Методичні вказівки
до виконання лабораторних робіт з курсу
«Операційні системи та системне програмування»
для студентів спеціальності
122 – «Комп’ютерні науки»

Міністерство освіти і науки України
Вінницький національний технічний університет

Методичні вказівки
до виконання лабораторних робіт з курсу
«Операційні системи та системне програмування»
для студентів спеціальності
122 – «Комп’ютерні науки»

Вінниця
ВНТУ
2018

Рекомендовано до друку Методичною радою Вінницького національного технічного університету Міністерства освіти і науки України (протокол № 6 від 14.02.2018 р.).

Рецензенти:
П. І. Кулаков, доктор технічних наук, професор
О. О. Коваленко, кандидат технічних наук, доцент

Методичні вказівки до виконання лабораторних робіт з курсу «Операційні системи та системне програмування» для студентів спеціальності 122 – «Комп'ютерні науки» / Уклад. : О. М. Бевз, М. В. Барабан, Є. А. Паламарчук, В. В. Гармаш. – Вінниця : ВНТУ, 2018. – 28 с.

У даних методичних вказівках розглянуті теоретичні відомості та практичні рекомендації для створення системних додатків керування ресурсами операційної системи Windows, завдання для лабораторних робіт та рекомендована література. Методичні вказівки розроблені відповідно до плану кафедри та програми дисципліни «Операційні системи та системне програмування».

ЗМІСТ

Лабораторна робота № 1

Тема: «Процеси в ОС Windows».

Мета: вивчення функцій WinAPI для створення та управління процесами, набуття практичних навичок створення додатків, що керують поведінкою програми, завантаженої в пам'ять.

Теоретичні відомості

Процес зазвичай визначають як екземпляр виконуваної програми, і він складається з двох компонентів [1]:

- об'єкта ядра, через який операційна система управляє процесом. В ньому зберігається статистична інформація про процес;
- адресного простору, в якому міститься код і дані всіх EXE- і DLL-модулів. Саме в ньому знаходяться області пам'яті, які динамічно розподіляються для стеків потоків й інших потреб.

Процеси інертні. Щоб процес щось виконав, в ньому потрібно створити потік. Саме потоки відповідають за виконання коду, що міститься в адресному просторі процесу. В принципі, один процес може мати декілька потоків, і тоді вони «одночасно» виконують код в адресному просторі процесу. Для цього кожен потік повинен мати у своєму розпорядженні власний набір регістрів процесора і власний стек. У кожному процесі є мінімум один потік. При створенні процесу система автоматично створює його перший потік, який називається головним потоком [2].

Щоб усі ці потоки працювали, операційна система відводить кожному з них певний процесорний час. Виділяючи потокам відрізки часу (так звані кванти), вона створює тим самим ілюзію одночасної роботи потоків.

У Windows під процесом розуміється об'єкт ядра, якому належать системні ресурси, що використовуються виконуваним додатком. Тому можна сказати, що в Windows процесом є виконуваний додаток. Виконання кожного процесу починається з первинного потоку. Під час своєї роботи процес може створювати інші потоки. Виконання процесу закінчується при завершенні роботи всіх його потоків. Кожен процес в операційній системі Windows має такі ресурси:

- віртуальний адресний простір;
- робоча множина сторінок в реальній пам'яті;
- маркер доступу, що містить інформацію для системи безпеки;
- таблиця для зберігання дескрипторів об'єктів ядра.

Крім дескриптора кожен процес в Windows має свій ідентифікатор, який є унікальним для процесів, що виконуються в системі. Ідентифікатори процесів використовуються, головним чином, службовими

програмами, які дозволяють користувачам системи відстежувати роботу процесів.

Новий процес в Windows створюється викликом функції CreateProcess, яка має такий прототип:

```
BOOL CreateProcess(
LPCTSTR lpApplicationName, // назва виконуваного модуля
LPTSTR lpCommandLine, // командний рядок
LPSECURITY_ATTRIBUTES lpProcessAttributes, // захист процесу
LPSECURITY_ATTRIBUTES lpThreadAttributes, // захист потоку
BOOL bInheritHandles, // ознака успадкування дескриптора
DWORD dwCreationFlags, // прапорці створення процесу
LPVOID lpEnvironment, // блок нового середовища оточення
LPCTSTR lpCurrentDirectory, // поточний каталог
LPSTARTUPINFO lpStartUpInfo, // вигляд головного вікна
LPPROCESS_INFORMATION lpProcessInformation, // інформація про процес
);
```

Функція CreateProcess повертає нульове значення, якщо процес був створений успішно. В іншому випадку ця функція повертає значення FALSE. Процес, який створює новий процес, називається батьківським процесом (parent process) щодо створюваного процесу. Новий процес, який створюється іншим процесом, називається дочірнім процесом (child process) відносно процесу-батька [3].

Перший параметр функції CreateProcess lpApplicationName визначає рядок з назвою файлу, який має тип exe (виконуваний) і буде запускатися при створенні нового процесу. Цей рядок має закінчуватися нулем і містити повний шлях до виконуваного файлу.

Назва нового процесу може і не містити повний шлях до exe-файлу, а тільки назву самого exe-файлу. При використанні параметра lpCommandLine система для запуску нового процесу здійснює пошук необхідного exe-файлу в такій послідовності каталогів [3]:

- каталог, з якого запущено програму;
- поточний каталог батьківського процесу;
- системний каталог Windows;
- каталог Windows;
- каталоги, які перераховані в змінній PATH середовища оточення.

Приклад застосування функції CreateProcess:

```
char lpszAppName[] = «C:\\ConsoleProcess.exe»; // рядок з назвою додатка
STARTUPINFO si; // оголошення структури типу STARTUPINFO
PROCESS_INFORMATION piApp; // оголошення структури типу
// PROCESS_INFORMATION
```

```
ZeroMemory(&si, sizeof(STARTUPINFO)); // заповнення нулями пам’яті для структури
//STARTUPINFO
si.cb = sizeof(STARTUPINFO); // визначення змінної «розмір структури
                                    //STARTUPINFO»
CreateProcess(lpszAppName, // виклик функції створення нового консольного
// процесу
NULL,
 NULL,
 NULL,
 FALSE,
CREATE_NEW_CONSOLE,
NULL,
NULL,
&si,
&piApp)
```

Хід роботи

Розробити дві програми. Перша програма має виводити на консоль свою назву та параметри командного рядка. Друга програма має створювати процес, в якому виконується перша програма.

