МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Київський національний університет будівництва та архітектури

СИСТЕМНЕ ПРОГРАМУВАННЯ

Методичні вказівки
до виконання лабораторних робіт
для здобувачів І рівня вищої освіти (бакалавр)
122 “Комп`ютерні науки”,
123 “Комп’ютерні системи та мережі”,
133 “Автоматизація та комп’ютерно-інтегровані технології”,
012 “Професійна освіта. Комп’ютерні технології”

Київ 2024

УДК 681.3

С34

Укладачі: В.М. Хроленко, канд. техн. наук, доцент;

В.Г. Голенков, старший викладач

Рецензент О.В. Горда, канд. техн. наук, доцент

Відповідальна за випуск Т.А. Гончаренко, канд. техн. наук, доцент

Затверджено на засіданні кафедри інформаційних технологій, прококол № 07 від 09 лютого 2024 року.

Системне програмування: методичні вказівки до виконання лабораторних робіт / уклад.: В.М. Хроленко, В.Г. Голенков. – К.: КНУБА, 2024. – 40 с.

Містить мету, завдання та зміст лабораторних робіт з дисципліни “Системне програмування” для студентів 2-го курсу, загальні відомості з систем числення, теоретичні дані, план роботи, контрольні запитання.

Призначено для здобувачів ОПП першого рівня вищої освіти (бакалавр) спеціальностей 122 “Комп’ютерні науки”, 123 “Комп’ютерні системи та мережі”, 133 “Автоматизація та комп’ютерно-інтегровані технології”, 012 “Професійна освіта. Комп’ютерні технології”.

ЗМІСТ

Загальні положення

Організація та порядок проведення лабораторних робіт

Попередня підготовка студентів до кожної лабораторної роботи та розуміння її мети та змісту – важливі умови набуття стійких практичних навичок. Тому, взявшись за виконання лабораторної роботи, студент повинен: досконало вивчити зміст роботи та порядок її виконання; повторити теоретичний матеріал, пов'язаний з виконанням даної роботи; підготувати алгоритм та текст програми завдання.

Лабораторні роботи виконуються студентом індивідуально. Лабораторна робота завершується складанням звіту та здачею заліку з неї, після перевірки викладачем правильно працюючої програми на комп'ютері. Лабораторна робота зараховується, якщо схеми алгоритмів та текст програми виконані правильно та охайно, а відповіді студента на запитання викладача є конкретними та повними.

У ході лабораторної роботи студент має вивчити принципи роботи програми та розуміти алгоритми її виконання.

Загальні вказівки до виконання лабораторних робіт

Програма лабораторної роботи може бути успішно виконана лише за умови ретельної підготовки та продуманих дій студента (мається на увазі складання алгоритму, написання та реалізація програми та створення звіту з лабораторної роботи).

Одержавши дозвіл викладача на роботу з комп'ютером, студент стає до виконання лабораторної роботи. Якщо текст програми реалізації завдання на лабораторну роботу знаходиться на особистому носії студента (USB, лазерному диску тощо), останній в присутності викладача перевіряє носій на наявність вірусів, після чого переписує програму на жорсткий диск і працює з ним. Після закінчення роботи викладач приймає комп'ютер в робочому стані у студента.

Небажано робити перерву у відлагодженні програми. Якщо з'явились сумніви у правильності одержаного результату, слід спробувати покрокову відладку програми.

По закінченні роботи слід уважно проаналізувати результати та зробити висновки. Якщо результати реалізації не викликають сумніву, результати слід представити для перевірки викладачеві. В залежності від правильності результатів або стилю програмування викладач дає вказівку або припинити виконання даної лабораторної роботи (при цьому зараховується відлагоджена програма), або повторно виконати роботу, перед цим переробивши у разі потреби програму.

Обробка результатів програм та оформлення звіту з лабораторної роботи

Кожен студент має самостійно проаналізувати результати реалізації програми та скласти звіт з лабораторної роботи. Звіт виконується на аркушах формату А4 та має містити такі частини: титульний аркуш з номером та назвою лабораторної роботи, прізвищем студента та викладача; мету роботи; індивідуальне завдання; схему алгоритму; текст програми, віддрукований на принтері (або написаний від руки у разі неможливості його віддрукувати); контрольний приклад.

Оформлення титульного аркушу та схеми алгоритмів, наведені у звіті, мають відповідати існуючим стандартам.

Лабораторна робота №1
Робота з двійковими, шістнадцятковими, BCD-числами та ASCII-кодами

Перетворення чисел

Перетворення десяткових чисел у двійкові може бути виконано методом віднімання або методом ділення.

Метод віднімання полягає у тому, що від десяткового числа віднімається найбільша можлива двійкова вага та у відповідну їй позицію записують 1. Від отриманого результату віднімають нову найбільшу можливу вагу та у відповідну їй позицію записують 1. Процес провадять доти, поки не отримаємо нульовий результат, після чого у всі ті позиції бітів, ваги яких не віднімали, записують 0.

Наприклад. Перевести 74 у двійкову систему числення

_74
64 $64=2^6$, позиція біта 6=1
_10
8 $8=2^3$, позиція біта 3=1
_ 2
2 $2=2^1$, позиція біта 1=1
0

В усі інші біти (0,2,4,5) записують 0.

Отримуємо результат: 1001010.

Для виконання зворотнього перетворення з двійкового числа у десяткове необхідно значення кожного розряду помножити на його вагу та одержані результати додати. Крайній правий розряд має вагу 2^0, наступний – 2^1, надалі – 2^2, 2^3 тощо. Наприклад. Перетворимо у десяткове число останнє одержане двійкове число 1001010:

$0*2^0+1*2^1+0*2^2+1*2^3+0*2^4+0*2^5+0*2^6=74$.

У методі ділення на першому кроці десяткове число, а на наступних – частка ділиться на основу системи числення, у яку перетворюється число, доти, поки частка не зробиться меншою, ніж основа системи числення. Отримані залишки від ділення утворюють число, перетворене у потрібну

систему числення. Слід пам'ятати, що першим зліва (старшим) розрядом числа є остання отримана частка, другим – останній залишок і так далі; останнім (наймолодшим) – перший залишок.

Наприклад. Перевести 74 у двійкову систему числення.

74:2=37, залишок 0

37:2=18, залишок 1

18:2=9, залишок 0

9:2=4, залишок 1

4:2=2, залишок 0

2:2=1, залишок 0

Отримуємо результат 1001010.

Метод ділення використовується також для перетворення десяткових чисел у шіснадцяткові. При цьому числа 10, 11, 12, 13, 14, 15 необхідно замінити символами A, B, C, D, E, F відповідно.

Наприклад. Перетворити 74 у шістнадцяткове число:

74:16=4, залишок 10 ^

Результат: 4A.

Перетворення з двійкового у шістнадцяткове число виконується шляхом виділення кожних чотирьох бітів (починаючи з правого боку) та перетворення окремо їх у шістнадцяткові розряди за допомогою шістнадцяткових кодів.

