МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КРЕМЕНЧУЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ МИХАЙЛА ОСТРОГРАДСЬКОГО

МЕТОДИЧНІ ВКАЗІВКИ
ЩОДО ПРАКТИЧНИХ ЗАНЯТЬ І САМОСТІЙНОЇ РОБОТИ
З НАВЧАЛЬНОЇ ДИСЦИПЛІНИ
“СИСТЕМНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ”
ДЛЯ СТУДЕНТІВ ДЕННОЇ ТА ЗАОЧНОЇ ФОРМ НАВЧАННЯ
ЗІ СПЕЦІАЛЬНОСТІ 123 – «КОМП'ЮТЕРНА ІНЖЕНЕРІЯ»
ЧАСТИНА ІІ

КРЕМЕНЧУК 2020

Методичні вказівки щодо практичних занять і самостійної роботи з навчальної дисципліни «Системне програмне забезпечення» для студентів денної та заочної форм навчання зі спеціальності 123 – «Комп'ютерна інженерія» Частина ІІ

Укладач старш. викл. Ю.В. Зілінський

Рецензент доц. Ю.В. Лашко

Кафедра комп’ютерних та інформаційних систем

Затверджено методичною радою КрНУ імені Михайла Остроградського

Протокол № _____ від « ___ » _____ 20___ р.

Голова методичної ради _____ проф. В. В. Костін

ЗМІСТ

ВСТУП

Практичні заняття з навчальної дисципліни «Системне програмне забезпечення» сприяють поглибленню теоретичних знань, які студенти отримують у лекційному матеріалі й значною мірою дозволяють підготувати студента до подальшого виконання лабораторних робіт з дисципліни.

Кожне практичне заняття містить набір завдань і перелік питань для самостійного опрацювання. Самостійне виконання набору завдань і робота з літературою дозволяють закріпити матеріал практичного заняття і поглибити теоретичні знання з теми заняття. Студенти отримують завдання по закінченні практичного заняття і повинні виконати його до наступного практичного заняття (або в терміни зазначені викладачем, але не раніше наступного практичного заняття). Завдання виконується студентом у позааудиторний час, у межах годин самостійної роботи, зазначених у робочій навчальній програмі.

Наведені в методичних вказівках приклади розроблено в безкоштовно поширюваному інтегрованому середовищі розробки програм RadAsm для безкоштовно поширюваного пакета асемблера MASM32 v10. Виконання завдань практичних занять може виконуватися з використанням пакетів асемблерів TASM, FASM, NASM, HLA, GoASM та C-компіляторів LCC, Borland C++, Microsoft Visual C++, які також підтримуються RadAsm.

Тематика практичних занять даних методичних вказівок перетинається з тематикою лабораторних робіт з курсу та охоплює питання керування процесами і різних методів організації їх взаємодії.

Друга частина є логічним продовженням першої частини цих методичних вказівок і разом вони охоплюють сім практичних занять, передбачених у шостому навчальному семестрі робочою навчальною програмою дисципліни «Системне програмне забезпечення» для бакалаврів зі спеціальності 123 – «Комп'ютерна інженерія».

1 ПЕРЕЛІК ПРАКТИЧНИХ ЗАНЯТЬ
ПРАКТИЧНЕ ЗАНЯТТЯ № 1

Тема. Керування процесами. Створення та знищення процесів

Мета: продемонструвати різні способи створення процесів та способи керування як дочірніми, так і не дочірніми процесами.

Короткі теоретичні відомості

1.1 Створення процесів

Для створення процесів у програмному інтерфейсі ОС Windows NT передбачена універсальна функція

```
BOOL CreateProcess(
    IN LPCTSTR lpApplicationName,
    IN LPTSTR  lpCommandLine OPTIONAL,
    IN LPSECURITY_ATTRIBUTES lpProcessAttributes OPTIONAL,
    IN LPSECURITY_ATTRIBUTES lpThreadAttributes OPTIONAL,
    IN BOOL bInheritHandles,
    IN DWORD dwCreationFlags,
    IN LPVOID lpEnvironment OPTIONAL,
    IN LPCTSTR lpCurrentDirectory OPTIONAL,
    IN LPSTARTUPINFO lpStartupInfo,
    IN OUT LPPROCESS_INFORMATION lpProcessInformation);
```

lpApplicationName і lpCommandLine покажчики на рядки, що містять повний або частковий шлях до програмного файла й командний рядок, що містить параметри запуску процесу.

lpProcessAttributes і lpThreadAttributes покажчики на атрибути захисту, відповідно, процесу і його головного потоку.

bInheritHandles – прапор спадкування ідентифікаторів батьківського процесу породженими (дочірніми) процесами. TRUE – дозволити спадкування, FALSE - скасувати спадкування.

dwCreationFlags – визначає базовий клас пріоритету процесу (REALTIME_PRIORITY_CLASS, HIGH_PRIORITY_CLASS,

NORMAL_PRIORITY_CLASS, IDLE_PRIORITY_CLASS), а також спосіб, яким буде запущений процес (тут не розглядається).

lpEnvironment – адреса блока середовища виконання (застаріло).

lpCurrentDirectory – адреса символьного рядка, що містить шлях до каталогу, який буде використаний як робочий при запуску процесу. Якщо цей параметр заданий як NULL, як робочий каталог для нового процесу буде використаний робочий каталог батьківського процесу.

lpStartupInfo – покажчик на структуру типу STARTUPINFO, що визначає параметри створення процесу і має наступний вигляд:

```
STARTUPINFOA STRUCT
  cb              DWORD ?  ; розмір структури в байтах
  lpReserved      DWORD ?  ; зарезервовано
  lpDesktop       DWORD ?  ; робочий стіл і станція для процесу
  lpTitle         DWORD ?  ; заголовок вікна консольного процесу
  dwX             DWORD ?  ; координата кута вікна в пікселах
  dwY             DWORD ?  ; координата кута вікна в пікселах
  dwXSize         DWORD ?  ; ширина вікна в пікселах
  dwYSize         DWORD ?  ; висота вікна в пікселах
  dwXCountChars   DWORD ?  ; ширина консольного вікна
  dwYCountChars   DWORD ?  ; висота консольного вікна
  dwFillAttribute DWORD ?  ; атрибути тексту консольного вікна
  dwFlags         DWORD ?  ; заповнені поля структури  (див. нижче)
  wShowWindow     WORD ?   ; зовнішній вигляд вікна додатка
  cbReserved2     WORD ?   ; зарезервовано
  lpReserved2     DWORD ?  ; зарезервовано
  hStdInput       DWORD ?  ; консоль уведення
  hStdOutput      DWORD ?  ; консоль виведення
  hStdError       DWORD ?  ; консоль виведення повідомлень
STARTUPINFOA ENDS
```

Як правило, всі елементи цієї структури не заповнюються «вручну». Замість цього використовують виклик функції VOID GetStartupInfo(LPSTARTUPINFO lpStartupInfo), яка повертає за адресою lpStartupInfo структуру STARTUPINFO, що буде використовуватися за замовчанням операційною системою при створенні нового процесу. Після цього виконується коректування значень елементів структури у відповідність зі своїми потребами, а в елементі dwFlags установлюється комбінація прапорів, що визначають уміст яких полів структури STARTUPINFO змінено. Операційна система буде враховувати ці елементи STARTUPINFO при створенні нового процесу відповідно наступним прапорцям:

STARTF_USESHOWWINDOW – wShowWindow
STARTF_USEPOSITION – dwX, dwY
STARTF_USESIZE – dwXSize, dwYSize
STARTF_USECOUNTCHARS – dwXCountChars, dwYCountChars
STARTF_USEFILLATTRIBUTE – dwFillAttribute
STARTF_USESTDHANDLES – hStdInput, hStdOutput й hStdError

lpProcessInformation – адреса структури типу PROCESS_INFORMATION, у яку в разі вдалого завершення функції повертається інформація про створений процес. Елементі цієї структури в подальшому можуть бути використані батьківським процесом для керування дочірнім процесом і визначена вона у такий спосіб:

```
PROCESS_INFORMATION STRUCT
  hProcess      DWORD ?  ; ідентифікатор процесу
  hThread       DWORD ?  ; ідентифікатор головного потоку процесу
  dwProcessId   DWORD ?  ; системний номер процесу
  dwThreadId    DWORD ?  ; системний номер головного потоку процесу
PROCESS_INFORMATION ENDS
```

Якщо функція CreateProcess завершується успішно, вона повертає значення TRUE і заповнену структуру типу PROCESS_INFORMATION, у протилежному випадку повертається значення FALSE.

Процес також може бути створений з використанням функції оболонки ShellExecute. Вона дозволяє створювати процеси як у явному вигляді, так і породжувати їх у неявному вигляді. Під неявним тут мається на увазі те, що в деяких випадках, загалом кажучи, невідомо, який програмний файл буде образом процесу. Прототип функції ShellExecute має наступний вигляд:

```
HINSTANCE ShellExecute(HWND hwnd,
    LPCTSTR lpOperation,
    LPCTSTR lpFile,
    LPCTSTR lpParameters,
    LPCTSTR lpDirectory,
    INT nShowCmd);
```

hwnd – ідентифікатор вікна, що буде використовуватися для відображення інтерфейсу (GUI) користувача або повідомлень про помилку.

lpOperation – покажчик на рядок, що закінчується нулем і визначає дію, яку треба виконати:

- «edit» – запускає редактор і відкриває документ для редагування. Якщо lpFile – не файл документа, функція завершується з помилкою;
- «explore» – відкриває в Explorer папку, зазначену в lpFile;
- «find» – ініціалізує в Explorer пошук, що починається з папки, зазначеної в lpFile;
- «open» – відкриває файл, зазначений у lpFile. Файл може бути програмним файлом, файлом документа, або папкою. У випадку програмного файла створюється процес. Файл документа відкривається асоційованою програмою, тобто виконується неявне створення процесу. Папка відкривається в окремому вікні Explorer;
- «print» – друкує файл документа, зазначений у lpFile. Якщо lpFile указує не на файл документа, функція завершується з помилкою.

lpFile – покажчик на рядок, що закінчується нулем і задає об'єкт для виконання lpOperation. Об'єкт повинен задаватися повним ім'ям. Не всі дії

підтримуються для всіх об'єктів. Наприклад, не всі типи документів підтримують операцію «print».

lpParameters – якщо параметр lpFile вказує на ім'я програмного файла, то lpParameters може містити покажчик на рядок, що закінчується нулем, і задає параметри командного рядка, який буде передано додатку. Якщо lpFile указує на файл документа, lpParameters повинен бути NULL.

lpDirectory – покажчик на рядок, що закінчується нулем, і задає робочий каталог для створюваного процесу.

nShowCmd – задає параметри відображення вікна додатка (аналогічний параметру функції ShowWindow):

- SW_HIDE приховує вікно й активізує інше вікно;
- SW_MAXIMIZE максимізує вікно;
- SW_MINIMIZE мінімізує вікно й активізує наступне вікно верхнього рівня;
- SW_RESTORE активізує й відображає вікно. Якщо вікно мінімізоване або максимізоване, Windows відновлює його в оригінальному розмірі й позиції;
- SW_SHOW активізує вікно й відображає його в поточному розмірі й позиції;
- SW_SHOWDEFAULT установлює стан показу, заснований на прапорці SW_*, заданому в структурі STARTUPINFO переданої функції CreateProcess у програмі, що запустила додаток;
- SW_SHOWMAXIMIZED активізує вікно й відображає його як максимізоване вікно;
- SW_SHOWMINIMIZED активізує вікно й відображає це як мінімізоване вікно;
- SW_SHOWMINNOACTIVE відображає вікно як мінімізоване вікно. Активне вікно залишається активним;

- SW_SHOWNA відображає вікно в його поточному стані. Активне вікно залишається активним;
- SW_SHOWNOACTIVATE відображає вікно в його останніх розмірах і позиції. Активне вікно залишається активним;
- SW_SHOWNORMAL активізує й відображає вікно. Якщо вікно мінімізоване або максимізоване, Windows відновлює його до оригінального розміру й позиції.

У випадку вдалого завершення функція повертає значення більше ніж 32, у протилежному випадку – код менший або рівний 32, що дозволяє конкретизувати помилку.

1.2 Знищення процесів

Знищити процес можна за допомогою функції BOOL TerminateProcess (HANDLE hProcess, UINT uExitCode). Параметр uExitCode аналогічний параметру функції ExitProcess і являє собою довільний код, що буде повернено операційній системі після завершення процесу. Як параметр hProcess функції TerminateProcess необхідно передати ідентифікатор знищуваного процесу. Але якщо процес, що підлягає знищенню, не є дочірнім, то невідомий і ідентифікатор цього процесу, який повертає функція CreateProcess при його створенні. Про ідентифікатор процесу створеного за допомогою ShellExecute взагалі говорити не доводиться.

У програмному інтерфейсі Windows NT існує набір спеціальних засобів, що складають так званий інтерфейс Win Debug API. Функції цього інтерфейсу використовуються налагоджувачами третього кільця захисту і дозволяють завантажувати програму або приєднуватися до запущеної програми для налагодження, одержувати інформацію про програму, що налагоджується, одержувати повідомлення про події, пов'язані з налагодженням, змінювати процес/потік, що налагоджується і т.і.

