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 усіх форм навчання напряму 6.050103 Програмна інженерія / Упоряд.: Качко О.Г., Мельникова Р.В. – Харків: ХНУРЕ, 2016. – 40 с.

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

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

ЗМІСТ

ЗАГАЛЬНІ ПОЛОЖЕННЯ

Мета методичних вказівок – надати допомогу студентам у ході підготовки, виконанні та оформленні результатів виконання лабораторних робіт з дисципліни, основним призначенням якої є вивчення принципів побудови сучасних операційних систем і найбільш ефективного використання їх функцій під час розробки програм та їх застосування.

Лабораторні роботи з дисципліни вчать практичному використанню теоретичних положень, які вивчені на лекціях, закріплюють та розширюють практичні навички, що були отримані на практичних заняттях з курсу.

Кожна лабораторна робота потребує попередньої підготовки, а саме:

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

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

- виконувати правила техніки безпеки під час роботи з ЕОМ;
- виконувати правила поведінки в лабораторії ЕОМ;
- приходити на заняття чітко за розкладом;
- не допускається зміна прав доступу до системних ресурсів.

До роботи допускаються тільки підготовлені студенти, які склали та захистили програми. В ході допуску викладач перевіряє наявність цих програм і ступінь самостійності їх складання.

Студенти самостійно виправляють помилки у програмі. Сама помилка та дії, необхідні для її виправлення, заносяться до звіту з лабораторної роботи. Найбільш характерні помилки обговорюються з групою в ході виконання лабораторних робіт.

Звіт має включати в себе:

– назву лабораторної роботи;

– мету лабораторної роботи;

– завдання і тексти програм, тести, помилки, які знайдені в ході виконання тестів, результати виконання програм, висновки.

Як шаблон для підготовки звіту можна використовувати цей документ в електронному вигляді. На титульному аркуші звіту слід 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. В процесі отримання заліку з лабораторних робіт в кінці семестру студент має подати викладачу повний звіт з усіх лабораторних робіт 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 задання одного символу. Цього не достатньо для задання текстів, у яких є речення різними мовами. UNICODE-кодування застосовує два байти під час кодування одного символу. При такому кодуванні максимальна кількість кодів символів 65536 порівняно з 256 для ASCII-кодування, що забезпечує можливість використання усіх наявних на сьогодні мов. Більшість текстових файлів сьогодні мають ASCII-кодування. Кодування імен файлів залежить від наявної файлової системи, в електронних листах можна використовувати обидва типи кодування. Метою даної лабораторної роботи є навчитися опрацьовувати тексти для обох типів кодування, причому 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

Під час використання ASCII-кодування використовується тип даних char. Даному типу char виділяється один байт. Приклади використання цього типу:

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

В ході використання 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 універсального оголошення типу даних та їх ініціалізації:

```
#ifdef UNICODE
    typedef     wchar_t     TCHAR ;
    #define     TEXT(a)     L##a
#else
    typedef     char        TCHAR ;
    #define     TEXT(a)     a
#endif
```

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

Розгляньте цей макрос! Він має 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, wcscpy, wcscat, wcslen,… .

Визначимо універсальну функцію, наприклад, для визначення довжини

рядку (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 у файлі tchar.h, який необхідно підключити для використання універсального кодування. Цей файл автоматично підключається Visual Studio з використанням непорожнього проекту. В цьому випадку створюється файл stdafx.h. Відкрийте його та знайдіть рядок #include <tchar.h>. Оберіть цей рядок та за допомогою правої кнопки 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 самий 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 файла. Необхідно визначити, яке кодування використовується для задання 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 у буфері;

lpi – ознаки, за якими 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 за більшістю ознак типу Unicode, FALSE для типу ASCII.

Функція для перетворення ASCII в Unicode MultiByteToWideChar :

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

Функція повертає кількість символів у рядку-результаті.