Наприклад. Перетворити двійкове число 01001010 у шістнадцяткове: 0100 1010 ---- ---- 2^2=4 2^3+2^1=10 4 A

Результат: 4A.

Зворотня операція, тобто переклад шістнадцяткового числа у двійкове, потребує виконання дій у зворотньому порядку.

Переведення шістнадцяткового числа у десяткове виконується за правилом переведення двійкового у десяткове. При цьому вага кожного розряду змінюється на наступні: 16^0,16^1,16^2 тощо.

Наприклад. Перетворимо шістнадцяткове число 4A у десяткове:

$A*16^0+4*16^1=10*16^0+4*16^1=10+64=74$.

Арифметичні операції, доповняльний код

Операція додавання двійкових чисел проводиться аналогічно операції додавання десяткових чисел, тільки перенесення 1 у старший розряд виконується при наявності 1 в обох розрядах, що додаються.

Операція віднімання чисел в ЕОМ замінюється на операцію додавання. Від'ємне число замінюється доповняльним кодом, і до нього додається зменшуване.

Операція множення зводиться до операції зсуву ліворуч. Зсув на один біт ліворуч збільшує значення у 2 рази. У випадку множення на непарне число операція множення замінюється на операцію зсуву та додавання.

Операція ділення у загальному випадку замінюється на операції зсуву праворуч та віднімання (фактично додавання). Зсув на 1 біт праворуч зменшує значення у 2 рази. При зсуві ліворуч праві звільнені біти заповнюються 0. При зсуві праворуч чисел без знака звільнені ліві біти заповнюються 0, а чисел зі знаком – звільнені біти заповнюються значенням знакового біту.

Щоб знайти доповняльний код від'ємного числа, необхідно узяти його додатню форму, доповнити нулями зліва до байта, інвертувати кожний біт, а потім додати одиницю до одержаного результату.

Наприклад. Перетворити число -74 у доповняльний код:

01001010

10110101 (результат інвертування)

+ 1 (додавання одиниці)

10110110 (доповняльний код) (-74)

Якщо отриманий код перетворити у десяткове число, то воно не буде дорівнювати 74. Проте сума двійкових кодів чисел +74 та -74 дає нульовий результат, що говорить про слушність даного підходу.

01001010 (+74)

+10110110 (-74) (доповняльний код)

(1) 00000000

Вісім біт мають нульове значення, а перенесення одиничного біта ліворуч ігнорується.

Для визначення абсолютного значення від'ємного числа просто повторюють операції інвертування над доповняльним кодом та додають одиницю.

Наприклад. 10110110 -74

01001001 результат інвертування

+ 1 додати одиницю

01001010 74

З цього виходить, що коли число знакове і його 7 або 15 біт має одиницю (у залежності від того, де воно розміщене – у байті або у слові), то комп'ютер це число сприймає як від'ємне, представлене у доповняльному коді. У числах без знаку старший біт виділяється під значення, а не під знак, тому у байті розміщується число без знаку яке має значення від 0 до 255 та знакове число (від –127 до +128).

ASCII – коди, двійкові числа та двійково-кодовані десяткові числа

Дані, які вводяться у ЕОМ з клавіатури, записуються у пам'ять у вигляді ASCII – символів. Аналогічно виведення інформації на екран здійснюється у кодах ASCII. ASCII – код кожного введеного/виведеного символа записується в окремому байті. Наприклад, число 74 після введення з клавіатури у пам'яті буде зберігатися у двох байтах та мати наступний ASCII-код у двійковому форматі: 0011 0111 0011 0100, який у шістнадцятковому форматі має значення 3734.

Однак для виконання у ЕОМ арифметичних операцій над десятковим числом 74 використовується його двійкове значення (двійкове число): 0100 1010, яке у шістнадцятковому форматі записується як 4A.

Не слід плутати представлення чисел у символьному форматі (ASCII-код) та у форматі арифметичних даних (двійкове число). Двійкове число отримується з десяткового числа шляхом перетворення його методами ділення, віднімання або іншими подібними. ASCII – код десяткового числа може бути сформований з його упакованого BCD – числа шляхом запису кожних його чотирьох бітів у окремому байті, а саме у його правій частині, а у його лівій частині – двійкової величини 0011 (див. рис.1).

Можна сказати, що число зберігається у неупакованому форматі, якщо воно записане у вигляді ASCII – кода, та в упакованому форматі, якщо – у вигляді двійкового значення числа, оскільки у першому випадку для зберігання одного розряду шістнадцяткового числа, відповідно десятковому числу, що вводиться, потрібно пам'яті у 1 байт, а у другому випадку – половину байта.

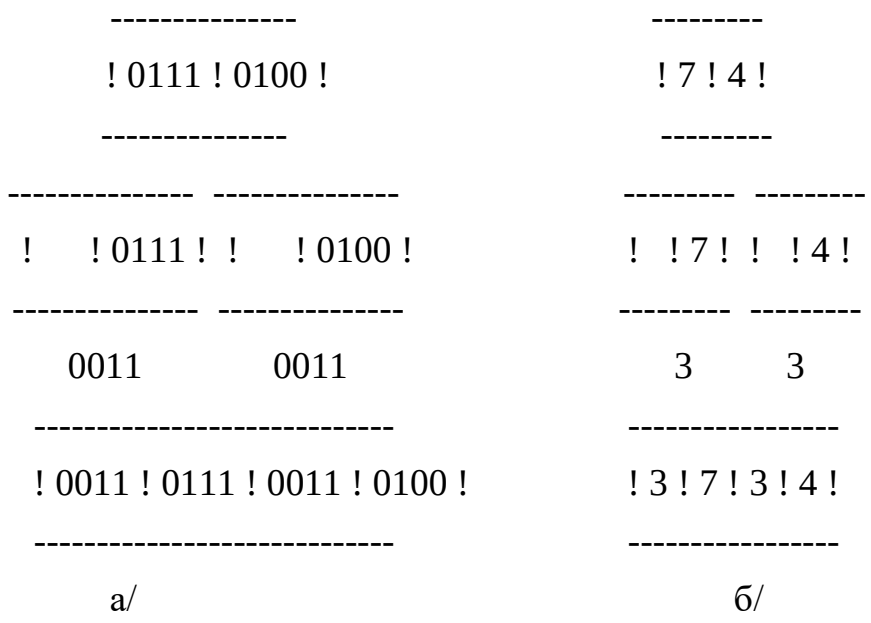

Рис. 1. Формування ASCII-кодів для десяткового числа 74
а/ у двійковому форматі
б/ у шістнадцятковому форматі.

Якщо кожну половину байта використовувати не під один шістнадцятковий розряд, а під-один десятковий, то у цьому випадку ми отримаємо нову форму представлення чисел, які називають упакованими двійково-кодованими десятковими числами (BCD).

Наприклад. Записати число 74 у вигляді упакованого BCD числа.

Для цього необхідно число 7 перетворити на двійкове 0111, а число 4 – на двійкове 0100, потім помістити їх у байт пам'яті таким чином: 01110100. Отриманий результат є упаковане BCD число числа 74.