Робота з функціями інтерфейсу Win Debug API передбачає попереднє відкриття процесу за допомогою функції

```
HANDLE WINAPI OpenProcess(
    DWORD    dwDesiredAccess,
    BOOL     bInheritHandle,
    DWORD    dwProcessId);
```

dwDesiredAccess – визначає права доступу до процесу. Можливість успішного відкриття процесу залежить від поточного рівня привілеїв процесу, що відкриває. Щоб відкрити будь-який, у тому числі й системний процес та одержати повні права доступу (PROCESS_ALL_ACCESS) процес, що відкриває повинен мати привілей SeDebugPrivilege. Ним, у більшості випадків, володіють тільки адміністратори, тому що це дозволяє маніпулювати усіма без винятку процесами системи. Отримати за необхідності привілей SeDebugPrivilege не маючи повноважень облікового запису адміністратора можна з використанням функцій API OpenProcessToken, LookupPrivilegeValue, AdjustTokenPrivileges.

bInheritHandle – якщо цей параметр TRUE, ідентифікатор може успадковуватися. Якщо параметр FALSE, ідентифікатор не може бути успадкований.

dwProcessId – PID процесу, що відкривається. Його можна одержати з використанням API раніше розглянутих інтерфейсів ToolHelp32, PSAPI, Native API.

Якщо виклик завершився не вдало функція повертає NULL, інакше – ідентифікатор процесу, який використовується для подальшої роботи з ним, і який зокрема можна використовувати як параметр функції TerminateProcess.

1.3 Приклади програмування

Використання вище розглянутих методів створення та знищення процесів демонструють наступні два приклади програмування.

Приклад 0.1 Консольний додаток Starter.exe створює процеси для:

- додатка з графічним інтерфейсом Блокнот (notepad.exe) без параметрів та з параметрами командного рядка;

- сеансу командного інтерпретатора cmd.exe з параметрами командного рядка у вікні із заданими параметрами відображення;
- додатків, призначених за замовчанням для відкриття файлів різних типів.

```
; Файл Starter.inc
include windows.inc
include user32.inc
include kernel32.inc
include masm32.inc
include shell32.inc
include C:\MASM32\MACROS\strings.mac
includelib user32.lib
includelib kernel32.lib
includelib masm32.lib
includelib    shell32.lib
Main      PROTO
.data?
StartInfo             STARTUPINFO <>
ProcessInfo1        PROCESS_INFORMATION <>
ProcessInfo2        PROCESS_INFORMATION <>
ProcessInfo3        PROCESS_INFORMATION <>
; Файл Starter.asm
.386
.model flat, stdcall
option casemap:none
include Starter.inc
.code
start:
  invoke Main
  invoke ExitProcess,0
```

```
Main proc
invoke AllocConsole
invoke FreeConsole
invoke SetConsoleTitle, $CTA0("Starter")
invoke ClearScreen
invoke GetStartupInfo, ADDR StartInfo
invoke CreateProcess,$CTA0("C:\\WINDOWS\\SYSTEM32\\notepad.exe"),\
        NULL, NULL, NULL, FALSE, NORMAL_PRIORITY_CLASS,\
        NULL, NULL, ADDR StartInfo, ADDR ProcessInfo1
 invoke CreateProcess,0,\
        $CTA0("C:\\WINDOWS\\SYSTEM32\\notepad.exe schedlgu.txt"),\
        NULL,NULL,FALSE, NORMAL_PRIORITY_CLASS, NULL,\
        $CTA0("C:\\WINDOWS"), ADDR StartInfo, ADDR ProcessInfo2
mov StartInfo.dwFillAttribute, 1Eh
mov StartInfo.dwX, 0
mov StartInfo.dwY, 0
mov StartInfo.dwXSize, 1200
mov StartInfo.dwYSize, 1000
mov  StartInfo.dwFlags,STARTF_USEPOSITION  OR  STARTF_USESIZE  OR
STARTF_USEFILLATTRIBUTE
invoke CreateProcess, NULL, \
        $CTA0("C:\\WINDOWS\\SYSTEM32\\cmd.exe /k dir|more"),\
        NULL, NULL, FALSE, CREATE_NEW_CONSOLE, NULL,\
        $CTA0("C:\\WINDOWS"), ADDR StartInfo, ADDR ProcessInfo3
invoke ShellExecute,0,$CTA0("open"), $CTA0("notepad.exe"),\
                0, 0, SW_MAXIMIZE
invoke ShellExecute,0,$CTA0("open"), $CTA0("notepad.exe"), \
        $CTA0("C:\\WINDOWS\\schedlgu.txt"),0,SW_MAXIMIZE

invoke ShellExecute,0,$CTA0("open"),
```

```
        $CTA0("C:\\WINDOWS\\schedlgu.txt"), 0, 0, SW_MAXIMIZE
invoke ShellExecute,0,$CTA0("edit"),\
        $CTA0("C:\\WINDOWS\\schedlgu.txt"), 0, 0, SW_MAXIMIZE
invoke ShellExecute,0,$CTA0("open"),\
        $CTA0("C:\\WINDOWS\\Web\\Wallpaper\\Windows XP.jpg"),\
        0,0,SW_MAXIMIZE
invoke ShellExecute,0,$CTA0("explore"),\
        $CTA0("C:\\WINDOWS"), 0, 0, SW_MAXIMIZE
invoke CloseHandle,ProcessInfo1.hProcess
invoke CloseHandle,ProcessInfo2.hProcess
invoke CloseHandle,ProcessInfo3.hProcess
ret
Main endp
end start
```

Приклад 0.2 Керування недочірнім процесом продемонструє консольний додаток ProcessKiller.exe, який знищує процес, ім'я образу якого отримує як параметр командного рядка.

```
; Файл ProcessKiller.Inc
include windows.inc
include user32.inc
include kernel32.inc
include masm32.inc
include advapi32.inc
include C:\MASM32\MACROS\strings.mac
includelib  user32.lib
includelib  kernel32.lib
includelib  masm32.lib
includelib  advapi32.lib
Main      PROTO
```

```
.data?
hSnapshot          HANDLE ?
ProcessInfo        PROCESSENTRY32  <>
szProcessName      BYTE MAX_PATH+1 dup (?)
hProcess           HANDLE ?
ExitCode           DWORD ?
; Файл ProcessKiller.Asm
.386
.model flat, stdcall
option casemap:none
include ProcessKiller.Inc
.code
start:  invoke Main
        invoke ExitProcess,0
Main proc
invoke SetConsoleTitle, $CTA0("Process Killer")
invoke ClearScreen
lea esi, szProcessName
invoke GetCL,1, esi
.if eax==1
        invoke  CreateToolhelp32Snapshot, TH32CS_SNAPPROCESS, NULL
        mov   hSnapshot, eax
        mov   ProcessInfo.dwSize, sizeof ProcessInfo
        invoke  Process32First, hSnapshot, ADDR ProcessInfo
        lea edi,ProcessInfo.szExeFile
        .while eax!=0
                invoke lstrcmpi,esi,edi
                .if eax==0
                        invoke OpenProcess, PROCESS_ALL_ACCESS,\
                                TRUE, ProcessInfo.th32ProcessID
```

```
                .if eax==NULL
                        invoke MessageBox,0,\
                                $CTA0("Не можу відкрити процес"),0,MB_OK
                        invoke CloseHandle,hSnapshot
                        ret
                .endif
                mov hProcess,eax
                invoke TerminateProcess,eax ,-1
                invoke Sleep,5000
                invoke GetExitCodeProcess, hProcess, ADDR ExitCode
                .if (eax!=0)&&(dword ptr ExitCode==STILL_ACTIVE)
                        invoke TerminateProcess, hProcess, -1
                        .if eax==0
                                invoke MessageBox,0,\
                                $CTA0("Не можу знищити процес"),0,MB_OK
                        .endif
                .endif
                invoke CloseHandle,hSnapshot
                invoke CloseHandle,hProcess
                ret
            .endif
        invoke  Process32Next, hSnapshot, ADDR ProcessInfo
        .endw
        invoke MessageBox,0,$CTA0("Процес не існує"),0,MB_OK
        invoke CloseHandle,hSnapshot
.else
        invoke MessageBox,0,$CTA0("Не задано ім’я образу процесу"),0,MB_OK
.endif
ret
Main endp
```

end start

1.4 Завдання до практичного заняття

Для визначення номера варіанта потрібно перевести номер залікової книжки або студентського квитка у двійкову систему числення, виділити три молодші біти ($b_2b_1b_0$) і скористатися таблицею 1.1.

Таблиця 0.1

Біти $b_2b_1b_0$	Розробити, налагодити і протестувати додаток Windows, який за умови, що процес не є системним (образ процесу не розташовується в системному каталозі):
000	знищує всі процеси з пріоритетами HIGH_PRIORITY_CLASS
001	знищує процес, що використовує найбільше віртуальної пам'яті
010	знищує процес із найбільшою кількістю потоків
011	знищує «найстаріший» процес у системі
100	знищує процес із найбільшим загальним часом роботи всіх потоків у режимі ядра
101	знищує процес із найбільшим загальним часом роботи всіх потоків у режимі користувача
110	знищує процес, що використовує найбільше ресурсів сторінкового файла
111	знищує процес із найбільшою кількістю відкритих дескрипторів

1.5 Завдання до самостійної роботи

Питання, які студент має опрацювати самостійно

1. Інтерфейси відлагодження Windows NT:
   а) призначення, відмінності й методика використання бібліотек psapi.dll, imagehlp.dll та dbghelp.dll;
   б) функції EnumDeviceDrivers, GetDeviceDriverFileName;
   в) функції EnumProcess, EnumProcessModules.

2. Робота з маркерами доступу і зміна рівня привілеїв процесу за допомогою функцій OpenProcessToken, LookUpPrivilegeValue, AdjustTokenPrivelege.

Література: [3, Глава 43, с. 894 – 918; 8, Глава 1, с. 54 – 75].

ПРАКТИЧНЕ ЗАНЯТТЯ № 2

Тема. Організація взаємодії між процесами за допомогою файлів, що відображені на пам'ять

Мета: набуття практичних навичок організації взаємодії поміж процесами за допомогою файлів, що відображені на пам'ять та синхронізації обміну даними між процесами за допомогою подій, м'ютексів та семафорів.

Короткі теоретичні відомості

1.1 Організація взаємодії поміж процесами

При розробці розподілених (або) клієнт-серверних додатків неминуче виникає необхідність організації обміну даними між різними процесами. Тривалий час така необхідність була потрібна для процесів працюючих на одному комп'ютері, а з поширенням мереж коло подібних задач істотно розширилося.

ОС Windows NT, як і переважна більшість сучасних операційних систем універсального призначення, працює в захищеному режимі роботи процесора. Зважаючи на те, що одним із основних принципів організації захищеного режиму є ізоляція адресних просторів процесів з метою захисту їх від взаємного впливу, організація обміну даними навіть між процесами на одному комп'ютері виглядає проблематично.

Усі потоки одного процесу працюють у єдиному адресному просторі надаваному їм цим процесом, область глобальних змінних поділювана всіма потоками і вони спільно використовують усі інші ресурси процесу. В цьому випадку питання про середовище для передачі даних навіть не виникає і залишаються тільки питання синхронізації доступу до поділюваних і спільно використовуваних ресурсів.

У випадку ж процесів, середовище для передачі даних повинна забезпечити операційна система. У загальному випадку організація взаємодії між процесами (InterProcess Communication, IPC) припускає надання операційною системою не тільки середовища для обміну даними, але й засобів синхронізації цього обміну. Для організації IPC у Windows NT можуть використовуватися різні механізми і протягом наступних практичних занять ми розглянемо лише декілька з них: відображені в пам'яті файли, анонімні канали, іменовані канали, сокети.

1.2 Організація IPC за допомогою об'єкта «проекція файла»

Практично всі механізми організації взаємодії між процесами, так чи інакше, базуються на спільному використанні об'єкта «розділ» (section object), що являє собою блок пам'яті, доступний двом і більше процесам, і є низькорівневим базовим механізмом спільного використання даних.

Об'єкт «розділ» використовується при створенні об'єкта «проекція файла» (file-mapping object). Win API функції CreateFileMapping та MapViewOfFile(Ex) базуються на Native API функціях, які використовуються ядром для створення розділу і для відображення або проектування об'єкта «розділ» у віртуальний адресний простір заданого процесу, і за своїми функціональними можливостями є по суті їх дещо вкороченими аналогами. Тому файли, відображені на пам'ять, дозволяють реалізувати IPC через поділювану пам'ять при максимальній швидкодії з мінімумом витрат.

Спільне використання даних за допомогою об'єкта «розділ» виконується за наступним сценарієм. Один процес створює відображуваний на пам’ять файл, викликаючи функцію CreateFileMapping (див. практичне заняття № 2 попередньої частини). Функція CreateFileMapping дозволяє, зокрема, створити відображення файла на пам'ять не пов'язане ні з яким дисковим файлом. Для цього як параметр ідентифікатора файла hFile у функцію потрібно передати значення 0FFFFFFFFh.

Об'єкт «розділ» є об'єктом ядра, і йому при створенні можна присвоїти ім'я. Таким чином, за допомогою CreateFileMapping можна створити іменований об'єкт ядра «розділ» і використати цей ресурс разом з іншими процесами.

Потім, процес викликом функції MapViewOfFile відображає подання об'єкта «розділ» на свій адресний простір. Інший процес може відкрити цей самий відображуваний файл за допомогою функції

```
HANDLE OpenFileMapping(
  DWORD   dwDesiredAccess, // режим доступу
  BOOL    bInheritHandle,  // прапор спадкування
  LPCTSTR lpName);         // адреса імені відображення файла
```

Через параметр lpName цієї функції потрібно передати ім'я відображення, що відкривається. Ім'я повинне бути задане точно так, як при створенні відображення функцією CreateFileMapping.

Параметр dwDesiredAccess обумовлює необхідний режим доступу до відображення й указується точно також, як і для функції MapViewOfFile.

Параметр bInheritHandle визначає можливість спадкування ідентифікатора відображення. Якщо він дорівнює TRUE, породжені процеси можуть успадковувати ідентифікатор, якщо FALSE – то ні.

