МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ
УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

.

МЕТОДИЧНІ ВКАЗІВКИ

до практичних занять та самостійної роботи
з дисципліни «ОПЕРАЦІЙНІ СИСТЕМИ»

для студентів усіх форм навчання спеціальностей

6.050103 «Програмна інженерія»

ЗАТВЕРДЖЕНО
кафедрою ПІ
Протокол №18 від 13.04.16

ХАРКІВ 2016

Методичні вказівки до практичних занять та самостійної роботи з дисципліни «Операційні системи» для студентів спеціальності 6.050103 Програмна інженерія / Упоряд. Качко О.Г., Мельникова Р.В. – Харків: ХНУРЕ, 2016. –62 с.

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

Рецензент

Навчальне видання

МЕТОДИЧНІ ВКАЗІВКИ

до практичних занять та самостійної роботи
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 200_, 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

ВСТУП

Дисципліна «Операційні системи» направлена на вивчення теорії, прийомів та методів створення та ефективного використання операційних систем. Дисципліна має велике значення в формуванні спеціаліста-професіонала в галузі програмної інженерії незалежно від технологій, які будуть використовуватися в практичній діяльності.

Сьогодні найбільш поширеними є операційні системи фірми Microsoft, тому при виконанні практичних занять передбачається використання операційних систем цієї фірми. Але основні функції, які вивчаються, можуть бути реалізовані в середовищі будь яких операційних систем. Якість навчання підвищиться, якщо студенти при виконанні практичних завдань розглянуть можливість їх виконання для інших операційних систем, наприклад, Linux.

Як і для інших дисциплін комп'ютерного направлення велике значення має самостійна робота студента. Без самостійного вивчення і постійної практики виконання практичних завдань, лабораторних робіт, контрольних робіт, неможливо професійно опанувати та грамотно використовувати функції операційних систем при рішенні реальних задач.

Рекомендуємо наступну методику підготовки до практичних занять по дисципліні "Операційні системи":

вивчити теоретичний матеріал до практичного заняття;

розібрати рішення задач, які наведені в даних методичних вказівках до заданого практичного заняття;

скласти самостійно програми, які наведені в даних методичних вказівках до заданого практичного заняття[1];

реалізувати складені програми на комп'ютері;

перевірити роботу програми на тестових прикладах.

Після практичного заняття необхідно:

скласти тести відповідно домашньому завданню;

скласти програми відповідно домашньому завданню;

---

[1] Якщо цей і наступний пункти неможливо виконати до практичного заняття, то його обовязково треба виконати після нього. Якщо студент не може його виконати після практичного заняття, треба звернутися до викладача за консультацією

реалізувати ці програми на комп’ютері;

перевірити роботу програм на тестових прикладах.

Успіх вивчення гарантується, якщо кожне практичне заняття виконується по запропонованій технології. Більшість задач з вказівок приведені з рішенням у виді готових програм. Це дозволяє студентам в міру необхідності розбирати готові рішення та аналізувати їх роботу за допомогою відлагоджувача .

В якості базового середовища для розробки програм використовується Microsoft Visual Studio Net версії 2013 або вище.

1 СТВОРЕННЯ УНІВЕРСАЛЬНИХ ДОДАТКІВ ДЛЯ UNICODE ТА ANSI КОДУВАНЬ. ВИЗНАЧЕННЯ ХАРАКТЕРИСТИК ПРОЦЕСОРА ТА ОПЕРАЦІЙНОЇ СИСТЕМИ

1.1 Мета заняття

Велика кількість функцій операційних систем використовують в якості параметрів рядки символів. Для кодування символів сьогодні використовується найчастіше 2-х байтове кодування, яке забезпечує можливість завдання 65536 різних символів. Це достатньо для зберігання рядків, які використовують різні алфавіти. Але багато програмних засобів розроблене для символьних даних, для кодування яких застосовувалось однобайтове кодування (ASCII). Для забезпечення можливості використання сучасних та раніше розроблених модулів використовується універсальне кодування, коли одна функція може використовуватись як для одно так і для двох байтового кодування. Мета лабораторної роботи – навчитись використовувати як конкретні кодування: ASCII, UNICODE, так і універсальне кодування, що дозволить створювати універсальні функції

1.2 Методичні вказівки з організації самостійної роботи студентів

1.2.1 Формування консольних додатків для Visual Studio Team System 2013

По документації до Visual Studio Net вивчить засоби створення консольного проекту для C++.

Так, для Visual Studio Team System 2013, необхідно обрати File→New→Project→Win32 Project і, після завдання папки для зберігання проекту та його імені, в новій формі обрати Application Setting і встановити Console application.

В результаті сформується файл, який містить:

```
#include "stdafx.h"

int _tmain(int argc, _TCHAR* argv[]){
return 0;
}
```

Файл stdafx.h, який формується автоматично, може використовуватися для завдання усіх заголовчих файлів, які будуть потрібні для програми. Цей файл використовується для попередньої компіляції усіх заголовчих файлів, які задані в файлі stdafx.h. Це може зменшити час, необхідний для компіляції проекту. Якщо цей файл використовується, його треба підключати до усіх .cpp файлів, а компіляцію треба починати з файлу stdafx.cpp, який створюється автоматично разом з stdafx.h. Якщо цей файл використовувати не планується, можно відключити його використання. Для цього необхідно:в Project → <Ім'я проекту> Properties→Configuration Properties→C/C++→Precompiled Headers → Create /Use Precompiled Headers замість Use Precompiled Headers обрати Not Using Precompiled Headers. Після цього можно з проекту виключити файл stdafx.cpp та видалити підключення stdafx.h.

1.2.2 Формування універсальних додатків

По [3] вивчити можливість використання ANSI та UNICODE символів та особливості створення програм, які працездатні незалежно від вибору типу кодування. ASCII кодування символів використовує один байт для завдання одного символу. Тобто можна задати 256 різних символів. Цього часто буває недостатньо. Тому частіше використовується UNICODE кодування, при якому один символ займає 2 байти і всього можна задати 65536 різних символів. Сучасні операційні системи, як за правило використовують UNICODE кодування по замовченню. Але усі старі програми використовували ASCII кодування. Програмісти і в сучасних програмах в якості імені файлу, наприклад, також використовують рядки типу "ASCII.TXT". Такий рядок записується в пам'ять їх ASCII масив. Тому серед функцій WINAPI? які використовують окремі символи або масиви, є 2 типу: функцій: для роботи з ASCII та UNICODE символами. Перші функції наприкінці мають букву A, інші – W. Якщо остання буква функції не задана, то середовище використовує те кодування, яке задане в властивостях проекту. Для визначення та завдання цієї властивості необхідно обрати Project → <Ім'я проекту> Properties→Configuration Properties→General→Caracter Set. Якщо встановлено Use Unicode Caracter Set, то використовується Unicode кодування, якщо Use Multi-Byte Caracter Set – то використовується ASCII. Тут же можна змінити засіб кодування. Починаючі з Windows 2000 фактично реалізовані функції для Unicode. Функції для ASCII виділяють динамічно пам'ять для відповідних рядків в Unicode, перетворюють рядок в Unicode і записують його в виділену пам'ять, виконують необхідну функцію, перетворюють результат, якщо необхідно в ASCII рядок, звільняють пам'ять. Таким

чином, швидкість виконання функцій для ASCII менше, ніж для Unicode. Ось чому дуже важливо навчитися писати додатки, які працюють для обох режимів кодування.

Як створювати такі додатки?

Як ми знаємо, в разі ASCII кодування використовується тип символу char. Приклади завдання окремого символу і рядка:

```
char ASCIIChar = 'a', ASCIIArray [] = "abc";
```

В разі Unicode кодування використовується тип символу wchar_t. Приклади завдання окремого символу і рядка:

```
wchar_t UnicodeChar = L'a', UnicodeArray [] = L"abc";
```

Зверніть увагу на завдання значень для символу та рядка!

Напишемо макрос, який дозволить використовувати обидва варіанта визначення.

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

Змінна UNICODE визначається при завданні в середовищі типу кодування (Use Unicode Caracter Set, або Use Multi-Byte Caracter Set).

Код для встановлення визначеного режиму:

```
TCHAR UniversalChar = TEXT('a'), UniversalArray [] = TEXT("abc");
printf ("sizeof (UniversalChar) = %d sizeof (UniversalArray) = %d\n",
        sizeof (UniversalChar), sizeof (UniversalArray));
```

Результат роботи коду UniversalChar = 1 UniversalArray = 4 для ASCII кодування і UniversalChar = 2 UniversalArray = 8 для Unicode кодування.

Такі визначення зроблені в файлі TCHAR.H, тому для створення універсальних додатків необхідно, перш за все, підключити цей файл. В файлі TCHAR.H не тільки визначено універсальний тип даних (TCHAR), а і задані функції для роботи з рядками. Так, усі функції, які мають ім’я, яке починається з префікса str, в універсальному варіанті мають префікс _tcs. Введена заміна і для інших функцій стандартних бібліотек, які обробляють окремі символи та рядки. Якщо Вам в програмі необхідно виконати обробку символу або рядка за допомогою стандартної функції рекомендуємо знайти цю функцію в

файлі TCHAR.H і замінити її універсальним варіантом цієї функції. Так, ім’я головної функції консольного додатку main замінено на його універсальний варіант _tmain.

Завдання для самостійної роботи. Вивчить вміст файлу tchar.h. Його вивчення навчить вас створювати універсальні додатки не тільки для кодування різними способами, але і для інших випадків, колди треба створити код, який дуже просто настроїти відповідно необхідним властивостям!

1.2.3 Функції для перетворення типів кодування

Функція для перетворення ASCII кодування в UNICODE.

```
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 зворотного перетворення

```
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.

1.2.4 Функція для визначення типу кодування

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

pBuffer – буфер, в якому текст для перевірки;

lpi – признаки. По яким виконується перевірка. Початковий стан -1. Результат – 1 в бітах, по яким це текст в UNICODE.

Функція повертає:

TRUE - імовірно текст в UNICODE;

FALSE - імовірно текст в ASCII.

1.3 Контрольні запитання і завдання для самостійної роботи

1 Які типи проектів можна створювати в Microsoft Visual Studio?

2 Як створити консольний додаток?

3 Для чого використовується файл stdafx.h. Як його використовувати?

4 Який файл треба підключити для забезпечення універсальної обробки ASCII і UNICODE рядків?

5 Перевірте можливість введення / виведення рядків в UNICODE за допомогою функцій з iostream

6 Для чого використовується директива #pragma once на початку стандартних файлів типу h?