Неупаковані BCD числа виходять з упакованих BCD чисел шляхом запису кожних 4-х його бітів в окремому байті, а саме у його правій частині.

Наприклад. Перетворити число 74 у неупаковане BCD число.

Упаковане BCD число 01110100, якому відповідає десяткове число 74, записується у неупакованому форматі таким чином:

! 000001110 ! 00000100 !

Упаковані BCD числа використовуються для зберігання великих чисел у RAM пам'яті комп'ютера та виконання над ними різних арифметичних операцій. Крім того, перетворення таких величин у ASCII-коди здійснюється в кілька разів швидше, ніж двійкових чисел.

Адресація пам'яті

Для звернення до будь-якої комірки пам'яті (байту або слова – два байти) процесор обчислює її фізичну адресу, яка формується з адреси сегмента та зміщення за допомогою операції додавання.

Сегмент пам'яті може починатися тільки з адреси, кратної 16 байтам (параграфу). Це призводить до того, що молодший розряд шістнадцяткової адреси сегмента завжди дорівнює 0. Тому у 16-бітний регістр процесора записують старші 4 шістнадцяткові розряди адреси сегменту, а молодший нульовий не записують, але при обчисленні фізичної адреси він завжди враховується. Розмір сегментів може змінюватися від 16 байт до 64 Кбайт (65336 байт). Дійсний розмір сегменту визначається вмістом програми. Сегменти можуть перекриватися один одним; тому один і той самий байт пам'яті може мати різні логічні адреси (комбінація значень адреси сегмента та зміщення), визначені різними, але при цьому еквівалентними парами сегмент – зміщення.

Застосування такого підходу до адресації пам'яті дозволяє при наявності двох 16-бітних регістрів адресувати пам'ять розміром не в 64 Кбайта, а в один Мбайт.

Програма роботи

1. Виконати над десятковими даними a, b, c такі операції:

- перетворити у двійкові числа. Від'ємні числа подати у доповняльному коді;
- перетворити у шістнадцяткові числа;
- перетворити в упаковані двійково-кодовані десяткові числа (BCD) та записати у двійковому форматі;
- перетворити у неупаковані BCD та записати у двійковому форматі;

- перетворити у неупаковані BCD та записати у шістнадцятковому форматі;
- перетворити у ASCII-коди та записати у двійковому форматі;
- перетворити у ASCII-коди та записати у шістнадцятковому форматі.

Варіанти завдань вказані у табл. 1.1.

Результати переведення десяткових чисел оформити у вигляді таблиці (див. табл. 1.2).

2. Виконати такі арифметичні та логічні операції:

- додати у двійковому форматі значення a та c;
- додати у шістнадцятковому форматі значення a та b;
- встановити біти від 0 до 3 змінної A в одиничний стан за допомогою операції AND (логічне "І") та OR (логічне "АБО");
- інвертувати біти змінної B за допомогою операції XOR (що виключають "АБО");
- помножити a на 4, c на 4;
- розділити b на 4, c на 4.

Результати записати у таблицю (див. табл. 1.3 та табл. 1.4).

3. Зобразити фізичну адресу пам'яті за допомогою трьох логічних адрес, які складаються з адреси сегмента та зміщення.

Результати оформити у таблицю (див. табл. 1.5).

Таблиця 1.1

№ п/п	a	b	c	Фізична адреса
1	2	3	4	5
1	171	87	-48	2EAF1
2	136	69	-52	1AABB
3	178	62	-47	AAAFF
4	109	99	-89	A1E8E
5	114	38	-76	B2C1A
6	135	71	-39	9FBE8
7	139	75	-37	5BF2C

8	158	85	-58	61AAC
9	149	94	-35	1A8FF
10	125	12	-91	24ECD
11	138	26	-78	E4AF8
12	141	48	-34	89A0B
13	156	74	-43	64FBA
14	165	69	-87	FAA77
15	169	96	-56	DDBA6
16	173	81	-69	BDDFF
17	137	47	-78	5FB79
18	147	91	-89	F097A
19	175	28	-52	A12BC
20	192	65	-25	FABC4
21	143	84	-98	BAFFF
22	132	61	-92	9A8B0
23	176	56	-72	80EB4
24	194	54	-93	45ABC
25	186	76	-39	75FF5
26	190	83	-45	5EA0B
27	123	74	-57	4AF15
28	173	86	-43	6F7EE
29	196	58	-77	ABCDE
30	187	53	-95	45DC9
31	166	47	-57	4AE28
32	187	55	-79	15A8C
33	148	78	-59	FD76B
34	177	83	-67	D56D3
35	139	38	-91	BA06E

Таблиця 1.2.

№ п/п	Десяткові			Двійкові Шістнадцяткові Упаковані BCD (у двійковому форматі) Неупаковані BCD (у двійковому форматі) Неупаковані BCD (у шіснадцятковому форматі) Символьні (у двійковому форматі) Символьні (у шістнадцятковому форматі		
	A	B	C	A	B	C
27	123	74	-57	01111011 7B 000100100011 00000001000000100 0000011 010203 001100010011 01000110011 313233	01001010 4A 01110100 000001110000010 0 0704 001101110011010 0 3734	11000111 C7 01010111 00000101000000 111 0507 00110101001101 111 3537

Таблиця 1.3.

Арифметичні та логічні операції

№ п/п	Додавання двійкових чисел a та c	Додавання шістнадцяткових чисел a та b	Встановити біти від 0 до 3 в 1 операц. AND та OR	Інвертувати біти b операцією XOR
27	123 – 57 = 66 01111011 (a) + <u>11000111</u> (c) 01000010 Перевірка: $2^6 + 2^1 = 64 + 2 = 66$	123 + 74 = 197 7B + <u>4A</u> C5 Перевірка: 5 + 12*16 = 5 + 192 = 197	Маскування 01111011 & <u>11110000</u> 01110000 Установка: 01110000 V <u>00001111</u> 01111111	01001010 + <u>11111111</u> 10110101

Таблиця 1.4.

Арифметичні операції множення та ділення

№ п/п	Множення двійкових чисел а – без знаку a * 4	Множення Двійкових чисел с – знакове c * 4	Ділення двійкових чисел b – без знаку b : 4	Ділення двійкових чисел с – знакове c : 4
27	123 * 4 = 492	-57 * 4 = -228	74 : 4 = 18	-57 : 4 = -14
	$4 = 2^2$ Зсув ліворуч на 2 біти	$4 = 2^2$ Зсув ліворуч на 2 біти	$4 = 2^2$ Зсув праворуч на 2 біти	$4 = 2^2$ Зсув праворуч на 2 біти
	Число розміщується у байті Початкове значення: 01111011 Результат: 11101100 $2^7 + 2^6 + 2^5 + 2^3 + 2^2 = 128 + 64 + 32 + 8 + 4 = 236$ Результат помилковий, переповнення розрядної сітки	*Число розміщується у байті* Початкове значення: 11000111 Результат: 00011100 $2^4 + 2^3 + 2^2 = 16 + 8 + 4 = 28$ Результат помилковий, переповнення розрядної сітки	*Число розміщується у байті* Початкове значення: 01001010 Результат: 00010010 $2^4 + 2^1 = 16 + 2 = 18$	*Число розміщується у байті* Початкове значення: 11000111 Результат: 11110001 Абсолютне значення: 00001110 + 100001111

	Число розміщується у слові Початкове значення: 0000000001111011 Результат: 0000000111101100 $2^8 + 2^7 + 2^6 + 2^5 + +2^3 + 2^2 = 256 + +128 + 64 + 32 + 8 + 4 = 492$	*Число розміщується у слові* Початкове значення: 1111111111000111 Результат: 1111111100011100 Абсолютне значення: 0000000011100011 + 10000000011100100 $2^7 + 2^6 + 2^5 + 2^2 = 128 + 64 + 32 + 4 = 228$		$2^3 + 2^2 + 2^1 + + 2^0 = 8 + 4 + + 2 + 1 = 15$ $15 – 1 = 14$

Таблиця 1. 5.