У випадку помилки OpenFileMapping повертає нульове значення, а у випадку успіху – ідентифікатор відображення.

Відкривши відображення і маючи його ідентифікатор, інший процес може відобразити його викликом функції MapViewOfFile, але вже до свого адресного простору. У результаті ті самі фізичні сторінки пам'яті стають доступними в обох процесах, що дозволяє їм легко передавати через цю область великі порції даних, тому що будь-які зміни на цих сторінках в одному процесі відбиваються на їх подання в іншому процесі.

При організації IPC за допомогою об'єкта «проекція файла» операційна система надає лише середовище для передавання даних, але при цьому не передбачає ніяких спеціальних засобів синхронізації, вбудованих в об’єкт

«проекція файла». Хоча будь-які зміни на сторінках пам'яті в одному процесі відбиваються на їх подання в іншому процесі, але цей інший процес не має змогу це автоматично «відчути».

1.3 Синхронізація IPC

Зважаючи на те, що одиницею планування в операційній системі є потік і що саме потоки містять і виконують код, а процеси лише надають для цього ресурси, задачі синхронізації взаємодії поміж процесами все одно зводяться, урешті-решт, до задач синхронізації потоків.

Можна виділити два класи задач синхронізації потоків. Перший клас задач пов'язаний із синхронізацією декількох потоків у рамках одного процесу, а другий клас – із синхронізацією декількох потоків приналежних різним процесам, і саме в цьому випадку можна говорити вже про синхронізацію IPC.

Хоча, по суті, і в першому й у другому випадку справа зводиться до синхронізації потоків, але набір застосовуваних для цього методів трохи різний. Деякі з методів синхронізації можуть застосовуватися винятково до потоків, що працюють у рамках одного процесу, інші орієнтовані на синхронізацію потоків різних процесів, хоча й можуть застосовуватися в рамках одного процесу.

У найпростішому випадку задачу синхронізації можна сформулювати як забезпечення послідовного (тобто неодночасного) доступу різних потоків до спільно використовуваних ресурсів. У загальному випадку синхронізація потоків полягає в тому, що деякі з них повинні припиняти своє виконання доти, поки не відбудуться ті чи інші події, пов'язані з іншими потоками.

У випадку організації IPC за допомогою об'єкта «проекція файла» такою подією можна вважати будь-які зміни на сторінках пам'яті виконані одним процесом, які відбиваються на їх подання в іншому процесі, і які цей інший процес має дочекатися перед тим як починати будь-які дії щодо їх обробки.

1.3.1 Об’єкти синхронізації ядра і Wait-функції

Певні об'єкти ядра операційної системи Microsoft Windows NT можуть перебувати у двох станах – сигнальному і несигнальному. До таких об'єктів ядра, яким притаманна властивість перебувати у сигнальному або несигнальному станах, належать процеси, потоки, завдання, файли, консольне введення, повідомлення про зміну файлів, події, семафори, м’ютекси, таймери очікування. Чотири останні з перерахованих об’єктів безпосередньо призначаються для розв’язання задач синхронізації і вони отримали назву примітивів синхронізації ядра.

Характерною рисою багатьох об’єктів ядра є те, що вони можуть мати імена. Наявність імен у всіх примітивів синхронізації ядра, робить можливим їх використання для синхронізації потоків, що належать різним процесам. Імена об'єктів ядра відображаються у так званий простір імен такого компонента ОС як Диспетчер Об’єктів (Object Manager). Простір імен Диспетчера Об'єктів глобальний і іменовані об'єкти служать для надання системних ресурсів у спільне використання.

З будь-якого потоку будь-якого процесу знаючи ім'я об'єкта, його тип і маючи відповідні права доступу можна відкрити об'єкт й одержати його ідентифікатор для подальшого використання. Перехід з одного стану в інший здійснюється за правилами, визначеним в операційній системі для кожного з об'єктів ядра. Знаючи ці правила можна відповідним чином будувати алгоритми синхронізації потоків.

Раніше ми говорили про те, що в загальному випадку завдання синхронізації потоків полягає в тому, що деякі з них повинні припиняти своє виконання доти, поки не відбудуться ті чи інші події, пов'язані з іншими потоками. У якості таких подій можуть розглядатися переходи певних об'єктів ядра з несигнального стану в сигнальний.

Операційна система надає спеціальний набір API – так звані Wait-функції, які дозволяють потоку в будь-який момент призупинитися й чекати переходу з несигнального в сигнальний стан певного об'єкта ядра. Потік, що викликав

одну із Wait-функцій і перейшов у результаті цього в стан очікування не одержує ні кванта процесорного часу – він виключається на певний час із черги планування.

Функція DWORD WaitForSingleObject(HANDLE hHandle, DWORD dwMilliseconds) припиняє потік і переводить його в режим очікування доти, поки об'єкт ядра з ідентифікатором hHandle не перейде з несигнального в сигнальний стан або не закінчиться час очікування в dwMilliseconds мілісекунд. Якщо параметр dwMilliseconds заданий константою INFINITE, потік буде перебувати в стані очікування доти, поки об'єкт ядра з ідентифікатором hHandle не перейде з несигнального в сигнальний стан, тобто невизначений час.

У випадку помилки функція повертає значення WAIT_FAILED. Якщо ж функція завершилася успішно, вона може повернути одне з наступних трьох значень: WAIT_OBJECT_0, WAIT_TIMEOUT або WAIT_ABANDONED.

Якщо об'єкт ядра з ідентифікатором hHandle, для якого виконувалося очікування, перейшов у сигнальний стан функція повертає значення WAIT_OBJECT_0. Якщо час очікування, заданий у параметрі dwMilliseconds минув, але об'єкт так і не перейшов у сигнальний стан, повертається значення WAIT_TIMEOUT. При нескінченному очікуванні з dwMilliseconds заданим константою INFINITE цей код завершення не повернеться ніколи. Код завершення WAIT_ABANDONED повертається для об'єкта синхронізації Mutex (розглядається далі), що не був звільнений потоком, що завершив свою роботу.

Ще більш гнучкі можливості надає функція

```
DWORD WaitForMultipleObjects(
  DWORD nCount,              // кількість ідентифікаторів у масиві
  const HANDLE* lpHandles,   // адреса масиву ідентифікаторів
  BOOL bWaitAll,             // тип очікування
  DWORD dwMilliseconds);     // час очікування в мілісекундах або INFINITE
```

Функція припиняє потік і переводить його в режим очікування доти, поки хоча б один або всі об'єкти ядра з ідентифікаторами в масиві за адресою

lpHandles не перейдуть із несигнального в сигнальний стан або не закінчиться час очікування в dwMilliseconds мілісекунд.

Параметр nCount задає кількість ідентифікаторів (різних) об'єктів ядра в масиві за адресою lpHandles.

Якщо параметр bWaitAll дорівнює TRUE, потік переводиться в стан очікування доти, поки всі об'єкти, ідентифікатори яких зберігаються в масиві lpHandles, не перейдуть у сигнальний стан. У тому випадку, коли значення параметра bWaitAll дорівнює FALSE, очікування припиняється, коли один з об'єктів, ідентифікатори яких зберігаються в масиві lpHandles, не перейде в сигнальний стан.

Якщо параметр dwMilliseconds заданий константою INFINITE, потік буде перебувати у стані очікування відповідно до того, як визначено параметром bWaitAll.

Функція WaitForMultipleObjects може повернути одне з наступних значень:

- WAIT_FAILED при помилці;
- WAIT_TIMEOUT якщо час очікування, заданий у параметрі dwMilliseconds минув, але об'єкти так і не перейшли в сигнальний стан відповідно до того, як визначено параметром bWaitAll;
- значення в діапазоні від WAIT_OBJECT_0 до (WAIT_OBJECT_0 + nCount - 1), що, залежно від вмісту параметра bWaitAll, або означає, що всі очікувані ідентифікатори перейшли у сигнальний стан (якщо bWaitAll був дорівнює TRUE), або це значення, якщо з нього відняти константу WAIT_OBJECT_0, дорівнює індексу ідентифікатора об'єкта в масиві ідентифікаторів lpHandles що перейшов у сигнальний стан;
- значення в діапазоні від WAIT_ABANDONED_0 до (WAIT_ABANDONED_0 + nCount - 1), якщо очікування для об'єктів Mutex було скасовано. Індекс відповідного об'єкта в масиві lpHandles можна визначити, якщо відняти з коду повернення значення WAIT_ABANDONED_0.

Wait-функції працюють з ідентифікаторами будь-яких об’єктів ядра для яких характерна наявність сигнального і несигнального стану. Тип об'єкта синхронізації має значення при його створенні й відкритті, бо для відкриття різних об'єктів ядра в загальному випадку використовуються різні API-функції, хоча при цьому для знищення більшості об’єктів використовується єдина функція CloseHandle.

Для всіх об'єктів ядра характерна наявність у їх структурі лічильника числа користувачів об’єкта, і функція CloseHandle нічого не знищує, а просто виконує зменшення лічильника користувачів об'єкта, ідентифікатор якого вона отримує як параметр. Коли якийсь процес завершується, лічильники всіх використовуваних їм об'єктів ядра також автоматично зменшуються на 1. Як тільки лічильник якого-небудь об'єкта приймає значення нуль, ядро знищує цей об'єкт.

Об'єкти синхронізації можуть спільно використовуватися різними процесами, і фактично вони будуть знищені тільки тоді, коли стануть нікому не потрібними, тобто або процеси, що їх використовують, завершаться, або відмовляться від їхніх послуг викликом функції CloseHandle.

1.3.2 Синхронізація з використанням подій

Операційна система Microsoft Windows NT дозволяє створювати об'єкти синхронізації, які називаються подіями (event object). Ці об'єкти можуть перебувати в сигнальному або несигнальному стані, причому установка стану виконується викликом відповідної функції.

Важливою можливістю, забезпечуваною об'єктами подій, є те, що перехід у сигнальний стан єдиного об'єкта події здатен вивести зі стану очікування одночасно декілька потоків.

Об'єкти події діляться на такі, що скидаються вручну, й на такі, що скидаються автоматично, і це їхня властивість встановлюється при їхньому створенні.

Події, що скидаються вручну можуть сигналізувати одночасно всім потокам, що очікують настання цієї події, і переводяться в несигнальний стан програмно, викликом спеціальної функції.

Події, що скидаються автоматично переводяться в несигнальний стан самостійно після звільнення одного з потоків, що очікують цю подію, тоді як інші потоки, продовжують очікувати переходу події в сигнальний стан.

Схема використання подій досить проста. Один із потоків створює об'єкт-подія. У процесі створення або пізніше цей потік встановлює подію у вихідний стан – сигнальний або не сигнальний. Викликаючи функції WaitForSingleObject або WaitForMultipleObjects, потік може виконувати очікування моменту, коли подія перейде в сигнальний стан.

1.3.2.1 Ініціалізація подій

Для створення події використовується функція CreateEvent, прототип якої наведений нижче:

```
HANDLE CreateEvent(
  LPSECURITY_ATTRIBUTES lpEventAttributes, // атрибути захисту
  BOOL    bManualReset,        // прапор ручного скидання події
  BOOL    bInitialState,       // прапор початкового стану події
  LPCTSTR lpName);             // адреса імені об'єкта-події
```

Параметр lpEventAttributes задає атрибути захисту й у більшості випадків може бути зазначений як NULL.

За допомогою параметра bManualReset можна вибрати один із двох режимів роботи об'єкта-події: ручний або автоматичний. Якщо значення цього параметра дорівнює TRUE, подію потрібно скидати вручну за допомогою функції ResetEvent. Якщо ж для параметра bManualReset указати значення FALSE, подію буде скинуто (тобто переведено в несигнальний стан) автоматично відразу після того як потік завершить очікування цієї події.

Параметр bInitialState визначає початковий стан події. Якщо цей параметр дорівнює TRUE, об'єкт-подія створюється в сигнальному стані, а якщо FALSE – у несигнальному.

Для того щоб подією могли користуватися потоки, створені різними процесами, необхідно за допомогою параметра lpName задати ім'я об'єкта-події. У тому випадку, коли подія використовується потоками, що працюють у рамках одного процесу, ім'я події можна не задавати, указавши параметр lpName як NULL.

У випадку успішного завершення функція CreateEvent повертає ідентифікатор події, яким потрібно буде користуватися при виконанні всіх операцій над об'єктом-подією. При помилці повертається значення NULL.

Якщо подія використовується потоками, створеними тільки в рамках одного процесу, її не потрібно відкривати. Як параметр функціям, що змінюють стан об'єкта-події, можна передавати ідентифікатор об'єкта-події, отриманий при його створенні від функції CreateEvent.

Якщо ж подія використовується для синхронізації потоків, що належать різним процесам, при створенні події необхідно задати її ім'я. Потік, належить іншому процесу, повинен відкрити об'єкт-подію за допомогою функції OpenEvent, передавши їй ім'я цього об'єкта.

```
HANDLE OpenEvent(
  DWORD   fdwAccess,          // прапори доступу
  BOOL    fInherit,           // прапор спадкування
  LPCTSTR lpszEventName);  // адреса імені об'єкта-події
```

Прапори доступу, передані через параметр fdwAccess, визначають необхідний рівень доступу до об'єкта-події. Цей параметр може бути комбінацією наступних значень:

- EVENT_ALL_ACCESS – зазначені всі можливі прапори доступу;
- EVENT_MODIFY_STATE – отриманий ідентифікатор можна буде використати для функцій SetEvent і ResetEvent (див. далі);

- SYNCHRONIZE – отриманий ідентифікатор можна буде використати в будь-яких функціях очікування події.