1.4 Приклади аудиторних задач

1 Визначити тип кодування, який використовується в середовищі по замовченню?

```
#include <tchar.h>
int _tmain(int argc, TCHAR* argv[])
{
        char ASCIIChar = 'a', ASCIIArray [] = "abc";
        wchar_t UnicodeChar = L'a', UnicodeArray [] = L"abc";
```

```
TCHAR UniversalChar = TEXT('a'), UniversalArray [] = TEXT("abc");
printf ("sizeof (UniversalChar) = %d sizeof (UniversalArray) = %d\n",
        sizeof (UniversalChar), sizeof (UniversalArray));
return 0;
}
```

Змінити тип кодування і перевірити отримані значення.

Залишити тип кодування UNICODE

2 Задані ім’я каталогу і ім’я файлу. Створити повне ім’я файлу. Ім’я каталогу може включати та не включати символ \ в кінці. Код повинен працювати без помилок для обох способів кодування.

Функція:

```
enum ERRORS
{
        OK = 0,         BAD_PARAM
};
int CreateFileName (const TCHAR *tcPath, const TCHAR *tcName, TCHAR *tcFullName )
{
int res = BAD_PARAM;
if (tcPath && tcFullName)
{
        _tcscpy (tcFullName, tcPath);
        size_t len = _tcslen (tcFullName);
        if (tcFullName [len - 1] !=  _TEXT('\\'))
                tcFullName [len++] =  _TEXT('\\');
        _tcscpy (tcFullName + len, tcName);
        res = OK;
}
return res;
}
```

Фрагмент головної програми:

```
TCHAR tcPath1 [MAX_PATH] = _TEXT("C:\\Temp");
TCHAR tcPath2 [MAX_PATH] = _TEXT("C:\\Temp\\");
TCHAR tcName [MAX_PATH] = _TEXT("x.txt");
```

```
TCHAR tcFullName [MAX_PATH];
int res = CreateFileName (tcPath1, tcName, tcFullName );
_tprintf (_TEXT("tcFullName = %s\nres = %d\n\n"), tcFullName, res);
res = CreateFileName (tcPath2, tcName, tcFullName );
_tprintf (_TEXT("tcFullName = %s\nres = %d\n\n"), tcFullName, res);
```

Обов’язково треба перевірити роботу програми для обох способів кодування!

3. Задано рядок в форматі ASCII. Перетворити цей рядок в масив символів UNICODE, а потім знову в ASCII. Перевірити правильність перетворення.

```
char str1 []= "Hello world";
wchar_t str2 [sizeof (str1)];
char str3 [sizeof (str1)];
int n = MultiByteToWideChar(
        CP_ACP,          // Кодова сторінка, звичайно задається CP_ACP
        0,               // 0
        str1,            // Рядок, який перетворюється
        sizeof (str1),   // Розмір (в байтах)
        str2,            // Рядок - результат
        sizeof (str1)    // Розмір в wchar_t символах.
);
if (n)
        n = WideCharToMultiByte(
                CP_ACP,  // Кодова сторінка, звичайно задається CP_ACP
                0,       // 0
                str2,    // Рядок, який перетворюється
                n,       // Розмір
                str3,    // Рядок - результат
                n,       // Розмір
                NULL,    // Адреса символу, яким заміняється
                NULL     // Покажчик на прапорець, який вказує на
        );
BOOL bRes = strcmp (str1, str3) == 0;
TCHAR *Res = bRes? _T("OK") : _T("Error");
_tprintf (_T("Res = %s\n"), Res);
```

Задача 4

Задано 2 тексти. Для завдання текстів можна використовувати довільний формат, в тому числі різний. Перевірити, чи однакові тексти (відомо, що вони написані англійською мовою. В тексті можуть бути маленькі та великі букви. При перевірці регістр не враховувати.

Функція для порівняння

```
BOOL CompareTexts (PVOID Text1, size_t n1, PVOID Text2, size_t n2){
    int iPriz1 = -1, iPriz2 = -1;
    BOOL b1 = IsTextUnicode( Text1, n1, &iPriz1);
    BOOL b2 = IsTextUnicode( Text2, n2, &iPriz2);
    wchar_t *p1 = 0, *p2 = 0;
    // Якщо рядок1 не UNICODE, перетворюємо його
    if (!b1) {
        p1 = new wchar_t [n1];
        n1 = MultiByteToWideChar(
            CP_ACP,      // Кодова сторінка, звичайно задається CP_ACP
            0,           // 0
        (char*)Text1,    // Рядок, який перетворюється
        n1,              // Розмір (в байтах)
        p1,              // Рядок - результат
        n1               // Розмір в wchar_t символах.
    );
    }
    else
        p1 = (wchar_t*)Text1;
    // Якщо рядок2 не UNICODE, перетворюємо його
    if (!b2) {
        p2 = new wchar_t [n2];
        n2 = MultiByteToWideChar(
            CP_ACP,      // Кодова сторінка, звичайно задається CP_ACP
            0,           // 0
            (char*)Text2,// Рядок, який перетворюється
            n2,          // Розмір (в байтах)
            p2,          // Рядок - результат
```

```
                    n2              // Розмір в wchar_t символах.
                );
        }
        else
                p2 = (wchar_t*)Text2;
        // Порівняння рядків без урахування регістру
        BOOL b = wcsicmp (p1, p2) == 0;
        // Визволення пам’яті, якщо треба
        if (!b1) delete []p1;
        if (!b2) delete []p2;
        // Повернення результату
        return b;
}
```

Головна програма (фрагмент)

```
char cstr1[] = "Hello world";
char cstr2[] = "HELLO WORLD";
char cstr3[] = "HILLO WORLD";
wchar_t wstr1[] = L"Hello world";
wchar_t wstr2[] = L"HELLO WORLD";
// cstr1 == cstr2
bRes =  CompareTexts (cstr1, sizeof (cstr1), cstr2, sizeof (cstr2));
Res = bRes? _T("OK") : _T("Error");
_tprintf (_T("cstr1 == cstr2 Res = %s\n"), Res);
// cstr1 == wstr2
bRes =  CompareTexts (cstr1, sizeof (cstr1), wstr2, sizeof (wstr2));
Res = bRes? _T("OK") : _T("Error");
_tprintf (_T("cstr1 == wcstr2 Res = %s\n"), Res);
// wstr1 == cstr2
bRes =  CompareTexts (wstr1, sizeof (wstr1), cstr2, sizeof (cstr2));
Res = bRes? _T("OK") : _T("Error");
_tprintf (_T("wstr1 == cstr2 Res = %s\n"), Res);
// wstr1 == wstr2
bRes =  CompareTexts (wstr1, sizeof (wstr1), wstr2, sizeof (wstr2));
Res = bRes? _T("OK") : _T("Error");
```

```
_tprintf (_T("wstr1 == wstr2 Res = %s\n"), Res);
// cstr1 != cstr3
bRes =  CompareTexts (cstr1, sizeof (cstr1), cstr3, sizeof (cstr3));
Res = bRes? _T("OK") : _T("Error");
_tprintf (_T("cstr1 == cstr3 Res = %s\n"), Res);
```

Зверніть увагу на вибір даних для перевірки функції!!!

1.5 Приклади домашніх задач

Приклад 1. Скласти функцію для порівняння двох каталогів, якщо відомо, що каталоги регістро незалежні і можуть закінчуватися символом \. Цей символ можу бути відсутнім.

Приклад 2. Рядки тексту упорядкувати в порядку зростання, якщо відомо, що символи цих рядків задані в ASCII або в Unicode

Приклад 3. Рядки тексту упорядкувати в порядку зростання, якщо тип кодування невідомий.

2 БІБЛІОТЕКИ

2.1 Мета заняття

Вивчити прийоми та методи створення та використання бібліотек.

2.2 Методичні вказівки до організації самостійної роботи

Сучасні операційні системи широко використовують бібліотеки. Ядро використовує статичні бібліотеки для модулів, які використовуються тільки ядром і динамічні бібліотеки для зв’язку модулів ядра і користувача та для модулів ядра. Додатки користувачів також використовують бібліотеки обох типів. Бібліотеки використовуються, щоб не копіювати модулі. які вже відлагоджені, та використовувати їх багатократно.

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 2013, 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.

2.2.3 Використання DLL

Є два режими використання DLL.

Перший режим (статичне завантаження). 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). В разі використання цього режиму необхідно крім самої DLL мати відповідний .lib файл, який містить список усіх функцій DLL, що експортуються. Цей файл необхідно додати до проекту, який використовує DLL. Далі функції DLL можна використовувати як функції. визначені в проекті.

Другий режим (Динамічне завантаження). Завантаження та вивантаження DLL виконується за допомогою функцій WinApi тоді, коли необхідно використовувати функції DLL (коли необхідність в функціях відпадає). Для використання необхідно:

– підключити файл заголовків windows.h;

– визначити типи показників на функції, що експортуються (typedef);

– перед використанням першої функції бібліотеки завантажити бібліотеку за допомогою функції LoadLibrary і перевірити успішність завантаження (в разі успіху функція повертає значення. яке відрізняється від 0;

– для усіх функцій бібліотеки, які будуть використовуватись, визначити їх адреси (функція GetProcAddress). Знайдені адреси функцій перетворити до відповідного типу. Перевірити успішність визначення адреси (адреса повинна відрізнятися від 0.

– для виклику функції використовувати її адресу.

– після використання останньої функції бібліотеки вивантажити її з пам’яті (функція FreeLibrary).

2.2.4 Контрольні питання та завдання

1 Навіщо використовуються бібліотеки?

2 Які режими використання динамічних бібліотек Ви знаєте? Порівняйте ці режими.

3 Дослідить режими компілятора, які використовуються при створенні проекту типу DLL.

2.2.5 Приклади аудиторних задач

Приклад 1. Створіть бібліотеку функцій для генерування ключів. Шифрування та розшифрування (метод RSA).

Опис методу.

Формуються 2 простих числа (p, q).

Формується додаток цих чисел, цей додаток називається модулем перетворення n = p * q.

Формується функція Ейлера для цих чисел по формулі: φ = (p - 1) * (q - 1).

Генерація ключів. Ключ для шифрування даних E – випадкове число, яке взаємно просте з φ, ключ для розшифрування даних D обчислюється з формули D * E = 1 (mod φ).

Шифрувати можна довільні дані, значення яких менше модуля n^2. Для шифрування використовується ключ E, який в криптографії називають відкритим ключем. Для шифрування повідомлення M використовується формула $M' = M^E \bmod (n)$. Отримане значення M' – це зашифроване значення.

