Українська, Англійська. В кожній бібліотеці додати рядки (тобто додати власні дані за шаблоном):

Прізвище, Факультет, Група, Дисципліна

8) у головній програмі по запиту визначити мову, завантажити потрібну бібліотеку та вивести задану інформацію на обраній мові.

9) додати в DllMain виведення значення параметру lpReserved для статичного використання (в Dll попередньої роботи) та динамічного режимів (в Dll поточної роботи). Зробити висновки;

Пункти 10 – 15 виконувати для отримання найвищої оцінки (уважно ознайомтеся з поясненнями у п. 3.2.5.3)

10) додати новий проект для підрахунку контрольної суми Dll і запису цієї суми в кінець Dll. Ім’я Dll передати в списку параметрів проекту. Додати до цього проекту функцію для контролю обчисленого контрольного значення для перевірки правильності його формування;

11) використати цей проект для запису контрольної суми в кінець Dll. Для роботи з файлом використовувати довільні функції. Звертаємо увагу на необхідність визначення файлу як двійкового файлу (Чому?);

12) перевірити правильність роботи програми по формуванню контрольного елементу;

13) перевірити, що після додавання контрольного елементу бібліотека продовжує нормально функціонувати;

14) додати до Dll функцію контролю цілісності та зробити необхідні зміни в гілці DLL_PROCESS_ATTACH;

15 після побудови Dll перевірити її роботу.

3.4 Зміст звіту

Звіт має містити:

- тексти програм для усіх створених проектів;
- результати роботи для усіх створених проектів;
- висновки.

3.5 Контрольні запитання та завдання

1. Чим відрізняється динамічний режим використання бібліотеки від статичного?
2. Як визначаються адреси функцій?
3. Чим відрізняється виклик функції з динамічної бібліотеки від виклику функції, визначеній в додатку?
4. Як програмно узнати, в якому режимі використовується бібліотека?.
5. В якому випадку використовуються ресурсні бібліотеки?

4 КЕРУВАННЯ ЗОВНІШНІМИ ПРИСТРОЯМИ. НЕСТАНДАРТНІ ПРИСТРОЇ

4.1 Мета роботи

Навчитися практичному використанню функцій WINAPI для роботи з файлами

4.2 Підготовка до роботи

Для підготовки до роботи необхідно навчитися використовувати наступні групи функцій:

1) створення (відкриття) файлів та каталогів CreateFile, CreateDirectory;
2) закриття файлів (CloseHandle);
3) копіювання файлів (CopyFile);
4) знищення файлів (DeleteFile);
5) функції для пошуку файлів FindFirstFile, FindFirstFileEx, FindNextFile, FindClose;
6) функція для визначення розміру файлу GetFileSize;
7) функції для введення – виведення даних з файлів ReadFile, WriteFile;
8) функції для позиціонування покажчика в файлі та встановлення кінця файлу при зменшенні його розміру SetFilePointerEx , SetEndOfFile;
9) функції для визначення та встановлення атрибутів файлів GetFileAttributes, SetFileAttributes.

4.3 Завдання до лабораторної роботи

Є 3 різних завдання. Студент обирає одне з завдань по своєму бажанню.

4.3.1 Моделювання поштової скриньки

В новому каталозі створити об’єкт **Поштова скринька**. Структура поштової скриньки: кількість повідомлень, загальний розмір усіх повідомлень, максимальний розмір поштової скриньки, Повідомлення 1, Повідомлення 2,… . Кожне повідомлення повинно задаватися в виді: розмір повідомлення, тіло повідомлення. Максимальний розмір поштової скриньки задається при створенні поштової скриньки.

Для об’єкту визначити функції додавання листів, читання листів з видаленням та без видалення, видалення заданого листа та усіх листів, визначення кількості листів, а також визначення загального числа поштових скриньок.

4.3.2 Створення класу для роботи з пристроями, файлами та каталогами (на найвищу оцінку)

1) Вивчити усі функції, які використовуються для роботи з пристроями, файлами та каталогами в C#;

2) Визначити клас (класи) для мови C++, інтерфейс для яких подібний інтерфейсу C#;

3) Реалізувати функції класу (класів) за допомогою функцій WinAPI;

4) Порівняти швидкодію функцій при їх використанні для C#, C в разі використання великих файлів.

