

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ
УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

МЕТОДИЧНІ ВКАЗІВКИ
до лабораторних занять з дисципліни
«ОПЕРАЦІЙНІ СИСТЕМИ»

для студентів усіх форм навчання
напряму 6.050103 «Програмна інженерія»

ЗАТВЕРДЖЕНО
кафедрою ПІ
Протокол №18 від 13.04.16

ХАРКІВ 2016

Методичні вказівки до лабораторних занять з дисципліни «Операційні системи» для студентів спеціальності 6.050103 Програмна інженерія / Упоряд. Качко О.Г., Мельникова Р.В. – Харків: ХНУРЕ, 2016. –38 с.

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

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

ВСТУП.....	5
1 РОЗРОБКА УНІВЕРСАЛЬНИХ ДОДАТКІВ ДЛЯ РІЗНИХ ТИПІВ КОДУВАНЬ СИМВОЛЬНОЇ ІНФОРМАЦІЇ	7
1.1 Мета роботи.....	7
1.2 Підготовка до роботи. Теоретичні положення	7
1.3 Завдання до лабораторної роботи. Порядок виконання	10
1.4 Зміст звіту.....	12
1.5 Контрольні запитання і завдання	12
2 СТВОРЕННЯ ТА ВИКОРИСТАННЯ БІБЛІОТЕК	13
2.1 Мета роботи.....	13
2.2 Підготовка до роботи	13
2.3 Порядок виконання лабораторної роботи	16
2.4 Зміст звіту.....	17
2.5 Контрольні запитання та завдання.....	17
3 КЕРУВАННЯ ЗОВНІШНІМИ ПРИСТРОЯМИ. НЕСТАНДАРТНІ ПРИСТРОЇ....	18
3.1 Мета роботи.....	18
3.2 Підготовка до роботи	18
3.3 Завдання до лабораторної роботи	18
3.4 Контрольні питання и завдання	19
4 КЕРУВАННЯ ПАМ'ЯТТЮ.....	20
4.1 Мета роботи.....	20
4.2 Підготовка до роботи	20
4.3 Завдання до лабораторної роботи	27
4.4 Зміст звіту.....	27
4.5 Контрольні питання й завдання	27
5 КЕРУВАННЯ ПРОЦЕСАМИ.....	29
5.1 Мета роботи.....	29
5.2 Підготовка до роботи і порядок її виконання	29
5.3 Порядок виконання лабораторної роботи	30
5.4 Зміст звіту.....	32

5.5	Контрольні запитання і завдання	32
6	КЕРУВАННЯ ПОТОКАМИ	33
6.1	Мета роботи.....	33
6.2	Підготовка до роботи і порядок її виконання	33
6.3	Зміст звіту	35
6.4	Контрольні питання й завдання	35
	СПИСОК ЛІТЕРАТУРИ	36

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 розширюють практичні навички, 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.

Усі лабораторні роботи виконуються з застосуванням ЕОМ. При цьому необхідно:

- виконувати правила техніки безпеки при роботі з ЕОМ;
- виконувати правила поведінки в лабораторії ЕОМ;
- приходити на заняття строго за розкладом;
- не допускається зміна прав доступу до системних ресурсів.

До роботи допускаються тільки підготовлені студенти, які склали та захистили програми. При допуску викладач перевіряє наявність цих програм та ступень самостійності їх складання.

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 можна використовувати цей документ в електронному вигляді. На титульному листі звіту треба написати автора звіту.

Без наявності електронної копії звіту для поточної лабораторної роботи 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 на ЕОМ.

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 типи 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 Типи даних char і wchar_t. Універсальне завдання типу