Для розшифровки використовується ключ D, який в криптографії називають особистим ключем. В даному випадку розшифрувати повідомлення зможе тільки власник особистого ключа. Зашифрувати повідомлення для нього зможе користувач, який має його відкритий ключ. Для розшифрування використовується формула $M = M'^D \bmod (n)$.

Найпростіший опис цього методу можна знайти в [5, с. 49-65].

Визначимо склад бібліотеки.

Для математичних операцій потрібні функції:

– генерація простих чисел;
– обчислення найбільшого загального дільника;
– обчислення ступеня по модулю.

Файл заголовків для математичних функцій бібліотеки (RSAMath.h)

Для додавання цього файлу до проекту треба обрати проект (у нас це RSA), права кнопка мишки та обрати Add, з меню, що випадає, обрати AddItem. З типів Item обрати Header File та задати ім'я файлу RSAMath.h.

Вміст файлу.

```
#pragma once
unsigned Simple ();     // Просте
unsigned Nod (unsigned x, unsigned y); // Найбільший загальний дільник
unsigned PowMod (unsigned x, unsigned y, unsigned m);
```

[2] Для подолання цього обмеження повідомлення для шифрування ділиться на блоки, кожний блок задовольняє обмеженню.

Для генерації ключів будемо використовувати функцію для генерації модуля, відкритого та особистого ключів.

Для шифрування – розшифрування фактично використовується функція обчислення ступеня.

Всі функції, які потрібні для зовнішнього користувача, (функції, що експортуються) визначимо в файлі rsa.h, який додаємо до проекту як попередній файл заголовків.

Для визначення цих функцій в DLL використовується _declspec (dllexport), для визначення цих функцій в програмі, яка використовує ці функції використовується _declspec (dllimport). Створення універсального файлу заголовків для DLL та програми користувача.

Вміст файлу RSA.h

```
#pragma once
#ifdef RSA_EXPORTS
#define RSAAPI        extern "C"  _declspec (dllexport)
#else
#define RSAAPI        extern "C"  _declspec (dllimport)
#endif
// Шифрування
RSAAPI bool Crypt(unsigned OpenMsg, unsigned E, unsigned n, unsigned *CryptMsg);
// Розшифрування
RSAAPI bool DeCrypt(unsigned CryptMsg, unsigned D, unsigned n, unsigned *OpenMsg);
// Генерація ключів
RSAAPI unsigned EncryptAndDecryptKey(unsigned *E, unsigned *D);
```

Перша частина файлу заголовків визначає константу RSAAPI, яка приймає значення _declspec (dllexport) для DLL (саме в проекті DLL середовище створює константу RSA_EXPORTS) і значення _declspec (dllimport) для решти проектів.

Визначення extern "C" використовується для забезпечення можливості використання DLLяк для C, так і для C++ файлів.

При визначенні функцій, що експортуються, використовується саме RSAAPI.

Для реалізації математичних функцій додаємо до проекту файл MATH.cpp. Додавання цього файлу виконується аналогічно додаванню файлу заголовків, але в якості типу обираємо cpp файл.

Вміст файлу MATH.cpp.

```
#include "stdafx.h"
#include <stdlib.h>
#include "RSAMath.h"
// Просте в інтервалі 0..99
unsigned Simple (){
	unsigned x = rand ()%100;
	x|=1;
	while (1){
		unsigned p;
		for (unsigned i = 3; ;i +=2){
			p = i * i;
			if (p > x) break;
			if (x % i == 0)  break;
		}
		if (p >x) return x;
		x +=2;
	}
}

// Найбільший загальний дільник
unsigned gcd (unsigned x, unsigned y){
	while (x && y){
		if (x > y)x%=y;else	y%=x;
	}
	return x + y;
}
// Обчислення ступеня по модулю
unsigned PowMod (unsigned x, unsigned y, unsigned m){
	unsigned  r = 1;
	for (unsigned i = 0; i < y; i++){
		r *= x;
		r%=m;
	}
```

```
        return r;
}
```

Після додавання першого файлу типа cpp спробуйте виконати побудову проекту (з головного меню обираємо BUILD/Build Solution)/ в разі наявності синтаксичних помилок, виправте їх. Рекомендуємо виправляти не тільки помилки, а і попередження!!!

Для функцій, що експортуються,. використовуємо файл RSA.cpp, який створено середовищем VS2013;

Вміст файлу RSA.cpp (перший рядок сформовано середовищем, його залишаємо обов’язково на першому місці)

```
#include "stdafx.h"
#include <stdlib.h>
#include "RsaMath.h"
#include "rsa.h"
RSAAPI unsigned EncryptAndDecryptKey (unsigned *E, unsigned *D){
        int p, q;
        do{
                p = Simple();   q = Simple();
        } while (p == q);
        unsigned n = p * q;
        unsigned fi = (p - 1) * (q - 1);
        for (*E = 3; gcd(*E, fi) != 1; *E += 2);
        for (*D = 3; (*E) * (*D) % fi != 1; *D += 1);
        return n;
}
// Шифрування
RSAAPI   bool Crypt (unsigned OpenMsg, unsigned E, unsigned n, unsigned *CryptMsg){
        if (OpenMsg > n) return false;
        *CryptMsg = PowMod (OpenMsg, E, n);
        return true;
}
// Розшифрування
RSAAPI   bool DeCrypt (unsigned CryptMsg,  unsigned D, unsigned n, unsigned *OpenMsg){
```

```
    if (CryptMsg > n) return false;
    *OpenMsg = PowMod (CryptMsg, D, n);
    return true;
}
```

Після додавання наступного cpp файлу виконуємо побудову проекту для виправлення помилок та попереджень.

Для завдання імен функцій, що експортуються, використовується def файл. Який треба додати до проекту. Цей файл додаємо до проекту як і решта файлів, але не має можливості обрати тип файлу, тому для цього файлу треба визначити і ім'я і тип.

В цей файл після ключового слова EXPORTS записуються імена усіх функцій, що експортуються.

Вміст def файлу

```
LIBRARY
EXPORTS
EncryptAndDecryptKey
Crypt
DeCrypt
```

Цей файл використовується компонувальником для завдання списку функцій, що експортуються. Тому його треба визначити для компонувальника. Для цього обираємо наш проект, права кнопка мишки, обираємо Properties, далі Linker, далі Input і для Module Definition File вказуємо ім'я цього файлу.

Будуємо проект.

В разі успішної побудови в каталозі Debug, каталогу, де створено solution, повинні знаходитись RSA.dll та RSA.lib файли. Якщо цих файлів немає, перевірте усі кроки створення solution.

Приклад 2.

Створити проект для використання бібліотеки RSA.dll в статичному режимі завантаження

1. Створюємо консольний проект (StaticRSA)
2. Додаємо до проекту посилання на бібліотеку (RSA.lib). Для цього в головному меню обираємо Project, далі Reference, add New Reference та обираємо потрібну бібліотеку. Якщо усі дії виконуються в межах одного solution, середовище обере бібліотеку сама, залишиться лише обрати ОК.

Вміст файлу StaticRSA

```
#include <stdio.h>
#include <stdio.h>
#include <stdlib.h>
#include "..\rsa\rsa.h"
int _tmain (){
unsigned E, D;
unsigned n = EncryptAndDecryptKey (&E, &D);
unsigned OpenMsg, CryptMsg, AfterDecryptMsg;
int i;
for (i = 0; i < 10; i++) {
        OpenMsg = rand () % n;
        Crypt (OpenMsg, E, n, &CryptMsg);
        DeCrypt (CryptMsg,  D, n, &AfterDecryptMsg);
        if (OpenMsg == AfterDecryptMsg)
                printf ("OpenMsg = %u CryptMsg = %u AfterDecryptMsg = %u - OK\n",
OpenMsg, CryptMsg, AfterDecryptMsg);
        else
                printf ("OpenMsg = %u CryptMsg = %u AfterDecryptMsg = %u -
Error\n", OpenMsg, CryptMsg, AfterDecryptMsg);
return 0;
}
```

Для визначення останньої програми, як програми, з якої треба починати виконання, обираємо цей проект, права кнопка мишки, та Set AsStartUpProject

Результати роботи програми:

OpenMsg = 840 CryptMsg = 1976 AfterDecryptMsg = 840 - OK

OpenMsg = 1777 CryptMsg = 741 AfterDecryptMsg = 1777 - OK

OpenMsg = 2687 CryptMsg = 1118 AfterDecryptMsg = 2687 - OK

OpenMsg = 1989 CryptMsg = 2042 AfterDecryptMsg = 1989 - OK

OpenMsg = 490 CryptMsg = 2045 AfterDecryptMsg = 490 - OK

OpenMsg = 1888 CryptMsg = 1481 AfterDecryptMsg = 1888 - OK

OpenMsg = 2239 CryptMsg = 1589 AfterDecryptMsg = 2239 - OK

OpenMsg = 2488 CryptMsg = 1983 AfterDecryptMsg = 2488 - OK

OpenMsg = 211 CryptMsg = 1054 AfterDecryptMsg = 211 - OK

OpenMsg = 675 CryptMsg = 2080 AfterDecryptMsg = 675 - OK

Як видно з результатів, значення до шифрування (OpenMsg) співпадає зі значенням після розшифрування (AfterDecryptMsg) для усіх даних, тому робимо висновок про правильну роботу функцій.

Приклад 3. Перевірити роботу бібліотеки при динамічному режимі використання.

Додаємо новий консольний проект DynamicRSA.