4.3.3 Розробка найпростішої файлової системи (на найвищу оцінку)

Вимоги

1 В якості сховища для файлів застосовується пам'ять

2 В якості метаданих застосовуються;

Таблиця з розмірами решти таблиць (1 блок)

Каталог (містить ім’я файлу, його розмір, атрібути та номер першого блоку)

Бітова карта для зайнятих та вільних блоків.

3 Файли розташовуються в сусідніх блоках, фрагментація файлів не передбачена.

4 Реалізувати функції:

Форматування;

Відкриття файлу ;

Запис в файл;

Читання файлу;

Видалення файлу;

Відновлення видаленого файлу

4.4 Зміст звіту

Звіт має містити:

- тексти програм для створених проектів;
- результати роботи для створених проектів;
- висновки.

4.5 Контрольні питання и завдання

1. Які прапорці необхідно використовувати для функції CreateFile для створення нового і відкриття існуючого файлу?
2. Задайте прапорці для функції CreateFile для дозволу введення – виведення з файлу іншими програмами;
3. Задайте прапорці для функції CreateFile для файлу, який спочатку читається, а потім модифікується;
4. Яка функція використовується для визначення розміру файлу?
5. Які параметри необхідно визначити для файлу, розмір якого менше $2^{32} – 1$?
6. За допомогою якої функції можна зміститися в файлі на задану величину?
7. Як задати відносно чого виконується зміщення?

5 КЕРУВАННЯ ПАМ'ЯТТЮ. ЧАСТИНА 1

5.1 Мета роботи

Вивчити особливості використання та функції для роботи з віртуальною та фізичною пам'яттю.

5.2 Підготовка до роботи

5.2.1 Відображення віртуальної пам’яті на оперативну

Віртуальна пам'ять визначається режимом роботи процесору (32 бітний або 64 бітний) і зазвичай значно перевищує розмір оперативної пам’яті, ось чому визначається алгоритм перетворення поточної адреси, яка обмежена адресацією віртуальної пам’яті, в адресу оперативної пам’яті. Алгоритм перетворення визначається операційною системою. Якщо для поточних даних (команд) немає вільної оперативної пам’яті необхідно спочатку визволити пам'ять, зайняту командами (даними), які зараз не використовуються, і застосувати визволену пам'ять для активних команд (даних). Для цього використовується алгоритми вивантаження - завантаження сторінок. Вимоги до алгоритму: мінімізація кількості вивантаження. Вивчить різні алгоритми роботи зі сторінками.

5.2.2 Функції OS Windows для роботи з пам’яттю

5.2.2.1 Класифікація функцій

- інформаційні;
- функції керування віртуальною та фізичною пам'яттю;
- відображення файлів на пам'ять;
- функції керування купою.

5.2.3 Інформаційні функції

Функція GetSystemInfo використовується для визначення системної інформації, а саме інформації по процесору та пам'яті:

void WINAPI GetSystemInfo(LPSYSTEM_INFO lpSystemInfo);

де:

typedef struct _SYSTEM_INFO {

```
union {
        DWORD dwOemId;
        struct {
                WORD wProcessorArchitecture;
                WORD wReserved;
        };
};
DWORD dwPageSize;
LPVOID lpMinimumApplicationAddress;
LPVOID lpMaximumApplicationAddress;
DWORD_PTR dwActiveProcessorMask;
DWORD dwNumberOfProcessors;
DWORD dwProcessorType;
DWORD dwAllocationGranularity;
WORD wProcessorLevel;  WORD wProcessorRevision;
} SYSTEM_INFO;
```

Поля, які пов’язані з пам’яттю, виділені.

Функція GlobalMemoryStatusEx визначає стан усіх типів пам’яті , а саме. скільки є всього, скільки доступно, і який процент зайнято:

```
BOOL WINAPI GlobalMemoryStatusEx(LPMEMORYSTATUSEX lpBuffer);
```

Структура MEMORYSTATUSEX:

```
typedef struct _MEMORYSTATUSEX {
        DWORD dwLength;
        DWORD dwMemoryLoad;
        DWORDLONG ullTotalPhys;
        DWORDLONG ullAvailPhys;
        DWORDLONG ullTotalPageFile;
        DWORDLONG ullAvailPageFile;
        DWORDLONG ullTotalVirtual;
        DWORDLONG ullAvailVirtual;
        DWORDLONG ullAvailExtendedVirtual;
} MEMORYSTATUSEX,  *LPMEMORYSTATUSEX;
```

В цій структурі поле dwLength означає розмір структури з даними, його треба

задати перед викликом функції GlobalMemoryStatusEx.