Параметр fInherit визначає можливість спадкування отриманого ідентифікатора. Якщо цей параметр дорівнює TRUE, ідентифікатор може успадковуватися дочірніми процесами. Якщо ж він дорівнює FALSE, спадкування не допускається.

Через параметр lpszEventName передається адреса символьного рядка, що містить ім'я об'єкта-події.

За допомогою функції OpenEvent кілька потоків можуть відкрити один і той самий об'єкт-подію і потім виконувати одночасне очікування для цього об'єкта.

1.3.2.2 Управління станом подій

Потік може встановити подію в сигнальний стан, використовуючи функцію BOOL SetEvent(HANDLE hEvent). Як єдиний параметр цієї функції необхідно передати ідентифікатор об'єкта-події, отриманий від функції CreateEvent або OpenEvent. При успішному завершенні повертається значення TRUE, при помилці – FALSE.

Якщо подія є такою, що скидається автоматично, то вона автоматично повертається в несигнальний стан уже після звільнення тільки одного з потоків, що очікують. За відсутності потоків, що очікують настання цієї події, вона залишається в сигнальному стані доти, поки такий потік не з'явиться, після чого цей потік відразу ж звільняється.

З іншого боку, якщо подія є такою, що скидається вручну, то вона залишається в сигнальному стані доти, поки який-небудь потік не викличе функцію скидання події. У цей час усі потоки, що очікують, звільняються, але до виконання такого скидання події, інші потоки можуть, як переходити в стан її очікування, так і звільнятися.

Скидання події (тобто установка її в несигнальний стан) виконується функцією BOOL ResetEvent(HANDLE hEvent). Якщо потік створив подію, що

працює в автоматичному режимі, вона буде скидатися й без допомоги цієї функції, якщо тільки який-небудь потік виконував очікування цієї події й подія відбулася.

1.3.3 Синхронізація з використанням м’ютексів

У випадку організації IPC за допомогою об'єкта «проекція файла» використання подій і Wait-функцій дозволяє зафіксувати моменти, в які зміни на сторінках пам'яті виконані одним процесом, відбиваються на їх подання в іншому процесі, але не вирішує ще однієї проблеми такого обміну.

Windows NT підтримує так звану симетричну мультипроцесорну обробку (англ. Symmetric Multiprocessing, або SMP) у якій два або більше однакові процесори підключаються до загальної пам'яті. Система планування Windows NT за умови наявності двох або більше процесорів уміє балансувати навантаження на процесори і навіть дозволяє закріпити виконання коду певного потоку на визначеному процесорі.

За таких обставин можливе виникнення ситуації, коли два або більше потоків, що паралельно виконуються на двох або більше процесорах можуть одночасно звертатися до одного і того самого ресурсу розміщеного в пам’яті. З урахуванням можливості витіснення будь-якого потоку на будь якому процесорі у будь який час, наслідки такого звертання до ресурсу стають просто непередбачуваними.

Для розв’язання вище зазначеної і подібних до неї проблем в операційній системі передбачена наявність відповідних засобів синхронізації, одним із яких є м’ютекси (Mutex), які одержали свою назву від вираження «mutually exclusive», що означає «взаємно виключний» і, відповідно до своєї назви, використовуються для забезпечення послідовного (неодночасного) доступу до спільно використовуваних ресурсів.

М’ютекс є об'єктом (примітивом синхронізації) ядра і структура, що забезпечує функціонування об'єкта ядра «м’ютекс» розміщується в пам'яті ядра. У цій структурі міститься лічильник числа користувачів об'єкта «м’ютекс»,

ідентифікатор потоку і лічильник рекурсії. Ідентифікатор потоку визначає, який потік захопив м’ютекс, а лічильник рекурсії – скільки разів.

Якщо ідентифікатор потоку в м’ютексі дорівнює 0 (у самого потоку не може бути такого ідентифікатора), м’ютекс не захоплений жодним з потоків і перебуває у вільному (сигнальному) стані. Якщо ідентифікатор потоку в м’ютексі не дорівнює 0, то м’ютекс захоплений одним із потоків і перебуває в зайнятому (несигнальному) стані.

Організація послідовного доступу до ресурсів з використанням об'єктів Mutex можлива тому, що в кожен момент тільки один потік може володіти цим об'єктом.

1.3.3.1 Ініціалізація м’ютекса

Створення об'єкта ядра Mutex виконується за допомогою функції

```
HANDLE CreateMutex(
  LPSECURITY_ATTRIBUTES lpMutexAttributes, // атрибути захисту
  BOOL bInitialOwner,          // початковий стан
  LPCTSTR lpName);             // ім'я об'єкта Mutex
```

lpMutexAttributes – адреса екземпляра структури типу SECURITY_ATTRIBUTES, що визначає атрибути захисту об'єкта.

Якщо bInitialOwner має значення TRUE, потік, що створює об'єкт Mutex, стає його власником відразу після створення. Якщо ж значення цього параметра дорівнює FALSE, після створення об'єкт Mutex не буде належати жодному потоку й буде вільний поки його явно не захопить який не будь потік.

lpName – покажчик на ім'я об'єкта Mutex. Як значення параметра lpName функції CreateMutex можна вказати й значення NULL, але не іменований м’ютекс може бути використаний тільки для синхронізації потоків у рамках одного процесу.

Функція CreateMutex повертає ідентифікатор створеного об'єкта Mutex або NULL при помилці.

Процес, що створив м’ютекс, уже має його ідентифікатор, а потоки інших процесів для відкриття об'єкта Mutex повинні скористатися функцією

```
HANDLE OpenMutex(
  DWORD dwDesiredAccess,   // необхідний доступ
  BOOL bInheritHandle,     // прапор спадкування
  LPCTSTR lpName);         // адреса імені об'єкта Mutex
```

dwDesiredAccess – прапори, що визначають необхідний рівень доступу до об'єкта Mutex. Якщо при створенні об'єкта задавалися параметри безпеки за замовчанням, то для всіх об'єктів синхронізації потоків приналежним різним процесам стандартними правами доступу є DELETE, READ_CONTROL, SYNCHRONIZE, WRITE_DAC, WRITE_OWNER.

Для об'єкта Mutex додатковими прапорами є:

- MUTEX_MODIFY_STATE – отриманий ідентифікатор можна буде використати у функції ReleaseMutex (див. далі) для відмови від володіння об'єктом Mutex;
- MUTEX_ALL_ACCESS – усі можливі прапори доступу.

Параметр bInheritHandle визначає можливість спадкування отриманого ідентифікатора. Якщо цей параметр дорівнює TRUE, ідентифікатор може успадковуватися дочірніми процесами, якщо ж він дорівнює FALSE, спадкування не допускається.

1.3.3.2 Захоплення і звільнення м’ютекса

Об'єкт Mutex, що не має власника, перебуває в сигнальному стані. Якщо потік не захопив об'єкт Mutex при його створенні, то знаючи його ідентифікатор, отриманий від функції CreateMutex, потік може пізніше заволодіти об'єктом Mutex за допомогою однієї з Wait-функцій.

Знаючи ідентифікатор об'єкта Mutex, отриманий від функцій OpenMutex потік також може заволодіти об'єктом за допомогою однієї з Wait-функцій. Коли який-небудь потік, що належить будь-якому процесу, стає власником об'єкта Mutex, останній переходить у несигнальний стан.

Для відмови потоком від володіння об'єктом Mutex з ідентифікатором hMutex використовується функція BOOL ReleaseMutex(HANDLE hMutex). Функція повертає нульове значення у випадку успіху, і ненульове – у випадку невдачі. Причиною невдачі функції може бути її виклик потоком, що не володіє об'єктом Mutex. Відмова від володіння об'єктом Mutex переводить його в сигнальний стан.

1.4 Приклади програмування

Приклади програмування цього і наступних практичних занять демонструють організацію IPC, і, внаслідок цього, будуть складатися з вихідних кодів як мінімум двох додатків – клієнта і сервера. Поняття сервер і клієнт використовуються в контексті програмної концепції «клієнт-сервер», що визначає закріплені за додатками ролі.

Організацію взаємодії між процесами з використанням файлів, що відображені на пам'ять продемонструємо на прикладі двох консольних додатків MMFServer.exe (сервер) і MMFClient.exe (клієнт).

Сервер створює два примітиви синхронізації «подія» й іменоване відображення файла на пам'ять, не пов'язане ні з яким файлом, тобто просто створює іменований об’єкт «секція». Після цього сервер створює дочірній процес для клієнта.

Клієнт відкриває іменоване відображення і об’єкти «подія» й після цього використовує файл, відображений на пам'ять для передачі серверу через нього повідомлень, що вводяться користувачем із клавіатури. Сервер читає з відображення файла повідомлення й виводить їх у своєму консольному вікні, використовуючи функцію WaitForMultipleObjects й примітиви синхронізації «подія» для упорядкування процесу обміну й завершення роботи додатків.

```
; Файл interface.inc
include windows.inc
include user32.inc
include kernel32.inc
```

```
include masm32.inc
include C:\MASM32\MACROS\strings.mac
includelib  user32.lib
includelib  kernel32.lib
includelib  masm32.lib
Main      PROTO
.data
szEventDataName         db "$EventDataName",0
szEventCancelName       db "$EventCancelName",0
szMMFName               db "$InterprocessCommunicationMMF$",0
.data?
hEventData          HANDLE ?
hEventCancel        HANDLE ?
hFileMap            HANDLE ?
hFileMem            HANDLE ?
; Файл MMFServer.inc
include ..\interface.inc
.data?
ProcessInfo             PROCESS_INFORMATION <>
StartInfo               STARTUPINFO <>
; Файл MMFServer.asm
.386
.model flat, stdcall
option casemap :none
include MMFServer.inc
.code
start:  invoke Main
        invoke UnmapViewOfFile, hFileMem
        invoke CloseHandle, hFileMap
        invoke CloseHandle, hEventData
```

```
        invoke CloseHandle, hEventCancel
        invoke CloseHandle,ProcessInfo.hProcess
        invoke ExitProcess,0
Main proc
invoke CreateEvent, NULL, FALSE, FALSE,ADDR szEventDataName
.if eax==NULL
        invoke MessageBox,0,\
                $CTA0("Не вдалося створити подію"), ADDR szEventDataName,\
                MB_OK OR MB_ICONERROR OR MB_APPLMODAL
        ret
.endif
mov hEventData,eax
invoke CreateEvent, NULL, FALSE, FALSE, ADDR szEventCancelName
.if eax==NULL
        invoke MessageBox,0,\
                $CTA0("Не вдалося створити подію "), ADDR szEventCancelName,\
                MB_OK OR MB_ICONERROR OR MB_APPLMODAL
        ret
.endif
mov hEventCancel,eax
invoke CreateFileMapping, 0FFFFFFFFh, NULL, PAGE_READWRITE,\
                0, 4096, ADDR szMMFName
.if eax==INVALID_HANDLE_VALUE
        invoke MessageBox,0,\
                $CTA0("Не вдалося створити відображення файла на пам’ять"),\
                0, MB_OK OR MB_ICONERROR OR MB_APPLMODAL
        ret
.endif
mov hFileMap, eax
invoke MapViewOfFile, eax, FILE_MAP_READ OR FILE_MAP_WRITE, 0, 0, 0
```

```
.if eax==NULL
        invoke MessageBox,0,\
                $CTA0("Не вдалося відобразити файл на пам’ять"),\
                0, MB_OK OR MB_ICONERROR OR MB_APPLMODAL
        ret
.endif
mov hFileMem,eax
invoke GetStartupInfo, ADDR StartInfo
mov StartInfo.dwFillAttribute, 1Fh
mov StartInfo.dwX, 0
mov StartInfo.dwY, 0
mov StartInfo.dwXSize, 800
mov StartInfo.dwYSize, 300
mov StartInfo.dwFlags, STARTF_USEPOSITION OR STARTF_USESIZE OR
STARTF_USEFILLATTRIBUTE
invoke CreateProcess, NULL,  $CTA0("MMFClient.exe"), NULL, NULL, FALSE,\
        CREATE_NEW_CONSOLE, NULL, NULL,\
        ADDR StartInfo, ADDR ProcessInfo
.if eax==NULL
        invoke MessageBox, NULL,\
                $CTA0("Не вдалося створити процес"), \
                0, MB_OK OR MB_ICONERROR
        ret
.endif
.while (TRUE)
        invoke WaitForMultipleObjects, 2, ADDR hEventData, FALSE, INFINITE
        sub eax,WAIT_OBJECT_0
        .break .if (eax==1)
        invoke StdOut, hFileMem
.endw
```

```
ret
Main endp
end start
; Файл MMFClient.inc
include ..\interface.inc
; Файл MMFClient.asm
.386
.model flat, stdcall
option casemap :none
include MMFClient.inc
.code
start:  invoke Main
        invoke ExitProcess,0
Main proc
LOCAL       Buffer[4096]:BYTE
invoke OpenEvent, EVENT_ALL_ACCESS, FALSE, ADDR szEventDataName
.if eax==NULL
        invoke MessageBox,0,\
                $CTA0("Не вдалося відкрити подію"),\
                ADDR szEventDataName,\
                MB_OK OR MB_ICONERROR OR MB_APPLMODAL
        ret
.endif
mov hEventData,eax
invoke OpenEvent,EVENT_ALL_ACCESS, FALSE, ADDR szEventCancelName
.if eax==NULL
        invoke MessageBox,0,\
                $CTA0("Не вдалося відкрити подію"),\
                ADDR szEventCancelName,\
                MB_OK OR MB_ICONERROR OR MB_APPLMODAL
```

```
            invoke CloseHandle,hEventData
        ret
.endif
mov hEventCancel,eax
invoke OpenFileMapping,FILE_MAP_READ OR FILE_MAP_WRITE,\
        FALSE,ADDR szMMFName
.if eax!=NULL
        mov hFileMap,eax
        invoke MapViewOfFile, eax, FILE_MAP_READ OR FILE_MAP_WRITE,\
        0, 0, 0
        .if eax!=NULL
                mov hFileMem,eax
                invoke StdOut, $CTA0("Press Enter to exit...\n\n")
                .while (TRUE)
                        invoke StdIn, ADDR Buffer,LENGTHOF Buffer
                        lea esi,Buffer
                        mov edi,hFileMem
                        .break .if Buffer==13
                        .repeat
                                lodsb
                                stosb
                        .until al==10
                        xor al,al
                        stosb
                        invoke SetEvent,hEventData
                .endw
                invoke SetEvent,hEventCancel
                invoke CloseHandle, hFileMem
        .else
                invoke MessageBox,0,\
```

```
                $CTA0("Не вдалося відобразити файл на пам’ять"),\
                0,MB_OK OR MB_ICONERROR OR MB_APPLMODAL
        .endif
        invoke UnmapViewOfFile, hFileMap
.else
        invoke MessageBox,0,\
                $CTA0("Не вдалося відкрити відображення файла"),\
                ADDR szMMFName,\
                MB_OK OR MB_ICONERROR OR MB_APPLMODAL
        invoke UnmapViewOfFile, hFileMap
.endif
invoke CloseHandle,hEventData
invoke CloseHandle,hEventCancel
ret
Main endp
end start
```

1.5 Завдання до практичного заняття

Для визначення номера варіанта потрібно перевести номер залікової книжки або студентського квитка у двійкову систему числення, виділити два молодші біти (b_1b_0) і скористатися таблицею 2.1.