Функція для перетворення Unicode в ASCII

```
int WideCharToMultiByte(
 UINT CodePage,          // Кодова сторінка, зазвичай 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 // Покажчик на прапорець, який вказує на
```

```
    // те, чи використовувався символ за замовчуванням у
    // попередньому  параметрі. Може бути NULL.
);
```

Функція повертає кількість символів у 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 за даними макросами в командному рядку. Для цього визначимо командний рядок для трансляції програми:

Properties→C/C++→Command Line

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

3. Переключіть режим задання символу на протилежний. Для підключення режиму з UNICODE в режим ASCII використовують:

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

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

4. Після переключення 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 літер 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 у UNICODE. Для сортування використовувати універсальну стандартну функцію qsort:

```
void qsort(
void *base,          // Масив, що упорядковуємо
size_t num,          // Кількість елементів масиву;
size_t width,              // Ширина елемента 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 у цьому файлі переставити ці символи у зворотному порядку. Символи кінця рядка залишити у правильному порядку (це завдання для отримання найвищої оцінки).

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 російською (українською) мовами;

– тексти розробленої програми з коментарями;

– тести для перевірки правильності програми;

– Висновки з роботи.

1.5 Контрольні запитання та завдання

1. Навіщо створювати універсальні додатки для роботи з ASCII, UNICODE?

2. Як забезпечити універсальність оголошення 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 необхідно змінити в програмі для упорядкування масиву ASCII-символів?

2 СТВОРЕННЯ ТА ВИКОРИСТАННЯ БІБЛІОТЕК

2.1 Мета роботи

Вивчити прийоми та методи створення й 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 бібліотеку, яку було створено у середовищі C++ Builder, можна використовувати в середовищі Visual Studio та навпаки;

– зі зміною коду бібліотеки не потрібне повторне компонування додатків, які використовують дану бібліотеку, ось чому операційна система використовує цей тип бібліотек для модулів, які можуть змінюватися залежно від версії та в разі помилок;

– якщо декілька додатків використовують одну й ту саму 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 і 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 задання імені проекту обирається тип DLL і поки що обираємо порожню 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 компонування DLL виконується як і для проектів інших типів.

Внаслідок виконання буде створено один або 2 файли: файл *.dll і, *.lib, якщо є хоча б одна експортована функція. Імена файлів за замовчуванням 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 інші 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 в ході виконання 2.2.4. Для цього в Solution Explorer обрати проект з програмою, яка використовує DLL, правою кнопкою мишки обрати Reference та обрати

необхідну бібліотеку. У 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 за складністю однакове 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 з завантаженням 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, слід визначити адресу цієї функції. Для цього використовується функція

FARPROC WINAPI GetProcAddress(HMODULE hModule, LPCSTR lpProcName);

де hModule – дескриптор бібліотеки;

lpProcName – імя 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 звільнить усі ресурси при завершенням програми. DLL належить до ресурсів.

2.2.4 Алгоритм шифрування RSA

1. Формується 2 простих числа (p, q, p!=q).

2. Формується додаток для цих чисел, далі називатимемо цей добуток модулем n = p * q.

3. Формується функція Ейлера для цих чисел за формулою:

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

4. Формуються ключі для криптоперетворень. Ключ для шифрування 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.

5. Шифрувати можна довільні дані, значення яких менше ніж n[1]. Для шифрування даних використовується ключ E, який у 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

[1] Для подолання цього обмеження дані поділяються на блоки, розмір кожного блока задовольняє обмеженню.

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

2.3 Порядок виконання лабораторної роботи

1. Визначіть необхідні функції для реалізації алгоритму RSA. Визначіть серед цих функцій ті, які повинні використовуватися в зовнішніх програмах.

2. Визначіть умову з виклику для функцій, якщо відомо, що функції слід використовувати в C, C++ файлах і вони мають бути максимально надійними.

3. Складіть файл заголовків таким чином, щоб цей файл заголовків можна було б використовувати для динамічної бібліотеки і 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 – виведення різними мовами.

2.4 Зміст звіту

Звіт має містити:

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 такі 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 файла GetFileSize.
7. Функції для введення – виведення даних з файлів ReadFile, WriteFile.
8. Функції для позиціонування покажчика у файлі та встановлення кінця файла зі зменшенням 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 за своїм бажанням.

3.3.1 Моделювання поштової скриньки

У 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 має задаватися у вигляді: розмір повідомлення, тіло

повідомлення. Максимальний розмір поштової скриньки задається під час створення поштової скриньки.