Функція VirtualQueryEx визначає стан адресного простору для заданого процесу:

```
SIZE_T WINAPI VirtualQueryEx(
        HANDLE hProcess,
        LPCVOID lpAddress,
        PMEMORY_BASIC_INFORMATION lpBuffer,
        SIZE_T dwLength
);
```

Де:

- HANDLE hProcess – дескриптор процесу, для поточного процесу можна використовувати функції GetCurrentProcess
- LPCVOID lpAddress – адреса пам’яті, стан якої аналізується;
- MEMORY_BASIC_INFORMATION lpBuffer – структура, куди записується результати аналізу;
- SIZE_T dwLength – розмір структури з результатами.

Структура MEMORY_BASIC_INFORMATION

```
typedef struct _MEMORY_BASIC_INFORMATION
{
PVOID BaseAddress;
PVOID AllocationBase;
DWORD AllocationProtect;
SIZE_T RegionSize;
DWORD State;
DWORD Protect;
DWORD Type;
}MEMORY_BASIC_INFORMATION, *PMEMORY_BASIC_INFORMATION;
```

Де:

- BaseAddress – найближча адреса на границі сторінки;
- AllocationBase – адреса початку регіону;
- AllocationProtect – права доступу, які призначені при виділенні пам’яті, задаються константами:

PAGE_EXECUTE, PAGE_EXECUTE_READ, PAGE_EXECUTE_READWRITE,

PAGE_EXECUTE_WRITECOPY, PAGE_NOACCESS, PAGE_READONLY, PAGE_READWRITE, PAGE_WRITECOPY

- RegionSize – розмір регіону;
- State, стан регіону, задається константами:
  - MEM_COMMIT – фізична пам’ять;
  - MEM_RESERVE – віртуальна пам’ять;
  - MEM_FREE – вільна пам’ять;
- Protect – реальні права доступу, задаються як для поля AllocationProtect.;
- Type – задається константами:
  - *MEM_IMAGE MEM_MAPPED*
  - *MEM_PRIVATE*

5.2.4 Функції для керування віртуальною та фізичною пам’яттю

Функція VirtualAllocEx використовується для виділення пам’яті заданого типу або перепризначення типу пам’яті:

```
LPVOID WINAPI VirtualAllocEx(
  __in      HANDLE hProcess,
  __in      LPVOID lpAddress,
  __in      SIZE_T dwSize,
  __in      DWORD flAllocationType,
  __in      DWORD flProtect
);
```

Де:

- hProcess – дескриптор процесу;
- lpAddress – адреса пам'яті, якщо змінюється її тип, 0 якщо пам'ять виділяється;
- dwSize – потрібний розмір (байт);
- flAllocationType – тип пам'яті; задається константами:
  - MEM_COMMIT;
  - MEM_RESERVE;
  - MEM_RESET;
- flProtect – спосіб доступу, задається константами:
  - PAGE_EXECUTE, PAGE_EXECUTE_READ,

PAGE_EXECUTE_READWRITE, PAGE_EXECUTE_WRITECOPY, PAGE_NOACCESS, PAGE_READONLY, PAGE_READWRITE, PAGE_WRITECOPY.

Функція VirtualFree використовується для звільнення пам’яті:

```
BOOL WINAPI VirtualFreeEx(
  __in      HANDLE hProcess,
  __in      LPVOID lpAddress,
  __in      SIZE_T dwSize,
  __in      DWORD dwFreeType
);
```

Де:

- hProcess – дескриптор процесу;
- lpAddress – адреса пам'яті, що визволяється;
- dwSize – розмір пам'яті, 0. якщо уся пам'ять.
- dwFreeType – тип визволення, задається константами:
  - MEM_DECOMMIT
  - MEM_RELEASE

5.2.5 Функції для відображення файлів на пам'ять

Використовуються, якщо необхідно:

- записи файлу обробляти в довільному порядку;
- один і той же файл використовується різними програмами;
- декілька програм використовують загальну пам'ять.

Функція CreateFile використовується для отримання інформації про файл, а саме, де розташоване, які права доступу, розмір.