Обчислювання логічної адреси

№ п/п	Фізична адреса		№ варі-анту	Логічна адреса				Перевірка
	Шістнадцяткова	Десяткова		Шістнадцяткова		Десяткова		
				Адреса сегменту	Адреса зміщення	Адреса сегменту	Адреса зміщення	
27	4AF15	306965	1	4AF1	5	19185	5	19185*16+5 =306965
			2	4AF0	15	19184	21	19184*16+2 =306965
			3	3FFF	AF25	16383	44837	16383*16+ 44837=30695

Лабораторна робота №2

Створення резидентної програми

Резидентні програми

Програми, які після виконання залишаються у пам’яті, називаються резидентними. Резидентні програми можуть використовуватись у якості утиліт для інших програм. Зазвичай такі програми викликаються через вектор переривання, який не використовується. Операційна система розглядає такі програми як свою складову, захищаючи їх від накладення інших програм, які будуть завантажені згодом. Резидентні програми зазвичай пишуться у форматі COM. Програми, написані у форматі EXE, залишити резидентними у пам’яті трохи складніше.

Завершення програми перериванням 27H залишає її резидентною у пам’яті. CS повинен вказувати на початок PSP для того, щоб ця функція працювала правильно. В програмах COM CS одразу встановлюється відповідним чином, тому треба просто завершити програму перериванням 27H. В програмах EXE CS спочатку вказує на перший байт, який слідує за PSP (тобто 100H). При нормальному завершенні EXE – програми остання інструкція RET виштовхує зі стеку перші покладені туди значення: PUSH DX / MOV AX, 0 / PUSH AX. Оскільки DS первісно вказує на початок PSP, при отриманні цих значень зі стеку лічильник команд вказує на зміщення 0 у PSP, де при ініціалізації записується інструкція INT 20H. Тому INT 20H виконується, а це стандартна функція для завершення програми та передачі управління в операційну систему. Щоб змусити переривання 27H працювати в EXE – програмі, треба помістити 27H у другий байт PSP (перший містить машинний код інструкції INT), а потім завершити програму звичайним RET. Для обох типів файлів, перш ніж здійснити переривання 27H, DX повинен містити зміщення кінця програми, яке відраховується від початку PSP.

Вектор переривання встановлюється за допомогою функції 25H переривання 21H (тут використовується вектор 70H). Процедура має закінчуватись IRET. Окрім самої процедури, програма, що встановлюється, не повинна робити нічого, окрім ініціалізації вектора переривання,

присвоєння DX значення зміщення кінця процедури та завершення. Для COM – файлів просто помістіть оператор INT 27H у кінець програми. Для EXE – файлів помістіть цей оператор у перше слово PSP та завершіть програму звичайним оператором RET.

Нижче наведено приклади для обох типів файлів (COM та EXE). В них установлено мітку FINISH для відзначення кінця процедури переривання. Для COM – файлів FINISH – дає зміщення від початку PSP, як і потрібно для переривання 27H. Для EXE – файлів зміщення – відраховується від першого байту, який слідує за PSP, тому до нього необхідно додати 100H, щоб перерахувати на початок PSP.

Для написання резидентної програми необхідно:

1. Написати програму, яка розмістить резидентну програму в оперативній пам'яті.
2. Написати програму запуску на виконання резидентної програми.
3. Відсікти в програмі носії команди, що не належать програмі.
4. Для того, щоб резидентну програму залишити в оперативній пам'яті, тобто зробити її резидентною, потрібно використовувати переривання BIOS int 27h або функцію 31h int 21h.
5. У регістрі DX фіксується розмір резидентної програми.
6. При використанні функції 31h переривання int 21h формується регістр AX, в який поміщається код завершення програми.
7. Щоб правильно встановити розмір резидентної програми при використанні вказаних переривань, необхідно врахувати, що в модулі COM значення регістра CS вказує на початок PSP (слово стану) програми, а в модулі EXE – на першу команду програми, тобто на першу адресу, розташовану за PSP.
8. У модулі COM значення регістра CS містить початок PSP (розмір PSP – 256 байт або 100H).
9. У EXE – модулі значення регістрів CS і DX вказують на 101H байт, тобто щоб правильно визначити розмір резидентної програми треба до значення регістра DX додати 100H, а в параграфах зробити зсув на 4.

Резидентна COM програма

```
TITLE
CODE SEGMENT
ASSUME CS:CODE, DS:CODE
ORG 100H
PRIM PROC FAR
PUSH DS – збереження регістрів
SUB AX,AX
PUSH AX
PUSH CS
POP DS
BEGIN: JMP CHORT SETUP – перехід до мітки
ROUTINE PROC FAR
PUSH DS
PUSHCS
POP DS
Вставити тіло резидентної програми:
MOV DX.OFFSET MES1
INT 21H
POP DS
IRET
MES1 DB ’Я резидентна COM програма’
ROUTINE ENDP
```

Встановлення вектора переривання:

SETUP: MOV DX,OFFSET ROUTINE – переслати адрес зміщення ROUTINE в регіт.

DX MOV AX,SEG ROUTINE – переслати адресу сегмента проц. ROUTINE в регістр AX.