Контрольні запитання

1. Визначення процесу операційної системи Windows.
2. Поняття батьківського і дочірнього процесів.
3. Параметри функції CreateProcess і їх призначення.

Лабораторна робота № 2

Тема: «Потоки в ОС Windows».

Мета: вивчення функцій WinAPI для створення та управління потоками; набуття практичних навичок взаємодії потоків з глобальними змінними.

Теоретичні відомості

Потоком в Windows називається об'єкт ядра, якому операційна система виділяє процесорний час для виконання програми. Кожному потоку належать такі ресурси:

- код виконуваної функції;
- набір регістрів процесора;
- стек для роботи програми;
- стек для роботи операційної системи;
- маркер доступу, який містить інформацію для системи безпеки.

Всі ці ресурси утворюють контекст потоку в Windows. Крім дескриптора кожен потік в Windows також має свій ідентифікатор, який є унікальним для потоків, що виконуються в системі. Ідентифікатори потоків використовуються службовими програмами, які дозволяють користувачам системи відстежувати роботу потоків [3].

В операційних системах Windows розрізняються потоки двох типів:

- системні потоки;
- потоки користувача.

Системні потоки виконують різні сервіси операційної системи і запускаються ядром операційної системи. Потоки користувача служать для вирішення завдань користувача і запускаються додатком.

У чинному додатку розрізняються потоки двох типів:

- робочі потоки (working threads);
- потоки інтерфейсу користувача (user interface threads).

Робочі потоки виконують різні фонові завдання в додатку. Потоки інтерфейсу користувача пов'язані з вікнами і виконують обробку повідомлень, що з'являються в цих вікнах. Кожна програма має, принаймні, один потік, який називається первинним (primary) або головним (Main) потоком. У консольних додатках це потік, який виконує функцію main. У додатках з графічним інтерфейсом це потік, який виконує функцію WinMain.

Створюється потік функцією CreateThread, яка має такий прототип:

```
HANDLE CreateThread(
LPSECURITY_ATTRIBUTES lpThreadAttributes, // атрибути захисту
DWORD dwStackSize, // розмір стека потоку в байтах
LPTHREAD_START_ROUTINE lpStartAddress, // адреса функції
LPVOID lpParameter, // адреса параметра
DWORD dwCreationFlags, // прапорці створення потоку
LPDWORD lpThreadId // ідентифікатор потоку
);
```

При успішному завершенні функція CreateThread повертає дескриптор створеного потоку і його ідентифікатор, який є унікальним для всієї системи. В іншому випадку ця функція повертає значення NULL.

Параметр функції CreateThread lpThreadAttributes встановлює атрибути захисту створюваного потоку. До тих пір, поки ми не вивчимо систему безпеки в Windows, ми будемо встановлювати значення цього параметра в NULL при виклику майже всіх функцій ядра Windows. В даному випадку це означає, що операційна система сама встановить атрибути захисту потоку, використовуючи налаштування за замовчуванням.

Параметр dwStackSize визначає розмір стека, який виділяється потоку при запуску. Якщо цей параметр дорівнює нулю, то потоку виділяється стек, розмір якого за замовчуванням дорівнює 1 Мбайту. Це найменший розмір стека, який може бути виділений потоку. Якщо величина параметра dwStackSize менше значення, заданого за замовчуванням, то все одно потоку виділяється стек розміром в 1 Мбайт. Операційна система Windows округлює розмір стека до однієї сторінки пам'яті, який зазвичай дорівнює 4 Кбайтам.

Параметр lpStartAddress вказує на функцію, яку виконує потік. Ця функція повинна мати такий прототип:

DWORD WINAPI назва_функції_потоку(LPVOID lpParameters);

Функції потоку може бути переданий єдиний параметр lpParameter, який вказує на порожній тип. Це обмеження випливає з того, що функція потоку викликається операційною системою, а не прикладною програмою. Програми операційної системи є виконуваними модулями і тому вони мають викликати лише функції, сигнатура яких заздалегідь визначена. Тому для потоків визначили найпростіший список параметрів, який містить тільки покажчик. Через те, що функції потоків викликаються операційною системою, вони також отримали назву функцій зворотного виклику.

Параметр dwCreationFlags визначає, в якому стані буде створений потік. Якщо значення цього параметра дорівнює 0 (нулю), то функція потоку починає виконуватися відразу після створення потоку. Якщо ж значення цього параметра дорівнює CREATE_SUSPENDED, то потік створюється в призупиненому стані. В майбутньому цей потік можна запустити викликом функції ResumeThread.

Параметр lpThreadId є вихідним, тобто його значення встановлює Windows. Цей параметр має вказувати на змінну, в яку Windows помістить ідентифікатор потоку. Цей ідентифікатор унікальний для всієї системи і може надалі використовуватися для посилань на потік. Ідентифікатор потоку використовується, головним чином, системними функціями і рідко – функціями програми. Ідентифікатор потоку дійсний тільки на час існування потоку. Після завершення потоку той же ідентифікатор може бути присвоєний іншому потоку. В операційній системі Windows 98 цей параметр не може дорівнює NULL. У Windows NT і 2000 допускається встановлення його значення в NULL. Тоді операційна система не повертатиме ідентифікатор потоку [3].

Приклад формування вторинного потоку:

```
DWORD WINAPI Add(LPVOID iNum) // вхідна функція потоку
{
cout << «Thread is started.» << endl; // виведення рядка
cout << «Thread is finished.» << endl; // виведення рядка
return 0;
}
HANDLE hThread; // дескриптор вторинного потоку
DWORD IDThread; // унікальний числовий ідентифікатор потоку
int inc // вхідний параметр вхідної функції потоку
hThread = CreateThread(NULL, // запуск
0,                  // та
Add,                // створення
(void*)inc,         // вторинного
0,                  // потоку
&IDThread);
if (hThread == NULL)   return GetLastError(); // перевірка правильності роботи
                                              // функції CreateThread
```

Хід роботи

Відповідно до свого варіанта:

1. Розробити програму, в якій первинний потік виводить на консоль значення глобальної змінної, а вторинний потік інкрементує цю змінну і також виводить значення на консоль.
2. Розробити програму, в якій у первинному потоці виконується введення символів з клавіатури, а вторинний потік виконує підрахунок лексем у даному рядку.
3. Розробити програму, в якій первинний потік призупиняє роботу вторинного потоку при введенні з клавіатури символа «t». Вторинний потік в циклі має виконувати інкрементацію глобальної змінної. Після завершення вторинного потоку первинний потік має вивести цю змінну на екран.
4. Розробити програму, в якій вторинний потік циклічно збільшує глобальну змінну. Первинний потік має, залежно від введеного з клавіатури символа, або призупиняти роботу вторинного потоку і виводити значення глобальної змінної на екран, або поновлювати роботу вторинного потоку та виводити значення глобальної змінної на екран.