Функція CreateFileMapping використовується для виділення віртуальної пам'яті (побудови відповідних записів в каталозі та таблицях сторінок)

```
HANDLE WINAPI CreateFileMapping(
  __in      HANDLE hFile,
  __in      LPSECURITY_ATTRIBUTES lpAttributes,
  __in      DWORD flProtect,
  __in      DWORD dwMaximumSizeHigh,
```

```
  __in      DWORD dwMaximumSizeLow,
  __in      LPCTSTR lpName
);
```

Де:

- hFile – дескриптор відповідного файлу, якщо виділяється загальна пам'ять, то задається INVALID_HANDLE_VALUE;
- lpAttributes – параметри безпеки (0);
- flProtect – режим доступу, задається константами:
    - PAGE_READONLY, PAGE_READWRITE, PAGE_WRITECOPY, PAGE_EXECUTE_READ PAGE_EXECUTE_READWRITE

- dwMaximumSizeHigh, dwMaximumSizeLow – максимальний розмір, для якого треба виділяти віртуальну пам'ять;
- lpName – ім'я відповідного об'єкту ядра.

Функція MapViewOfFile виконує виділення фізичної пам’яті та читання файлу в цю пам’ять:

```
LPVOID WINAPI MapViewOfFileEx(
  __in      HANDLE hFileMappingObject,
  __in      DWORD dwDesiredAccess,
  __in      DWORD dwFileOffsetHigh,
  __in      DWORD dwFileOffsetLow,
  __in      SIZE_T dwNumberOfBytesToMap,
  __in      LPVOID lpBaseAddress
);
```

Де:

- hFileMappingObject – дескриптор відображеного об’єкту;
- dwDesiredAccess – режим доступу до пам’яті, задається константами:
    - *FILE_MAP_ALL_ACCESS, FILE_MAP_COPY, FILE_MAP_EXECUTE, FILE_MAP_READ, FILE_MAP_WRITE;*
- dwFileOffsetHigh, dwFileOffsetLow – зміщення по відношенню до початку файлу;
- dwNumberOfBytesToMap – кількість байтів, які треба відобразити;

- lpBaseAddress – адреса пам’яті, яку використовувати для відображення (краще задати як 0!!!);

Після цієї функції можна використовувати дані з пам’яті.

Функція UnmapViewOfFile визволяє фізичну пам'ять:

BOOL WINAPI UnmapViewOfFile(LPCVOID lpBaseAddress);

Де: lpBaseAddress – адреса, яку повернула функція MapViewOfFileEx.

Функцію CloseHandle треба викликати для дескриптора – відображення та дескриптору файлу, якщо функція CreateFile використовувалась.

BOOL WINAPI CloseHandle(HANDLE hObject);

Де: hObject – дескриптор.

5.3 Завдання до лабораторної роботи

У даній лабораторній роботі необхідно виконати наступне.

1. Скласти програму для формування системної інформації про різні типи пам'яті (GetSystemInfo, GlobalMemoryStatusEx) (Програма №1);
2. В програмі №1 після виконання GetSystemInfo зробіть точку останова і визначте адреси початку і кінця програми. Програму №1 не завершуйте;
3. Скласти другу програму для формування системної інформації про різні типи пам'яті (GetSystemInfo, GlobalMemoryStatusEx) (Програма №2);
4. В програмі №2 після виконання GetSystemInfo зробіть точку останова і визначте адреси початку і кінця програми. Програму 2 не завершуйте;
5. Порівняйте результати і поясніть їх;
6. За допомогою функції VirtualQuery побудуйте список блоків пам’яті;
7. Виділіть пам'ять за допомогою функції VirtualAlloc та дослідіть зміни в списку вільних блоків. Яку стратегію використовує ОС? (для цього слід ЩЕ РАЗ ПОБУДУВАТИ список блоків па пам’яті та порівняти);
8. Реалізуйте функції виділення та визволення пам’яті за допомогою стратегії Найменший достатній;
9. Для отримання найвищої оцінки переробіть функції для роботи з поштовою

скринькою (попередня лабораторна робота) з використанням функцій відображення файлів.

5.4 Зміст звіту

Звіт має містити наступні частини:

- повний опис функцій для роботи з пам'яттю, що були використані у лабораторній роботі;
- тексти програми;
- пояснення отриманих результатів;
- висновки.

