# Лабораторная работа №2

# Понятие процесса и работа с Win32 API

### Задание

1. Используя представленные ниже материалы, разработать на Visual C++ консольное приложение, выполняющее следующие команды:

**run word** – запускает приложение Microsoft Word (использовать WinExec)

**run excel** – запускает приложение Microsoft Excel (использовать CreateProcess)

**open** *имя\_документа –* открывает документ с именем *имя\_документа*  (в формате Microsoft Word или Microsoft Excel)

**exit** – завершение работы приложения

Путь к программам Word и Excel задается в программе в виде константы. Приложение должно контролировать корректность ввода команд, при неверном задании пути выдавать или команды соответствующие сообщение.

1. Найдите в документации MSDN три любые функции управления процессом или получения информации о его состоянии. Включите вызов этих функций в программу.

## Процессы в Win32

Процесс (process) - это объект операционной системы, представляющий контейнер для набора ресурсов, используемых при выполнении экземпляра программы. В Windows процесс включает следующее:

* закрытое виртуальное адресное пространство – диапазон адресов виртуальной памяти, которым может пользоваться процесс;
* исполняемую программу – начальный код и данные, проецируемые на виртуальное адресное пространство процесса;
* список открытых описателей различных системных ресурсов – семафоров, коммуникационных портов, файлов и других объектов, доступных всем потокам в данном процессе;
* контекст защиты (security context), (иногда называемый маркером доступа (accesstoken)) идентифицирующий пользователя, группы безопасности и привилегии, сопоставленные с процессом;
* уникальный идентификатор процесса;
* куча процесса;
* переменные окружения;
* минимум один поток (см. далее).

На первый взгляд понятия программа и процесс практически одинаковые, однако, их следует отличать друг от друга. Программа представляет собой статический набор команд, а процесс можно представить как программу во время исполнения.

# Управление процессами

## Создание процесса

В ОС Windows процесс создается с помощью вызова Win32 API функций CreateProcess, WinExec или ShellExecute. Функции CreateProcess, WinExec создают новый процесс. Функция ShellExecute имеет большие возможности, но чаще всего она используется для открытия файла определенного формата связанным с данным форматом приложением, таким образом, создание нового процесса является косвенным результатом выполнения данной функции.

### Функция CreateProcess

Основной функцией для создания процесса является CreateProcess. Данная функция создает новый процесс с единственным первичным потоком. Десять параметров функции позволяют указать основные параметры создаваемого процесса.

BOOL CreateProcess(

LPCTSTR lpApplicationName,

LPTSTR lpCommandLine,

LPSECURITY\_ATTRIBUTES lpProcessAttributes,

LPSECURITY\_ATTRIBUTES lpThreadAttributes,

BOOL bInheritHandles,

DWORD dwCreationFlags,

LPVOID lpEnvironment,

LPCTSTR lpCurrentDirectory,

LPSTARTUPINFO lpStartupInfo,

LPPROCESS\_INFORMATION lpProcessInformation

);

При вызове CreateProcess, система создает объект ядра «процесс» (структуру данных, через которую операционная система управляет процессом). Затем система создает для нового процесса виртуальное адресное пространство и загружает в него код и данные исполняемого файла и необходимых библиотек (DLL). Далее система формирует объект ядра «поток» для первичного потока (primary) нового процесса. Первичный поток отвечает за инициализацию процесса, обработку системных сигналов и событий. Этот поток “живет” до того момента, когда управление не будет возвращено операционной системе для завершения процесса. Первичный поток начинает с исполнения стартового кода программы (для C/C++ это функции WinMain, wWinMain, main или wmain).

Рассмотрим параметры и возвращаемое значение функции CreateProcess.

Параметры lpApplicationName и lpCommandLine используются вместе для указания исполняемой программы и аргументов командной строки. Если файл с указанным именем не найден, Windows приступает к поиску заданного файла и делает это в следующем порядке:

1. Каталог, содержащий ЕХЕ-файл вызывающего процесса.

2. Текущий каталог вызывающего процесса.

3. Системный каталог Windows.

4. Основной каталог Windows.

5. Каталоги, перечисленные в переменной окружения PATH.

lpProcessAttributes и lpThreadAttributes указатели на структуры атрибутов защиты процесса и потока. Значениям NULL соответствует использование атрибутов защиты, заданных по умолчанию.

Флаг bInheritHandles показывает, наследует ли новый процесс открытый дескрипторы (файлов, файлов отображенных на память и т.д.) из вызывающего (родительского) процесса производящего запуск. Унаследованные дескрипторы имеют те же значения и права доступа, что и оригиналы.