Контрольні запитання

1. Визначення потоку операційної системи Windows.
2. Типи потоків операційної системи Windows.
3. Параметри функції CreateThread.

Лабораторна робота № 3

Тема: «Синхронізація потоків з використанням м’ютексів».

Мета: вивчення функцій WinAPI для роботи з м’ютексами; набуття практичних навичок реалізації взаємодії та синхронізації потоків за допомогою м’ютексів.

Теоретичні відомості

Для вирішення проблеми взаємного виключення між паралельними потоками, що виконуються в контекстах різних процесів, в операційних системах Windows використовується об’єкт ядра м’ютекс.

М’ютекс знаходиться в сигнальному стані, якщо він не належить жодному потоку. В іншому випадку м’ютекс знаходиться в несигнальному стані. Одночасно м’ютекс може належати тільки одному потоку. Потоки, що чекають сигнального стану м’ютекса, обслуговуються в порядку FIFO (First In First Out), тобто потоки стають в чергу до м’ютексів з порядком обслуговування «першим прийшов – першим вийшов» [3].

Для доступу до існуючих м’ютексів потік може використовувати одну з функцій CreateMutex або OpenMutex. Функція CreateMutex використовується в тих випадках, коли потік не знає, створений чи ні м’ютекс із вказаною назвою іншим потоком. У цьому випадку значення параметра bInitialOwner потрібно встановити в FALSE через те, що неможливо визначити, який з потоків створює м’ютекс. Якщо потік використовує для доступу до створеного м’ютекса функцію CreateMutex, то він отримує повний доступ до цього м’ютекса. Для того, щоб отримати доступ до вже створеного м’ютекса, потік може також використовувати функцію OpenMutex, яка має такий прототип:

```
HANDLE OpenMutex (
DWORD dwDesiredAccess, // доступ до м’ютекса
BOOL bInheritHandle // властивість наслідування
LPCTSTR lpName // назва м’ютекса
);
```

Параметр dwDesiredAccess цієї функції може приймати одне з двох значень:

- MUTEX_ALL_ACCESS – повний доступ;
- SYNCHRONIZE – синхронізація.

У першому випадку потік отримує повний доступ до м’ютексів. У другому випадку потік може використовувати м’ютекс тільки у функціях очікування, щоб захопити м’ютекс, або в функції ReleaseMutex для його звільнення.

Параметр bInheritHandle визначає властивість наслідування м’ютекса. Якщо значення цього параметра дорівнює TRUE, то дескриптор м’ютекса, що відкривається, є успадкованим. В іншому випадку дескриптор не успадковується.

У разі успішного завершення функція OpenMutex повертає дескриптор відкритого м’ютекса, а в разі невдачі ця функція повертає значення NULL.

Приклад використання м’ютекса:

```
HANDLE hMutex; // оголошення дескриптора для об’єкта ядра м’ютекс
hMutex = CreateMutex(NULL, FALSE, «DemoMutex»); // створення м’ютекса
if (hMutex == NULL) // перевірка правильності роботи функції CreateMutex
{
cout << «Create mutex failed.» << endl;
cout << «Press any key to exit.» << endl;
cin.get();
return GetLastError();
}
WaitForSingleObject(hMutex, INFINITE); // захоплення м’ютекса
ReleaseMutex(hMutex); // звільнення м’ютекса
```

Хід роботи

Розробити дві програми. В першій програмі циклічно виконується виведення десяти рядків в консоль, які складаються з десяти чисел від нуля до дев’яти. Друга програма має також виводити циклічно у консоль першої програми десять рядків, які складаються з десяти чисел від десяти до дев’ятнадцяти. Виведення рядків всіх чисел має бути виконано від нуля до дев’ятнадцяти.

Контрольні запитання

1. Визначення і призначення м’ютексів.
2. Різниця між функціями CreateMutex і OpenMutex.
3. Типи станів м’ютексів.

Лабораторна робота № 4

Тема: «Робота з віртуальною пам’яттю в ОС Windows».

Мета: вивчення функцій WinAPI для роботи з віртуальною пам’яттю; набуття практичних навичок створення та управління віртуальними сторінками.

Теоретичні відомості

Лінійна адреса процесу в Windows складається з 32 бітів і знаходиться в межах від 0×00000000 до 0×FFFFFFFF. Це теоретично дозволяє процесу звертатися до 4 Гбайтів логічної пам’яті. В операційних системах сімейства Windows NT процесу доступні два молодших гігабайти цієї пам’яті з діапазоном адрес від 0×00000000 до 0×7FFFFFFF, а її старші два гігабайти з діапазоном адрес від 0×80000000 до 0×FFFFFFFF використовуються системою [2].

В операційних системах Windows віртуальна адреса процесу відрізняється від лінійної адреси цього ж процесу тільки інтерпретацією байтів лінійної адреси. Тому можна сказати, що кожному процесу в Windows також доступно два гігабайти віртуальної пам’яті. Це не означає, що процес може використовувати всю цю пам’ять одночасно. Кількість віртуальної пам’яті, що є доступною процесу, залежить від ємності фізичної пам’яті і дисків. Щоб обмежити процес у використанні віртуальної пам’яті, деякі сторінки в таблиці сторінок можуть бути позначені як недоступні.

З точки зору процесу сторінки його віртуальної пам’яті можуть знаходитися в одному з трьох станів:

- вільні для використання (free);
- виділені процесу для використання (committed);
- зарезервовані, але не використовуються процесом (reserved).