5.5 Контрольні питання й завдання

1. Дайте визначення різних типів пам'яті.
2. Поясните дії, необхідні для виділення пам'яті з погляду програміста і операційної системи.
3. Яка інформація про пам'ять може бути отримана і де використовується ця інформація?
4. Які стратегії виділення пам’яті ви знаєте?
5. Що буде, якщо:
    - програміст забув звільнити пам'ять?
    - у зв'язку з аварійним завершенням програми вона не дійшла до коду звільнення?
6. Що буде, якщо не передбачена перевірка благополучності виділення пам'яті і використовується фактично невиділена пам'ять?
7. Що буде, якщо використовується більший обсяг пам'яті, чим виділено?

6 КЕРУВАННЯ ПАМ'ЯТТЮ. ЧАСТИНА 2

6.1 Мета роботи

Вивчити особливості захисту критичних даних, використання динамічної та кеш пам’яті

6.2 Підготовка до роботи

При підготовці до роботи вивчити класи функцій для:

- захисту критичних даних від вивантаження на диск;
- роботи з динамічною пам’яттю (Heap).

А також вивчить архітектуру та особливості ефективного використання кеш пам’яті.

6.2.1 Захисту критичних даних від вивантаження на диск

Операційна система в разі, якщо немає вільних сторінок для виділення оперативної пам’яті, зайняті сторінки записує на диск (swapping) і виділяє визволену пам'ять по потребі. Якщо в пам’яті знаходяться критичні дані, наприклад, особиста інформація користувача, ці дані треба захистити від swapping. Саме для цього і використовуються функції цього класу. Цей же клас функцій застосовують, якщо швидкість обробки даних – критична ознака цих даних.

Функція VirtualLock гарантує, що регіон пам’яті, адреса якого lpAddress і розмір dwSize, не буде розміщено на диску:

```
BOOL VirtualLock(    LPVOID lpAddress, SIZE_T dwSize );
```

Функція VirtualUnLock розблоковує вказаний діапазон сторінок:

```
BOOL VirtualUnlock(  LPVOID lpAddress,   SIZE_T dwSize);
```

Алгоритм використання функцій:

- для виділення пам’яті треба обов’язково застосовувати функцію VirtualAlloc

(чому?);

- треба виділяти найменший достатній обсяг пам’яті (чому?);
- після використання критичних даних пам'ять потрібно заповнити «сміттям» (для чого?);
- функцію VirtualUnlock треба викликати зразу після заповнення пам’яті сміттям.

6.2.2 Алгоритми заміщення сторінок

Для оцінки алгоритмів застосовується кількість заміщень при заданому кількості робочої множини та загальній кількості сторінок. ЯКІ ТРЕБА застосовувати.

6.2.2.1 Оптимальний алгоритм

Заміщується сторінка, яку найдовше не буде застосовуватись

6.2.2.2 FCFS (Перша завантажена, перша заміщується

6.2.2.3 Годинник

Усі сторінки створюють коло, тобто після останньої сторінки іде перша.

Поточна сторінка спочатку дорівнює 0, після кожної операції заміщення або знаходження поточна сторінка збільшується на 1. Якщо сторінка знайдена, для неї біт доступу встановлюється в 1, і операція завершується. Якщо сторінка не знайдена після проходження циклу, починається нове коло. Якщо біт доступу дорівнює 1, він встановлюється в 0. Якщо наступна сторінка має біт доступу 0, вона заміщується новою сторінкою, для неї біт доступу встановлюється в 1

6.2.2.4 LRU (Який найдовше не застосовувався)

6.2.3 Кеш пам'ять

6.2.3.1 Архітектура кешу

Нагадуємо, що для виконання команд та обробки даних процесором, вони повинні бути відповідно в внутрішніх кешах команд та даних. Час завантаження даних а кеш значно більше ніж час виконання команд процесору, ось чому завантажуються команди (дані) блоками. Розмір блоку для сучасних процесорів зазвичай b = 64 байти. Кожний рядок кешу може зберігати n блоків, наприклад n = 2, (4, 8, …). Відповідний кеш називається 2 (4, 8, …) асоціативним. Розмір внутрішнього кешу визначається 64 * n * l, де

l – кількість рядків кешу. Зазвичай внутрішній кеш для сучасних комп'ютерів дорівнює 32К, тобто, якщо кеш 4-асоциативний, то кількість рядків кешу дорівнює l = 32 * 1024 / (64 * 4) = 128.