MOV AL,70H – номер вектору переривання

MOV AH,25H – встановлення наших значень у вектор переривання

INT 21H – встановлення вектору

Залишити тільки резидентну програму:

LEA DX OFFSET FINTSH – запис розміру резидентної програми в регістр DX

INT 27H – збереження програми в пам’яті

RET – повертання до процедури

PRIM ENDP – закінчення процедури

CODE ENDS

END PRIM

Резидентна EXE програма

```
TITLE 'РЕЗИДЕНТНА EXE ПРОГРАММА'
CODE SEGMENT
ASSUME CS:CODE, DS:CODE
PRIM PROC FAR
PUSH DS – збереження регістрів
SUB AX,AX
PUSH AX
PUSH CS
POP DS
JMP SHORT SETUP
ROUTINE PROC FAR – спеціальні вимоги ОС
PUSH DS
PUSHCS
POPDS
```

Вставити тіло резидентної програми:

```
MOV AH,9
MOV DX,OFFSET RES
INT21H
POP DS
```

IRET – повернення з преривання (отримує зі стеку адресу повернення та регістр флагів)

RES DB ’Я резидентна EXE програма’

FINISH LABEL BYTE – мітка закінчення процедури

ROUTINE ENDP

Встановлення вектора переривання:

SETUP: MOV DX,OFFSET ROUTINE

MOV AX,SEG ROUTINE

MOV DS.AX

MOV AH,25H

MOV AL,62H

INT21H

Завершення програми:

MOV DX,OFFSET FINISH

ADD DX,100H – додаємо до регістра КЮн

MOV CL,4 – зсув на 4 розряди

SHR DX,CL

INC DX

MOV AL.0

MOV AH,3IH

INT 21H

PRIM ENDP

CODE ENDS

END PRIM

Функція 31H переривання 21H працює аналогічно, за винятком того, що в DX повинна міститись кількість 16-байтних параграфів, потрібних процедурі. Перевагою цієї функції є те, що вона передає «батьківській» програмі код виходу, який дає інформацію про статус процедури. «Батьківська» програма отримує цей код за допомогою функції 31H переривання 21H.

Програма роботи

Завершити резидентно програму власного переривання. Вектори переривань наведені у таблиці 2.1.

Таблиця 2.1

№ п/п	Номер переривання	№ п/п	Номер переривання
1	2	3	4
1	40h	17	50h
2	41h	18	51h
3	42h	19	52h
4	43h	20	53h
5	44h	21	54h
6	45h	22	55h
7	46h	23	56h
8	47h	24	57h
9	48h	25	58h
10	49h	26	59h
11	4Ah	27	5Ah
12	4Bh	28	5Bh

13	4Ch	29	5Ch
14	4Dh	30	5Dh
15	4Eh	31	5Eh
16	4Fh	32	5Fh

Лабораторна робота №3

Використання мови асемблера при програмуванні на C та C++

Використання асемблерних вставок при програмуванні на C та C++

Причини підключення модулів на мові асемблера до програм на Turbo C та Borland C++ ті ж самі, що й при програмуванні на мові Pascal – це швидкість та отримання доступу до найнижчих рівнів обладнання для генерування найшвидшого, найкомпактнішого та найкращого машинного коду для будь-якої програми. Але й пастки, що підстережують на цьому шляху, також ідентичні – зниження переносимості програм та підвищення вірогідності збоїв. Для більшості програм компілятори Borland C та C++ генерують компактний та швидкий машинний код, з яким важко конкурувати. У багатьох випадках єдиний спосіб додати програмі швидкості – трошки доповнити сучасний «C», використовуючи один з методів:

- використання вбудованих операторів (операторів Inline);
- використання зовнішніх функцій.

Оператори Inline вставляють мову асемблера безпосередньо у вихідну програму на C та C++. Ця методика швидка та проста, але має декілька недоліків. Зовнішні функції, хоча їх важче використовувати, ніж оператори

inline, мають ту перевагу, що дозволяють найбільш повно використовувати всі можливості Turbo Assembler.

Написання програми мовою С з асемблерними вставками

Асемблерні оператори inline починаються словом asm, за яким слідує інструкція асемблера та її операнди. Наприклад, щоб синхронізувати програму із зовнішнім сигналом переривання, можна написати:

```
/* очікування переривання */
asm sti
asm hlt
printf («Переривання отримано\n»);
```

Інший приклад.

```
/*Завдання значень двох змінних, їх сумування та запис у третю змінну*/
#include <stdio.h>
int main(void)
{
int nl, n2, n3; /*оголошення 3-х змінних цілоцисельного типу*/
n1=5, n2=10;/*ініціалізація оголошених змінних*/
ASM MOV AX, [n1]; /*пересилання значення nl в регістр AX*/
ASM ADD AX, [n2]; /*додавання до значення регістру AX значення n2 {nl+n2}*/
ASM MOV [n3], AX; /* пересилання значення регістру AX в n3*/
    printf ("Значення N3 = ", N3);
return 0;
}
```

Основні кроки підключення зовнішніх процедур на мові Асемблер

1. У програму на С оголосити асемблерну процедуру з параметром EXTERNAL.

2. Оголосити асемблерну процедуру як таку, що спільно використовується з іншими зовнішніми програмами. Для цього перед

процедурою на асемблері (наприклад, Prim1) дописати команду PUBLIK.

3. Якщо в програмі на С оголошуються глобальні змінні, то в Асемблерній програмі необхідно ці ж змінні оголосити з параметром EXTERN.

В цьому випадку змінна з Асемблерного модуля буде доступна і в модулі С.

4. В асемблерному модулі імена функцій, що підключаються, і змінних, які передаються програмі на С, треба писати з символом «підкреслення» на початку назви, оскільки компілятор С при компіляції програми до назви додає символ «підкреслення».

Виклик Асемблерної процедури з програми на мові С без передачі параметрів

Програма на С.

```
# include <stdio.h>
external void PRASM ();
external int N1, N2, N3;
int main ()
{
N1=5; N2=10;
PRASM ();
printf ("Значення N3 = ", N3);
return 0;
}
```

Програма на Асемблері:

```
TITLE "PRASM"
DATA SEGMENT
EXTERNAL _N1: WORD; /*змінні, що передаються */
EXTERNAL _N2: WORD; /*змінні, що передаються*/
EXTERNAL _N3 DW 5
PUBLIC N3 – оголошена змінна, що буде доступною для ін. модулів
DATA ENDS
```

```
CODE SEGMENT
ASSUME CS: CODE, CS: DATA
PUBLIC _PRASM
_PRASM PROC
Вимога ОС:
MOV AX, [_Nl]
ADD AX, [_N2]
MOV [_N3], AX
RET
_PRASM ENDP
CODE ENDS
END_PRASM
```

Виклик Асемблерної процедури з програми на мові С з передачею параметрів

Програма на С.

```
# include <stdio.h>
external void PRASM (int p1, int p2, int far *p3;)
int N1, N2;
int *N3=0;
main ();
{
N1=5; N2=10;
printf ("Перед виконанням PRASM N3=", N3);
PRASM (N1, N2,N3);
printf ("Після виконання PRASM N3=", N3")
return 0;
}
```

Програма на Асемблері:

```
TITLE "PRASM"
CODE SEGMENT
ASSUME CS: CODE
PUBLIC _PRASM
```

_PRASM PROC NEAR

Для звернення до змінних за назвою (а не адресою в стеку):

PUSH BP – використовуємо регістр BP для отримання значення зі стеку

MOV BP, SP

PUSH ES

PUSH DI

ARG (N3: WORD, N2. WORD, N1:DWORD) ARG SIZE

MOV AX, [_N1]

ADD AX, [_N2]

LES DI, [_N3]; адресу заміщення перемінної N3 записуємо у регістр DI

MOV [WORD ES:DI], AX

POP DI

POP ES

POP BP

RET ARGSIZE – вказується довжина значень змінних, записаних в стеку

_PRASM ENDP

CODE ENDS

END _PRASM

Програма роботи

Робота складається з трьох частин, кожна з яких захищається окремо, як самостійна лабораторна робота. У першій частині створюється програма на мові С або С++ зі вкрапленням у неї вставок на асемблері, яка підраховує математичний вираз. Підрахунок виразу здійснюється за допомогою асемблерних вставок. Математичні вирази наведені у таблиці 3.1.