Параметр dwCreationFlags задает параметры создания процесса и его приоритет. Некоторые возможные значения данного флага перечислены ниже.

|  |  |
| --- | --- |
| CREATE\_NEW\_CONSOLE | Новый процесс получает новую консоль вместо того, чтобы унаследовать родительскую. |
| CREATE\_NEW\_PROCESS\_GROUP | Создаваемый процесс - корневой процесс новой группы. |
| CREATE\_SUSPENDED | Первичная нить процесса создается в спящем (suspended) состоянии и не выполняется до вызова функции ResumeThread |
| REALTIME\_PRIORITY\_CLASS | Процесс получает наивысший приоритет – Real-time (реального времени). Потоки в этом процессе обязаны немедленно реагировать на события, обеспечивая выполнение критических по времени задач. Такие потоки вытесняют даже компоненты операционной системы. |
| HIGH\_PRIORITY\_CLASS | Потоки в процессе с приоритетом High (высокий) должны немедленно реагировать на события, обеспечивая выполнение критических по времени задач. Этот класс присвоен, например системной утилите Task Manager. |
| ABOVE\_NORMAL\_PRIORITY\_CLASS | Класс приоритета Above Normal (выше среднего), промежуточный между Normal и High. Данный класс впервые введен в Windows 2000. |
| NORMAL\_PRIORITY\_CLASS | Потоки в процессе c приоритетом Normal (нормальный) не предъявляют особых требований к выделению им процессорного времени. |
| BELOW\_NORMAL\_PRIORITY\_CLASS | Процесс получит класс приоритета Below Normal (ниже среднего), промежуточный между Normal и High. Данный класс впервые введен в Windows 2000. |
| IDLE\_PRIORITY\_CLASS | Потоки в процессе с приоритетом Idle выполняются, когда система не заняты другой работой. Этот класс приоритета обычно используется для утилит, работающих в фоновом режиме, экранных заставок и приложений, собирающих статистическую информацию. |

lpEnvironment казывает на блок переменных окружения нового процесса. Если передаваемое значение будет NULL, то будет использован блок родительского процесса. Блок среды это список строк с нулевым окончанием типа «имя=значение». Использование переменных окружения наиболее простой способ передачи информации в порождаемый процесс.

lpCurrentDirectory – указатель но строку, содержащую путь к текущему каталогу нового процесса. Если значение не задано, то в качестве текущего каталога будет использоваться каталог родительского (порождающего) процесса.

lpStartupInfo указатель на структуру STARTUPINFO, которая описывает внешний вид основного окна создаваемого процесса и содержит дескрипторы основных устройств нового процесса. Большинство приложений порождает процессы с атрибутами по умолчанию, в этом случае должны быть инициализировать все элементы структуры STARTUPINFO нулевыми значениями.

lpProcessInformation – указатель на структуру PROCESS\_INFORMATION, в которую Windows будут помещены, возвращаемые функцией значения дескрипторов и уникальные идентификаторы созданного процесса и его главного потока.

Как видно из описания функции CreateProcess, возвращаемое ей значение не является дескриптором созданного объекта типа HANDLE, как это происходит в большинстве функций. Если системе удастся создать новый процесс и его первичный поток, тогда CreateProcess вернет ненулевое значение, иначе будет возвращено нулевое значение.

#include "stdafx.h"

#include "windows.h"

using namespace std;

int \_tmain(int argc, \_TCHAR\* argv[])

{

STARTUPINFO cif;

ZeroMemory(&cif,sizeof(STARTUPINFO));

PROCESS\_INFORMATION pi;

if (CreateProcess("c:\\windows\\notepad.exe",NULL,

NULL,NULL,FALSE,NULL,NULL,NULL,&cif,&pi))

{

cout << "Process ID: " << pi.dwProcessId << endl;

cout << "Process handle: " << pi.hProcess << endl;

Sleep(5000); // задержка на 5 секунд

TerminateProcess(pi.hProcess,NO\_ERROR);// завершение } //процесса

}

### Функция WinExec

Вызов функции WinExec наиболее простой способ порождения нового процесса. Хотя эта функция реализована только для совместимости с Win16-приложениями, приложения Win32 должны реализовывать данную возможность через CreateProcess. Фактически WinExec является оберткой для CreateProcess - внутри WinExec происходит вызов CreateProcess с параметрами по умолчанию.

UINT WinExec(

LPCSTR lpCmdLine,

UINT uCmdShow

);

Параметр lpCmdLine – ссылка на заканчивающуюся нулевым значением строку, содержащую командную строку для запуска приложения.

Параметр uCmdShow задает способ отображения окна программы (SW\_HIDE, SW\_MAXIMIZE, SW\_MINIMIZE, SW\_SHOWNORMAL и т.д.).

Результатом функции WinExec является целое число. Если код возврата больше 31, то функция выполнена успешно, иначе в процессе выполнения функции произошла ошибка и возвращенное функцией значение следует сравнить со специальными константами для выяснения причины неудачи.