Для об’єкта визначити функції додавання листів, читання листів з видаленням та без видалення, видалення заданого листа та усіх листів, визначення кількості листів, а також визначення загальної кількості 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 у 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 в ході їх використання для C#, C у разі використання великих файлів.

3.4 Контрольні запитання та завдання

1. Які прапорці необхідно використовувати для функції CreateFile для створення нового і відкриття існуючого файла?

2. Задайте прапорці для функції CreateFile для дозволу введення – виведення з файла іншими програмами.

3. Задайте прапорці для функції CreateFile для файла, який спочатку читається, а потім модифікується.

4. Яка функція використовується для визначення розміру файла?

5. Які параметри необхідно визначити для файла, 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

Під час підготовки 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 щодо процесора 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 загалом, 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

слід 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 записуються 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 в процесі виділення 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 необхідно 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 файла обробляти в довільному порядку;

2) один і той самим 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 у 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 файла, 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 слід виділяти віртуальну пам'ять.

lpName – ім'я відповідного об'єкта ядра.

4.2.3.3 Функція MapViewOfFile

Виконується виділення фізичної пам’яті та читання файла в цю пам’ять.

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

Де:

hFileMappingObject – дескриптор відображеного об’єкта;

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 відносно до початку файла;

dwNumberOfBytesToMap – кількість байтів, які слід 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

Очищує 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 необхідно викликати для дескриптора – відображення та дескриптора файла, 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 вивільнення пам’яті за допомогою стратегії найменший достатній.

3. Реалізувати алгоритм заміщення сторінок.

4. Реалізувати алгоритм LRU, який використовується для 4-спрямованого кеша.

5. Переробити програму для поштової скриньки з використанням відображення файла на пам'ять (на оцінку Відмінно).

4.4 Зміст звіту

Звіт повинен вміщувати такі частини:

– повний опис функцій для роботи з пам'яттю, що були використані у лабораторній роботі;

– тексти програми;

– пояснення отриманих результатів;

– висновки.

4.5 Контрольні запитання та 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 відбудеться, якщо:

– програміст забув очистити пам'ять?

– у зв'язку з аварійним завершенням програми вона не дійшла до коду очищення?

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, ніж 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

Вивчіть функцію CreateProcess та 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 має 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 такі 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 дескриптора.

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 усіх використаних функцій WINAPI;

2) складені макроси;

3) тексти всіх складених програм;

4) висновки з роботи з переліком вмінь, які отримані внаслідок виконання лабораторної роботи.

5.5 Контрольні запитання та завдання

1. Як необхідно підготувати структуру STARTUPINFO перед створенням процесу?

2. Чим відрізняється запуск програм, що виконуються від запуску командного файла?

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 за ідентифікатором процесу його дескриптор?

7. Як довідатися ім'я програми, що виконується за дескриптором 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_ROUTIN 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 з керування 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. У потоковій функції використовувати цикл, завдяки якому час виконання потокової функції буде більше одного кванта часу. Порахувати, скільки разів викликатиметься потокова функція.

Очікувати завершення хоча б одного потоку, рахувати, скільки разів викликатиметься потокова функція.

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

6.2.2 Задача 2

Визначити максимальну кількість потоків, які можна запустити та очікувати 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 і використати їх, якщо є кілька потоків виробника та споживача.

У головній програмі передбачити, що споживач записує імена файлів у 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 додає запис у кінець списку новин. Потік читача читає останню новину.

6.2.5 Задача 5

У тій самій програмі реалізувати потоки для філософів, що обідають. Потік має 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 і блокувань з використання потоків в одній програмі.

6.2.6 Завдання на найвищу оцінку

Створити додаткову програму, яка запускає декілька разів першу програму з потоками відповідно до заданих 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 в ході вирішення конкретних завдань.

6.4 Контрольні запитання та 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 та м’ютексом?

6. Які за кожним способом синхронізації переваги й недоліки порівняно з іншими способами?

7. Які способи синхронізації процесів ви знаєте?

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

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

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

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

3. Конспект лекцій «Програмування мовою С++» по курсу / Упоряд. О.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. Деревянко А.С., Солощук М.Н. Операционные системы: Учеб. 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. – Загол. с экрана.