6.2.3.2 Адресація даних в кеші

Кеш значно менше оперативної пам'яті.

Розглянемо алгоритм перетворення адреси пам'яті A в адресу в кеш пам'яті (line, nblock, offset)в разі використання n = 4- направленого кешу для блоку розміром b = 64 байта і кількості рядків l = 128. Тут line – номер рядка кешу (0..127), nblock – номер блоку в рядку (0..3), offset – зміщення в блоці (0..63):

- визначаємо зміщення даних відносно початку блоку: offset = A & 0x3F
- визначаємо номер рядка: line = (offset >> 6) & 7F

Як визначити номер блоку? Для цього використовується алгоритм LRU.

6.3 Завдання до лабораторної роботи

У даній лабораторній роботі необхідно виконати наступне:

1) Реалізувати алгоритми заміщення сторінок (оптимальний, годинник, LRU). Порівняти алгоритми;

2) Реалізувати алгоритм LRU для роботи з кешом (для стандартних параметрів кешу)

3) Реалізувати функції встановлення паролю та перевірки паролю. Функція встановлення паролю просить задати пароль 2 рази, порівнює їх, якщо вони співпадають і пароль задовольняє вимогам до «хорошого» пароля https://yablyk.com/339648-kak-proverit-nadezhnost-parolya-onlajn-i-uznat-naskolko-bystro-ego-mozhno-vzlomat/ для заданого часу злому

4) обчислити для пароля геш та повернути цей геш. (можна обрати довільну функцію для обчислення гешу, наприклад, https://github.com/maciejczyzewski/retter/blob/master/algorithms/MD5/md5.c). Функція перевірки пароля після введення пароля обчислює його геш і порівнює його з заданим. При складанні функцій забезпечте безпечне зберігання паролів;

6.4 Зміст звіту

Звіт має містити наступні частини:

- повний опис функцій для роботи з пам'яттю, що були використані у лабораторній роботі;
- тексти програми;
- пояснення отриманих результатів;
- висновки.

6.5 Контрольні питання й завдання

1 Для чого треба використовувати функції VirtualLock, VirtualUnlock?

2 В якому випадку має сенс використання купи?

3 В якому випадку має сенс використання нестандартної купи?

4 Скільки пам’яті фактично виділяється по запиту виділення динамічної пам’яті для блоку розміром size?

5 Які вимоги до алгоритму завантаження даних в кеш память?

6 Алгоритми LRU для кешу при n = 2, n = 8.

7 Чи завжди «справедливий» алгоритм LRU для кешу?

7 КЕРУВАННЯ ПРОЦЕСАМИ ТА ПОТОКАМИ

7.1 Мета роботи

Навчитися створювати процеси та потоки при виконанні програм

7.2 Підготовка до роботи і порядок її виконання

Вивчить функцію CreateProcess і її використання для запуску програм. Складіть макрос для спрощення використання цієї функції.

```
BOOL WINAPI CreateProcess(
        LPCTSTR lpApplicationName, // Ім'я додатка
        LPTSTR lpCommandLine,// Командний рядок
        LPSECURITY_ATTRIBUTES lpProcessAttributes, // Атрибути
        LPSECURITY_ATTRIBUTES lpThreadAttributes,  // безпеки
        BOOL bInheritHandles, // Спадкування дескрипторів
        DWORD dwCreationFlags, // Прапорці створення
        LPVOID lpEnvironment, // Середа
        LPCTSTR lpCurrentDirectory,// поточний каталог
        LPSTARTUPINFO lpStartupInfo, // Структура з вхідними даними
        LPPROCESS_INFORMATION lpProcessInformation
        // Структура з дескрипторами створеного процесу та потоку
);
```