Пояснимо ці стани більш детально. Спочатку, при запуску процесу, всі сторінки віртуальної пам’яті вважаються вільними, крім тих, в які завантажена сама програма. Адреси завантаження всіх модулів можна дізнатися, вибравши, при налаштуванні програми, в середовищі розробки Visual C ++ в пункті меню Debug | Modules. Щоб розподілити для використання вільні або зарезервовані сторінки віртуальної пам’яті, процес має викликати функцію VirtualAlloc. Тільки після успішного завершення цієї функції процес може використовувати розподілену йому віртуальну пам’ять. Третій стан характеризує віртуальні сторінки як зарезервовані. Це означає, що ці віртуальні сторінки зарезервовані процесом для подальшого використання і не будуть виділятися системою для використання процесу без точного вказання процесом їх адреси. Потрібно зазначити, що при

резервуванні віртуальних сторінок реальна пам’ять під ці сторінки не виділяється. Для резервування або розподілу області віртуальної пам’яті процес має викликати функцію VirtualAlloc, прототип якої:

```
LPVOID VirtualAlloc(
LPVOID lpAddress, // область для розподілу або резервування
SIZE_T dwSize, // розмір області
DWORD flAllocationType, // тип розподілу
DWORD flProtect // тип захисту доступу
);
```

У разі успішного завершення ця функція повертає адресу віртуальної пам’яті, яка розподілена або зарезервована процесом, а в разі невдачі – NULL. При цьому відзначимо, якщо розподіл віртуальної пам’яті функцією VirtualAlloc завершується успішно, то виділена пам’ять автоматично ініціалізується нулями.

Параметр функції VirtualAlloc lpAddress встановлюється програмою, що викликається, і вказує системі початкову адресу віртуальної пам’яті, яку процес хоче зарезервувати або розподілити. Ця електронна адреса може вказувати як на вільну, так і на зарезервовану раніше віртуальну пам’ять. При встановленні цієї адреси слід розрізняти такі ситуації:

- в разі резервування віртуальної пам’яті цю адресу вирівнюють системою до межі в 64 Кбайти, яка передує вказаній адресі;
- в разі розподілу віртуальної пам’яті з зарезервованої раніше області ця адреса округлюється операційною системою до межі віртуальної сторінки, що містить цю адресу;
- в разі, якщо параметр lpAddress дорівнює NULL, то операційна система сама вибирає початкову адресу області віртуальної пам’яті.

Параметр dwSize встановлюється викликом програми і вказує розмір розподіленої або резервованої області віртуальної пам’яті в байтах. Якщо параметр lpAddress дорівнює NULL, то система округлює цю величину в більшу сторону до кратності розміру віртуальної сторінки. Якщо ж пам’ять розподіляється за конкретними адресами, то ця пам’ять буде охоплювати всі сторінки, які містять байти з діапазону від lpAddress до lpAddress + dwSize. Параметр flAllocationType встановлюється програмою і вказує на тип операції, яку виконує функція VirtualAlloc. Значенням цього параметра може бути будь-яка комбінація з наведених нижче прапорців:

- MEM_COMMIT – розподілити пам’ять програмі;
- MEM_RESERVE – зарезервувати область фізичної пам’яті.

В операційних системах Windows NT / 2000 в цьому параметрі можуть встановлюватися такі прапорці:

- MEM_RESET – пам’ять тимчасово не використовується;
- MEM_TOP_DOWN – розподілити пам’ять, починаючи з найбільшої з вільних адрес.

Крім того, в операційній системі Windows 98 може бути встановлений прапорець:

- MEM_WRITE_WATCH – запам’ятовувати адреси віртуальних сторінок, до яких було проведено запис.

Згодом адреси цих сторінок можна дізнатися, викликавши функцію GetWriteWatch. Очистити список таких сторінок можна за допомогою виклику функції ResetWriteWatch. Через те, що використання цих функцій можливо тільки в операційній системі Windows 98, то ми їх детально розглядати не будемо.

Параметр flProtect встановлює атрибути доступу до області віртуальної пам’яті, які дозволяють виконувати над сторінками віртуальної пам’яті тільки певні операції. Цей параметр може бути комбінацією таких прапорців:

- PAGE_READONLY – дозволяє тільки читання віртуальних сторінок;
- PAGE_READWRITE – дозволяє читання та запис у віртуальних сторінках;
- PAGE_EXECUTE – дозволяє тільки виконання коду у віртуальних сторінках;
- PAGE_EXECUTE_READ – дозволяє виконання і читання коду в віртуальних сторінках;
- PAGE_EXECUTE_READWRITE – дозволяє виконання, читання і запис віртуальних сторінок;
- PAGE_NOACCESS – немає доступу до віртуальних сторінок;
- PAGE_NOCACHE – віртуальні сторінки можна не поміщати в кеш.

Відзначимо, що спроба читання або запису в сторінку, яка призначена тільки для виконання коду (прапорець PAGE_EXECUTE), викличе помилку доступу (access violation). Ця ж помилка виникне в разі будь-якого доступу до віртуальної сторінки, яка має прапорець PAGE_NOACCESS.

В операційних системах Windows NT / 2000 в параметрі flProtect може бути також встановлений прапорець:

- PAGE_GUARD – сторінка під охороною.

Цей прапорець використовується для визначення моменту, коли процесу необхідно виділити додаткову віртуальну пам’ять. Наприклад, цей прапорець можна встановити для останньої віртуальної сторінки області, яка використовується процесом при роботі з деякими даними. Тоді, при спробі доступу до цієї сторінки, генерується програмний виняток і процес буде знати, що віртуальна пам’ять для даних закінчується. Відзначимо також, що прапорець PAGE_GUARD не може бути використаний спільно з прапорцем PAGE_NOACCESS. Після завершення роботи з віртуальною пам’яттю, її необхідно звільнити, використовуючи функцію VirtualFree, яка має такий прототип:

BOOL VirtualFree(

```
LPVOID lpAddress, // адреса області віртуальної пам’яті
SIZE_T dwSize, // розмір області
DWORD dwFreeType // тип операції
);
```

У разі успішного завершення ця функція повертає нульове значення, а в разі невдачі – FALSE.

Параметр dwFreeType може приймати будь-яку комбінацію нижченаведених двох прапорців:

- MEM_DECOMMIT – скасувати розподіл віртуальної пам’яті;
- MEM_RELEASE – звільнити віртуальну пам’ять.

Якщо використовується прапорець MEM_DECOMMIT, то пам’ять не звільняється, а залишається в зарезервованому стані. Щоб звільнити зарезервовану віртуальну пам’ять, потрібно встановити прапорець MEM_RELEASE. У загальному випадку щодо встановлення значення цього параметра можна сказати, що воно повинно відповідати стану області пам’яті, з якою працює функція VirtualFree.

Параметр lpAddress має вказувати на базову адресу області, для якої потрібно скасувати розподіл або звільнити пам’ять. Якщо в параметрі dwFreeType встановлений прапорець MEM_RELEASE, то ця адреса має збігатися з адресою, яку повернула функція VirtualAlloc.