Головна програма проекту

```
#include "stdafx.h"
#include <windows.h>
// Визначення типів для адрес функцій
typedef bool (*TCRYPT)(unsigned, unsigned, unsigned, unsigned *);
typedef bool(*TDECRYPT)(unsigned, unsigned, unsigned, unsigned *);
typedef unsigned (*TENCRYPTANDDECRYPTKEY)(unsigned *, unsigned *);
int _tmain(int argc, _TCHAR* argv[]){
// Завантаження бібліотеки
HMODULE h = LoadLibrary(_T("RSA.dll"));
if (h != 0){
// Визначення адрес для функцій, що експортуються
        TCRYPT Crypt = (TCRYPT)GetProcAddress(h, "Crypt");
        TDECRYPT DeCrypt = (TDECRYPT)GetProcAddress(h, "DeCrypt");
        TENCRYPTANDDECRYPTKEY EncryptAndDecryptKey =
(TENCRYPTANDDECRYPTKEY)GetProcAddress(h, "EncryptAndDecryptKey");
        if (Crypt && DeCrypt && EncryptAndDecryptKey){
        // Використання функцій
                unsigned E, D;
                unsigned n = EncryptAndDecryptKey(&E, &D);
                unsigned OpenMsg, CryptMsg, AfterDecryptMsg;
                int i;
                for (i = 0; i < 10; i++) {
                        OpenMsg = rand() % n;
                        Crypt(OpenMsg, E, n, &CryptMsg);
                        DeCrypt(CryptMsg, D, n, &AfterDecryptMsg);
                        if (OpenMsg == AfterDecryptMsg)
```

```
                printf("OpenMsg=%u CryptMsg=%u AfterDecryptMsg=%u- OK\n",
OpenMsg, CryptMsg, AfterDecryptMsg);
                else
                printf("OpenMsg = %u CryptMsg = %u AfterDecryptMsg = %u - Error\n",
OpenMsg, CryptMsg, AfterDecryptMsg);
            }
        }
        }
        return 0;
        }
```

2.2.6 Приклади домашних задач

Приклад 1. Скласти функції для виконання арифметичних операцій +, -, *, / для 64-бітних чисел з урахування переповнень.

Приклад 2. Створити динамічну бібліотеку з цими функціями.

Приклад 3. Створити проект для перевірки бібліотеки в режимі статичного завантаження.

Приклад 4. Створити проект для перевірки бібліотеки в режимі динамічного завантаження.

3 . ВВЕДЕННЯ – ВИВЕДЕННЯ ДАНИХ. РОБОТА З ФАЙЛАМИ

3.1 Мета заняття

Придбати практичні навики роботи з файлами за допомогою функцій WinApi

3.2 Методичні вказівки по організації самостійної роботи

В курсах програмування та об’єктна-орієнтованого програмування вивчаються функції для роботи з файлами, але це тільки обгортки для функцій операційної системи. Як за правило, обгортки створюються для спрощення роботи з функціями, але одночасно вони не враховують усі можливості, які дають функції операційної системи. Ось чому вивчення функцій ОС важливо.

Вивчить розділ 14. Робота з файлами по [2, с. 148-164]. Зверніть увагу на основні функції: CreateFile, ReadFile, WriteFile, CloseHandle, а також на функції для роботи з

показником файлу. Після вивчення функцій виконайте самостійно завдання з [2]. Далі розгляньте питання, пов’язані з введенням – виведенням по [6, с. 10-24]

3.3 Контрольні питання та завдання для самостійної роботи

Чим відрізняються функції введення – виведення для консолі від відповідних функцій для файлів?

Яку відповідь повертає функція CreateFile в разі помилки, в разі успішного запиту?

Що буде, якщо програміст забув закрити файл, якщо це трапилось в функції, яка використовується багато разів?

Чому дорівнює старша та молодша частини зміщення, якщо необхідно зміститися в файлі на величину a ($a < 2^{32}$) відносно початку, відносно кінця файлу?

Який шаблон необхідно задати для пошуку файлів, ім’я яких починається з літери a, тип файлу з літери b, для завдання типу завжди використовується 3 символи?

Як узнати, що функція пошуку знайшла каталог, а не файл??

3.4 Приклади аудиторних задач

Скласти програму для моделювання роботи поштової скриньки. Поштова скринька – це файл зі структурою: загальна кількість листів, загальний обсяг усіх листів (в байтах), і далі листи. Кожний лист має структуру: довжина листа (байтів), лист.

Програма повинна мати наступні функції:

Створення поштової скриньки, додавання листа, читання листа, видалення листа, визначення кількості листів, знищення поштової скриньки

Вимоги до програми:

програма повинна працювати коректно для обох способів кодування: ASCII и UNICODE.

Файл заголовків для об’єкта MAILBOX.h.

```
#pragma once
#include <windows.h>
#include <tchar.h>
#include <stdlib.h>
#include <windows.h>
#include <string.h>
enum{
```

```
MAILBOX_OK,
MAILBOX_CREATE,
MAILBOX_FILE,
MAILBOX_NUMBER,
MAILBOX_CS
};

typedef struct _MAILBOX_TITLE
{
	TCHAR fName[MAX_PATH];
	DWORD MaxSize, MessageCounts;
}MAILBOX_TITLE, *PMAILBOX_TITLE;
class MAILBOX{
	MAILBOX_TITLE Title;
	DWORD dwError;
	static int count;
	DWORD dwCurSize;
	DWORD dwCS; // Контрольна сума
public:
	MAILBOX(LPCTSTR  fName, size_t MaxSize = 0);
	MAILBOX& operator+=(LPCTSTR  Msg);  // Додати лист
	MAILBOX& operator-=(DWORD  Number); // Видалити лист за номером
	DWORD operator()(){                                    // Кількість листів
		return Title.MessageCounts;
	}
	// Кількість листів
	DWORD ReadCounts(){
		return Title.MessageCounts;
	}
	// Читання листа з заданим номером
	DWORD RdMsg(TCHAR *res, DWORD i);
	// Знищити усі листи
	void Clear();
	DWORD GetLastError(){ return dwError; }
```

```
        ~MAILBOX(){ count--; }
};
// Контрольна сума
DWORD GetControlSum(PBYTE pMem, DWORD dwCount);
```

Визначення функцій класу (Файл mailBox.cpp)

```
// MailBox.cpp : Defines the entry point for the console application.
//#include "stdafx.h"
#include <stdio.h>
#include "MailBox.h"
#include "mailBox.h"
DWORD GetControlSumForFile(LPCTSTR fName){
        HANDLE h = CreateFile(fName, GENERIC_READ | GENERIC_WRITE,
        FILE_SHARE_READ, 0, OPEN_EXISTING, 0, 0);
        if (h == INVALID_HANDLE_VALUE) return MAILBOX_FILE;
        DWORD dwSize = GetFileSize(h, 0);
        PBYTE pMem = new BYTE[dwSize];
        DWORD dwCount, dwCS;
        BOOL b = ReadFile(h, pMem, dwSize, &dwCount, 0);
        if (b)  {
                dwCS = GetControlSum(pMem, dwSize);
                b = WriteFile(h, &dwCS, sizeof(dwCS), &dwCount, 0);
        }
        delete[]pMem;
        CloseHandle(h);
        return b ? MAILBOX_OK : MAILBOX_FILE;
}
DWORD GetControlSum(PBYTE pMem, DWORD dwCount){
        DWORD dwCS = 0;
        DWORD dwFull = dwCount / 4;
        PDWORD p32Mem = (PDWORD)pMem;
        for (DWORD i = 0; i < dwFull; ++i)
                dwCS += p32Mem[i];
```

```
        DWORD dwNotFull = dwCount % 4;
        if (dwNotFull) {
                pMem += dwFull * 4;
                DWORD dwNumber = 0;
                memcpy(&dwNumber, pMem, dwNotFull);
                dwCS += dwNumber;
        }
        return dwCS;
}
int MAILBOX::count = 0;
MAILBOX::MAILBOX(LPCTSTR  fName, size_t MaxSize){
        BOOL b;
        DWORD dwCount;
        _tcscpy_s(Title.fName, fName);
        HANDLE h = CreateFile(fName, GENERIC_READ, FILE_SHARE_READ, 0,
        OPEN_EXISTING, 0, 0);
        if (h == INVALID_HANDLE_VALUE){
                if (MaxSize){
                        Title.MaxSize = MaxSize;
                        Title.MessageCounts = 0;
                        h = CreateFile(fName, GENERIC_WRITE,
                FILE_SHARE_READ, 0, CREATE_NEW, 0, 0);
                if (h == INVALID_HANDLE_VALUE)      {
                        dwError = MAILBOX_FILE;
                        return;
                }
                dwCS = GetControlSum((PBYTE)&Title, sizeof(Title));
                b = WriteFile(h, &Title, sizeof(Title), &dwCount, 0);
                if (b) b = WriteFile(h, &dwCS, sizeof(dwCS), &dwCount, 0);
                CloseHandle(h);
                dwCurSize = sizeof(Title) + sizeof(dwCS);
                if (!b)   {
                        dwError = MAILBOX_FILE;
                        return;
```

```
                }
                dwError = MAILBOX_OK;
        }
        else{
                dwError = MAILBOX_CREATE;
                return;
        }
}
else{
        dwCurSize = GetFileSize(h, 0);
        PBYTE pMem = new BYTE[dwCurSize];
        b = ReadFile(h, pMem, dwCurSize, &dwCount, 0);
        if (b){
                dwCS = *(DWORD*)(pMem + dwCurSize - 4);
                DWORD CalcCS = GetControlSum(pMem, dwCurSize - 4);
                dwError = CalcCS == dwCS ? MAILBOX_OK : MAILBOX_CS;
                PMAILBOX_TITLE pTitle = (PMAILBOX_TITLE)pMem;
                Title = *pTitle;
        }
        else{
                dwError = MAILBOX_FILE;
        }
        delete[]pMem;
        CloseHandle(h);
}
}
MAILBOX& MAILBOX::operator+=(LPCTSTR  Msg){
        DWORD dwLen = _tcslen(Msg) * sizeof(DWORD);
        dwError = MAILBOX_CREATE;
        if (Title.MaxSize >= dwCurSize + dwLen + sizeof(DWORD)){
                Title.MessageCounts += 1;
                dwError = MAILBOX_FILE;
                HANDLE h = CreateFile(Title.fName, GENERIC_READ |
GENERIC_WRITE, FILE_SHARE_READ, 0, OPEN_EXISTING, 0, 0);
```

```
            BOOL b;
            DWORD dwCount;
            if (h != INVALID_HANDLE_VALUE)      {
                b = WriteFile(h, &Title, sizeof(Title), &dwCount, 0);
                if (b)   {
                    LONG dwHigeOffs = -1;
                    SetFilePointer(h, -4, &dwHigeOffs, FILE_END);
                    b = WriteFile(h, &dwLen, sizeof(dwLen), &dwCount, 0);
                    if (b) b = WriteFile(h, Msg, dwLen, &dwCount, 0);
                    CloseHandle(h); h = INVALID_HANDLE_VALUE;
                    if (b)   {
                        dwError = GetControlSumForFile(Title.fName);
                    }
                }
                if (h != INVALID_HANDLE_VALUE)      CloseHandle(h);
            }
        }
        return *this;
        }
        DWORD MAILBOX::RdMsg(TCHAR *msg, DWORD i){
            BOOL b = TRUE;
            DWORD dwLen, dwCount;
            dwError = MAILBOX_NUMBER;
            if (i < Title.MessageCounts)  {
                dwError = MAILBOX_FILE;
                HANDLE h = CreateFile(Title.fName, GENERIC_READ,
FILE_SHARE_READ, 0, OPEN_EXISTING, 0, 0);
            if (h != INVALID_HANDLE_VALUE){
                SetFilePointer(h, sizeof(Title), 0, FILE_BEGIN);
                for (DWORD j = 0; j < i; ++j){
                    b = ReadFile(h, &dwLen, sizeof(dwLen), &dwCount, 0);
                    if (!b) break;
                    SetFilePointer(h, dwLen, 0, FILE_CURRENT);
                }
```

```
                    if (b) b = ReadFile(h, &dwLen, sizeof(dwLen), &dwCount, 0);
                    if (b && msg)
                    {
                        b = ReadFile(h, msg, dwLen, &dwCount, 0);
                        msg[dwLen / sizeof(TCHAR)] = 0;
                    }
                    dwError = b ? MAILBOX_OK : MAILBOX_FILE;
                    CloseHandle(h);
                }

        }
        return dwLen;
        }

        MAILBOX& MAILBOX::operator -=(DWORD i){
        dwError = MAILBOX_NUMBER;
        if (i < Title.MessageCounts)
        {
                Title.MessageCounts -= 1;
                dwError = MAILBOX_FILE;
                HANDLE h = CreateFile(Title.fName, GENERIC_READ | GENERIC_WRITE,
FILE_SHARE_READ, 0, OPEN_EXISTING, 0, 0);
                dwCurSize = GetFileSize(h, 0);
                BOOL b;
                DWORD dwCount, dwLen = 0;
                if (h != INVALID_HANDLE_VALUE)
                {
                    b = WriteFile(h, &Title, sizeof(Title), &dwCount, 0);
                    DWORD dwOld = sizeof(Title), dwNew, dwCnt;
                    if (b)
                    {
                        for (DWORD j = 0; j < i; ++j)
                        {
                            b = ReadFile(h, &dwLen, sizeof(dwLen), &dwCount, 0);
```

```
                        if (!b) break;
                        dwOld = SetFilePointer(h, dwLen, 0, FILE_CURRENT);

                    }
                    b = ReadFile(h, &dwLen, sizeof(dwLen), &dwCount, 0);
                    dwNew = SetFilePointer(h, dwLen, 0, FILE_CURRENT);
                    dwCnt = dwCurSize - dwNew - 4;
                    PBYTE pMem = new BYTE[dwCnt];
                    SetFilePointer(h, dwNew, 0, FILE_BEGIN);
                    b = ReadFile(h, pMem, dwCnt, &dwCount, 0);
                    if (b)
                    {
                        SetFilePointer(h, dwOld, 0, FILE_BEGIN);
                        b = WriteFile(h, pMem, dwCnt, &dwCount, 0);
                        SetEndOfFile(h);
                    }
                    delete[]pMem;
                    dwError = b ? MAILBOX_OK : MAILBOX_FILE;
                }
                if (h != INVALID_HANDLE_VALUE)
                    CloseHandle(h);
                if (b) dwError = GetControlSumForFile(Title.fName);
            }
        }
        return *this;
    }
```