Необхідні структури:

```
typedef struct _STARTUPINFO {
    DWORD cb;
    LPTSTR lpReserved;
    LPTSTR lpDesktop;
    LPTSTR lpTitle;
    DWORD dwX, dwY, dwXSize, dwYSize;
    DWORD dwXCountChars, dwYCountChars;
    DWORD dwFillAttribute;
    DWORD dwFlags;
    WORD wShowWindow;
    WORD cbReserved2;
    LPBYTE lpReserved2;
```

```
        HANDLE hStdInput, hStdOutput;
        HANDLE hStdError;
} STARTUPINFO, *LPSTARTUPINFO;

typedef struct _PROCESS_INFORMATION {
        HANDLE hProcess;
        HANDLE hThread;
        DWORD dwProcessId;
        DWORD dwThreadId;
} PROCESS_INFORMATION, *LPPROCESS_INFORMATION;
```

Вивчить функцію для створення потоків.

```
HANDLE CreateThread(
        LPSECURITY_ATTRIBUTES lpThreadAttributes, // Атрибути безпеки, 0
        SIZE_T dwStackSize,                       // Розмір стеку, 0
        LPTHREAD_START_ROUTINE lpStartAddress,    // Потокова функція
         LPVOID lpParameter,                      // Параметр
        DWORD dwCreationFlags,                    // Флаги створення, 0
        LPDWORD lpThreadId                        // Ідентифікатор потоку, 0
);
```

Потокова функція (Заголовок)

```
DWORD WINAPI fun (PVOID);
```

Чекання завершення.

Один об'єкт

```
DWORD WaitForSingleObject(
        HANDLE hHandle,
        DWORD dwMilliseconds              INFINITE
);
```

Декілька об'єктів

```
DWORD WaitForMultipleObjects(
        DWORD nCount,
        const HANDLE *lpHandles,
        BOOL bWaitAll,                    bWaitAll = TRUE, якщо чекати всі
```

DWORD dwMilliseconds

);

7.3 Порядок виконання лабораторної роботи

1. Скласти 3 програми. Перша програма має запустити текстовий редактор і створити текстові файли в заданій папці. Друга програма приймає в якості параметру час запуску редактору і для всіх створених першою програмою файлів визначає їх розмір, кількість рядків і довжину кожного рядка. Для отримання відмінної оцінки програма повинна підтримувати створення як ASCII, так і UNICODE текстових файлів. Третя програма запускає по черзі спочатку першу, а потім другу програму.
2. Створити 10 потоків. Кожний потік повинен виводити рядки:
   Begin <Номер потоку> і End <Номер потоку>.
   Чекати завершення хоч одного потоку.
   Порахувати, скільки разів виконується потокова функція.

При виконанні програми для пункту 2 зверніть увагу на: порядок виконання потоків, оператори виведення даних в потоках, чи усі потоки будуть завершені після повернення в головну програму

3. Створити послідовну та паралельну функцію для обчислення добутку для квадратних матриць. Порівняти час виконання цих функцій.

7.4 Зміст звіту

Звіт має містити:

- опис всіх використаних функцій WINAPI;
- тексти всіх складених програм;
- висновки по роботі з переліком вмінь, які отримані в результаті виконання лабораторної роботи.

7.5 Контрольні запитання і завдання

1. Як необхідно підготувати структуру STARTUPINFO перед створенням процесу?
2. Чим відрізняється запуск програм, що виконуються від запуску командного файлу?
3. Як довідатися встановлені зовнішні пристрої?
4. Як довідатися, які з файлів знову створені або модифікувалися?
5. Як можна довідатися, які процеси виконуються?
6. Як довідатися по ідентифікаторі процесу його дескриптор?
7. Як довідатися ім'я програми, що виконується по дескрипторові процесу?

8 КЕРУВАННЯ ПОТОКАМИ ОДНОГО ТА ДЕКІЛЬКОХ ПРОЦЕСІВ

8.1 Мета роботи

Вивчити методи керування потоками.

Вивчити об'єкти операційної системи для синхронізації процесів і методику їхнього використання

8.2 Підготовка до роботи і порядок її виконання

1. Вивчити розподіл часу між потоками і можливості програміста по керуванню цим розподілом. Пріоритетне обслуговування потоків (конспект лекцій https://docs.microsoft.com/en-us/windows/win32/api/processthreadsapi/);
2. Вивчити способи синхронізації потоків (конспект лекцій, https://docs.microsoft.com/en-us/windows/win32/api/synchapi/):
   - без операційної системи;
   - за допомогою критичних секцій;
3. Вивчити засоби синхронізації процесів (конспект лекцій, https://docs.microsoft.com/en-us/windows/win32/api/synchapi/):
   - м'ютекси;
   - семафори;
   - таймери, що очкуються.
   -

Далі наведений перелік задач, які треба вирішити.