Параметр dwSize задає в байтах розмір області віртуальної пам’яті, розподіл якої потрібно скасувати. Якщо в параметрі dwFreeType встановлений прапорець MEM_RELEASE, то значення параметра dwSize має дорівнювати нулю [2].

Приклад роботи з віртуальною пам’яттю:

```
int *a; // вказівник на масив цілих чисел
const int size = 1000; // розмірність масиву
// розподіляємо віртуальну пам’ять
a = (int*)VirtualAlloc(
NULL,
size * sizeof(int),
MEM_COMMIT,
PAGE_READWRITE);
// перевірка роботи функції VirtualAlloc
if(!a)
{
cout << «Virtual allocation failed.» << endl;
return GetLastError();
}
// виведення адреси виділеної віртуальної пам’яті
cout << «Virtual memory address:» << a << endl;
// звільняємо віртуальну пам’ять
if (!VirtualFree(a, 0, MEM_RELEASE))
```

```
{
cout << «Memory release failed.» << endl;
return GetLastError();
}
return 0;
}
```

Хід роботи

Відповідно до свого варіанта:

1. Розробити програму, яка виконує розподілення віртуальної пам’яті для масиву чисел типу «int», кількість яких становить 1000. Виводить адресу віртуальної пам’яті цього масиву. А після цього звільняє віртуальну пам’ять.
2. Розробити програму, яка виконує розподілення віртуальної пам’яті для масиву чисел типу «int», кількість яких становить 1000, з адреси 0×00890000. А після цього звільняє віртуальну пам'ять.
3. Розробити програму, яка спочатку виконує резервування віртуальної пам’яті з адреси 0×00880000, а потім цю віртуальну пам’ять надає процесу.
4. Розробити програму, яка формує сторінку віртуальної пам’яті і встановлює її в стан PAGE_GUARD. Сформувати вилучення типу EXCEPTION_GUARD_PAGE для цієї сторінки. Потім виконати читання та запис в цю сторінку числа типу «int».

Контрольні запитання

1. Визначення моделі віртуальної пам’яті процесу.
2. Параметри функції VirtualAlloc.
3. Атрибути та типи віртуальних сторінок.

Лабораторна робота № 5

Тема: «Файли, що відображаються в пам’ять».

Мета: вивчення функцій WinAPI для роботи з файлами, що відображаються в пам’ять; набуття практичних навичок створення та управління файлами, що відображаються в пам’ять.

Теоретичні відомості

Операційна система створює файли підкачування з віртуальними сторінками, які система відображає в адресні простори процесів. В операційних системах Windows реалізований механізм, який дозволяє відображати в адресний простір процесу не тільки вміст файлів підкачування, але і вміст звичайних файлів. Тобто, в цьому випадку файл або його частину розглядають як набір віртуальних сторінок процесу, які мають послідовні логічні адреси. Файл, який відображений в адресному просторі процесу, називається поданням або видом файлу (file view). Після відображення файлу в адресний простір процесу доступ до виду може здійснюватися за допомогою покажчика, як до звичайних даних в адресному просторі процесу. Кілька процесів можуть одночасно відображати один і той же файл у свій адресний простір. У цьому випадку операційна система забезпечує узгодженість вмісту файлу для всіх процесів, якщо доступ до цих даних здійснюється як до області віртуальної пам’яті процесу. Тобто, для доступу до файлу, який відображений в пам’ять, не використовується функція WriteFile. Така узгодженість даних, що зберігаються в файлі, відображеному в пам’ять декількома процесами, називається когерентністю даних. Однак потрібно зазначити, що когерентність даних для файлу, відображеного в пам’ять, не підтримується в тому випадку, якщо цей файл відображається в адресний простір процесів, які виконуються на інших комп’ютерах локальної мережі [3].

Тепер коротко опишемо загальну послідовність дій, які необхідно виконати для роботи з відображеним у пам’ять файлом. Ці дії можуть бути розбиті на такі кроки:

- відкрити файл, який буде відображатися в пам’ять;
- створити об’єкт ядра, який виконує відображення файлу;
- відобразити файл або його частину в адресний простір процесу;
- виконати необхідну роботу з видом файлу;
- скасувати відображення файлу;
- закрити об’єкт ядра для відображення файлу;
- закрити файл, який відображався в пам’ять.

Якщо в пам’ять відображається існуючий файл, то, в першу чергу, цей файл має бути відкритий для доступу, використовуючи функцію CreateFile.

Це робиться для того, щоб отримати дескриптор файлу. Надалі цей дескриптор використовується при створенні об'єкта, що відображає файл в пам'ять процесу. Якщо відображений файл використовується просто для обміну даними між процесами, то створювати для цього спеціальний файл на диску не обов'язково. Після того як файл відкритий, створюється об'єкт, що відображає цей файл в пам'ять. Під об'єктом, що відображає файл в пам'ять, можна розуміти об'єкт ядра операційної системи, який виконує відображення файлу в адресний простір процесу. Можна також уявити, що цей об'єкт дозволяє розглядати файл, що відображається в пам'ять, як файл підкачування. Для створення цього об'єкта використовується функція CreateFileMapping, яка має такий прототип:

HANDLE CreateFileMapping(
HANDLE hFile, // дескриптор файлу
LPSECURITY_ATTRIBUTES, lpAttributes // атрибути захисту
DWORD flProtect, // прапорці доступу до файлу
DWORD dwMaximumSizeHigh, // старше подвійне слово розміру об'єкта
DWORD dwMaximumSizeLow, // молодше подвійне слово розміру об'єкта
LPCTSTR lpName // назва об'єкта відображення
);

У разі успішного завершення ця функція повертає дескриптор об'єкта, що відображає файл в пам'ять, а в разі невдачі – NULL. Параметри цієї функції мають таке призначення. Параметр hFile повинен містити дескриптор відкритого файлу, для якого буде створюватися об'єкт, що відображає цей файл в пам'ять процесу. Параметр lpAttributes, зазвичай, вказує на атрибути захисту для об'єкта, що відображає файл в пам'ять. Параметр flProtect містить прапорці, які задають режими доступу у вигляді файла в пам'яті процесу. Цей параметр може набувати одне зі значень:

- PAGE_READONLY – з відображення файлу можна тільки читати дані;
- PAGE_READWRITE – дозволяє читання та запис даних у відображення файлу;
- PAGE_WRITECOPY – дозволяє читання та запис даних у відображення файлу, але під час запису створюється нова копія відображення файлу [2].