Головна програма (файл main.cpp)

```
#include "stdafx.h"
#include <windows.h>
#include <stdio.h>
#include "MailBox.h"
#include <tchar.h>
#include <iostream>
```

```
using namespace std;

//int MAILBOX::count = 0;

int _tmain()
{
MAILBOX MailBox(_T("mb.txt"), 1000);
MailBox += _T("First");
MailBox += _T("Second");
MailBox += _T("Third");

TCHAR Msg[80];

for (int i = 0; i < MailBox.ReadCounts(); ++i)
{
        MailBox.RdMsg(Msg, i);

        _tprintf(_T("%s\n"), Msg);
}

_tprintf(_T("\n"));
_tprintf(_T("\n"));


MailBox -= 2;

for (int i = 0; i < MailBox.ReadCounts(); ++i)
{
        MailBox.RdMsg(Msg, i);

        _tprintf(_T("%s\n"), Msg);
}

}
```

3.5 Приклади домашніх задач

Задано файл з DLL. Порахувати контрольну суму цього файлу і записати її в кінець DLL. Для підрахунку контрольної суми використовувати алгоритм, як для поштової скриньки

Скласти функцію для перевірки контрольної суми при завантаженні DLL.

Вставити перевірку контрольної суми в головну функцію DLL.

Забезпечити автоматичне формування контрольної суми після змінення DLL.

4 АЛГОРИТМИ ПЛАНУВАННЯ

4.1 Мета заняття

Метою заняття є вивчення різних алгоритмів планування процесів та оцінка їх ефективності

4.2 Методичні вказівки по організації самостійної роботи

При підготовці до практичного заняття вивчите тему лекції Планування та диспетчеризація процесів. Зверніть увагу на критерії оцінки алгоритмів планування, а саме:

1. Максимальне завантаження процесорів виконаним доцільної роботи – для досягнення необхідно зменшити кількість переключень між процесами

2. Мінімальний час виконання процесів – для досягнення необхідно, щоб середній час виконання програм був мінімальним.

3. Мінімальний час відгуку на запуск програми - для досягнення необхідно, щоб середній час між створенням процесу і початком його виконання був мінімальним.

4. Мінімальний час відгуку на введення виведення даних - для досягнення необхідно, щоб середній час між розблокуванням програми і часом отримання кванту часу був мінімальним.

5 Справедливий – використовується для систем з багатьма користувачами для досягнення необхідно, щоб час, виділений для кожного користувача, був приблизно однаковим незалежно від кількості процесів, створених кожним користувачем.

Розгляньте алгоритми планування, а саме:

1. Алгоритм FCFS – перший прийшов, перший обслуговується використовується для систем без витіснення.

2. Алгоритм RR - перший прийшов, перший обслуговується використовується для систем з витісненням.

3. Алгоритм SJF – першим виконується саму коротку дільницю процесу.

4 Алгоритми з урахуванням пріоритетів.

4.3 Контрольні питання та завдання

1 Для чого використовуються алгоритми планування?

2 Для чого використовуються алгоритми диспетчеризації?

3 В який алгоритм переходить алгоритм RR, якщо квант часу збільшити до нескінченості?

4 Що таке статичний пріоритет?

5 Що таке динамічний пріоритет?

4.4 Приклади аудиторних задач

1 Скласти функції для додавання процесу в чергу та визначення процесу для виконання.

2. Задані 4 процеси P0, P1, P2, P3.

Інформація про процеси визначена в таблиці.

Процес	Час початку	Тривалість
P0	0	10
P1	1	8
P2	2	6
P3	3	4

Порівняти алгоритми FCFS, RR, SJF по першим 3 критеріям.

Алгоритм FCFS

0 P0 (10) 0..10 (10)

10 P1 (8) 1..18 (17)

18 P2 (6) 2..24 (22)

24 P3 (4) 3..28 (25)

Кількість переключень між процесами 4

Середній час виконання (10 + (18 – 1) + (24 – 2) + (28 – 3))/4 = 18,5

Середній час чекання запуску (0 + (10 - 1) + (18 - 2) + (24 - 3)) / 4 = 11,5

`Quant`

Алгоритм RR (квант часу = 2)

0 P0 (2) 8

2 P1 (2) 6

4 P2 (2) 4

6 P3 (2) 2

8 P0 (2) 6

10 P1 (2) 4

12 P2 (2) 2

14 P3 (2) 0

16 P0 (2) 4

18 P1 (2) 2

20 P2 (2) 0

22 P0 (2) 2

24 P1 (2) 0

26 P0 (2) 0

Кількість переключень між процесами **14**

Середній час виконання ((28 - 0) + (24 – 1) + (20 – 2) + (14 – 3))/4 = 20

Середній час чекання запуску (0 + (2 - 1) + (4 – 2) + (6 – 3))/4 = 1,5

Алгоритм **SJF**

0	P0 (2)	P0(8)
2	P0(8), P1(8). P2 (6)	P2 (4)
4	P0(8), P1(8). P2 (4), P3 (4)	P2 (2)
6	P0(8), P1(8). P2 (2), P3 (4)	P2 (0)
8	P0(8), P1(8). **P2 (0)**, P3 (4)	P3 (2)
10	P0(8), P1(8). P3 (2)	P3 (0)
12	P0(8), P1(8). **P3 (0)**	P0 (6)

14	P0(6), P1(8).	P0 (4)
16	P0(4), P1(8).	P0 (2)
18	P0(2), P1(8).	P0 (0)
20	**P0(0)**, P1(8).	P1 (6)
22	P1 (6)	
28	**P1 (0)**	

Кількість переключень між процесами 13

Середній час виконання ((20 - 0) + (28 – 1) + (8 – 2) + (12 – 3))/4 = 15,5

Середній час чекання запуску (0 + (2 – 1) + (2 - 2) + (4 - 3)/4 = 0,5

Висновки. По 1 критерію найкращий – алгоритм FCFS

По 2 і 3 критеріям – алгоритм **SJF. не достаток SJF – треба знати заздалегідь час виконання**

Приклад 3. Для процесів, заданих в прикладі 2, визначені статистичні пріоритети, рівні 3, 2, 1, 0 відповідно (0 пріоритет найвищій)

Визначити, як зміняться характеристики процесів

Витіснення виконується, якщо збіг квант часу, або з’явився процес, який має пріоритет вище, ніж у процесу, який виконується.

0	P0 (10)	P0 (9)
1	P1(8), P0(9),.	P1 (7)
2	P2 (6), P1(7), P0(9).	P2 (5)
3	P3 (4), P2 (5), P1(7), P0(9)	P3 (2)
5	P3 (2), P2 (5), P1(7), P0(9)	P3 (0)
7	**P3 (0)**, P2 (5), P1(7), P0(9)	P2 (3)
9	P2 (3), P1(7), P0(9)	P2 (1)
11	P2 (1), P1(7), P0(9)	P2 (1)
12	**P2 (0)**, P1(7), P0(9)	P1 (5)
14	P1(5), P0(9)	P1 (3)
16	P1(3), P0(9)	P1 (1)
18	P1(1), P0(9)	P1 (0)
19	**P1(0)**, P0(9)	P0 (7)
21	P0(7)	
28	**P0(0)**	

Кількість переключень між процесами 15

Середній час виконання ((28 - 0) + (19 – 1) + (12 – 2) + (7 – 3))/4 = 15

Середній час чекання запуску (0 + (1 – 1) + (2 - 2) + (3- 3)/4 = 0

Якщо останніми запускаються програми з найвищим пріоритетом, то час чекання запуску = 0!