Розробити три додатки операційної системи Windows NT, що демонструють організацію обміну даними поміж процесами. Перший додаток (клієнт) повинен створити об’єкт «проекція файла» і процеси для двох інших додатків (серверів). Після старту сервери повинні виконати завдання клієнта згідно з номером варіанта, наведеному в таблиці 2.1, передати результати виконання завдання клієнту з використанням об’єкта «проекція файла» і завершити роботу. Отримавши від серверів результати виконання завдання, клієнтський додаток повинен їх обробити та вивести на екран.

Таблиця 0.1

Біти b_1b_0	Завдання для 1-го та 2-го серверів
00	1-й: Надати вміст зазначеного каталогу 2-й: Повернути інформацію про зазначений процес та його модулі
01	1-й: Надати мережеве ім’я комп’ютера, ім’я користувача, що в даний час зареєструвався в системі та час, що пройшов з моменту останнього завантаження операційної системи. 2-й: Надати перелік усіх доступних мережевих ресурсів.
10	1-й: Надати інформацію про кількість, імена, типи, розмір встановлених логічних дискових пристроїв. 2-й: Надати мітку, серійний номер і тип файлових систем кожного з логічних дисків.
11	1-й: Надати інформацію про загальний і доступний розмір віртуальної та фізичної пам’яті. 2-й: Надати інформацію про загальний розмір зазначеного логічного диску та наявний розмір вільного місця.

1.6 Завдання до самостійної роботи

Питання, які студент має опрацювати самостійно

1. Механізм APC і керування APC-чергами.
2. Безпечне завершення потоку з використанням APC.
3. Завершення асинхронних запитів введення/виведення:
   а) сигналізація об’єкта ядра, що керує пристроєм;
   б) сигналізація об’єкта ядра, що керує повідомленнями;
   в) сповіщальне введення/виведення;
   г) порти завершення введення/виведення.

Література: [1, Глава 3, с. 158 – 168; 3, Глава 26, с. 485 – 498, Глава 28, с. 536 – 539; 4, Глава 2, с. 29 – 40; 7, Глава 2, с. 360 – 355].

ПРАКТИЧНЕ ЗАНЯТТЯ № 3

Тема. Організація взаємодії між процесами за допомогою іменованих каналів

Мета: набуття практичних навичок організації взаємодії між процесами за допомогою іменованих каналів та синхронізації обміну даними між процесами за допомогою м’ютексів та семафорів.

Короткі теоретичні відомості

1.7 Канали передачі даних

Канали (pipes) – це найпростіший надаваний операційною системою механізм реалізації IPC, підтримуваний не тільки Microsoft Windows NT, але й Unix-подібними ОС. Цей засіб дозволяє організувати передачу даних, як між локальними процесами, так і між процесами, запущеними на різних робочих станціях у мережі.

Для каналів використовується характерна для UNIX файлова абстракція, що дозволяє працювати з ними як з файлами за допомогою основного API вводу/виводу, й тому вони досить прості у використанні.

Через канал можна передавати дані тільки між двома процесами. Один із процесів створює канал, а інший відкриває його за допомогою CreateFile і після цього обидва процеси можуть передавати дані через канал в одну або обидві сторони, використовуючи для цього функції, призначені для роботи з файлами, такі як ReadFile й WriteFile. Додатки можуть виконувати над каналами як синхронні, так й асинхронні операції, аналогічно тому, як це можна робити з файлами.

1.8 Режими передачі даних

Протокол – набір правил, що дозволяє здійснювати з'єднання й обмін даними між його учасниками. Протокол описує структуру переданих даних (синтаксис повідомлення), способи передачі даних, операції керування передачею й контролю її стану, методи обробки помилок.

Протокол називають орієнтованим на передачу повідомлень, якщо для кожної команди запису він передає байти блоком в окремому дискретному повідомленні. Це означає, що приймач одержить дані у вигляді окремого повідомлення відправника зі збереженням границь повідомлень, при якому кожне повідомлення прочитується повністю.

Протокол, що не зберігає межі повідомлень, зазвичай називають протоколом, заснованим на потоці (stream-based). Потоковий відправник безупинно передає дані, а одержувач зчитує стільки даних, скільки є в наявності, незалежно від границь повідомлень. Це означає, що відправник і одержувач точно не знають, скільки байтів зчитується або записується у певний момент часу. Таким чином, запис однієї кількості байтів відправником не означає читання тієї ж кількості з іншої сторони одержувачем.

Надійність, або гарантована доставка, має на увазі, що кожен байт даних буде доставлений від відправника зазначеному одержувачеві без змін. Ненадійний протокол не гарантує ні доставку кожного байта, ні цілісність даних. Протокол, що зберігає порядок даних, гарантує, що одержувач прийме ці дані в тому самому порядку, у якому вони були відправлені. Відповідно, протокол, що не зберігає порядок байтів, не дає такої гарантії.

У більшості випадків надійність і порядок доставки нерозривно пов'язані з тим, орієнтований протокол на з'єднання чи ні. Орієнтовані на з'єднання протоколи перед будь-яким обміном даними встановлюють канал зв'язку між двома сторонами, що беруть участь в обміні. Це гарантує існування маршруту між двома сторонами й те, що обидві сторони будуть коректно обмінюватися інформацією.

Установлення каналу зв'язку між двома учасниками тягне додаткові витрати. Гарантована доставка даних ще більше збільшує витрати, викликані додатковими обчисленнями для перевірки правильності передачі даних. З іншого боку, протокол, не орієнтований на передачу даних, не гарантує, що приймач фактично приймає дані, як правило, не орієнтований на з'єднання, і, як наслідок, працює швидше.

Передача по каналу зв'язку може виконуватися в дуплексному, напівдуплексному й симплексному режимі.

Дуплексний (двосторонній) режим дозволяє по одному каналу зв'язку одночасно передавати інформацію в обох напрямках. Він може бути асиметричним, якщо пропускна здатність в одному з напрямків істотно розрізняється із пропускною здатністю в протилежному, або симетричним, якщо ці значення приблизно однакові.

Напівдуплексний режим дозволяє передавати інформацію в прямому й зворотному напрямку, але не одночасно, а по черзі.

Симплексний (однобічний) режим передбачає тільки один напрямок передачі інформації, а в зустрічному напрямку можуть передаватися тільки допоміжні службові дані.

1.9 Іменовані канали

Іменовані канали (named pipes) забезпечують надійну однобічну й двобічну передачу даних між процесами на одній або різних робочих станціях у локальній мережі. Іменовані канали використовують два режими передачі даних – побайтовий (потоковий) і повідомлень.

Розробка додатків, що працюють з іменованими каналами, не представляє складності й не вимагає особливих знань механізму роботи основних мережних протоколів, таких як TCP/IP, UDP/IP, NetBIOS або IPX. Деталі роботи протоколів приховані від додатка, тому що для обміну даними між процесами через мережу іменовані канали використовують провайдер мережі Microsoft (Microsoft Network Provider, MSNP).

Дуже важливо й те, що іменовані канали дозволяють скористатися вбудованими можливостями захисту Windows NT. Іменовані канали пропонують ряд можливостей, які роблять їх корисними як універсальний механізм реалізації додатків на основі IPC, включаючи додатки, що вимагають мережного доступу до файлів і клієнт-серверні системи. До числа згаданих можливостей належать наступні:

1. Іменовані канали підтримують обмін повідомленнями, тому процес, що виконує читання, може зчитувати повідомлення змінної довжини саме в тому вигляді, у якому вони були послані процесом, що виконує запис;

2. Іменовані канали є двонаправленими, що дозволяє здійснювати обмін повідомленнями між двома процесами за допомогою єдиного каналу;

3. Взаємодія за допомогою іменованого каналу здійснюється однаковим чином як для процесів, що виконуються на одній і тій самій, так і на різних машинах мережі;

4. Є набір допоміжних функцій, що спрощують обслуговування взаємодії «запит/відповідь» і клієнт-серверних з'єднань.

1.9.1 Імена каналів

Назви іменованих каналів повинні задовольняти формату Universal Naming Convention (UNC). Імена UNC – це стандартний спосіб доступу до файлів і пристроїв без призначення цим об'єктам букви локального диска, спроектованого на віддалену файлову систему. Це дозволяє додаткам не залежати від імен дисків і прозоро працювати з мережею. Зокрема, не треба турбуватися, що не вистачить букв для загальних ресурсів, які підключають.

Імена UNC мають вигляд \\сервер\ресурс\[шлях]\ім'я. Перша частина – \\сервер, починається із двох зворотних косих рисок і імені віддаленого сервера, на якому перебуває потрібний ресурс. Друга – \ресурс – це ім'я загального ресурсу, наприклад, папки у файловій системі, до якої відкритий загальний доступ користувачам мережі. Третя частина – \шлях – це не обов'язкова частина; якщо ім'я загального ресурсу відповідає папці у файловій системі, то \шлях позначає частковий шлях до потрібного файла, що починається від цієї папки. Четверта частина – \ім'я – це ім'я об'єкта, наприклад, ім'я файла у папці, до якої відкритий загальний доступ користувачам мережі.

Для іменованих каналів UNC-імена мають вигляд \\сервер\pipe\ім'я якщо процес відкриває канал, створений на іншій робочій станції мережі, або

\\.\pipe\ім'я якщо процес створює канал чи відкриває канал на своїй робочій станції.

Основний API вводу/виводу працює з UNC-іменами й звертання до файлів по мережі за допомогою UNC-імен приховує від додатка деталі формування мережного з'єднання, що організуються відповідними компонентами ОС.

1.9.2 Реалізації каналів

У Windows NT реалізація іменованих каналів зміщена убік організації взаємодії «клієнт-сервер», і працюють вони багато в чому подібно сокетам. Найпростіший сервер іменованих каналів створює канал, з'єднується з єдиним клієнтом, виконує транзакцію (обмін) з клієнтом, розриває з'єднання із клієнтом, закриває ідентифікатор каналу, і, можливо, завершується.

Модель найпростішого сервера іменованих каналів легко реалізована, але мало ефективна, тому що володіє малою функціональністю, що не достатньо для рішення цілого ряду практичних завдань. Більш характерною для сервера є робота з декількома клієнтами.

Сервер іменованих каналів міг би використати єдиний канал для зв'язку з багатьма клієнтами, з'єднуючи й від'єднуючи кожного клієнта один за іншим, але продуктивність такого рішення, особливо при значному числі клієнтів, була б дуже низкою. Більш характерною для сервера є одночасна робота відразу з декількома клієнтами, тим більше, що клієнти можуть розташовуватися на різних робочих станціях мережі.

Сервер іменованих каналів міг би створити кілька каналів з різними іменами для роботи з різними клієнтами («кожному по каналу»), але це знижує масштабованість клієнт-серверного додатка. Для кожного клієнта доведеться компілювати власний код, що розрізняється тільки ім'ям використовуваного каналу. Можна передавати ім'я каналу як параметр командного рядка при старті клієнта, але яке ім'я? Можна всі імена каналів закласти в код єдиного клієнта й по черзі намагатися з'єднається з кожним із каналів, але необхідність

збільшення кількості клієнтів призведе до неминучої переробки й коду сервера й коду клієнта.

Windows NT підтримує так називані множинні реалізації іменованих каналів, допускаючи багаторазове створення й одночасне існування декількох незалежних екземплярів каналу, що мають однакові імена й параметри, що дозволяє просто й ефективно організувати роботу з декількома клієнтами одночасно.

З єдиною серверною системою можуть зв'язуватися одночасно декілька клієнтів, що використовують канали з тим самим ім'ям. Кожен клієнт буде мати власний екземпляр іменованого каналу, і сервер може використовувати цей же канал для відправлення відповіді клієнтові.