При використанні ASCII кодування використовується тип даних char. Даному типу 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 типу 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:

```
#ifndef UNICODE
    typedef wchar_t    TCHAR ;
    #define    TEXT(a)    L##a
#else
```

```

typedef      char      TCHAR ;
#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.

1.2.2 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, `wcsncpy`, `wscat`, `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.

1.2.3 Файл `tchar.h`. Пошук імен для універсальних функцій

Макроси, які наведені вище, а також багато інших, наведені в файлі `tchar.h`, які необхідно підключити для використання універсального кодування. Цей файл автоматично підключається Visual Studio при використанні непустого проекту. В цьому ат

падку створюється файл `stdafx.h`. Відкрийте його та знайдіть рядок `#include <tchar.h>`. Оберіть цей рядок та за допомогою правою кнопкою мишки відкрийте його. Знайдіть макроси, визначені вище.

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.

1.2.4 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 замінюється
                           // символ, що перетворюється, якщо він не може бути
                           // відображений. Дорівнює NULL, якщо використовується
                           // символ за замовчуванням.
    LPBOOL lpUsedDefaultChar // Показчик 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.

1.3 Завдання до лабораторної роботи. Порядок виконання

1. Складіть програму для програмної перевірки типу кодування, заданого за замовчуванням (визначте довжину в байтах типу TCHAR).

2. Визначте тип кодування по заданих макросах в командному рядку. Для цього визначимо командний рядок для трансляції програми:

Properties→C/C++→Command Line

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.

3. Перемикніть режим завдання символу на протилежний. Для перемикавання режиму з UNICODE в режим ASCII використовують:

Properties→General→Character Set→Use Multi-Byte Character Set

Для перемикавання з режиму ASCII в режим UNICODE використовують:
Properties→General→Character Set→UseUnicode Character Set.

4. Після перемикавання режиму знову перевірте тип символу за замовчуванням та командний рядок.

(Після трансляції!!!)

5. Задайте ПІБ членів своєї сім'ї в ASCII та виведіть задані значення. Для виведення букв кирилиці необхідно встановити локальні режими. Для цього використовуються файл заголовку і функція:

```
#include <locale.h>
```

```
...
```

```
TCHAR * _tsetlocale ( int category, const TCHAR *locale );
```

де:

category – зазвичай (LC_ALL);

locale – рядок, який визначається мовою, для якої встановлюється:

_T("Russian") – російська мова;

_T("Ukrainian") – українська мова.

Функція повертає 0, якщо функція не може бути виконана.

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

7. Виведіть отриманий масив. Перевірте можливість виведення кожним з 2-х способів:

– функція _tprintf, якщо встановлено локальні режими;

– функція MessageBox

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

8. Виконайте упорядкування масиву рядків, заданих в UNICODE. Для сортування використовувати універсальну стандартну функцію qsort:

```
void qsort(
```

```
void *base,          // Масив, що упорядковуємо
```

```
size_t num,          // Кількість елементів масиву;
```

```
size_t width,        // Ширину елементу масиву;
```

```
int (__cdecl *compare)(const void *, const void *) );
```

```
                // Функція для порівняння елементів масиву
```

9. Виконайте зворотне перетворення масиву з Unicode в ASCII.

10. Виведіть отриманий результат

11. Завдання для отримання найвищої оцінки. Задано текстовий файл. Незалежно від способу кодування символів в цьому файлі, переставити ці символи в зворотному порядку. Символи кінця рядка залишити в правильному порядку (це завдання для отримання найвищої оцінки).

1.4 Зміст звіту

Звіт повинен вміщувати:

- 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

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;
- 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.

Для створення DLL в середовищі використовується проект типу DLL (File→New→Projects→Win32). Після завдання імені проекту обирається тип DLL і, поки що, обираємо пусту DLL (An empty DLL project). Додамо до проекту необхідні файли з визначення 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 файли 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).

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.

1. Якщо немає необхідної DLL, програма не буде завантажена.
2. Використання по складності однаково з використанням статичних бібліотек.

Недоліки першого режиму:

1. Файл з розширенням `lib` є платформенно залежним, тобто Visual Studio, C++ Builder використовують різні файли.
2. 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.

3. Не може використовуватися для DLL, які вміщують тільки ресурси і не вміщують експортованих функцій.

Для другого режиму використання необхідно виконати такі кроки.

1. Підключити файл заголовків Windows.h
2. Перед використанням першої функції з DLL завантажити цю DLL. Для цього використовується функція LoadLibrary.

HMODULE WINAPI LoadLibrary(LPCTSTR lpFileName);

3. Після використання цієї функції необхідно перевірити успішність завантаження (результат не дорівнює 0).

4. 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 – ім'я функції (внутрішнє).

5. 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 DLL. Після використання функції визначення адреси, необхідно перевірити успішність цієї функції (адреса не дорівнює 0).

6. Далі функції викликаються як звичайні функції.

7. Після виклику останньої функції бібліотеку можна вивантажити з пам'яті. Для цього використовується функція FreeLibrary:

BOOL WINAPI FreeLibrary(HMODULE hModule);

8. Якщо функції DLL використовуються до кінця програми, функцію FreeLibrary можна не використовувати, операційна система визволить усі ресурси при завершенні програми. DLL відноситься до ресурсів.

2.2.4 Алгоритм шифрування RSA

1. Формується 2 простих числа (p, q, $p \neq q$).
2. Формується добуток для цих чисел, будемо далі називати цей добуток модулем $n = p * q$.
3. Формується функція Ейлера для цих чисел по формулі:

$$\varphi = (p - 1) * (q - 1).$$

4. Формуються ключі для крипто перетворень. Ключ для шифрування даних E – випадкове число, яке взаємно просте з φ , ключ для розшифрування даних D задовольняє формулі $D * E = 1 \bmod (\varphi)$. Остання формула означає, що $(D * E) \% \varphi = 1$.

5. Шифрувати можна довільні дані, значення яких менше ніж n^1 . Для шифрування 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.

6. 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. Визначите умову по виклику для функцій, якщо відомо, що функції треба використовувати в C, C++ файлах і вони повинні бути максимально надійними;
3. Складіть файл заголовків таким чином, щоб цей файл заголовків можна було б використовувати для динамічної бібліотек та програм, які використовують бібліотеку;
4. Реалізуйте функції бібліотек;
5. Створить динамічну бібліотеку;
6. Реалізуйте головну програму для динамічної бібліотеки для першого способу її використання.
7. Реалізуйте головну програму для динамічної бібліотеки для другого способу її використання;
8. Для отримання найвищої оцінки додайте до DLL підтримку введення –

¹ Для здолання цього обмеження дані діляться на блоки, розмір кожного блоку задовольняє обмеженню.

виведення на різних мовах;

2.4 Зміст звіту

Звіт повинен включати:

1. Правила формування проекту для динамічної бібліотеки та головної програми;
2. Текст функцій для бібліотеки;
3. Тексти головних програм (3);
4. Порівняльна характеристика способів застосування бібліотек

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

3 КЕРУВАННЯ ЗОВНІШНІМИ ПРИСТРОЯМИ. НЕСТАНДАРТНІ ПРИСТРОЇ

3.1 Мета роботи

Навчитися практичному використанню функцій WINAPI для роботи з файлами

3.2 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 SetFilePointer, SetEndOfFile;
9. Функції для визначення та встановлення атрибутів файлів GetFileAttributes, SetFileAttributes.

3.3 Завдання до лабораторної роботи

Є 2 різних завдання. Студент обирає одне з завдань по своєму бажанню

3.3.1 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.

Для отримання оцінки відмінно додати функцію для контролю цілісності поштової скриньки. Для контролю цілісності використовувати CRC або інші засоби контролю цілісності.

3.3.2 Створення класу для роботи з пристроями, файлами та каталогами

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

3.4 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?
- 8 Які засоби контролю цілісності ви знаєте?

4 КЕРУВАННЯ ПАМ'ЯТТЮ

4.1 Мета роботи

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

4.2 Підготовка до роботи

При підготовці до роботи вивчити класи функцій для роботи з пам'яттю WINAPI:

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

4.2.1 Інформаційні функції

4.2.1.1 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.

4.2.1.2 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

4.2.1.3 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

4.2.2 Функції для керування віртуальною та фізичною пам'яттю

4.2.2.1 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*

4.2.2.2 Функція VirtualFree

```
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
```

4.2.2.3 Функція VirtualProtectEx

Використовується, якщо треба змінити рівень доступу до пам'яті, який було призначено при виділенні. Можна змінити рівень доступу не для всього, а для частини виділеного регіону.

```
BOOL WINAPI VirtualProtectEx(
__in    HANDLE hProcess,
__in    LPVOID lpAddress,
__in    SIZE_T dwSize,
__in    DWORD flNewProtect,
__out   PDWORD lpflOldProtect
);
```

Де:

hProcess – дескриптор процесу;

lpAddress – адреса

dwSize – розмір;

flNewProtect – новий рівень доступу;

lpflOldProtect – старий рівень доступу.

4.2.2.4 Функція VirtualLock і VirtualUnlock

Призначення для запобігання запису критичних даних на диск .

