Лабораторна робота № 6

(4 години)

Тема: Керування процесами у Windows.
Мета: Ознайомитися із засобами створення процесів та сформувати навички керування процесами у Windows-програмах.

Короткі теоретичні відомості

Запуск процесу здійснюється через Провідник Windows, командний рядок або із іншого процесу за допомогою функції **CreateProcess()**.

```
BOOL CreateProcess(
  LPCTSTR lpApplicationName,                  // вказівник на ім'я виконуваного файлу
  LPTSTR lpCommandLine,                       // вказівник на командний рядок
  LPSECURITY_ATTRIBUTES lpProcessAttributes,  // вказівник на PSA
  LPSECURITY_ATTRIBUTES lpThreadAttributes,   // вказівник на TSA
  BOOL bInheritHandles,                       // прапорець наслідування дескриптора
  DWORD dwCreationFlags,                      // прапорці створення
  LPVOID lpEnvironment,                       // вказівник на новий блок оточення або NULL
  LPCTSTR lpCurrentDirectory,                 // вказівник на ім'я поточного каталогу або NULL
  LPSTARTUPINFO lpStartupInfo,                // вказівник на структуру STARTUPINFO
  LPPROCESS_INFORMATION lpProcessInformation  // вказівник на структуру
                                              // PROCESS_INFORMATION
);
```

Ім'я програми задається у першому (*lpApplicationName*) або другому (*lpCommandLine*) параметрах. Найчастіше перший параметр NULL, тоді другий параметр дозволяє задати повний командний рядок із аргументами, де перша лексема – ім'я виконуваного модуля.

lpProcessAttributes, *lpThreadAttributes* – атрибути безпеки процесу (PSA) і потоку (TSA), якщо NULL, то за умовчанням.

bInheritHandles – визначає, чи може новий процес наслідувати дескриптори від батьків.

dwCreationFlags – прапорці створення нового процесу, визначають рівень пріоритету нового процесу (за умовчанням NORMAL_PRIORITY_CLASS) та умови створення (для консольних програм CREATE_NEW_CONSOLE – процес одержує нову консоль; DETACHED_PROCESS – процес не має доступу до консолі батьківського процесу та ін.).

lpEnvironment – вказує на блок оточення нового процесу. Блок містить пари типу *ім'я - значення*. Якщо NULL, використовується батьківське оточення.

lpCurrentDirectory – шлях до каталогу нового процесу. Якщо NULL, використовується каталог батьківського процесу.

lpStartupInfo – вказівник на структуру STARTUPINFO, яка описує зовнішній вигляд вікна і містить дескриптори стандартних пристроїв нового процесу.

```
typedef struct _STARTUPINFO {
    DWORD   cb;               // розмір структури
    LPTSTR  lpReserved;       // NULL
    LPTSTR  lpDesktop;        // може бути NULL
    LPTSTR  lpTitle;          // ім'я консолі або NULL
    DWORD   dwX;              // положення
    DWORD   dwY;              // на екрані
    DWORD   dwXSize;          // ширина
    DWORD   dwYSize;          // і висота
    DWORD   dwXCountChars;    // для консолі ширина
    DWORD   dwYCountChars;    // і висота екранного буфера
    DWORD   dwFillAttribute;  // для консолі колір тексту і фону
    DWORD   dwFlags;          // прапорці використання полів структури
    WORD    wShowWindow;      // прапорці показу вікна
    WORD    cbReserved2;      // 0
```

```
    LPBYTE  lpReserved2;      // NULL
    HANDLE  hStdInput;        // дескриптор стандартного пристрою вводу
    HANDLE  hStdOutput;       // дескриптор стандартного пристрою виводу
    HANDLE  hStdError;        // дескриптор стандартного пристрою помилок
} STARTUPINFO, *LPSTARTUPINFO;
```

lpProcessInformation – вказівник на структуру для прийому дескрипторів та ідентифікаторів нового процесу і потоку.