Як можна бачити, значення цього параметра збігаються зі значеннями відповідного параметра функції VirtualAlloc, яка розподіляє віртуальну пам'ять процесу. Відзначимо, що режими доступу до об'єкта, який відображає файл в пам'ять, мають відповідати режимам доступу до файлу, для якого створюється цей об'єкт відображення.

Крім того, в параметрі flProtect може бути встановлена будь-яка комбінація прапорців, які визначають атрибути секцій виконуваних

файлів, заданих в переносному форматі (portable executable files). Ці прапорці розглядатися не будуть.

Параметри dwMaximumSizeHigh і dwMaximumSizeLow визначають, відповідно, значення старшої і молодшої частин, які в сукупності задають розмір об'єкта, що відображає файл в пам'ять. Якщо ці значення встановлені в 0, то об'єкт, що відображає файл в пам'ять, має такий саме розмір, що і файл. Відзначимо, що, якщо розмір цього об'єкта буде менше розміру файлу, то система не зможе відобразити весь файл в пам'ять. Якщо ж розмір об'єкта, що відображає файл, більший ніж розмір файлу, то розмір файлу збільшується до розміру об'єкта.

Останній параметр lpName використовується для надання назви об'єкту, що відображає файл в пам'ять. Як завжди, ця назва використовується для доступу до одного і того ж об'єкта в різних процесах. Якщо процес намагається отримати доступ до вже створеного об'єкта, що відображає файл, то прапорці доступу, встановлені в параметрі flProtect, мають відповідати прапорцям доступу, вже встановленим в існуючому об'єкті, що відображає файл [2].

Після того як був створений об'єкт, що відображає файл в пам'ять, файл або його частина мають бути відображені в пам'ять процесу. Іншими словами, має бути створений вид файлу, або його частини в адресному просторі процесу. Для відображення файлу або його частини в адресний простір процесу використовується функція MapViewOfFile, яка має такий прототип:

```
LPVOID MapViewOfFile(
HANDLE hFileMappingObject, // дескриптор об'єкта, який відображає файл
DWORD dwDesiredAccess, // режим доступу
DWORD dwFileOffsetHigh, // старше подвійне слово зміщення
DWORD dwFileOffsetLow, // молодше подвійне слово зміщення
SIZE_T dwNumberOfBytesToMap // кількість відображених байтів
);
```

У разі успішного завершення функція повертає покажчик на відображення файлу в адресному просторі процесу, а разі невдачі – NULL. Параметри цієї функції мають нижчеописане призначення.

Параметр hFileMappingObject має містити дескриптор об'єкта, що відображає файл в пам'ять, який був попередньо створений функцією CreateFileMapping.

Параметр dwDwsiredAccess задає режим доступу до відображення файлу і може приймати одне зі значень:

- FILE_MAP_WRITE – читання і запис у відображення файлу;
- FILE_MAP_READ – тільки читання з відображення файлу;
- FILE_MAP_ALL_ACCESS – читання і запис у відображення файлу;

- FILE_MAP_COPY – при записі в відображення файлу створюється його копія, а вихідний файл не змінюється.

Слід зазначити, що встановлене в цьому параметрі значення має відповідати режиму доступу, який встановлений для об'єкта, що відображає файл в пам'ять.

Параметри dwFileOffsetHigh і dwFileOffsetLow задають зсув від початку файлу або, іншими словами, перший байт файлу, починаючи з якого файл відображається в пам'ять. Цей зсув задається в байтах і має бути кратним гранулярності розподілу віртуальної пам'яті в системі (allocation granularity). Вона може бути визначена за допомогою виклику функції GetSystemInfo. Єдиним параметром цієї функції є покажчик на структуру SYSTEM_INFO в полі dwAllocationGranularity. Функція GetSystemInfo розміщує гранулярність (в байтах). Це значення залежить від архітектури комп'ютера і в більшості випадків складає 64 Кбайти.

Параметр dwNumberOfBytesToMap задає кількість байтів, які будуть відображатися в пам'ять з файлу. Якщо значення цього параметра дорівнює нулю, то в пам'ять буде відображений увесь файл. Якщо необхідно відобразити файл в адресний простір процесу, починаючи з деякої заданої віртуальної адреси, то для цього потрібно використовувати функцію MapViewOfFileEx, яка має такий прототип:

```
LPVOID MapViewOfFileEx(
HANDLE hFileMappingObject, // дескриптор об'єкта, який відображає файл
DWORD dwDesiredAccess, // режим доступу
DWORD dwFileOffsetHigh, // старше подвійне слово зсуву
DWORD dwFileOffsetLow, // молодше подвійне слово зсуву
SIZE_T dwNumberOfBytesToMap, // кількість відображених байтів
LPVOID lpBaseAddress // початковий адрес відображення файлу
);
```

Останній параметр цієї функції вказує на початкову віртуальну адресу відображеного файлу. Цей параметр може бути встановлений в NULL. В цьому випадку система сама вибере початкову адресу завантаження. Інші параметри цієї функції відповідають параметрам функції MapViewOfFile. Після закінчення роботи з відображенням файлу в пам'яті потрібно скасувати відображення файлу в адресний простір процесу. Скасування відображення файлу звільняє віртуальні адреси процесу. Потрібно особливо відзначити, що якщо відображення файлу в адресний простір процесу не скасовано, то система продовжує тримати відображений файл відкритим до тих пір, поки існує його відображення, незалежно від того, закритий цей файл функцією CloseHandle чи ні. Для скасування відображення файлу в пам'ять використовується функція UnmapViewOfFile, яка має такий прототип:

BOOL UnmapViewOfFile (LPCVOID lpBaseAddress);

У разі успішного завершення повертає нульове значення, а в разі невдачі – FALSE. Єдиним параметром цієї функції є початкова адреса відображення файлу. Ця електронна адреса має бути попередньо отримана однією з функцій MapViewOfFile або MapViewOfFileEx.