```
BOOL WINAPI VirtualLock(
__in    LPVOID lpAddress,
__in    SIZE_T dwSize
```


);

```

BOOL WINAPI VirtualUnlock(
    __in    LPVOID lpAddress,
    __in    SIZE_T dwSize
);

```

4.2.3 Функції для відображення файлів на пам'ять

Використовуються, якщо необхідно:

- 1 записи файлу обробляти в довільному порядку;
- 2 Один і той же файл використовується різними програмами
- 3 Декілька програм використовують загальну пам'ять

4.2.3.1 Функція CreateFile

Використовується для отримання інформації про файл, а саме, де розташоване, які права доступу, розмір

4.2.3.2 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.

4.2.3.3 Функція MapViewOfFile

Виконується 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 0!!!)

Після цієї функції можна використовувати дані з пам'яті

4.2.3.4 Функція UnmapViewOfFile

Визволяє фізичну пам'ять

```
BOOL WINAPI UnmapViewOfFile(LPCVOID lpBaseAddress);
```

lpBaseAddress – адреса, яку повернула функція MapViewOfFileEx

4.2.3.5 Функція CloseHandle

BOOL WINAPI CloseHandle(HANDLE hObject);

hObject – дескриптор.

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

4.3 Завдання до лабораторної роботи

У даній лабораторній роботі необхідно виконати наступне.

1. Скласти програму для формування системної інформації про віртуальну пам'ять і пояснити отримані результати;
2. За допомогою функції VirtualQuery побудувати список вільних блоків пам'яті. Реалізувати функції виділення та визволення пам'яті за допомогою стратегії Найменший достатній.
3. Реалізувати алгоритм заміщення сторінок.
4. Реалізувати алгоритм LRU, який використовується для 4-х направленої кешу
5. Переробити програму для поштової скриньки з використанням відображення файлу на пам'ять (на оцінку Відмінно)

4.4 Зміст звіту

Звіт повинен вміщувати наступні частини:

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

4.5 Контрольні питання й завдання

1. Дайте визначення різних типів пам'яті.
2. Поясніть 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?

5 КЕРУВАННЯ ПРОЦЕСАМИ

5.1 Мета роботи

Навчитися запускати програми і командні файли із програми.

5.2 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, // Середовище
    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;

```

5.3 Порядок виконання лабораторної роботи

1. Скласти макроси для спрощення використання функції `CreateProcess` для випадків негайного й відкладеного запуску програми.
2. Скласти 3 програми. Перша програма має запустити текстовий редактор і створити текстові файли в заданій папці. Друга програма 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. Для другої програми має бути встановлений нижчий пріоритет ніж у інших програм.
3. Для отримання інформації про процеси використати наступні функції (файл заголовків `tlhelp32.h`):
 - 3.1 Створити дескриптор для доступу до інформації;


```

HANDLE WINAPI CreateToolhelp32Snapshot(
    __in DWORD dwFlags,
    __in DWORD th32ProcessID
);

```

`dwFlags` - показує, яка інформація потрібна, наприклад, `TH32CS_SNAPPROCESS` – інформація про процеси;

`th32ProcessID` – ідентифікатор процесу, якщо 0-ідентифікатор поточного процесу

3.2 Отримати інформацію про перший процес

```

BOOL WINAPI Process32First(
    __in    HANDLE hSnapshot,
    __inout LPPROCESSENTRY32 lppe
);

```

hSnapshot – дескриптор процесу;

lppe - структура з інформацією про процес (PROCESSENTRY32).

3.3 Інформація про наступний процес

```

BOOL WINAPI Process32Next(
    __in    HANDLE hSnapshot,
    __out   LPPROCESSENTRY32 lppe
);

```

3.4 Структура PROCESSENTRY32:

```

typedef struct tagPROCESSENTRY32 {
    DWORD dwSize;
    DWORD cntUsage;
    DWORD th32ProcessID;
    ULONG_PTR th32DefaultHeapID;
    DWORD th32ModuleID;
    DWORD cntThreads;
    DWORD th32ParentProcessID;
    LONG pcPriClassBase;
    DWORD dwFlags;
    TCHAR szExeFile[MAX_PATH];
} PROCESSENTRY32, *PPROCESSENTRY32;

```

3.5 Функція CloseHandle для закриття дескриптору.