Перед викликом функції **CreateProcess()** необхідно ініціалізувати структуру типу STARTUPINFO. На практиці використовують інформацію із батьківського вікна, яку можна одержати за допомогою функції **GetStartupInfo()**.

```
VOID GetStartupInfo(
  LPSTARTUPINFO lpStartupInfo      // адреса структури STARTUPINFO
 );
```

Функція **CreateProcess()** визначає і заповнює даними поля структури типу PROCESS_INFORMATION**:**

```
typedef struct _PROCESS_INFORMATION {
    HANDLE hProcess;          // дескриптор процесу
    HANDLE hThread;           // дескриптор первинного потоку
    DWORD dwProcessId;        // ідентифікатор нового процесу
    DWORD dwThreadId;         // ідентифікатор первинного потоку
} PROCESS_INFORMATION;
```

Структура містить дескриптори та ідентифікатори створеного процесу і первинного потоку. Ідентифікатор процесу завжди залишається унікальним в системі, в той час як дескрипторів може бути кілька.

Для визначення моменту закінчення певного процесу використовують одну із функцій:

```
DWORD WaitForSingleObject(
  HANDLE hHandle,                  // дескриптор процесу, на який чекають
  DWORD dwMilliseconds             // інтервал очікування в мілісекундах
 );
DWORD WaitForMultipleObjects(
  DWORD nCount,                    // кількість дескрипторів у масиві дескрипторів об'єктів
  CONST HANDLE *lpHandles,         // вказівник на масив дескрипторів об'єктів
  BOOL bWaitAll,                   // прапорець очікування
  DWORD dwMilliseconds             // інтервал очікування в мілісекундах
 );
```

В параметрах цих функцій вказується або дескриптор процесу *hHandle* або масив дескрипторів процесів *lpHandles*. Кількість елементів масиву не повинна перевищувати значення MAXIMUM_WAIT_OBJECTS, яке дорівнює 64. Параметр *dwMilliseconds* задає інтервал очікування в мілісекундах або значення INFINITE. Параметр *bWaitAll* вказує на необхідність очікування завершення усіх процесів групи.

Функція може повернути одне із наступних значень:

- WAIT_OBJECT_0 – для **WaitForSingleObject()** об'єкт перейшов у сигнальний стан; для **WaitForMultipleObjects()** із *bWaitAll* = TRUE всі *nCount* об'єктів одночасно перейшли у сигнальний стан.
- WAIT_OBJECT_0+n, де $0 \le n < nCount$. Віднімітьте WAIT_OBJECT_0 від значення, що повертається, щоб визначити, який процес закінчився, якщо очікується завершення будь-якого із групи процесів. Якщо в сигнальний стан перейшло кілька об'єктів, то повертається найменше із можливих значень.
- WAIT_TIMEOUT указує, що період очікування вичерпався, але процес (процеси) не завершився.
- WAIT_FAILED вказує, що виклик функції завершився невдачею; наприклад, дескриптор не має доступу SYNCHRONIZE.

Завершується процес одним із способів:

- завершується ***вхідна функція первинного потоку*** (рекомендований спосіб). Функція завершується оператором **return**, який повертає код виходу із процесу.
- один із потоків процесу викликає **ExitProcess()**. Після того, як процес закінчився, він, або точніше, потік, що працює в процесі, може викликати функцію **ExitProcess()** із зазначенням коду виходу.

VOID ExitProcess (UINT *uExitCode*);

Ця функція не повертає значення. Вона завершує процес і всі його потоки. Код виходу відповідає процесу.

- потік іншого процесу викликає **TerminateProcess()** (небажаний спосіб). Процес не знає, що його зупиняють і може бути завершений некоректно (процес повинен зберегти дані на диск, звільнити свої ресурси (м'ютекси, семафори, події, пам'ять), виконати обробники завершення, закрити відкриті файли і т.д.). Виклик функції можливий, якщо дескриптор процесу, що завершується має доступ типу PROCESS_TERMINATE. Використовують у виключних випадках, коли іншими засобами завершити процес не вдається.
- усі потоки завершуються **ExitThread()**. Якщо всі потоки процесу закінчили виконання, операційна система завершує виконання процесу.