Приклад збереження даних у сторінковий файл:

```
HANDLE hMapping; // дескриптор об’єкта, що відображає файл у пам’ять
char MappingName[] = «MappingName»; // ім’я об’єкта ядра, що відображає
                                                    // файл у пам’ять
// створення об’єкта, що відображає файл у пам’ять
hMapping = CreateFileMapping(
INVALID_HANDLE_VALUE, // файл підкачування сторінки
NULL, // атрибути захисту за замовчуванням
PAGE_READWRITE, // режим доступу: читання і запис
                                    0, // значення старшого слова = 0
n * sizeof(int), // значення молодшого слова = довжина масиву
MappingName); // ім’я об’єкта відображення
// перевірка результату роботи функції CreateFileMapping
if (!hMapping)
{
cerr << «Create file mapping failed.» << endl;
return GetLastError();
}
// створення відображення файлу
ptr = (int*)MapViewOfFile(
hMapping, // дескриптор об’єкта файлу, що відображається в пам’ять
FILE_MAP_WRITE, // режим доступу до файлу, що відображається в пам’ять
0,0, // відображаємо файл з початку
0); // відображаємо весь файл
// запис масиву даних у сторінковий файл
for (int i = 0; i < n; ++i)
{
ptr[i] = i;
cout << ptr[i] << ' ';
}
```

Хід роботи

Відповідно до свого варіанта:

1. Розробити програму, яка виконує введення масиву цілих чисел у файл. Виконує зміну елементів цього масиву у файлі через відображення файлу у пам’ять. Результат роботи виводить на консоль.

2. Розробити програмне забезпечення, яке виконує обмін даними між двома процесами через відображення файлу у пам’ять. Як дані потрібно використати масив з десяти чисел типу «int». Перша програма має виконувати передавання цього масиву. Друга програма має виконувати прийом цього масиву.

3. Розробити програму, яка виконує введення послідовності десяти чисел типу «int» у файл. Виконує збільшення цієї послідовності з використанням відображення файлу у пам’ять. Результат роботи вивести на консоль.

4. Розробити програму для читання даних з файлу, що відображений в пам’ять іншого процесу.

Контрольні запитання

1. Поняття проекції файлу.
2. Яка послідовність дій, що їх необхідно виконати для роботи з відображеним у пам’яті файлом?
3. Визначення когерентності даних.

Лабораторна робота № 6

Тема: «Робота з динамічною пам’яттю».

Мета: вивчення функцій WinAPI для роботи з динамічною пам’яттю; набуття практичних навичок створення, виділення, перерозподілу та знищення динамічної пам’яті.

Теоретичні відомості

Динамічною пам’яттю називається розподілена процесом область віртуальної пам’яті, яка використовується ним для захоплення і звільнення блоків пам’яті, розмір яких менше розміру віртуальної сторінки. У Windows кожна динамічна пам’ять має свій дескриптор і, отже, є об’єктом ядра.

Для кожного процесу Windows за замовчуванням резервує одну динамічна пам’ять розміром в 1 Мбайт і одразу виділяє з неї 4 Кбайти віртуальної пам’яті для використання процесом. Функції malloc і free зі стандартної бібліотеки мови програмування C, а також оператори new і delete мови програмування C++ розподіляють пам’ять з динамічної пам’яті, яка зарезервована за процесом за замовчуванням. Дескриптор динамічної пам’яті, що створена для процесу за замовчуванням, можна отримати за допомогою функції GetProcessHeap. Функція має такий прототип:

HANDLE GetProcessHeap (VOID);

У разі успішного завершення ця функція повертає дескриптор динамічної пам’яті, а в разі невдачі – значення NULL [3].

Крім того, процес може динамічно створювати область пам’яті під час своєї роботи. Динамічне створення області пам’яті, як правило, спрямоване на прискорення роботи програми з динамічно розподіленою пам’яттю. Ця мета досягається двома способами. По-перше, динамічно створена пам’ять використовується для зберігання тільки однотипних об’єктів. Оскільки в цьому випадку унеможливлюється фрагментація області, то динамічний розподіл пам’яті виконується швидше, ніж у звичайній пам’яті. Та й пам’ять у цьому випадку використовується більш економно. По-друге, динамічно створена область пам’яті може бути серіалізованою, що також прискорює роботу з нею. Адже не потрібно синхронізувати доступ потоків до такої області пам’яті. Але в цьому випадку, на жаль, динамічну пам’ять може використовувати тільки один потік. Для доступу декількох потоків до такої області пам’яті ці потоки мають виконувати синхронізацію самостійно [3].

Для динамічного створення області пам’яті використовується функція HeapCreate, яка має такий прототип:

HANDLE HeapCreate(
DWORD flOptions, // атрибути розподілення динамічної пам’яті
SIZE_T dwInitialState, // початковий стан
SIZE_T dwMaximumSize // максимальний розмір динамічної пам’яті
);

У разі успішного завершення ця функція повертає дескриптор динамічної пам’яті, а в разі невдачі – значення NULL.

Параметр flOptions задає додаткові атрибути для створюваної динамічної пам’яті. Ці атрибути визначаються нижчевказаними прапорцями, які можуть бути встановлені в будь-якій комбінації:

- HEAP_GENERATE_EXCEPTIONS – в разі помилки функції, яка працює з динамічною пам’яттю, система буде генерувати виняток, а не повертати NULL, як вона робить це за замовчуванням;
- HEAP_NO_SERIALIZE – визначає чи є динамічна пам’ять серіалізованою.

Параметр dwInitialSize задає початковий розмір фізичної пам’яті, яка розподіляється динамічною пам’яттю. Цей розмір задається в байтах і округляється системою в більшу сторону до кратності розміру віртуальної сторінки.

Параметр dwMaximumSize задає максимальний розмір динамічної пам’яті в байтах, який округлюється системою в більшу сторону до кратності розміру віртуальної сторінки і не може перевищувати величини в 0×7FFF8 байтів. Якщо параметр dwMaximumSize визначає конкретний розмір динамічної пам’яті, то розмір динамічної пам’яті не може перевищувати цей розмір. Якщо ж параметр dwMaximumSize встановлений в 0 (нуль), то розмір області пам’яті обмежений тільки доступною віртуальною пам’яттю. Для знищення області пам’яті потрібно використовувати функцію HeapDestroy, яка має такий прототип:

BOOL HeapDestroy(
HANDLE hHeap // дескриптор динамічної пам’яті
);

У разі успішного завершення ця функція повертає нульове значення, а в разі невдачі – значення FALSE. Тут параметр hHeap задає дескриптор знищуваної динамічної пам’яті. Функцію HeapDestroy не потрібно застосовувати до динамічної пам’яті, створеної для процесу за замовчуванням, тому що в цьому випадку функція поверне значення TRUE, але сама динамічна пам’ять знищена не буде.