Приклад 4. Скласти функцію для визначення непустої черги з найвищим пріоритетом, якщо для кожної черги відводиться один біт 32 бітного слова. Найвищий пріоритет відповідає біту 31 (старшому біту)

4.5 Приклади домашніх завдань домашних задач

5 КЕРУВАННЯ ПА’МЯТТЮ

5.1 Мета заняття

Вивчення алгоритмів, прийомів та методів керування пам’яттю в програмах користувача за допомогою засобів операційних систем

5.2 Методичні вказівки до організації самостійної роботи

Вивчить загальні принципи керування пам’яттю [0, с. 313-342].

Зверніть увагу на наступні положення. Є 2 типа пам’яті, якою можна керувати в режимі користувача – віртуальна та фізична. Ви повинні розуміти різницю між цими класами пам’яті.!

Є 3 способи керування пам’яттю.

Перший спосіб [0, с. 368-398] дозволяю напряму виділяти віртуальну та фізичну пам'ять. Функції для реалізації цього способу мають ім’я виду Virtual…. Ці функції виділяють безперервну область пам’яті (регіон), розмір якого завжди ділиться на фіксоване значення. Для сучасних версій ОС WINDOWS це значення 4К. Адреса початку регіону завжди ділиться також на фіксоване значення, для сучасних версій ОС WINDOWS це значення 64К, ось чому виділяти віртуальну пам'ять для невеликих масивів не має сенсу. Крім того, ці функції потребують зміни в таблиці сторінок, а тому можуть бути виконані тільки в режимі ядра, що потребує витрат часу процесору на переключення.

Для роботи з великими масивами, розмір яких може перевищувати розмір оперативної пам’яті, цей метод ефективний. В цю групу функцій входять також функції для дослідження стану пам’яті процесу [0, с. 342-368]. Нижче наведено короткий опис цієї групи функцій.

Для отримання інформації про пам'ять необхідно послідовно виконати наступні кроки:

Крок 1. Для отримання загально системної інформації використовується функція:

VOID GetSystemInfo(LPSYSTEM_INFO lpSystemInfo).

Структура типу SYSTEM_INFO містить основну інформацію про операційну систему, в тому числі про пам'ять.

Поля структури, необхідні для дослідження пам’яті:

DWORD dwPageSize – розмір сторінки

LPVOID lpMinimumApplicationAddress- мінімальна адреса додатку;

LPVOID lpMaximumApplicationAddress-максимальна адреса додатку;

DWORD dwAllocationGranularity - гранулярність, яка визначає границю вирівнювання пам’яті, яка виділяється функцією VirtualAlloc.

Крок 2. Для отримання інформації, яка залежить від типу пам’яті, використовується функція GlobalMemoryStatusEx:GlobalMemoryStatusEx (&MemoryStatus), яка повертає результат в структурі MEMORYSTATUSEX.

Основні поля структури MEMORYSTATUSEX:

dwLength[3]- розмір структури в байтах;

ullAvailPhys[4] – розмір доступної фізичної пам'яті;

ullAvailVirtual – розмір доступної віртуальної пам'яті;

ullAvailPageFile– розмір доступної пам’яті в файлі підкачки;

ullTotalPhys – загальний розмір фізичної пам'яті;;

ullTotalVirtual – загальний розмір віртуальної пам'яті;

ullTotalPageFile– загальний розмір пам’яті в файлі підкачки;.

Крок 3. Дослідимо адресний простір заданого процесу (функція VirtualQueryEx).

---

[3] Треба задати до використання структури

[4] Усі поля структури MEMORYSTATUSEX, які визначають розмір, мають тип беззнакових 64-бітних чисел, про що вказує префікс `ull, цей тип позначається як` ULONGLONG

DWORD VirtualQueryEx(HANDLE hProcess, LPCVOID lpAddress, PMEMORY_BASIC_INFORMATION lpBuffer, SIZE_T dwLength),

де:

hProcess - дескриптор процесу, для якого досліджується пам'ять.;

lpAddress – адреса початку пам'яті, яка досліджується;

lpBuffer – адреса початку пам'яті для буфера з результатами;

dwLength – розмір буфера.

Інформація про пам'ять записується в структуру типу MEMORY_BASIC_INFORMATION, яка містить:

AllocationBase – адреса початку регіону;

AllocationProtect- режим доступу, заданий при виділенні регіону (PAGE_READONLY, , …);

RegionSize – розмір регіону;

State - MEM_COMMIT, MEM_FREE, MEM_RESERVE;

Спосіб 2 [0, с. 409-461] дозволяє відобразити дискову пам'ять на віртуальну. Цей спосіб використовується:

для завантаження файлів для виконання;;

для роботи з файлами, як з масивами даних при необхідності їх обробки в довільному порядку;

для виділення загальної пам’яті для декількох процесів.

3 спосіб [0, с. 461-475] передбачає виділення процесу пам’яті для використання її під час динамічного виділення (купа - HEAP). Цей спосіб доцільний для виділення невеликих фрагментів пам’яті з подальшим її визволенням. Використання своєї купи дозволяє локалізувати помилки, пов’язані з її застосуванням, прискорити операції з пам’яттю.

5.3 Контрольні питання та завдання для самостійної роботи

Чим відрізняється віртуальна пам'ять от фізичної?

Чи можна виділити фізичну пам'ять за допомогою функції VirtualAlloc?

Яка пам'ять виділяється при використанні відображення файлу на пам'ять. Поясніть атрибут WRITECOPY.

Чи можна змінити розмір файлу після його відображення в пам'ять?

В яких випадках рекомендується використовувати замість стандартної нестандартну купу?

5.4 Приклади аудиторних задач

Приклад 1. Скласти програму для визначення стану віртуальної та фізичної пам’яті.

```
#include "stdafx.h"
#include <windows.h>
#include <stdio.h>
int _tmain(){
// Системна інформація
        SYSTEM_INFO SystemInfo;
        GetSystemInfo (&SystemInfo);
        printf ("dwPageSize = %d\n", SystemInfo.dwPageSize);
        printf ("dwAllocationGranularity = %d\n", SystemInfo.dwAllocationGranularity);
        printf ("lpMinimumApplicationAddress = %#x\n",
        SystemInfo.lpMinimumApplicationAddress);
        printf ("lpMaximumApplicationAddress = %#x\n",
        SystemInfo.lpMaximumApplicationAddress);
//  Стан пам’яті
        MEMORYSTATUSEX MemoryStatus;
        MemoryStatus.dwLength = sizeof (MEMORYSTATUSEX);
        GlobalMemoryStatusEx (&MemoryStatus);
        printf ("ullAvailPhys = %I64x\n", MemoryStatus.ullAvailPhys);
        printf ("ullAvailVirtual = %I64x\n", MemoryStatus.ullAvailVirtual);
        printf ("ullAvailPageFile = %I64x\n", MemoryStatus.ullAvailPageFile);
        printf ("ullullTotalPhys = %I64x\n", MemoryStatus.ullTotalPhys);
        printf ("ullTotalVirtual = %I64x\n", MemoryStatus.ullTotalVirtual);
        printf ("ullTotalPageFile = %I64x\n", MemoryStatus.ullTotalPageFile);
// Базова  інформація про пам'ять процесору
        MEMORY_BASIC_INFORMATION Buffer;
        HANDLE hProcess = GetCurrentProcess();
        DWORD dwAddress = (DWORD)SystemInfo.lpMinimumApplicationAddress;
while (dwAddress < (DWORD)SystemInfo.lpMaximumApplicationAddress)
```

```
{
        DWORD dwSize = VirtualQueryEx(  hProcess, (const void *)dwAddress,
        &Buffer, sizeof (MEMORY_BASIC_INFORMATION));
        printf ("BaseAddress = %#x\n", dwAddress);
        printf ("RegionSize = %u\n", Buffer.RegionSize);
        switch(Buffer.State) {
                case MEM_COMMIT:printf ("MEM_COMMIT\n");break;
                case  MEM_FREE: printf ("MEM_FREE\n");break;
                case  MEM_RESERVE:printf ("MEM_RESERVE\n");break;
                default:printf ("UNKNOWN\n");
        }
        if (Buffer.State != MEM_FREE){
                switch (Buffer.AllocationProtect){
                case PAGE_READONLY: printf ("PAGE_READONLY\n");break;
                case PAGE_READWRITE: printf ("PAGE_READWRITE\n");break;
                case PAGE_EXECUTE: printf ("PAGE_READWRITE\n");break;
                case PAGE_EXECUTE_READ:
                printf ("PAGE_EXECUTE_READ\n");break;
                case PAGE_EXECUTE_READWRITE:
                printf ("PAGE_EXECUTE_READWRITE\n");break;
                case PAGE_EXECUTE_WRITECOPY:
                 printf ("PAGE_EXECUTE_WRITECOPY\n");break;
                case PAGE_NOACCESS: printf ("PAGE_NOACCESS\n");break;
                case PAGE_WRITECOPY: printf ("PAGE_WRITECOPY\n"); break;
                default: printf ("UNKNOWN\n");
        }
}
PRINTF ("\N\N");
dwAddress+=Buffer.RegionSize;
}
return 0;
}
```

Приклад 2. Створити список регіонів, а також списки вільних та зайнятих блоків пам’яті з використанням функції VirtualAlloc.