#include "stdafx.h"

#include "windows.h"

using namespace std;

int \_tmain(int argc, \_TCHAR\* argv[])

{

switch (WinExec("c:\\windows\\notepad.exe",SW\_SHOW))

{

case 0:

cout << "ERROR: The system is out of memory or resources.";

break;

case ERROR\_BAD\_FORMAT:

cout << "ERROR: The .exe file is invalid.";

break;

case ERROR\_PATH\_NOT\_FOUND:

cout << "ERROR: The specified path was not found.";

break;

case ERROR\_FILE\_NOT\_FOUND:

cout << "ERROR: The specified file was not found.";

break;

}

cin;

}

### Функция ShellExecute

Функция ShellExecute еще один простой вызов, который можно использовать для запуска программ. Эта функция во многом напоминает WinExec, однако поддерживает обработку типов файлов, зарегистрированных графической оболочкой операционной системы. Например, если при помощи ShellExecute попробовать запустить файл с расширением \*.txt**,** будет запущена программа Notepad или другая программа, которая используется в системе для просмотра текстовых файлов.

HINSTANCE ShellExecute(

HWND hwnd,

LPCTSTR lpOperation,

LPCTSTR lpFile,

LPCTSTR lpParameters,

LPCTSTR lpDirectory,

INT nShowCmd

);

Параметр hwnd - дескриптор окна родительского процесса, которое будет использовано для вывода сообщений при возникновении ошибок.

В качестве параметра lpOperation необходимо передать строку, указывающую какое действие следует выполнить с файлом. Допустимы команды «open»(открыть объект, который может быть исполняемым файлом, документом или папкой), «print» (распечатать файл) или «explore»(открыть папку для обзора в проводнике), «edit»(редактировать файл) и «find» (запускает процесс поиска в указанной папке). В качестве значения данного параметра можно передать NULL, в этом случае указанный объект будет открыт («open»).

Параметры lpFile и lpParameters используются для передачи в функцию имени файла и параметров командной строки (обычно NULL).

lpDirectory - это текущий каталог.

nShowCmd - задает способ отображения окна программы (аналогично WinExec).

Если код возврата функции больше 31, то функция выполнена успешно, иначе при выполнении функции произошла ошибка.

Функцию ShellExecuteможно использовать, например, для того, чтобы открыть корневой каталог диска С.

ShellExecute(handle, "open", "c:\\", NULL, NULL, SW\_SHOWNORMAL);

Практически аналогичным системным вызовом является вызов ShellExecuteEx. Этот вызов фактически является полным аналогом ShellExecute**,** однако в качестве аргумента он принимает указатель на структуру, поля которой во многом совпадают с аргументами вызова ShellExecute.Полезной особенностью функции является то, что ShellExecuteEx помещает в одно из полей этой структуры дескриптор вновь запущенного процесса.

## Завершение и прекращение выполнения процесса

Существует четыре варианта завершения процесса:

* входная функция первичного потока возвращает управление;
* один из потоков процесса вызывает функцию ExitProcess;
* поток другого процесса вызывает функцию TerminateProcess;
* все потоки процесса завершаются «по своей воле».

Рассмотрим каждый из перечисленных вариантов.

### Возврат управления входной функцией первичного потока

Приложение следует проектировать так, чтобы его процесс завершался только после возврата управления входной функцией первичного потока. В С++ это единственный способ, гарантирующий корректную очистку всех ресурсов, принадлежавших первичному потоку.

### Использование функции ExitProcess

Процесс завершается, когда один из его потоков вызывает ExitProcess:

VOID ExitProcess(UINT fuExitCode);

Эта функция завершает процесс и все его потоки, заносит в параметр fuExitCode код завершения процесса. Возвращаемого значения у ExitProcess нет, так как результат ее действия - завершение процесса. Вызов ExitProcess приводит к уничтожению процесса или потока, когда выполнение функции еще не завершилось.

### Использование функции TerminateProcess

Вызов функции TerminateProcess завершает процесс:

BOOL TerminateProcess( HANDLE hProcoss, UINT fuExitCode);

Особенность функции ExitProcess в том, что она может быть вызвана из любой потока произвольного процесса. Параметр hProcess идентифицирует описатель завершаемого процесса, а в параметре fuExitCode возвращается код завершения процесса.

Использовать TerminateProcess следует в случае, если иным способом завершить процесс не удается. Процесс не получает абсолютно никаких уведомлений о том, что он завершается, и приложение не может выполнить освобождение используемых ресурсов.

### Все потоки процесса завершаются

Может возникнуть ситуации, когда все потоки процесса завершены. Операционная система, обнаружив, что в процессе не исполняется ни один поток, немедленно завершает его. При этом код завершения процесса приравнивается коду завершения последнего потока.