1.10 Створення каналу

Створення іменованого каналу (або його реалізації) виконується за допомогою функції

```
HANDLE CreateNamedPipe(
  LPCTSTR lpName,              // адреса рядка імені каналу
  DWORD   dwOpenMode,          // режим відкриття каналу
  DWORD   dwPipeMode,          // режим роботи каналу
  DWORD   nMaxInstances,       // максимальна кількість реалізацій каналу
  DWORD   nOutBufferSize,      // розмір вихідного буфера в байтах
  DWORD   nInBufferSize,       // розмір вхідного буфера в байтах
  DWORD   nDefaultTimeOut,     // час очікування в мілісекундах
  LPSECURITY_ATTRIBUTES lpSecurityAttributes); // атрибути захисту
```

Через параметр lpName передається адреса рядка, що закривається нулем, з UNC-ім’ям каналу.

За допомогою параметра dwOpenMode можна вказати, чи буде даний канал використаний тільки для читання даних (PIPE_ACCESS_INBOUND), тільки для запису (PIPE_ACCESS_OUTBOUND) або одночасно для читання й запису (PIPE_ACCESS_DUPLEX). Перераховані вище параметри повинні бути

однакові для всіх реалізацій каналу. Додатково в параметрі dwOpenMode можна задати ще шість різних прапорів відповідальних за режим операцій вводу/виводу й безпеку, які можуть бути різні для різних реалізацій каналу і які тут не розглядаються.

Параметр dwPipeMode визначає режим роботи каналу і являє собою логічну суму констант із трьох різних груп. Константи в межах групи є взаємовиключними значеннями цього параметра.

PIPE_TYPE_BYTE або PIPE_TYPE_MESSAGE указують, відповідно, чи будуть дані записуватися в канал як потік байтів, або як повідомлення. Даний параметр повинен бути однаковий для всіх реалізацій каналу. За замовчуванням використовується PIPE_TYPE_BYTE.

PIPE_READMODE_BYTE або PIPE_READMODE_MESSAGE указують, відповідно, чи будуть дані зчитуватися як потік байтів, або як повідомлення. Значення PIPE_READMODE_MESSAGE вимагає одночасного використання значення PIPE_TYPE_MESSAGE. За замовчанням використовується PIPE_READMODE_BYTE.

PIPE_WAIT або PIPE_NOWAIT визначають, відповідно, буде чи не буде працювати канал у режимі, що блокує. У режимі, що блокує, потік виконуючий операцію уведення/виведення в каналі переводиться в стан очікування до завершення операцій. У режимі, що не блокує, якщо операція не може бути виконана негайно, відповідна функція завершується з помилкою. За замовчанням використовується PIPE_WAIT.

Параметр nMaxInstances визначає максимальну кількість реалізацій, які можуть бути створені для каналу. Можна вказувати тут значення від 1 до PIPE_UNLIMITED_INSTANCES. В останньому випадку максимальна кількість реалізацій обмежується тільки наявністю вільних системних ресурсів.

Параметри nOutBufferSize й nInBufferSize визначають, відповідно, розмір буферів, використовуваних для запису в канал і читання з каналу. При необхідності система може використати буфери інших, порівняно зі зазначеними, розмірів.

Параметр nDefaultTimeOut визначає час очікування для реалізації каналу. Для всіх реалізацій необхідно вказувати однакове значення цього параметра.

Через параметр lpPipeAttributes передається адреса змінною, в якій містяться атрибути захисту для створюваного каналу.

У випадку успіху функція CreateNamedPipe повертає ідентифікатор створеної реалізації каналу, який можна використовувати в операціях читання й запису, виконуваних за допомогою таких функцій як ReadFile й WriteFile.

При помилці функція CreateNamedPipe повертає значення INVALID_HANDLE_VALUE.

1.11 Функції підключення іменованих каналів

Після того як серверний процес створив канал, він може перейти в режим з'єднання з клієнтським процесом. З'єднання з боку сервера виконується за допомогою функції ConnectNamedPipe.

```
BOOL ConnectNamedPipe(
  HANDLE hNamedPipe,                // ідентифікатор іменованого каналу
  LPOVERLAPPED lpOverlapped);   // адреса структури OVERLAPPED
```

Через параметр hNamedPipe серверний процес передає цій функції ідентифікатор каналу, отриманий від функції CreateNamedPipe.

Параметр lpOverlapped використовується тільки для організації асинхронного обміну даними через канал. Якщо використовуються тільки синхронні операції, як значення для цього параметра потрібно вказати NULL. Якщо параметр lpOverlapped установлений рівним NULL, то функція ConnectNamedPipe здійснює повернення відразу ж після встановлення з'єднання з клієнтом.

Для каналу, створеного в синхронному режимі, що блокує (з використанням константи PIPE_WAIT), функція ConnectNamedPipe переходить у стан очікування з'єднання з клієнтським процесом.

Якщо канал створений у синхронному режимі, що не блокує, функція ConnectNamedPipe негайно повертає керування з кодом TRUE, якщо тільки

клієнт був відключений від даної реалізації каналу й можливе підключення цього клієнта. У протилежному випадку повертається значення FALSE. Подальший аналіз необхідно виконувати за допомогою функції GetLastError. Ця функція може повернути значення ERROR_PIPE_LISTENING (якщо до сервера ще не підключений жоден клієнт), ERROR_PIPE_CONNECTED (якщо клієнт уже підключений) або ERROR_NO_DATA (якщо попередній клієнт відключився від сервера, але ще не завершив з'єднання).

Після повернення з функції ConnectNamedPipe сервер може виконувати читання запитів за допомогою функції ReadFile і запис відповідей за допомогою функції WriteFile.

Якщо сервер працює з декількома клієнтськими процесами, то він може використовувати для цього кілька реалізацій каналу, причому ті самі реалізації можуть застосовуватися по черзі.

Установивши зв'язок з клієнтським процесом за допомогою функції ConnectNamedPipe, серверний процес може потім його розірвати, викликавши для цього функцію BOOL DisconnectNamedPipe(HANDLE hNamedPipe).

Через параметр hNamedPipe функції передається ідентифікатор реалізації каналу, отриманий від функції CreateNamedPipe.

У випадку успіху функція повертає значення TRUE, а при помилці – FALSE. Після успішного виклику функції реалізація каналу може бути знову використана для з'єднання з іншим клієнтським процесом. Якщо канал більше не потрібний, після відключення клієнтського процесу, серверний процес повинен закрити його ідентифікатор функцією CloseHandle.

Для з'єднання з каналом клієнтський процес використовує функцію CreateFile з параметром dwCreationDisposition рівним OPEN_EXISTING. Параметр lpFileName задається у вигляді UNC-імені згідно з правилами іменування каналів.

У випадку успішного завершення функція CreateFile повертає ідентифікатор реалізації каналу. При помилці повертається значення INVALID_HANDLE_VALUE.

Після успішного повернення з функції CreateFile клієнт може використовувати отриманий ідентифікатор для читання відповідей сервера за допомогою функції ReadFile і запису запитів за допомогою функції WriteFile.

Якщо канал більше не потрібний, клієнтський процес повинен закрити його ідентифікатор функцією CloseHandle.

1.12 Приклади програмування

Організацію взаємодії між процесами з використанням іменованих каналів продемонструємо на прикладі двох консольних додатків PipeServer.exe (сервер) і PipeClient.exe (клієнт).

Сервер створює іменований канал і переходить у режим очікування підключення до нього клієнта. Після старту клієнт відкриває канал і в циклі до натискання клавіші Enter передає через нього команди, що вводяться користувачем з клавіатури, для виконання їх сервером. Команди являють собою рядки символів, які сервер обробляє у такий спосіб:

- date – одержує поточну системну дату, переводить її в символьний вигляд і відправляє через канал назад клієнтові
- time – одержує поточний системний час, переводить його в символьний вигляд і відправляє через канал назад клієнтові
- exit – відправляє через канал клієнтові текстове повідомлення «exit» і завершує свою роботу. Прийнятий клієнтом з каналу рядок «exit» означає й для нього завершення роботи.
- Будь-який рядок, що не збігається з вищеперерахованими – відправляє через канал клієнтові текстове повідомлення «Bad command».

Клієнт після передачі команди в канал читає з нього відповідь, отриману від сервера і виводить її на свою консоль. Канал створюється в режимі, що блокує. Введення/виведення виконується в синхронному режимі і для упорядковування обміну ніяких додаткових засобів не використовується.

```
; Файл interface.inc
include windows.inc
```

```
include user32.inc
include kernel32.inc
include masm32.inc
include C:\MASM32\MACROS\strings.mac
includelib  user32.lib
includelib  kernel32.lib
includelib  masm32.lib
Main      PROTO
.data
szPipeName              db "\\.\\pipe\\$NamedPipe$",0
.data?
hPipe                   HANDLE ?
nSendReciveBytes        DWORD ?
InputBuffer             BYTE 512 dup (?)
OutputBuffer            BYTE 512 dup (?)
; Файл PipeServer.Inc
.data?
DateTime                                SYSTEMTIME <>
; Файл PipeServer.Asm
.686
.model flat, stdcall
option casemap :none
include ..\interface.inc
include PipeServer.inc
.code
start:
        invoke Main
        invoke ExitProcess,0
Main proc
invoke AllocConsole
```

```
invoke ClearScreen
invoke CreateNamedPipe, ADDR szPipeName,\
        PIPE_ACCESS_DUPLEX, PIPE_TYPE_MESSAGE OR PIPE_WAIT,\
        PIPE_UNLIMITED_INSTANCES,\
        LENGTHOF InputBuffer, LENGTHOF OutputBuffer, NULL, NULL
.if eax==INVALID_HANDLE_VALUE
        invoke StdOut, $CTA0("Cannot create pipe")
        ret
.elseif
        mov hPipe,eax
.endif
invoke StdOut, $CTA0("Connection . . .")
invoke ConnectNamedPipe,hPipe,NULL
invoke locate,11,0
invoke StdOut, $CTA0("successful")
invoke locate,0,2
lea esi,InputBuffer
lea edi,OutputBuffer
.while (TRUE)
        invoke ReadFile,hPipe,esi,LENGTHOF InputBuffer,\
                ADDR nSendReciveBytes, NULL
        or dword ptr [esi],20202020h                    ;
        invoke StdOut,esi
        .break .if dword ptr [esi]==74697865h
        mov ebx,edi
        .if dword ptr [esi]==65746164h
                invoke GetLocalTime,ADDR DateTime
                movzx ecx,DateTime.wDay
                invoke dwtoa,ecx,ebx
                invoke lstrlen,ebx
```

```
        mov byte ptr [ebx+eax],"/"
        lea ebx,[ebx+eax+1]
        movzx ecx,DateTime.wMonth
        invoke dwtoa,ecx,ebx
        invoke lstrlen,ebx
        mov byte ptr [ebx+eax],"/"
        lea ebx,[ebx+eax+1]
        movzx ecx,DateTime.wYear
        invoke dwtoa,ecx,ebx
        invoke lstrlen,ebx
        mov dword ptr [ebx+eax],0D0Ah
        invoke WriteFile,hPipe,edi, LENGTHOF OutputBuffer,\
                ADDR nSendReciveBytes, NULL
.elseif dword ptr [esi]==656D6974h
        invoke GetLocalTime,ADDR DateTime
        movzx ecx,DateTime.wHour
        invoke dwtoa,ecx,ebx
        invoke lstrlen,ebx
        mov byte ptr [ebx+eax],":"
        lea ebx,[ebx+eax+1]
        movzx ecx,DateTime.wMinute
        invoke dwtoa,ecx,ebx
        invoke lstrlen,ebx
        mov byte ptr [ebx+eax],":"
        lea ebx,[ebx+eax+1]
        movzx ecx,DateTime.wSecond
        invoke dwtoa,ecx,ebx
        invoke lstrlen,ebx
        mov dword ptr [ebx+eax],0D0Ah
        invoke WriteFile,hPipe,edi, LENGTHOF OutputBuffer,\
```

```
                    ADDR nSendReciveBytes, NULL
        .else
                invoke WriteFile,hPipe, $CTA0("Bad command\n"), 14,\
                    ADDR nSendReciveBytes, NULL
.endif
.endw
invoke WriteFile,hPipe,$CTA0("exit\n"), 6,\
        ADDR nSendReciveBytes, NULL
invoke CloseHandle, hPipe
ret
Main endp
end start

; Файл PipeClient.Asm
.386
.model flat, stdcall
option casemap :none
include ..\interface.inc
.code
start:  invoke Main
        invoke ExitProcess,0
Main proc
invoke AllocConsole
invoke ClearScreen
invoke StdOut, $CTA0("Connection . . . ")
invoke CreateFile, ADDR szPipeName,GENERIC_READ OR GENERIC_WRITE,\
            0, NULL, OPEN_EXISTING, 0, NULL
.if eax==INVALID_HANDLE_VALUE
        invoke StdOut, $CTA0("Cannot create pipe")
        ret
```

```
.else
        mov hPipe,eax
.endif
invoke locate,11,0
invoke StdOut, $CTA0("successful")
invoke locate,0,2
lea esi,InputBuffer
lea edi,OutputBuffer
.while (TRUE)
        mov ebx,edi
        invoke StdOut,$CTA0("\nCommand$ ")
        invoke StdIn,edi,LENGTHOF OutputBuffer
        mov al,0Dh
        mov ecx,LENGTHOF OutputBuffer
        push edi
        repne scasb
        mov byte ptr [edi+1],0h
        sub edi,ebx
        lea ebx,[edi+2]
        pop edi
        invoke WriteFile,hPipe, edi, ebx, ADDR nSendReciveBytes, NULL
        invoke ReadFile,hPipe,esi,\
                LENGTHOF InputBuffer, ADDR nSendReciveBytes, NULL
        invoke StdOut, esi
.break  .if dword ptr [esi]==74697865h
.endw
invoke CloseHandle, hPipe
ret
Main endp
end start
```

1.13 Завдання до практичного заняття

Для визначення номера варіанта потрібно перевести номер залікової книжки або студентського квитка у двійкову систему числення, виділити три молодших біти ($b_2b_1b_0$) і скористатися таблицею 3.1.