Структура елемента списку

```
typedef struct {
        PVOID pAddr;
        DWORD dwSize;
        DWORD dwState; // MEM_COMMIT, MEM_RESERVE. MEM_FREE
} REGION, *PREGION;
typedef list<REGION> REGIONLIST, *PREGIONLIST;
#define Align(size, al) ((size + al - 1)/al*al)
void CreateRegionsList (PREGIONLIST pList){
        SYSTEM_INFO si;
        GetSystemInfo (&si);
        DWORD dwStartAddress = (DWORD)si.lpMinimumApplicationAddress;
        DWORD dwEndAddress = (DWORD)si.lpMaximumApplicationAddress;
        DWORD dwAlign = si.dwAllocationGranularity;
        MEMORY_BASIC_INFORMATION mbi;
        HANDLE h = GetCurrentProcess();
        for (DWORD i = dwStartAddress; i < dwEndAddress; )      {
                REGION Item;
                VirtualQueryEx (h, (LPVOID)i, &mbi, sizeof (mbi));
                Item.pAddr = mbi.BaseAddress;
                Item.dwSize = Align (mbi.RegionSize, dwAlign);
                Item.dwState = mbi.State;
                pList->push_back (Item);
                i += Item.dwSize;
        }
}
// Список вільних та зайнятих блоків
void CreateRegionsList (PREGIONLIST pFreeList, PREGIONLIST pBuzyList){
        SYSTEM_INFO si;
        GetSystemInfo (&si);
        DWORD dwStartAddress = (DWORD)si.lpMinimumApplicationAddress;
        DWORD dwEndAddress = (DWORD)si.lpMaximumApplicationAddress;
```

```
        DWORD dwAlign = si.dwAllocationGranularity;
MEMORY_BASIC_INFORMATION mbi;
HANDLE h = GetCurrentProcess();
REGION Item;
for (DWORD i = dwStartAddress; i < dwEndAddress; ){
        VirtualQueryEx (h, (LPVOID)i, &mbi, sizeof (mbi));
        Item.pAddr = mbi.BaseAddress;
        Item.dwSize = Align (mbi.RegionSize, dwAlign);
        Item.dwState = mbi.State;
        if (mbi.State != MEM_FREE){
                pBuzyList->push_back (Item);
        }
        else
                pFreeList->push_back (Item);
        i += Item.dwSize;
}
}
// Виведення списку
void printRegionsList (PREGIONLIST pList){
        for (REGIONLIST::iterator i = pList->begin (); i != pList->end(); i++){
        _tprintf (_T("%p\t%x\t%s\n"),
        i->pAddr, i->dwSize, i->dwState == MEM_FREE? _T("FREE") : _T("BUZY"));
}
}
```

Приклад 3. Скласти функцію для заміщення сторінок з використанням методу LRU.

```
typedef struct{
        PVOID va, fa;
}PAGE, *PPAGE;
typedef list <PAGE>             PAGES, *PPAGES;
PVOID AddPage (PVOID fa[], size_t faCount, PPAGES Pages, PPAGE page){
BOOL bFind = FALSE;
PAGE Item;
size_t size = Pages->size();
```

```
PAGES::iterator i;
// Перевірка на наявность сторінки в памяті
for (i = Pages->begin(); i!= Pages->end(); ++i){
        Item = *i;
        if (Item.va == page->va){
                page->fa =      Item.fa;
                Pages->erase (i);
                bFind =  TRUE;
                break;
        }
}
//  Якщо сторінка не знайдена
if (!bFind){
        if (size == faCount){
        // Якщо вільних сторінок несає
                Item = *Pages->begin();
                page->fa = Item.fa;
                Pages->pop_front();
        }
        else{
        // Є вільні сторінки
                page->fa = fa [size];
        }
}
Pages->push_back (*page);
return page->fa;
}
```

Приклад 4 Реалізувати алгоритм LRU для 4- направленого кешу

```
int GetCacheNumber (int p[4], int n)
static int flags;
int RetValue = -1;
        for (int i = 0; i < 4; ++i){
        if (p [i] == n)   {
                if (i < 2){
```

```
                    flags |=1;
                    if (i & 1) flags &= ~2;else flags |= 2;
                }
                else{
                    flags &= ~1;
                    if (i & 1) flags &= ~4;else flags |= 4;
                }
                RetValue = i;
            }
        }
        if (RetValue == -1){
            if (flags & 1)   {
                if (flags & 4) RetValue = 3; else RetValue = 2;
                flags ^=4;
            }
            else     {
                if (flags & 2)   RetValue = 1; else      RetValue = 0;
                flags ^=2;
            }
            flags ^=1;
            p [RetValue] = n;
        }
        printf ("%d\t%d %d %d\t%d\n", n, flags & 1, (flags>> 1) &1, (flags>> 2) &1, RetValue);
        return RetValue;
        }
```

5.5 Приклади домашніх задач

Створити функції для виділення та визволення пам’яті, використовуючи списки вільних та зайнятих блоків, які побудовані в прикладі 2. При виділенні пам’яті виділяти найменший достатній. При визволенні об’єднати суміжні блоки.

6 КЕРУВАННЯ ПРОЦЕСАМИ

6.1 Мета заняття

Вивчити методи для створення процесів та передачі даних між процесами

6.2 Методичні вказівки до організації самостійної роботи

Вивчить призначення процесів, структуру відповідного об’єкту ядра, а також функцію CreateProcess для створення процесів [0, с. 48-98]. При вивченні функції зверніть увагу на параметри та їх використання по замовченню. В найчастіше можна використовувати таке створення нового процесу::

```
PROCESS_INFORMATION pi;
STARTUPINFO si = { sizeof (si)};
BOOL b = CreateProcess (0, Командний рядок, 0, 0, false, 0, 0, 0, &si, &pi);
```

Якщо додаток знаходиться в тій же папці, що і програма, яка його викликає, можна задавати ім’я додатку без відповідного шляху.

Перевірка успішності завершення функції:

```
if (!b){
        printf ("Error\n"); return 1;
}
```

Якщо продовження програми можливо тільки після завершення процесу, який створено, використовується функція:

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

в якій перший параметр задає дескриптор процесу, завершення якого чекаємо. Цей дескриптор отримуємо як поле структури PROCESS_INFORMATION (поле hProcess). Другий параметр визначає час (мілісекунд), протягом якого треба чекати завершення. Якщо чекати треба стільки, скільки необхідно, то в якості другого параметру задається константа INFINITE.

Для визначення причини завершення функції чекання, аналізується код повернення функції:

WAIT_OBJECT_0, процес завершено;

WAIT_TIMEOUT – час чекання сплив.

Очевидно, що якщо в якості часу чекання задано INFINITE, аналіз коду повернення не має сенсу.

Якщо запущено декілька додатків і необхідно чекати їх завершення, замість функції WaitForSingleObject використовується функція WaitForMultipleObject:

```
DWORD WaitForMultipleObjects(
DWORD nCount, // кількість дескрипторів в масиві
CONST HANDLE *lpHandles, // масив дескрипторів
BOOL fWaitAll,          // чекати завершення усіх (TRUE), одного (FALSE).
DWORD dwMilliseconds // час очікування – мілісекунд або INFINITE
);
```

Функція повертає значення, за яким можна визначити потік, який завершено в разі, коли *fWaitAll* = FALSE (Індекс дескриптору = Значення - WAIT_OBJECT_0).

6.3 Приклади аудиторних задач

Приклад 1. Скласти програму, яка запускає текстовий редактор, в цій програмі створити декілька текстових файлів і завершити її. Повернути код повернення текстового редактору.

```
#include "stdafx.h"
#include <windows.h>
int _tmain(int argc, _TCHAR* argv[]){
DWORD dwRet = 0;
STARTUPINFO si = {sizeof (si)};
PROCESS_INFORMATION pi;
TCHAR cl [] = _T("notepad.exe");
BOOL b = CreateProcess (0, cl,0, 0, FALSE, 0, 0, 0, &si, &pi);
if (!b){
        _tprintf (_T("CreateProcess Error\n"));
        dwRet =  1;
}
else{
```

```
        WaitForSingleObject (pi.hProcess, INFINITE);
        GetExitCodeProcess(pi.hProcess,  &dwRet);
    }
    return dwRet;
    }
```

Приклад 2. Скласти програму, яка для файлів, створених першою програмою, визначає їх імена та розмір.

```
    #include "stdafx.h"
    #include <windows.h>
    int _tmain(int argc, _TCHAR* argv[]){
    DWORD dwRet = 1;
    if (argc == 3){
        ULONGLONG ull;
        FILETIME ft;
        _stscanf (argv[1], _T("%d"), &ft.dwLowDateTime);
        _stscanf (argv[2], _T("%d"), &ft.dwHighDateTime);
        WIN32_FIND_DATA fd;
        TCHAR tcTemplate [] = _T("*.txt");
        HANDLE h = FindFirstFile (tcTemplate, &fd);
        if (h != INVALID_HANDLE_VALUE){
            dwRet = 0;
            do{
                if (CompareFileTime (&ft, &fd.ftLastWriteTime) < 0)
                {
                    _tprintf  (_T("%s %d\n"), fd.cFileName, fd.nFileSizeLow);
                }
            }while (FindNextFile (h, &fd));
            FindClose (h);
        }
    }
    else
        _tprintf (_T("Not Parameters\n"));
```

```
return dwRet;
}
```

Зверніть увагу, що друга програма в якості параметрів приймає час, коли було викликано текстовий редактор для визначення саме тих файлів, які були створені в результаті останнього виклику редактору.

Приклад 3. Скласти програму, яка спочатку викликає програму 1, а потім програму 2. Для першої програми визначити пріоритет нижче середнього.

```
#include "stdafx.h"
#include <windows.h>
int _tmain(int argc, _TCHAR* argv[]){
        DWORD dwRet = 0;
        STARTUPINFO si = {sizeof (si)};
        PROCESS_INFORMATION pi;
        SYSTEMTIME st; GetSystemTime (&st);
        FILETIME ft;  SystemTimeToFileTime (&st, &ft);
        TCHAR cl [] = _T("Process.exe");
        BOOL b = CreateProcess (0, cl, 0, 0, FALSE,
BELOW_NORMAL_PRIORITY_CLASS, 0, _T("../Debug"), &si, &pi );
        if (!b)  {
                _tprintf (_T("Process1.exe Error\n"));dwRet = 1;
        }
        else{
                WaitForSingleObject (pi.hProcess, INFINITE);
        GetExitCodeProcess( pi.hProcess, &dwRet);
        _tprintf (_T("Process1.exe OK. RetCode = %d\n"), dwRet);
        CloseHandle (pi.hThread);
        CloseHandle (pi.hProcess);
}
if (dwRet == 0){
        TCHAR buf [MAX_PATH];
        _stprintf (buf, _T("Process2.exe %d %d"), ft.dwLowDateTime,
ft.dwHighDateTime);
        BOOL b = CreateProcess (0, buf, 0, 0, FALSE, 0, 0, _T("../Debug"), &si, &pi );
```

```
        if (!b)  {
                _tprintf (_T("Process1.exe Error\n")); dwRet = 2;
        }
        else     {
                WaitForSingleObject (pi.hProcess, INFINITE);
                GetExitCodeProcess( pi.hProcess, &dwRet);
                _tprintf (_T("Process2.exe OK. RetCode = %d\n"), dwRet);
                CloseHandle (pi.hThread);
                CloseHandle (pi.hProcess);
        }
}
return dwRet;
}
```

6.4 Приклади домашніх задач

1 Змінити програму 2 та 3 для передачі часу створення файлів через параметри середовища

2 Вивчить приклад дослідження поточних процесів як елементи Process Manager. Запустіть запропонований приклад (https://msdn.microsoft.com/en-us/library/windows/desktop/ms686701(v=vs.85).aspx). Вивчить функції, які використовуються в цьому прикладі.