Вище зазначено, що процес, який закінчив свою роботу, повертає код завершення процесу операційній системі. Батьківський процес може дізнатися про цей код за допомогою функції **GetExitCodeProcess()**:

```
BOOL GetExitCodeProcess(

  HANDLE hProcess,          // дескриптор завершеного процесу
  LPDWORD lpExitCode        // адреса змінної для одержання коду
);
```

Завдання для виконання

1. Створити програму `run`, яка здійснює запуск іншого процесу (наприклад, компілятора TASM), чекає на його завершення, визначає код завершення процесу і виводить його на стандартний пристрій. Програма запускається командою типу: `run програма [файл_1] [файл_2] … [файл_N]`
2. Модифікувати попередню програму таким чином, щоб вона виводила результати своєї роботи у створений текстовий файл. Змінити стандартний пристрій виведення дочірнього процесу на дескриптор файлу (полю *hStdOutput* структури STARTUPINFO нового процесу слід присвоїти значення дескриптора файлу).
3. Змінити програму таким чином, щоб вона запускала кілька процесів (наприклад, TASM і TLINK), чекала на завершення кожного з цих процесів, виводила коди їх завершення. Порівняти коди завершення при різних випадках завершення програми (успішне завершення, помилка компіляції, відсутній асемблерний файл).
4. Створити програму, яка виводить на стандартний пристрій вміст кількох текстових файлів (аналог команди UNIX `cat`).

Контрольні питання

1. Що таке процес? Які ресурси доступні процесу?
2. Яким чином можна запустити процес Windows?
3. Який прототип має функція **CreateProcess()?** Яке призначення мають її параметри?
4. Яке призначення структур STARTUPINFO та PROCESS_INFORMATION?
5. Як можна використовувати поля-дескриптори структури STARTUPINFO?
6. Яка різниця між дескрипторами та ідентифікаторами процесів у Windows?

7. Які засоби можна використовувати для визначення моменту закінчення запущеного процесу?
8. Як можна перенаправити вхідні та вихідні потоки процесу?
9. В чому подібність і відмінність функцій **WaitForSingleObject()** та **WaitForMultipleObjects()**? За що відповідають їх параметри?
10. Які ви знаєте способи завершення процесів?

Додаток

Зразок програми для запуску іншого процесу

```
#include <windows.h>
#include <stdio.h>

int main () {
    STARTUPINFO si;
    PROCESS_INFORMATION pi;

    ZeroMemory (&si, sizeof (STARTUPINFO));          // заповнюємо структуру нулями
    si.cb = sizeof (STARTUPINFO);                    // ініціалізуємо потрібні поля

    if (!CreateProcess(
        NULL,
        "calc.exe",
        NULL,                          // атрибути безпеки процесу за умовчанням
        NULL,                          // атрибути безпеки потоку за умовчанням
        FALSE,                         // дескриптор без наслідування
        0,                             // звичайний пріоритет NORMAL_PRIORITY_CLASS
        NULL,                          // оточення викликаю чого процесу
        NULL,                          // поточний каталог від батьківського процесу
        &si,                           // структура STARTUPINFO
        &pi )                          // структура PROCESS_INFORMATION
    ) {
        printf ("The new process is not created.\n Check a name of the process.\n");
        return 0;
    }

    printf ("Calc handle is %ld\n", pi.hProcess);
    printf ("Calc ID is %ld\n", pi.dwProcessId);

    // чекаємо завершення дочірнього процесу
    WaitForSingleObject( pi.hProcess, INFINITE );
    printf ("The Calc is over");

    // закриємо дескриптори первинного потоку і процесу
    CloseHandle(pi.hThread);
    CloseHandle(pi.hProcess);

    return 0;
}
```