Розробити два додатки операційної системи Windows NT, що демонструють організацію обміну даними між процесами. Перший додаток (клієнт) повинен звертатися до другого додатку (серверу) із запитом на виконання завдання згідно з номером варіанта, наведеним у таблиці 3.1. Після виконання завдання, серверний додаток повинен передати результати виконання запиту клієнту, а той їх обробити та вивести на екран. Сервер повинен уміти працювати в локальній мережі Microsoft і обслуговувати одночасно до трьох клієнтів.

Таблиця 0.1

Біти $b_2b_1b_0$	Запити клієнта на виконання завдання
000	Надати вміст зазначеного каталогу.
001	Надати мережеве ім’я комп’ютера, ім’я користувача, що в даний час зареєструвався в системі та час, що пройшов з моменту останнього завантаження операційної системи.
010	Надати інформацію про кількість, імена, типи, розмір установлених логічних дискових пристроїв.
011	Надати перелік усіх доступних мережевих ресурсів.
100	Надати інформацію про загальний та доступний розмір віртуальної та фізичної пам’яті.
101	Надати інформацію про загальний розмір зазначеного логічного диску та наявний розмір вільного місця.
110	Надати мітку, серійний номер і тип файлових систем кожного з логічних дисків.
111	Повернути інформацію про зазначений процес та його модулі.

1.14 Завдання до самостійної роботи

Питання, які студент має опрацювати самостійно

1. Функції транзакцій іменованих каналів.
2. Визначення і зміна стану іменованого каналу.
3. Отримання інформації про іменований канал.
4. Уособлення (impersonate) іменованих каналів.

Література: [2, Глава 4, с. 78 – 101; 3, Глава 16, с. 285 – 303; 6, Глава 2; 7, Глава 11, с. 374 – 382].

ПРАКТИЧНЕ ЗАНЯТТЯ № 4

Тема. Організація взаємодії між процесами за допомогою анонімних каналів

Мета: набуття практичних навичок організації взаємодії між процесами за допомогою анонімних каналів, організації конвеєрної взаємодії команд та утиліт командного інтерпретатора ОС з консольними додатками та розробки програм з використанням стандартних елементів графічного інтерфейсу.

Короткі теоретичні відомості

1.1 Організація IPC батьківського і дочірнього процесів

Анонімні канали (anonymous pipes) Windows забезпечують односпрямовану побайтову передачу даних між процесами на локальному комп'ютері. За допомогою анонімного каналу не можна організувати IPC двох довільних процесів, не пов'язаних ні якими родинними відносинами. Причини таких обмежень стають зрозумілі при вивченні прототипу функції CreatePipe, що використовується для створення анонімного каналу:

```
BOOL CreatePipe(
  PHANDLE hReadPipe,
  PHANDLE hWritePipe,
  LPSECURITY_ATTRIBUTES lpPipeAttributes,
  DWORD nSize);
```

Канал може використовуватися або для запису в нього даних, або для читання. Тому при створенні каналу функція CreatePipe повертає два ідентифікатори, записуючи їх за адресами, заданими в параметрах hReadPipe й hWritePipe. Ідентифікатор, записаний за адресою hReadPipe, можна передавати як параметр функції ReadFile або ReadFileEx для виконання операції читання. Ідентифікатор, записаний за адресою hWritePipe, передається функції WriteFile або WriteFileEx для виконання операції запису.

Параметр nSize визначає розмір буфера для створюваного каналу. Якщо цей розмір зазначений як нуль, буде створений буфер з розміром, прийнятим за замовчанням. За необхідності система сама може змінити зазначений при створенні каналу розмір буфера.

Читання з використанням ідентифікатора читання каналу блокується, якщо канал порожній. У протилежному випадку в процесі читання буде прийнято стільки байтів, скільки є в каналі, аж до кількості, зазначеної при виклику функції ReadFile. Операція запису в заповнений канал, також буде блокована.

Через параметр lpPipeAttributes передається адреса структури з атрибутами захисту для створюваного каналу. Якщо вказати цей параметр як NULL, канал буде мати атрибути захисту, прийняті за замовчанням.

У випадку успіху функція CreatePipe повертає не нульове значення (TRUE), при помилці – FALSE.

Щоб канал можна було використати для IPC, повинен існувати ще один процес, і для цього процесу потрібно один з ідентифікаторів каналу. Тоді виникає питання про те, як передати іншому процесу ідентифікатор читання або записи каналу.

Звичайно, можна спробувати переслати ці ідентифікатори іншому процесу використовуючи будь-який інший механізм IPC, але в першу чергу тут виникає питання доцільності таких дій. Якщо між процесами вже вдалося реалізувати IPC, то навіщо їм після цього використовувати ще й анонімні канали?

Передачу ідентифікаторів створеного каналу іншому процесу можна виконати використовуючи механізм спадкування ідентифікаторів дочірніми процесами. Одним із найпоширеніших способів використання анонімних каналів є заміна ідентифікатора стандартного потоку введення або виведення дочірнього процесу, на один з ідентифікаторів каналу, що створив батьківський процес.

Обмеженням такого механізму IPC є те, що він дозволяє реалізувати взаємодію тільки батьківського і дочірнього процесу, і при цьому дочірній процес повинен бути реалізований у вигляді консольного додатка ОС, бо тільки для консольних додатків існує поняття потоків введення/виведення.

Щоб зробити ідентифікатор наслідуваним, батьківський процес повинен почати спеціальні дії, оскільки створювані ідентифікатори не стають такими за замовчанням.

Створити наслідуваний ідентифікатор можна або шляхом використання структури SECURITY_ATTRIBUTES у момент створення об'єкта, або шляхом копіювання існуючого ідентифікатора за допомогою DuplicateHandle.

У структурі SECURITY_ATTRIBUTES є прапор bInheritHandle, значення якого повинне бути встановлене рівним TRUE. Також в екземплярі структури значенням sizeof SECURITY_ATTRIBUTES повинен бути ініціалізований елемент nLength.

При виклику функції CreateProcess повинен бути заданий прапор bInheritHandles, що визначає, чи буде дочірній процес успадковувати копії ідентифікаторів батьківського процесу.

1.2 Організація конвеєрної IPC трьох процесів

Твердження про те, що не можливо організувати IPC двох довільних процесів, не пов'язаних ні якими родинними відносинами, безперечно, має сенс, але якщо ширше підійти до поняття «родинні відносини» процесів, то з використанням одного й того самого анонімного каналу можна організувати взаємодію між трьома процесами.

Батьківський та дочірній процеси пов’язані прямими родинними відносинами, але певний зв’язок мають і різні процеси створені одним і тим самим процесом, тобто ті, які мають спільних батьків. Безпосередньо між собою такі процеси не можуть організувати взаємодію з використанням анонімного каналу, але тут їм на допомогу може прийти посередник – батьківський процес.

Батьківський процес створює анонімний канал. При створенні каналу функція CreatePipe повертає два ідентифікатори – один для читання, а другий для запису. Батьківський процес робить ідентифікатори читання і запису каналу успадкованими шляхом установлення в TRUE елемента bInheritHandle та ініціалізації значенням sizeof SECURITY_ATTRIBUTES елемента nLength структури SECURITY_ATTRIBUTES у момент створення об'єкта.

Ідентифікатори читання і запису анонімного каналу батьківський процес використовує для організації взаємодії між собою створюваних ним двох дочірніх процесів шляхом заміни ідентифікаторів стандартних потоків введення та виведення дочірніх процесів на ідентифікатори каналу в елементах hStdOutput, hStdInput структури STARTUPINFO, з встановленням прапору STARTF_USESTDHANDLES в елементі dwFlags, та використовуючи механізм спадкування ідентифікаторів дочірніми процесами.

При виклику функції CreateProcess для створення дочірніх процесів повинен бути заданий прапор bInheritHandles, що визначає, чи буде дочірній процес успадковувати копії ідентифікаторів батьківського. Для організації взаємодії з дочірнім процесом батьківський процес виконує перенаправлення стандартного потоку виведення дочірнього процесу у свій стандартний потік виведення заміною ідентифікатора стандартного потоку виведення дочірнього процесу на ідентифікатор власного стандартного потоку виведення.

Обмеженням такого механізму використання анонімних каналів для організації IPC трьох процесів є те, що всі три процеси повинні бути реалізовані у вигляді консольних додатків ОС, бо тільки для консольних додатків існує поняття потоків введення/виведення.

Подібна методика реалізації IPC декількох процесів може використовуватися не тільки для перенаправлення введення/виведення, а й для організації конвеєра команд в оболонці командного інтерпретатору ОС.

1.3 Приклади програмування

Наступні приклади програмування демонструють організацію взаємодії графічного та консольного додатків користувача зі стандартними утилітами командного інтерфейсу операційної системи.

Приклад 0.1 GUI-додаток guiping.exe, що демонструє організацію взаємодії через анонімний канал графічного й консольного додатка Windows. У якості консольного додатка використовується утиліта командного інтерфейсу ping.exe. Додаток guiping.exe сам відправленням ніяких мережних пакетів не займається, а є просто графічною оболонкою (Рисунок 0.1) для ping.exe. Додаток guiping.exe створює канал, а потім створює процес для ping.exe, передаючи через структуру STARTUPINFO як ідентифікатор вихідного потоку консолі для ping.exe ідентифікатор каналу. Дочекавшись завершення ping.exe з використанням WaitForSingleObject, guiping.exe читає інформацію з каналу й відображає її з використанням елементів графічного інтерфейсу.

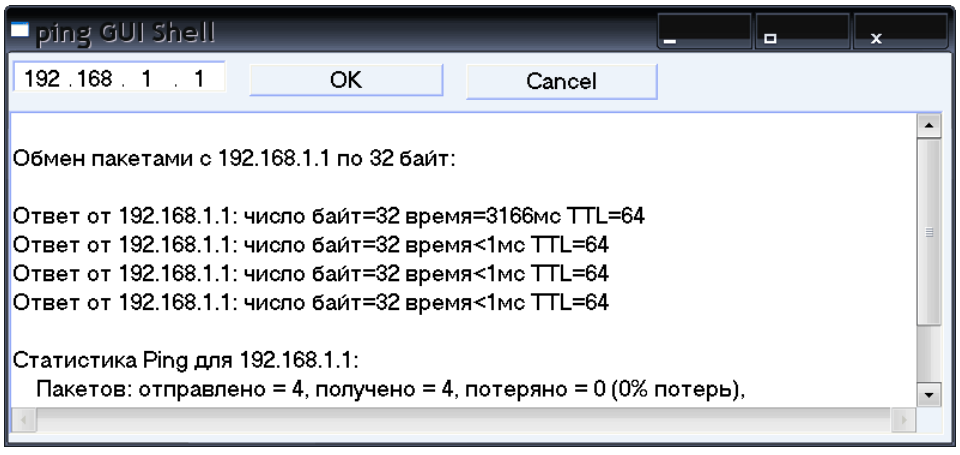

Рисунок 0.1

```
; Файл guipingdlg.rc
#define IDD_MAINDIALOG 101
#define IDC_RESULTEDIT 1001
```

```
#define IDC_IP 1002
#define IDC_OKBTN 1003
#define IDC_CANCELBTN 1004
IDD_MAINDIALOG DIALOGEX 3,3,286,108
CAPTION "ping GUI Shell"
FONT 14,"MS Sans Serif",700,0,204
STYLE 0x10CF0800
EXSTYLE 0x00000000
BEGIN
CONTROL "",IDC_RESULTEDIT,"Edit",0x503100C4,0,16,284,90,0x00000200
CONTROL "",IDC_IP,"SysIPAddress32",0x50010000,1,2,65,9,0x00000200
CONTROL "OK",IDC_OKBTN,"Button",0x50010000,73,3,59,9,0x00000000
CONTROL
"Cancel",IDC_CANCELBTN,"Button",0x50010000,139,3,58,10,0x00000000
END
; Файл guiping.inc
include windows.inc
include kernel32.inc
include user32.inc
include Comctl32.inc
include shell32.inc
include       C:\MASM32\MACROS\strings.mac
includelib kernel32.lib
includelib user32.lib
includelib Comctl32.lib
includelib shell32.lib
DlgProc                 PROTO     :HWND,:UINT,:WPARAM,:LPARAM
.const
IDD_MAINDIALOG                    EQU  101
IDC_RESULTEDIT                    EQU  1001
```

```
IDC_IP                  EQU 1002
IDC_OKBTN               EQU  1003
IDC_CANCELBTN           EQU 1004
.data
szExecute       db "cmd.exe /c c:\windows\system32\ping.exe "
IP              db 18 dup(0)
.data?
hInstance       HANDLE ?
StartInfo       STARTUPINFO <>
ProcessInfo     PROCESS_INFORMATION <>
Security        SECURITY_ATTRIBUTES <>
phRead          HANDLE ?
phWrite         HANDLE ?
StdOutBuffer    BYTE 4096 dup(?)
nBytesTransfer  DWORD ?
ExitCode        DWORD ?
; Файл guiping.asm
.386
.model flat, stdcall
option casemap :none
include guiping.inc
.code
start:  invoke GetModuleHandle,NULL
        mov   hInstance,eax
        invoke InitCommonControls
        invoke DialogBoxParam, hInstance, IDD_MAINDIALOG, NULL,\
                        addr DlgProc, NULL
        invoke ExitProcess, 0
DlgProc proc hWin:HWND,uMsg:UINT,wParam:WPARAM,lParam:LPARAM
LOCAL OutputBuffer[LENGTHOF StdOutBuffer]:BYTE
```

```
mov   eax,uMsg
.if eax== WM_COMMAND
        mov   eax,wParam
        and    eax,0FFFFh
        .if eax==IDC_OKBTN
        mov   Security.lpSecurityDescriptor,NULL
        mov   Security.bInheritHandle,TRUE
        invoke CreatePipe,ADDR phRead, ADDR phWrite,\
                addr Security, LENGTHOF StdOutBuffer
        invoke GetStartupInfo, ADDR StartInfo
        mov ebx,phWrite
        mov StartInfo.hStdOutput,ebx
        mov StartInfo.hStdError,ebx
        mov StartInfo.wShowWindow,SW_HIDE
        mov          StartInfo.dwFlags,          STARTF_USESHOWWINDOW          +
STARTF_USESTDHANDLES
        invoke GetDlgItemText, hWin, IDC_IP, ADDR IP, LENGTHOF IP
        invoke CreateProcess, NULL, ADDR szExecute, NULL, NULL,TRUE,\
                CREATE_NEW_CONSOLE, NULL,NULL,ADDR StartInfo,\
                ADDR ProcessInfo
        invoke WaitForSingleObject,ProcessInfo.hProcess,INFINITE
        invoke ReadFile,phRead, ADDR StdOutBuffer, LENGTHOF StdOutBuffer,\
                        ADDR nBytesTransfer, NULL
        invoke OemToChar,ADDR  StdOutBuffer, ADDR OutputBuffer
        invoke SetDlgItemText,hWin,IDC_RESULTEDIT,ADDR OutputBuffer
        .elseif eax==IDC_CANCELBTN
                invoke SendMessage,hWin,WM_CLOSE,0,0
        .endif
.elseif eax==WM_CLOSE
        invoke EndDialog,hWin,0
```

```
.else
        mov   eax,FALSE
        ret
.endif
mov   eax,TRUE
ret
DlgProc endp
end start
```