3.6 За допомогою цих функцій для кожного процесу визначити:

- його ім'я;
- DLL, які використовуються процесом;
- Кількість потоків для процесу

5.4 Зміст звіту

Звіт має містити:

1. Опис всіх використаних функцій WINAPI;
2. Складені макроси;
3. Тексти всіх складених програм;
4. Висновки по роботі з переліком вмінь, які отримані в результаті виконання лабораторної роботи.

5.5 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?

6 КЕРУВАННЯ ПОТОКАМИ

6.1 Мета роботи

Вивчити методи керування потоками, які використовують загальні ресурси.

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

Необхідно вивчити функції створення потоків: `CreateThread`.

`HANDLE CreateThread(`

`LPSECURITY_ATTRIBUTES SecurityAttributes, // Атрибути безпеки`

`DWORD StackSize, // Розмір стека`

`LPTHREAD_START_ROUTINE StartFunction, //Потокова функція`

`LPVOID ThreadParameter, // Параметр потокової функції`

`DWORD CreationFlags, // Прапорці створення потоку`

`LPDWORD ThreadId // Ідентифікатор потоку`

`);`

6.2 Підготовка до роботи і порядок її виконання

1. Вивчити особливості створення й завершення процесів, їх призначення, процеси-нащадки і властивості, що отримуються у спадок від процесу - батька (конспект лекцій, файл WIN32.HLP);

2. Вивчити функцію створення потоків: `CreateThread`(конспект лекцій, файл WIN32.HLP);

3. Вивчити потоки, способи їх створення і завершення (конспект лекцій, файл WIN32.HLP);

4. Вивчити розподіл часу між потоками і можливості програміста по керуванню цим розподілом. Пріоритетне обслуговування потоків (конспект лекцій, файл WIN32.HLP);

5. Вивчити способи синхронізації потоків (конспект лекцій, файл WIN32.HLP):

- без операційної системи (конспект лекцій, файл WIN32.HLP);
- за допомогою критичних секцій (конспект лекцій, файл WIN32.HLP);

6. Вивчити засоби синхронізації процесів (конспект лекцій, файл WIN32.HLP):

- м'ютекси;
- семафори;

таймери, що очкуються.

6.2.1 Задача 1

Запустити 10 потоків, в яких спочатку видати інформацію про початок потокової функції, в кінці – про її завершення. В потоковій функції використовувати цикл, завдяки якому час виконання потокової функції буде більше одного кванта часу. Рахувати, скільки разів буде викликатися потокова функція.

Чекати завершення хоча б одного потоку Рахувати, скільки разів буде викликатися потокова функція.

Зробити так, щоб інформація про початок та кінець для потокової функції виводився в файл тільки в режимі DEBUG, забезпечивши можливість використання функції різними програмами.

6.2.2 Задача 2

Визначити максимальну кількість потоків, які можна запустити та чекати їх завершення.

Для цього зробити цикл для кількості потоків 2, 4, 8, ..., в якому

- створити задану кількість потоків, які формують кількість викликів потокової функції;
- чекати їх завершення.
- Якщо кількість викликів потокової функції не співпадає з кількістю потоків, вийти з циклу.

6.2.3 Задача 3

Задача 3 або задача 4 вирішується на вибір студента

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

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

Перевірити можливість використання стандартної черги для роботи з потоками

6.2.4 Задача 4

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.

6.2.5 Задача 5

В тій же програмі 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.

6.2.6 Завдання на найвищу оцінку

Створити додаткову програму, яка запускає декілька раз першу програму з потоками відповідно з заданим розкладом (WaitableTimer). Внести зміну в об'єкти синхронізації, які потрібні.

6.3 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

6.4 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.

СПИСОК ЛІТЕРАТУРИ

- 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](https://ru.wikibooks.org/wiki/Операционные_системы) - 10.05.2016 г.- Загол. с экрана.

Навчальне видання

МЕТОДИЧНІ ВКАЗІВКИ
до лабораторних занять з дисципліни
«ОПЕРАЦІЙНІ СИСТЕМИ»
для студентів усіх форм навчання спеціальностей
6.050103 «Програмна інженерія»

Упорядники: КАЧКО Олена Григорівна
МЕЛЬНІКОВА Роксана Валеріївна

Відповідальний випусковий

Редактор

План 201_, 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