Таблиця 3.1

№ п/п	*Функція*
1	Z=(Y+S)*P-(C*(A+B)*P)/Y
2	Z=(A*B+C*P)/S*(X-1)

3	Z=(P+B)/(A-D)+(P-100)
4	Z=(P*S+X*Y)/X-(A*B+X)
5	Z=(A*P-B*Y)/(S+Y)+3*(A-D)
6	Z= (S*X-Y)/P*Y+X
7	Z=P/A*X/C*S/Y+C
8	Z=((A+B)*(P+S)-Y)/X
9	Z=(A*B*C+X)/P*D
10	Z=((P-Y)*S+(X+Y)*P)/Y
11	Z=((A-B)*C*D+P)/Y*D
12	Z=(S+X0/A*(P-Y)/D*(X+Y)/B-Y
13	Z=(A*B/C+C*D/A)*P-S
14	Z=A*B/C+S*X)/Y-D
15	Z=((P*S-Y)/X)*((Y+S)/A)
16	Z=S/(A+B)*Y/(B-D)*P(X+15)-(X*Y)
17	Z=(S*X+(P-Y)/A)-P*Y
18	Z=((P+A-B)*S)/Y*(X-4)

19	Z=((A+D)*(B-C)*S-P*Y)/X
20	Z=(A-B)*(S+Y)/D*Y-(X+P)*Y
21	Z=(P*Y/S-(S+X))*P+X*Y
22	Z=((X-Y)*P+S(P+Y))\(A*B+D)-2*(S+X)
23	Z=((A*B+C*D)*P+(S*Y-75))/C*D
24	Z=(X*X+Y*Y)/P*(S-100)-A*B
25	Z=((P+S)/(A-B)*C*D+(S+X)*Y)/(P+Y)
26	Z=((A+B-C+D)*(B-C)*D+(S+X-Y))/A*B
27	Z=(X-Y+S)*(P-A)/Y+P*X

У другій частині створюється програма на мові C або C++ та зовнішня процедура на асемблері, яка виконується окремо від головної програми на мові C або C++ і підраховує математичний вираз, наведений у таблиці 3.1. Треба зв’язати їх між собою без передачі параметрів, для чого треба здійснити наступне:

- написати асемблерну процедуру PROC типу NEAR або FAR;
- оголосити процедуру PROC PUBLIC, щоб зробити мітку зовнішньої процедури експортованою для компілятора C або C++;
- оголосити дану процедуру в C або C++ як EXTERNAL;
- створити виконавчу програму типу .EXE, для чого у командному рядку компілятора прописати команду tcc з відповідними опціями та підключенням створених заздалегідь файлами типу .OBJ.

У третій частині, як і в другій, створюється програма на мові Pascal та зовнішня процедура на асемблері, яка виконується окремо від головної програми на мові Pascal і підраховує математичний вираз, наведений у таблиці 3.1. Різниця полягає в тому, що зв’язок їх між собою відбувається з передачею параметрів, для чого треба здійснити те ж саме, що й у другій частині, тільки асемблерну процедуру PROC типу NEAR або FAR не оголошувати як EXTERNAL.

Лабораторна робота №4

Пріоритети потоків

Завдання

1. Створити програмне застосування з двома симетричними потоками для виведення даних з одного файлу, що здійснюють читання даних з різними пріоритетами.
2. Перевірити, який потік прочитає більше даних за однаковий проміжок часу. Приклад головного вікна на рис. 4.1.

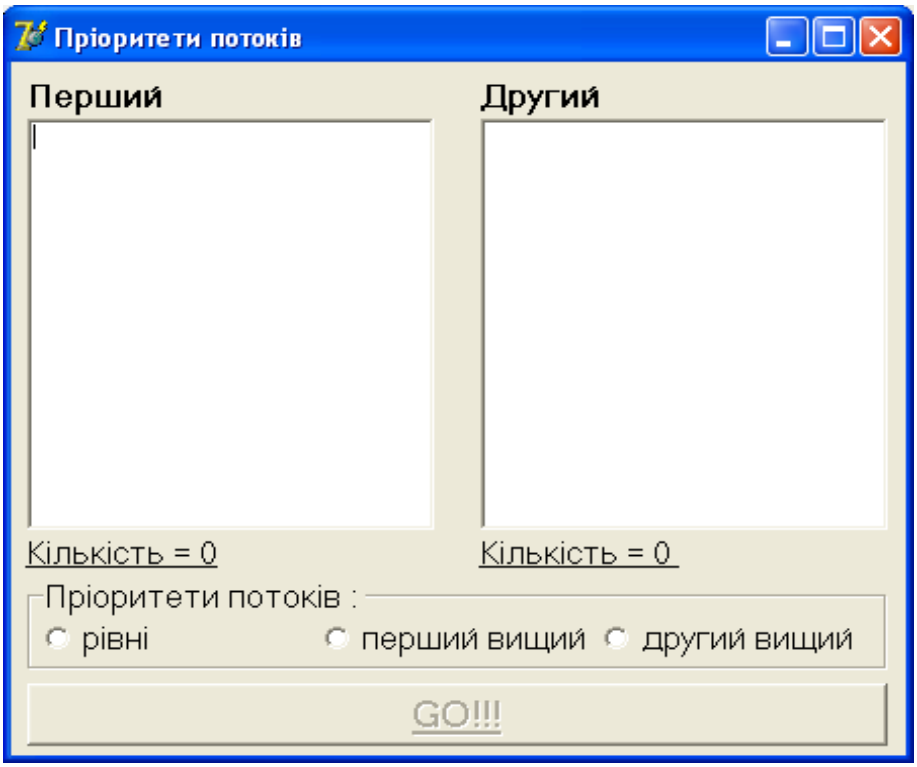

Рис. 4.1. Головне вікно програми

Теоретичні відомості

У класі TThread є властивість Priority, яка визначає пріоритет потоку. В ОС Windows потокам з більшим пріоритетом надаються різні кванти часу: чим вище пріоритет, тим більший квант. Пріоритети потоків (значення поля Priority) в Delphi наведені в таблиці 4.1.

Таблиця 4.1

Пріоритети потоків Delphi

tpIdle	Потік виконується, тільки коли ОС простоює. ОС не буде переривати інші процеси заради процесу з пріоритетьом tpIdle.
tpLowest	На два пункти нижче нормального.
tpLower	На один пункт нижче нормального.
tpNormal	Нормальний пріоритет.
tpHigher	На один пункт вище нормального.
tpHighest	На два пункти вище нормального.

tpTimeCritical	Найвищій пріоритет для програм користувача.

Головний потік застосування має за замовченням пріоритет tpNormal.