Приклад 0.2 Консольний додаток SortTaskList.exe, який виводить на власну консоль список усіх запущених у системі процесів. Список видається відсортованим за зменшенням розміру використовуваної процесом віртуальної пам'яті. Додаток SortTaskList.exe сам ні одержанням інформації про процеси, ні сортуванням не займається. SortTaskList.exe створює процеси для стандартних утиліт командного інтерфейсу ОС tasklist.exe й sort.exe із зазначенням відповідних параметрів командного рядка й перенаправляє вихідний потік tasklist.exe у вхідний потік sort.exe, а вихідний потік sort.exe у свій вхідний потік.

```
; Файл SortTaskList.inc
include windows.inc
include user32.inc
include kernel32.inc
include masm32.inc
include C:\MASM32\MACROS\strings.mac
includelib  user32.lib
includelib  kernel32.lib
includelib  masm32.lib
Main      PROTO
.data?
StartupInfo1          STARTUPINFO <>
```

```
StartupInfo2        STARTUPINFO <>
hReadPipe           HANDLE ?
hWritePipe          HANDLE ?
ProcessInfo1        PROCESS_INFORMATION <>
ProcessInfo2        PROCESS_INFORMATION <>
PSA                 SECURITY_ATTRIBUTES <>
hProcess            HANDLE 2 dup (?)
; Файл SortTaskList.asm
.386
.model flat, stdcall
option casemap :none
include SortTaskList.inc
.code
start:  invoke Main
        invoke ExitProcess,0
Main proc
LOCAL Buffer[128]:BYTE
invoke AllocConsole
invoke CharToOem, $CTA0("Інформація про процеси"), ADDR Buffer
invoke SetConsoleTitle,ADDR Buffer
invoke ClearScreen
mov PSA.nLength,SIZEOF PSA
mov PSA.lpSecurityDescriptor,NULL
mov PSA.bInheritHandle,TRUE
invoke GetStartupInfo,addr StartupInfo1
invoke CreatePipe,addr hReadPipe, addr hWritePipe,ADDR PSA,NULL
push hWritePipe
pop StartupInfo1.hStdOutput
mov StartupInfo1.dwFlags,STARTF_USESTDHANDLES
invoke CreateProcess, NULL,
```

```
        $CTA0("C:\\WINDOWS\\SYSTEM32\\tasklist.exe /nh"),\
        NULL, NULL, TRUE, NULL, NULL, \
        $CTA0("C:\\WINDOWS\\SYSTEM32\\wbem"),\
        ADDR StartupInfo1, ADDR ProcessInfo1
invoke CloseHandle,hWritePipe
invoke GetStartupInfo,addr StartupInfo2
invoke GetStdHandle,STD_OUTPUT_HANDLE
mov StartupInfo2.hStdOutput,eax
push hReadPipe
pop StartupInfo2.hStdInput
mov StartupInfo2.dwFlags,STARTF_USESTDHANDLES
invoke CreateProcess, NULL, \
        $CTA0("C:\\WINDOWS\\SYSTEM32\\sort.exe /+60 /R"),\
        NULL, NULL, TRUE, NULL,\
        NULL, NULL, ADDR StartupInfo2, ADDR ProcessInfo2
invoke CloseHandle,hReadPipe
lea esi,hProcess
push ProcessInfo1.hProcess
pop dword ptr [esi]
push ProcessInfo2.hProcess
pop dword ptr [esi+4]
invoke WaitForMultipleObjects,2,esi,TRUE,INFINITE
invoke CloseHandle,ProcessInfo1.hProcess
invoke CloseHandle,ProcessInfo2.hProcess
invoke StdIn,ADDR Buffer, LENGTHOF Buffer
ret
Main endp
end start
```

1.4 Завдання до практичного заняття

Для визначення номера варіанта потрібно перевести номер залікової книжки або студентського квитка у двійкову систему числення, виділити три молодших біти ($b_2b_1b_0$) і скористатися таблицею 4.1.

Розробити додаток з графічним інтерфейсом користувача, що виконує роль оболонки для виконання утиліт командного інтерфейсу операційної системи Windows NT згідно з номером варіанта, наведеним у таблиці 4.1.

Таблиця 0.1

Біти $b_2b_1b_0$	Утиліта командного інтерфейсу
000	attrib [{+r|-r}] [{+a|-a}] [{+s|-s}] [{+h|-h}] [[диск:][шлях] ім’я_файла] [/s[/d]]
001	comp [файл1] [файл2] [/d] [/a] [/l] [/n=кількість] [/c]
010	fc [/a] [/b] [/c] [/l] [/lbn] [/n] [/t] [/u] [/w] [/nnnn] [диск1:][шлях1]ім’яфайла1 [диск2:][шлях2]ім’яфайла2
011	ipconfig [/all] [/renew [адаптер]] [/release [адаптер]] [/flushdns] [/displaydns] [/registerdns] [/showclassid адаптер] [/setclassid адаптер [код_класу]]
100	start ["заголовок"] [/dшлях] [/i] [/min] [/max] [{/separate | /shared}] [{/low | /normal | /high | /realtime | /abovenormal | belownormal}] [/wait] [/b] [им’я_файлу] [параметри]
101	tasklist[.exe] [/s комп’ютер] [/u домен\користувач [/p пароль]] [/fo {TABLE|LIST|CSV}] [/nh] [/fi фільтр [/fi фільтр2 [...]]] [/m [модуль] | /svc | /v]
110	systeminfo[.exe] [/s комп’ютер [/u домен\користувач [/p пароль]]] [/fo {TABLE|LIST|CSV}] [/nh]
111	xcopy джерело [результат] [/w] [/p] [/c] [/v] [/q] [/f] [/l] [/g] [/d[:мм-дд-рррр]] [/u] [/i] [/s [/e]] [/t] [/k] [/r] [/h] [{/a|/m}] [/n] [/o] [/x]

	[/exclude:файл1[+[файл2]][+[файл3]] [{/y|/-y}] [/z]

1.5 Завдання до самостійної роботи

Питання, які студент має опрацювати самостійно

1. Організація IPC за допомогою поштових скриньок (mailslots):
   а) іменування поштових скриньок, режими роботи і розмір повідомлень;
   б) API для обслуговування поштових скриньок;
   в) відмінності поштових скриньок і каналів.

Література: [1, Глава 6, с. 353 – 354; 2, Глава 3, с. 63 – 77; 3, Глава 3, с. 307 – 321; 6, Глава 2; 7, Глава 11, с. 384 – 392].

2 КРИТЕРІЇ ОЦІНЮВАННЯ ЯКОСТІ ВИКОНАННЯ ПРАКТИЧНИХ ЗАВДАНЬ І САМОСТІЙНОЇ РОБОТИ СТУДЕНТАМИ

Виконання практичних завдань як і самостійної роботи проводиться студентом в позааудиторний час і дозволяє йому підвищити загальний бал, що враховується при проведені модульного та семестрового контролю.

За результатами виконання практичних завдань студент складає звіт за формою звіту з лабораторної роботи який повинен містити такі складові

1. Назва і мета практичного завдання.

2. Результати (за необхідності у вигляді Screenshot), отримані під час виконання завдання.

3. Повний вихідний текст усіх файлів проектів з їх детальними коментарями.

Максимальна кількість балів за кожне практичне завдання дорівнює 3 бали. Залік з виконання практичного завдання студент одержує після співбесіди з викладачем, у ході якого він повинен обґрунтувати вибір алгоритму розв’язання задачі, дати вичерпні пояснення за змістом звіту з виконання практичного завдання.

За результатами самостійної роботи студент складає звіт у формі реферату який повинен містити детальний огляд кожного запропонованого для самостійної роботи питання. Припускається оформлювати звіт за результатами самостійної роботи у вигляді розділу звіту з виконання практичного завдання.

Максимальна кількість балів за кожну самостійну роботу дорівнює 1 бал. Залік з виконання самостійної роботи студент одержує після співбесіди з викладачем, у ході якого він повинен продемонструвати теоретичні знання з кожного запропонованого для самостійної роботи питання, надати відповіді на додаткові питання викладача з теми питання.

Підсумкові бали оцінки виконання практичних завдань з навчальної дисципліни розраховуються з цілої частини від попередньо обчисленого числа $S_p = T_p*M_p/K_p/3+0.5$, де T_p – сума оцінок всіх практичних завданнь, K_p –

загальна кількість практичних завдань, M_p – максимальна кількість балів на цей вид контролю, що передбачена в робочій навчальній програмі дисципліни.

Підсумкові бали оцінки самостійної роботи з навчальної дисципліни розраховуються з цілої частини від попередньо обчисленого числа $S_s = T_s*M_s/K_s+0.5$, де T_s – сума оцінок всіх питань самостійної роботи, K_s – кількість питань самостійної роботи, M_s – максимальна кількість балів на цей вид контролю, що передбачена в робочій навчальній програмі дисципліни.

Підсумкові бали отримані студентом за виконання практичних завдань і самостійної роботи S_p і S_s додаються до суми балів за всі види навчальної діяльності студентами в шкалі оцінювання ECTS

Сума балів за всі види навчальної діяльності	Оцінка ECTS	Оцінка за національною шкалою	
		Для екзамену, курсового проекту (роботи), практики	Для заліку
90–100	A	Відмінно	Зараховано
82–89	B	Добре	
74–81	C		
64–73	D	Задовільно	
60–63	E		
35–59	FX	Незадовільно з можливістю повторного складання	Не зараховано з можливістю повторного складання
0–34	F	Незадовільно з обов’язковим повторним вивченням навчальної дисципліни	Не зараховано з обов’язковим повторним вивченням навчальної дисципліни

СПИСОК ЛІТЕРАТУРИ

1. Вильямс А. Системное программирование в Windows 2000 для профессионалов. − СПб.: Питер, 2001. − 624 с.: ил.
2. Джонс Э., Оланд Дж. Программирование в сетях Microsoft Windows. Мастер-класс / Пер. с англ. – СПб.: Питер; М.: Издательско-торговый дом «Русская редакция», 2001. – 608 с.: ил.
3. Побегайло А.П. Системное программирование в Windows. – СПб.: БХВ-Петербург, 2006. – 1056 с.: ил.
4. Рихтер Дж., Кларк Дж. Д. Программирование серверных приложений для Microsoft Windows 2000. Мастер-класс / Пер. с англ. – СПб.: Питер; М.: Издательско-торговый дом «Русская редакция», 2001. – 592 с.: ил.
5. Рихтер Дж. Windows для профессионалов: создание эффективных Win32 приложений с учетом специфики 64-разрядной версии Windows / Пер. С англ. − 4-е изд. − СПб.: Питер; М.: Издательско-торговый дом «Русская редакция», 2004. − 749 с.: ил.
6. Фролов А.В., Фролов Г.В. Программирование для Windows NT. − М.: Диалог-МИФИ, 1996. Том 27. − Часть 1. − 272 с.
7. Харт, Джонсон, М. Системное программирование в среде Windows. – 3-е издание: Пер. с англ. – М.: Издательский дом «Вильямс», 2005. – 592 с.
8. Шрайбер С. Недокументированные возможности Windows 2000. Библиотека программиста. – СПб.: Питер, 2002. – 544 с.: ил.

Методичні вказівки щодо практичних занять і самостійної роботи з навчальної дисципліни «Системне програмне забезпечення» для студентів денної та заочної форм навчання зі спеціальності 123 – «Комп'ютерна інженерія» Частина ІІ

Укладач старш. викл. Ю.В. Зілінський
Відповідальний за випуск в.о. зав. кафедри КІС проф. М. І. Гученко

Підп. до др. _____. Формат 60х84 1/16. Папір тип. Друк ризографія.
Ум. друк. арк. ___. Наклад _____ прим. Зам. № _____. Безкоштовно.

Видавничий відділ
Кременчуцького національного університету
імені Михайла Остроградського
вул. Першотравнева, 20, м. Кременчук, 39600