7 КЕРУВАННЯ ПОТОКАМИ

7.1 Мета заняття

Мета заняття навчитись створювати та використовувати потоки в умовах застосування багато ядерних процесорів

7.2 Методичні вказівки по організації самостійної роботи

Перш за все треба розібратися, коли має сенс створювати потоки. Створення потоків має сенс, якщо в програмі є обробка великих масивів даних, яка може бути виконана паралельно, або є функції, які потребують відносно великого часу виконання і можуть виконуватись паралельно.

Для того, щоб функція виконувалась паралельно з іншими функціями, її треба перетворити в потокову функцію. Потокова функція завжди має заголовок:

```
DWORD WINAPI Ім’я функції (PVOID Par)
```

Для цієї функції треба створити потік:

```
HANDLE WINAPI CreateThread(
  _In_opt_ LPSECURITY_ATTRIBUTES lpThreadAttributes,     // 0
  _In_ SIZE_T dwStackSize,                               // 0
  _In_ LPTHREAD_START_ROUTINE lpStartAddress,            // Ім’я функції
  _In_opt_ __drv_aliasesMem LPVOID lpParameter,          //  Параметр
  _In_ DWORD dwCreationFlags,                            // 0
  _Out_opt_ LPDWORD lpThreadId                           // 0
  );
```

В якості коментарів задані значення, які використовуються найчастіше.

Для очікування завершення усіх потоків використовується функція

```
DWORD WINAPI WaitForMultipleObjects(
  _In_ DWORD nCount,                                     // Кількість потоків
  _In_reads_(nCount) CONST HANDLE *lpHandles,            // Масив їх дескрипторів
  _In_ BOOL bWaitAll,                                    // -Чекати усіх .або одного
  _In_ DWORD dwMilliseconds                              // Час чекання
  );
```

Знищення об’єкту ядра Thread

```
CloseHandle (HANDLE)
```

Для визначення кількості потоків, які найкраще створювати, використовується кількість ядер процесору:

```
 SYSTEM_INFO si;
GetSystemInfo(&si);
```

```
DWORD p = si.dwNumberOfProcessors;
```

Вивчить розділи синхронізації потоків для одного та різних процесів [4, с. 187 - 242]. Треба пам’ятати, що синхронізація необхідна, якщо один потік змінює дані або файл, які використовуються іншими потоками.

Треба розібратися з способами використання, перевагами та недоліками наступних засобів синхронізації (Interlocked функції, критичні секції, об’єкти ядра).

7.3 Приклади аудиторних задач

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.

Потокова функція.

```
volatile DWORD dwCount = 0;
DWORD WINAPI ThreadFun1(PVOID pvPar){
//dwCount++;
InterlockedIncrement (&dwCount);
char *pcPar = (char *)pvPar;
printf("Begin %s Thread\n", pcPar);
// Затримка для збільшення часу виконання потоку
for (int i = 0; i < 100000; i++);
printf("End %s Thread\n", pcPar);
return 0;
}
```

Фрагмент головної програми

```
TCHAR *ThreadNumber[] = {
_T("1"), _T("2"), _T("3"), _T("4"), _T("5"),
_T("6"), _T("7"), _T("8"), _T("9"), _T("10") };
int i;
HANDLE hThread[MAXTHREADS];
```

```
for (i = 0; i < MAXTHREADS; i++){
hThread[i] = CreateThread(0, 0, ThreadFun1, ThreadNumber[i], 0);
if (!hThread[i]){
        printf("CreateThread Error for thread number ThreadNumber\n");
        continue;
}
}
WaitForMultipleObjects(MAXTHREADS, hThread, false, INFINITE);
printf("dwCount = %d\n", dwCount);
for (i = 0; i < MAXTHREADS; i++)
        CloseHandle(hThread[i]);
```

При виконанні цього прикладу на комп'ютері змінюйте час чекання в потоковій функції і поясніть отримані результати в залежності від часу чекання

Приклад 2.

Визначити максимальну кількість потоків, яку можна створювати і чекати завершення одночасно.

1 Створюємо найпростішу потокову функцію.

```
DWORD WINAPI ThreadFun2(PVOID pvPar){
InterlockedIncrement(&dwCount);
return 0;
}
```

2 Формуємо цикл в якому поступово збільшуємо кількість потоків і визначаємо кількість викликів потокової функції. Вихід з циклу, якщо кількість викликів не співпадає з кількістю створених потоків.

Фрагмент головної програми

```
for (i = 2;; i ++){
dwCount = 0;
HANDLE *ph = new HANDLE[i];
for (int j = 0; j < i; j++)
        ph[j] = CreateThread(0, 0, ThreadFun2, 0, 0, 0);
WaitForMultipleObjects(i, ph, TRUE, INFINITE);
if (dwCount != i)       break;
```

```
printf("threads = %d\n", i);
for (int j = 0; j < i; j++)          CloseHandle(ph[i]);
delete[]ph;
}
```

Отримана інформація дуже важлива при створенні практичних багато поточних задач з використанням для цього функцій WINAPI.

Приклад 3.

Скласти функції для послідовного та паралельного обчислення скалярного добутку двох векторів: s = $\sum_{i=0}^{n-1} a_i * b_i$.

Порівняти обчислені значення. Визначити час виконання для обох варіантів.

В цьому прикладі необхідно забезпечити паралельне виконання циклу

Послідовна функція:

```
float SecSM(float *a, float *b, size_t n){
float s = 0;
for (int i = 0; i < n; i++)          s += a[i] * b[i];
return s;
}
```

Структура для паралельної функції:

```
typedef struct{
float *a, *b, res;
size_t begin, end;
}SM_DATA, *PSM_DATA;
```

Паралельна функція

```
float ParSM(float *a, float *b, size_t n){
// Визначення кількості потоків
        SYSTEM_INFO si;
GetSystemInfo(&si);
DWORD p = si.dwNumberOfProcessors;
HANDLE h[16];
SM_DATA  sm[16];
// Кількість ітерацій для одного потоку
        size_t Step = (n + p - 1) / p;
```

```
// Заповнення структур для потокових функцій
for (int i = 0; i < p; i++){
        sm[i].a = a;
        sm[i].b = b;
        sm[i].begin = i * Step;
        sm[i].end = (i + 1)* Step;
}
sm[p - 1].end = n;
// Створення потоків (з урахуванням існування одного потоку)
for (int i = 0; i < p - 1; i++)
        h[i] = CreateThread(0, 0, ThreadSM, &sm[i], 0, 0);
// Обчислення для останньої порції ітерацій
ThreadSM(&sm[p - 1]);
// Чекаємо завершення усіх потоків
WaitForMultipleObjects(p - 1, h, true, INFINITE);
// Закриваємо потоки
for (int i = 0; i < p - 1; i++) CloseHandle (h[i]);
// Обчислення глобальної суми по локальним
float s = sm[0].res;
for (int i = 1; i < p; i++)
        s += sm[i].res;
return s;
}
```

Фрагмент головної програми.

```
float s;
double dStart, dFinish;
// Послідовне обчислення
dStart = omp_get_wtime();
s = SecSM(a1, b1, n * n);
dFinish = omp_get_wtime();
printf("s = %g time = %lg\n", s, dFinish - dStart);
// Паралельне обчислення
dStart = omp_get_wtime();
s = ParSM(a1, b1, n * n);
```

```
dFinish = omp_get_wtime();
printf("s = %g time = %lg\n", s, dFinish - dStart);
```

При виконанні цих функцій задавайте різні значення для розміру масивів. Чи вдалося вам досягнути підвищення продуктивності в паралельному режимі. Не забудьте, що вимір часу необхідно виконувати в режимі Release!

7.4 Приклади домашніх завдань

Реалізувати задачу Виробник - Споживач за допомогою потоків одного додатку, потоків різних додатків (http://life-prog.ru/view_os.php?id=50)

Реалізувати задачу Письменники – Читачі за допомогою потоків одного додатку, потоків різних додатків (https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%B4%D0%B0%D1%87%D0%B0_%D0%BE_%D1%87%D0%B8%D1%82%D0%B0%D1%82%D0%B5%D0%BB%D1%8F%D1%85-%D0%BF%D0%B8%D1%81%D0%B0%D1%82%D0%B5%D0%BB%D1%8F%D1%85)

Реалізувати задачу Філософи, що обідають за допомогою потоків одного додатку, потоків різних додатків (https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B1%D0%BB%D0%B5%D0%BC%D0%B0_%D0%BE%D0%B1%D0%B5%D0%B4%D0%B0%D1%8E%D1%89%D0%B8%D1%85_%D1%84%D0%B8%D0%BB%D0%BE%D1%81%D0%BE%D1%84%D0%BE%D0%B2)

СПИСОК ЛІТЕРАТУРИ

1 Бондаренко М.Ф., Качко О.Г. Операційні системи: навч. посібник. – Х., Компанія СМІТ, 2013. – 432 с.

2 Бондаренко М.Ф., Качко Е.Г. Операционные системы: учебн. пособие. – Х., Компания СМИТ, 2006. – 443 с.

3 Рихтер Дж. Windows для профессионалов: создание эффективных Win32-приложений с учетом специфики 64-разрядной версии Windows.- СПб: Питер; М.: Издательско-торговый дом «Русская редакция», 2001.- 752 с

4 Рихтер Дж., Кларк Дж. Программирование серверних приложений Microsoft для Windows 2000. Мастер класс- СПб: Питер; М.: Издательско-торговый дом «Русская редакция», 2001.- 592 с

5 Кнут Д. Искусство программирования для ЭВМ. Т. 2. Получисловые алгоритмы. Москва. Мир, 1977. – 723 с.

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

7 Intel Architecture Software Developer’s Manual. Volume 2. Instruction Set Reference (файл 24319102.PDF).

8 Рублинецкий В.И. Введение в компьютерную криптологию. Х.:Око, 1995. – 128 с.

9 Рихтер Дж., Кларк Дж. Д. Программирование серверных приложений для Microsoft Windows 2000/ - СПб: Питер; М.: Издательско-торговый дом «Русская редакция», 2001.- 592 с

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

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