Небажано його дочірнім потокам присвоювати вищі пріоритети, тому що «заклякне» обробка інтерфейсу з користувачем!!!

Хід роботи

1. Відкрийте новий проект.
2. Під час створення форми створіть на диску текстовий файл (MyFile – файлова змінна), в який за допомогою генератора випадкових чисел запишіть, наприклад, 100000 цілих.
3. Після створення файлу закрийте його, а потім відкрийте вже для читання (тепер для роботи потоків вже все готово, окрім самих потоків!).
4. Додайте до проекту новий модуль потоку TReadFileThread.
5. Опишіть в потоці чотири поля:

```
Value :integer;   // для зчитування даних з файлу
Count :longint;   // кількість прочитаних значень
MyMemo :TMemo;    // поле для виведення даних
MyLabel :TLabel;  // мітка для виведення кількості
```

6. Додайте до класу-потоку два методи (MyFile – змінна):

```
procedure TReadFileThread.ReadFile;
begin
if not eof(MyFile) then
try
readLn(MyFile, Value); inc(Count);
except
on EInOutError do Value:=0;
end
else terminate;
end;
```

```
procedure TReadFileThread.UpDateView;
begin
MyLabel.Caption:='Кількість = '+IntToStr(Count);
MyMemo.Lines.Add(FloatToStr(Value));
end;
```

7. В методі Execute потоку опишіть цикл для читання даних з файлу:

```
while not eof(MyFile) and not Terminated do
begin
Synchronize(readFile);
Synchronize(UpdateView);
end;
```

Поясніть для наведеного прикладу:

- чому метод UpdateView викликається за допомогою Synchronize?
- чому метод readFile викликається також через Synchronize?
- чому перевірка на eof здійснюється двічі: в Execute та в readFile?
- за якої умови переривається виконання потоку?

8. Опишіть два екземпляри класу TReadFileThread - T1 та T2.

```
Var T1, T2:TReadFileThread.
```

9. Збережіть застосування!!!

10. Повертаємося до модуля форми. Під'єднайте модуль з класом-потоком до модуля форми.

11. В обробнику OnCreate форми після відкриття текстового файлу на читання додайте створення екземплярів потоку T1 та T2:

```
T1:=TReadFileThread.Create (true);
T1.MyMemo:=Memo1;
T1.MyLabel:= Label2;
T1.FreeOnTerminate:=true;
```

// створення T2 аналогічно до T1

Поясніть, для чого під час створення потоку в конструктор передається значення true?

12. В обробнику OnClick для RadioGroup1 задайте присвоєння пріоритетів потокам, зробіть неактивною RadioGroup1, а кнопку навпаки активною:

```
case RadioGroup1.ItemIndex of
0 : begin
T1.Priority:= tpLower;     T2.Priority:= tpLower;
end;
1: … … …
end;
RadioGroup1.Enabled:=false;
Button1.Enabled:=true;
```

Увага!!! Не присвоюйте пріоритети потокам вищі за tpNormal.

Це може у разі некоректної роботи «підвісити» ВСЕ.

13. В обробнику OnClick для кнопки запустіть потоки на виконання:

```
T1.Resume; T2.Resume;
Button1.Enabled:=false;
```

14. Під час закриття форми не забудьте подбати про закриття текстового файлу та руйнацію потоків.

15. Збережіть застосування та запустіть на виконання.

16. Порівняйте кількість прочитаних даних з файлу, коли у потоків рівні або різні пріоритети.

Лабораторна робота №5

Використання м’ютексів для синхронізації потоків

Завдання

1. Створити три несиметричні потоки:

- потік, що змінює форму компонента Shape (3 стани – круг, прямокутник, еліпс);
- потік, що змінює колір компонента Shape (3 стани – червоний, синій, зелений);
- потік, що змінює розташування компонента Shape, тобто задає рух зверху вниз або знизу вверх.

Описати ієрархію класів для потоків (рис. 5.1).

2. Враховуючи те, що при роботі цих потоків компонент Shape буде ресурсом, який спільно використовується, необхідно синхронізувати роботу потоків. Для цього можна використати метод Synchronize або описати м’ютекс. Це і буде головною метою цієї роботи! Кожний потік, перш ніж змінювати параметри Shape, повинен захопити м’ютекс, а після проведення змін звільнити його.

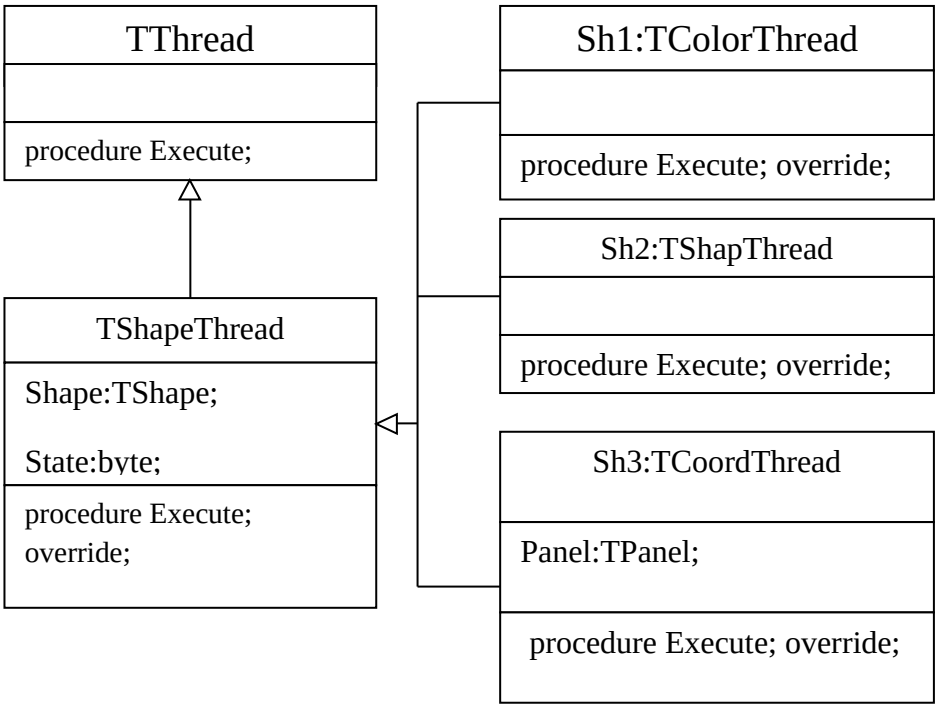

Рис. 5.1. Ієрархія класів потоків

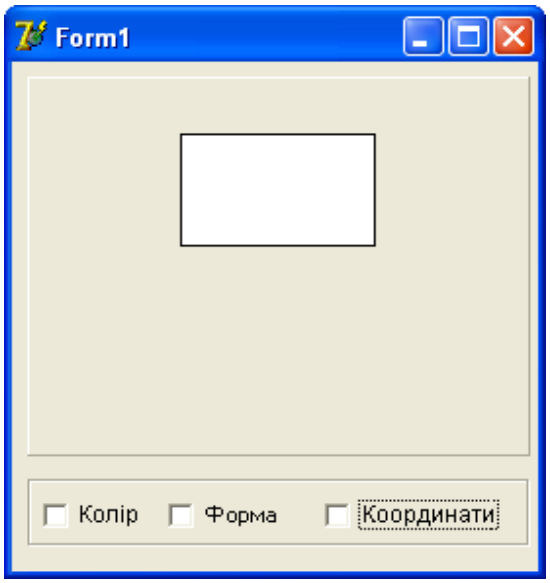

Рис. 5.2. Загальний вигляд головного вікна програми

Теоретичні відомості. Mutex (Mutually Exclusive)