Для розподілу пам’яті з області використовується функція HeapAlloc, яка має такий прототип:

LPVOID HeapAlloc(
HANDLE hHeap, // дескриптор динамічної пам’яті
DWORD dwFlags, // прапорці управління
SIZE_T dwBytes // розмір розподіленої пам’яті
);

У разі успішного завершення ця функція повертає адресу розподіленої пам’яті, а в разі невдачі можливі два варіанти роботи функції. Якщо прапорець HEAP_GENERATE_EXCEPTIONS не встановлено, то функція HeapAlloc в разі невдачі повертає значення NULL. Якщо ж цей прапорець встановлений, то в разі невдачі ця функція генерує один з винятків:

- STATUS_NO_MEMORY – не вистачає пам’яті або пошкодження динамічної пам’яті;
- STATUS_ACCESS_VIOLATION – пошкодження динамічної пам’яті або неправильні параметри функції.

Тепер коротко опишемо призначення параметрів цієї функції. У параметрі hHeap повинен бути встановлений дескриптор динамічної пам’яті, з якої розподіляється пам’ять. Параметр dwFlags керує режимом роботи функції і може бути встановлений в будь-яку комбінацію з нижченаведених прапорців управління:

HEAP_GENERATE_EXCEPTIONS – в разі невдачі функція згенерує виняток;

HEAP_NO_SERIALIZE – немає взаємного виключення при доступі до динамічної пам’яті;

HEAP_ZERO_MEMORY – розподілена пам’ять ініціалізується нулями.

Параметр dwBytes задає в байтах розмір пам’яті, яка буде розподілена з динамічної пам’яті. Якщо пам’ять розподілена з динамічної пам’яті і більше не використовується програмою, то її потрібно звільнити. Для цього використовується функція HeapFree, яка має такий прототип:

```
BOOL HeapFree(
HANDLE hHeap, // дескриптор динамічної пам’яті
DWORD dwFlags, // прапорці управління
LPVOID lpMem // адреса пам’яті
```

У разі успішного завершення ця функція повертає нульове значення, а в разі невдачі – значення FALSE.

Параметр hHeap цієї функції задає дескриптор динамічної пам’яті, а параметр lpMemory – адресу цієї пам’яті. Параметр dwFlags задає прапорці, які встановлюють режим роботи функції. Наразі в цьому параметрі може бути встановлений тільки один прапорець: HEAP_NO_SERIALIZE – немає взаємного виключення при доступі до динамічної пам’яті.

Приклад розподілення та звільнення пам’яті зі стандартної динамічної пам’яті процесу:

```
HANDLE hHeap; // дескриптор динамічної пам’яті
int *a = NULL; // покажчик на масив
int size = 1000; // розмірність масиву
// отримання дескриптором стандартної динамічної пам’яті
hHeap = GetProcessHeap();
// перевірка результату роботи функції GetProcessHeap
```

```
if (!hHeap) return GetLastError();
// розподіляєм пам’ять під масив розмірністю 1000
a = (int*)HeapAlloc(hHeap, HEAP_ZERO_MEMORY, size * sizeof(int));
// звільнюємо динамічну пам’ять
if (!HeapFree(hHeap, NULL, a))
{
cout << «Heap free failed.» << endl;
return GetLastError();
}
```

Хід роботи

Відповідно до свого варіанта:

1. Розробити програму, яка розподіляє пам’ять для масиву з 1000 елементів типу «int» зі стандартної динамічної пам’яті процесу;
2. Розробити програму, яка розподіляє пам’ять для масиву з 2048 елементів типу «int» з динамічно розподіленої пам’яті, розмір якої становить 16384 байти;
3. Розробити програму, яка перевіряє на помилку у випадку створення динамічної пам’яті, розмір якої становить 16384 байти:
4. Розробити програму обробки помилки при створенні динамічної пам’яті за допомогою виключень.

Контрольні запитання

1. Означення динамічної пам’яті.
2. Типи динамічної пам’яті.
3. Функції роботи з динамічною пам’яттю.

Список використаної літератури

1. Робота із процесами в ОС WINDOWS. Методичні вказівки до лабораторної роботи 2 з дисципліни «Операційні системи» для студентів базового напрямку 6.0804 «Комп'ютерні науки» / Укл. Вовчак І. Г. – Львів : Видавництво НУ «Львівська політехніка», 2008. – 4 с.

2. Назарр К. Windows via C/C++. Программирование на языке Visual C++ / К. Назарр, Дж. Рихтер. – Питер : Русская Редакция, 2009. – 896 с.

3. Побегайло А. П. Системное программирование в Windows / Побегайло А. П. – СПб. : БХВ-Петербург, 2006. – 1056 с.

4. Харченко В. П. Операційні системи та системи програмування : навч. посібник / Харченко В. П., Знаковська Є. А., Бородін В. А.– К. : НАУ, 2012. – 348 с.

5. Архангельский А. Я. Программирование в C++ Builder. / Архангельский А. Я. ; [7-е изд.]. – М. : «Бином», 2010. – 1304 с.

6. Дейтел Х. М. Операционные системы. Ч. 1. Основы и принципы / Дейтел Х. М., Дейтел П. Дж., Чофнес Д. Р. – М. : Бином, 2006.

7. Дейтел Х. М. Операционные системы. Ч. 2. Распределённые системы, сети, безопасность / Дейтел Х. М., Дейтел П. Дж., Чофнес Д. Р. – М. : Бином, 2006.

Навчальне видання

Методичні вказівки до виконання лабораторних робіт з курсу «Операційні системи та системне програмування» для студентів спеціальності 122 – «Комп'ютерні науки»

Укладачі: Олександр Миколайович Бевз
Марія Володимирівна Барабан
Євген Анатолійович Паламарчук
Володимир Володимирович Гармаш

Рукопис оформила М. Барабан

Редактор В. Дружиніна

Оригінал-макет підготував О. Ткачук

Підписано до друку 01.10.2018 р.
Формат 29,7×42¼. Папір офсетний.
Гарнітура Times New Roman.
Друк різографічний. Ум. друк. арк. 1,68.
Наклад 40 (1-й запуск 1 – 20) пр. Зам. № 2018-173.

Видавець та виготовлювач
Вінницький національний технічний університет,
інформаційний редакційно-видавничий центр.

ВНТУ, ГНК, к. 114.
Хмельницьке шосе, 95,
м. Вінниця, 21021.
Тел. (0432) 65-18-06.
press.vntu.edu.ua;
E-mail: kivc.vntu@gmail.com.

Свідоцтво суб'єкта видавничої справи
серія ДК № 3516 від 01.07.2009 р.