Задача №1

Реалізувати потокові функції для виробника та споживача, та використати їх, якщо є кілька потоків виробника та споживача.

В головній програмі передбачити, що «виробник» записує імена файлів в чергу, а «споживач» виводить їх зміст.

Перевірити можливість використання стандартної черги для роботи з потоками.

Задача №2

Реалізувати потоки для читача та письменника. Реалізувати головну програму, в якій створено декілька потоків обох типів. Потік письменника записує запис в кінець списку новин. Потік читача читає останню новину.

Задача №3

Реалізувати потоки для філософів, що обідають, Потік повинен забезпечити повний цикл операцій (думає, бере одну виделку, бере другу виделку, обідає, кладе одну виделку, кладе другу виделку), які виконуються задану кількість разів.

Перевірити відсутність гонок та блокувань при використанні потоків в одній програмі.

Завдання на найвищу оцінку

Створити додаткову програму, яка запускає декілька раз програму, у якій реалізовано задачу №1 або задачу №2 з потоками відповідно з заданим розкладом (WaitableTimer). Внести зміну в об’єкти синхронізації, які потрібні.

8.3 Зміст звіту

У звіт мають бути включені всі складені програми й обґрунтування використаних засобів синхронізації.

Висновки й рекомендації з використання способів синхронізації при рішенні конкретних завдань

8.4 Контрольні питання й завдання

1. У чому різниця між процесами й потоками?
2. У якому випадку необхідна синхронізація потоків?
3. Які засоби синхронізації Ви знаєте для потоків одного процесу?
4. Які засоби синхронізації не можна використовувати для потоків різних процесів і чому?
5. У чому полягає різниця між критичною секцією і м’ютексом?

6. По кожному способі синхронізації переваги й недоліки в порівнянні з іншими способами;

7. Які способи синхронізації процесів ви знаєте?

8. У чому особливість синхронізації за допомогою подій? приведіть приклади.

ПЕРЕЛІК ПОСИЛАННЬ

1 Бондаренко М.Ф., Качко О.Г. Операційні системи: навч. посібник. – Х.: Компанія СМІТ, 2008. – 432 с.

2 Качко Е.Г. Программирование на ассемблере. Учебное пособие по курсу «Системное программирование и операционные системы» - Харьков: ХНУРЭ, 2002 – 172 с.

3 Конспект лекцій «Програмування мовою С++» по курсу / Упоряд. Ог.Г. Качко – Харків: ХТУРЕ, 1999 – 148 с.

4 Рихтер Дж. Windows для профессионалов: создание эффективных Win 32 приложений с учетом специфики 64-разрядной версии Windows – СПб: Питер; М.: Издательско- торговый дом «Русская редакция», 2001. – 752 с.

5 Операционные системы: Деревянко А.С., Солощук М.Н. Учебное пособие. – Харьков: НТУ «ХПИ», 2003. – 574 с.

6 Таненбаум Э. Современные операционные системы. – СПБ: Питер, 2010 – 1120 с.

7 https://ru.wikibooks.org/wiki/Операционные системы - 10.05.2016 г.- Загол. с экрана.

8 Методичні вказівки до практичних занять та самостійної роботи з дисципліни «ОПЕРАЦІЙНІ СИСТЕМИ». Харків, ХНУРЕ, 2020

Навчальне видання

МЕТОДИЧНІ ВКАЗІВКИ

до лабораторних занять з дисципліни
«ОПЕРАЦІЙНІ СИСТЕМИ»

за спеціальністю
121 – ІНЖЕНЕРІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
освітньо-професійна програма «Програмна інженерія»
для здобувачів усіх форм навчання

Упорядники: КАЧКО Олена Григорівна
МЕЛЬНІКОВА Роксана Валеріївна

Відповідальний випусковий

Редактор

План 202_, поз.

Підп. до друку Формат 60×84 1/16. Спосіб друку – ризографія.
Умов.друк.арк. Облік. вид.арк. Тираж прим.
Зам. № Ціна договірна.

ХНУРЕ. Україна. 61166, Харків, просп. Науки, 14

Віддруковано в навчально-науковому
видавничо-поліграфічному центрі ХНУРЕ
61166, Харків, просп. Науки, 14