М’ютекс – це об'єкт синхронізації, який знаходиться в сигнальному стані лише тоді, коли він не належить жодному з процесів (потоків). Як тільки хоча б один процес захоплює м’ютекс, він переходить в несигнальний стан і залишається в ньому доти, поки не буде звільнений

власником. Така поведінка дозволяє використовувати м’ютекси для синхронізації спільного доступу декількох процесів до ресурсу, що розділяється.

Створення м’ютекса

Для створення м’ютекса використовується функція:

function CreateMutex(

lpMutexAttributes: PSecurityAttributes; *// Адреса структури TSecurityAttributes*

bInitialOwner: BOOL; *// Задає, чи буде процес володіти // м’ютексом відразу після створення*

lpName: PChar *// Унікальне ім’я м’ютекса*

): THandle; **stdcall**;

Функція повертає дескриптор створеного об'єкту, або 0.

Захоплення м’ютекса

Найпростішою функцією очікування на захоплення є:

function WaitForSingleObject (hHandle:THandle;

dwMilliseconds: DWORD): DWORD; **stdcall,** де hHandle – дескриптор об’єкта (у нашому випадку м’ютекс), dwMilliseconds – період очікування в мілісекундах.

Функція чекає переходу об'єкту hHandle в сигнальний (вільний) стан протягом dwMilliseconds мілісекунд. Якщо як параметр dwMilliseconds передати значення INFINITE, функція очікуватиме необмежений час.

Функція повертає одне з наступних значень:

WAIT_ABANDONED	Потік, що володів об'єктом, завершився, не перевівши об'єкт в сигнальний стан.
WAIT_OBJECT_0	Об'єкт перейшов в сигнальний стан.
WAIT_TIMEOUT	Закінчився термін чекання. Зазвичай в цьому випад-ку генерується помилка, або функція викликається в циклі до отримання іншого результату.
WAIT_FAILED	Сталася помилка, наприклад, невірне значення hHandle. Детальнішу інформацію можна отримати, викликавши функцію GetLastError.

Звільнення м'ютекса

Для повернення в несигнальний стан (звільнення) м'ютекса служить функція:

function ReleaseMutex(hMutex: THandle): BOOL; **stdcall**;

Технологія використання м'ютекса

Якщо декілька процесів через м'ютекс розділяють певні ресурси, кожен з них повинен містити наступний код для забезпечення коректного доступу до загального ресурсу:

```
// Очікування на захоплення
WaitForSingleObject(Mutex, INFINITE);
try
  // Обробка ресурсу. При цьому захват м'ютекса гарантує,
  // що інші процеси які намагаються отримати доступ
  // будуть зупинені на функції WaitForSingleObject
  ...
finally
  // Робота з ресурсом закінчена, звільняємо його
  // для інших процесів
  ReleaseMutex(Mutex);
end;
```

Завжди захищайте захоплення – звільнення об'єкту синхронізації за допомогою блоку try ... finally, інакше помилка під час роботи з ресурсом призведе до блокування роботи всіх процесів, що очікують його звільнення.

Хід роботи

1. Відкрийте новий проект, додайте до нього модуль для створення класів потоків.
2. Опишіть задану ієрархію класів.
3. Якщо ви у скруті, використайте додаток 2 даних методичних вказівок!
4. Опишіть три екземпляри класів-потоків:

```
var
```

Sh1:TColorThread; Sh2:TShapThread; Sh3:TCoordThread;

5. Під’єднайте створений модуль до модуля форми та збережіть застосування.

6. Розробіть інтерфейс з користувачем згідно до рис. 8 даної роботи.

7. В обробнику OnClick для кожного CheckBox задайте створення відповідного екземпляру класу – потока. Наприклад, для CheckBox1:

```
if CheckBox1.Checked then
  begin
    Sh1:= TColorThread.Create(True);
    Sh1.shap:=Shape1;
    Sh1.FreeOnTerminate := True;
    Sh1.Priority := tpLower;
    Sh1.Resume;
end
else
if Assigned(Sh1) then Sh1.Terminate;
```

Аналогічні дії задайте для CheckBox2 та CheckBox3, а також потоків Sh2 та Sh3.

8. Збережіть застосування та переходьте до створення м’ютекса.

9. Опишіть змінну-м’ютекс:

```
var
  Shape_Mutex: THandle;
```

10. Створіть м’ютекс під час створення форми:

```
Shape_Mutex:= CreateMutex(NIL, FALSE, ‘Shape_Mutex);
if Shape_Mutex = 0 then  RaiseLastWin32Error;
```

11. Закрийте м’ютекс під час завершення програми:

```
CloseHandle(Shape_Mutex);
```

12. Збережіть програму та запустіть її на виконання.

Список літератури

1. Modern Operating Systems [RENTAL EDITION] 5th. ANDREW S. TANENBAUM, HERBERT BOS. -Pearson Education, Inc., 2022, ISBN 978-0-13-761887-3.

2. Вільям Столлінгс: Операційні системи. Внутрішня структура та принципи проектування (9-е видання). -Діалектика, 2020, стр. 1264.

3. Operating Systems Design and Implementation 3rd Edition. Andrew Tanenbaum , Albert Woodhull, Pearson Education, Inc., 2006, ISBN 10: 0131429388 ISBN 13: 9780131429383.

4. Функції та принципи роботи математичного співпроцесора. URL: https://ukrbukva.net/

Навчально-методичне видання

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Київський національний університет будівництва та архітектури

СИСТЕМНЕ ПРОГРАМУВАННЯ

Методичні вказівки до виконання лабораторних робіт
для здобувачів ОПП першого рівня вищої освіти (бакалавр) спеціальностей
122 “Комп`ютерні науки”,
123 “Комп’ютерні системи та мережі”,
133 “Автоматизація та комп’ютерно-інтегровані технології”,
012 “Професійна освіта. Комп’ютерні технології”

Комп’ютерне верстання

Підписано до друку 22.02.2024 Формат 60 × 84 1/ 16

Ум. друк. арк. 1,16. Обл.-вид. арк. 1,25.

Електронний документ. Вид № 59/ІІІ-17.

Видавець і виготовлювач

Київський національний університет будівництва і архітектури

Повітрофлотський проспект, 31, Київ, Україна, 03037

Свідоцтво про внесення до Державного реєстру суб’єктів

Видавничої справи ДК №808 від 13.02.20