Windows API a základní práce s procesy ve Windows

13. dubna 2022

Operační systém Microsoft Windows poskytuje velmi širokou škálu funkcí, které mohou programy pro svůj běh využívat, počínaje základní prací se systémem, jako je práce s procesy a soubory, přes komunikaci po síti až po tvorbu grafických aplikací. Tato funkcionalita se souhrně označuje jako Windows API (nebo zkráceně WinAPI), je poskytována jako sada funkcí jazyka C a práce s tímto rozhraním má svá specifika.

1 Typový systém WinAPI

Rozhraní WinAPI využívá vlastní sadu datových typů, které narozdíl od jazyka C mají jednoznačně definované velikosti a rozsahy hodnot.

Mezi ty nejčastěji používané datové typy patří.

`DWORD`	32bitové neznaménkové celé číslo
`WORD`	16bitové neznaménkové celé číslo
`BYTE`	8bitové neznaménkové celé číslo
`BOOL`	pravdivostní hodnota
`VOID`	neplatná hodnota
`LPDWORD`	ukazatel na hodnotu typu `DWORD`
`LPWORD`	ukazatel na hodnotu typu `DWORD`
`LPBYTE`	ukazatel na hodnotu typu `BYTE`
`LPBOOL`	ukazatel na hodnotu typu `BOOL`
`LPVOID`	ukazatel na hodnotu typu `VOID`
`LPSTR`	ukazatel na hodnotu typu `char`, řetezec obsahující jednobytové znaky (ANSI)
`LPWSTR`	ukazatel na hodnotu typu `wchar_t`, řetezec obsahující jednobytové znaky (UNICODE)
`HANDLE`	obecný identifikátor objektu (technicky `void *`)

Tento výčet není ani v nejmenším úplný, další typy lze nalézt v dokumentaci.[1] Může se stát, že datové typy, které používá WinAPI, nebudou zcela kompatibilní s typovým systémem jazyka C/C++. Zejména se to týká novějších verzí překladače, které jsou při práci s datovými typy striktnější. V takovém případě je nutné upravit nastavení projektu a povolit benevolentnější práci s typy a to následovně: *Project* → *Properties* → *C/C++* → *Language* → *Conformance Mode* → ***Default***.

[1] `https://docs.microsoft.com/en-us/windows/win32/winprog/windows-data-types`

2 Práce s řetězci

Další úskalí, které WinAPI přináší, spočívá v práci s řetězci. WinAPI umožňuje pracovat s dvěma typy řetězců, které jsou označovány jako ANSI a Unicode, kdy první typ řetězců používá pro reprezentaci znaků jednobytové hodnoty (typ `char`) a druhý typ používá „široké znaky" o velikost dva byty (typ `wchar_t`).

Pracuje-li některá z funkcí WinAPI s řetezci, je nabízena ve dvou variantách, s příponou A nebo W podle typu řetězců s kterými pracuje, např. `CreateFileA` (ANSI) nebo `CreateFileW` (Unicode). Při programování se reálně používá makro `CreateFile`, které se podle nastavení projektu expanduje na `CreateFileA` nebo `CreateFileW`.

Standardně jsou k dispozici funkce pro práci s oběma typy řetězců a programátor má na výběr ze dvou možností: (i) Buď si zvolit jeden typ řetězců, nastavit jej v projektu, a ten konzistentně v rámci jedné aplikace používat nebo (ii) použít pomocné sady maker v hlavičkovém souboru `tchar.h`, které se expandují na daný typ funkcí pro práci s řetězci podle nastavení projektu. Toto řešení je univerzálnější, ale zápis kódu se bude lišit od tradičního kódu v jazyce C.

Rozdíly při práci s jednotlivými typy znaků nastiňuje následující tabulka.

typ znaku	`char`	`wchar_t`	`_TCHAR`
řetězcový literál	`"abc"`	`L"abc"`	`_T("abc")`
hlavní funkce	`main`	`wmain`	`_tmain`
délka řetězce	`strlen`	`wcslen`	`_tcslen`
kopie řetezce	`strcpy`	`wcscpy`	`_tcscpy`
spojení řetezců	`strcat`	`wcscat`	`_tcscat`
porovnání řetezců	`strcmp`	`wcscmp`	`_tcscmp`
formátovaný výstup	`printf`	`wprintf`	`_tprintf`

Jaký typ znaků bude použit lze nastavit ve vlastnostech projektu: *Project* → *Properties* → *Advanced* → *Character Set*.

3 Rozhraní pro práci se soubory

Základní principy práce s WinAPI a to, jak pracovat s řetezci univerzálním způsobem, si ukážema na práci se soubory. Pro práci se soubory má Windows vlastní sadu funkcí[2]. Klíčovou funkcí je `CreateFile`, která slouží k vytvoření nebo otevření souboru.[3]

Funkce `CreateFile`[4], jako svůj argument bere cestu k souboru a další příznaky, jak se souborem pracovat. Jedná se jednak o režim přístupu k souboru (zda jej chcem číst nebo do něj zapisovat), režim souběžného přístupu, zda může být otevřený soubor otevřen ještě jednou (např. pro čtení), jak se zachovat, pokud soubor existuje nebo neexistuje. Další příznaky, jako bezpečností atributy, atributy souboru nebo šablonu souboru nebudeme využívat. Otevření nebo vytvoření souboru pro zápis i čtení ilustruje následující kód.

[2] `https://docs.microsoft.com/en-us/windows/win32/api/fileapi/`

[3] Název `CreateFile` přirozeně asociuje vytvoření souboru, proto může být matoucí, že funkce slouží i k otevření existujícího souboru. Ve skutečnosti se slovo Create vztahuje k vytvoření objektu, který reprezentuje soubor a v tomto kontextu je používáno konzistentně i v dalších částech WinAPI.

[4] `https://docs.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-createfilew`

```c
#include <windows.h>
#include <tchar.h>

int _tmain(int argc, TCHAR* argv[])
{
        HANDLE hFile = CreateFile(
                _T("foo.txt"),                  // cesta k souboru
                GENERIC_READ | GENERIC_WRITE,   // rezim prace se souborem
                FILE_SHARE_READ,                // rezim sdileneho pristupu k souboru (pro cteni)
                NULL,                           // bezpecnostni atributy
                OPEN_ALWAYS,                    // jak nalozit s (ne)existujicimi soubory
                0,                              // priznaky souboru
                NULL);                          // odkaz na sablonu

        if (hFile == INVALID_HANDLE_VALUE) {
                _tprintf(_T("error opening file\n"));
                return -1;
        }

        _TCHAR* str = _T("Hello world!");
        if (WriteFile(hFile, str, sizeof(_TCHAR) * _tcslen(str), NULL, NULL))
                _tprintf(_T("ok"));

        CloseHandle(hFile);
        return 0;
}
```

Všimněme si, že návratovou hodnotou funkce `CreateFile` je hodnota typu `HANDLE`. Jedná se o unikátní identifikátor (v rámci spuštěného procesu) pro jednotlivé objekty poskytované jádrem operačního systému. Datový typ `HANDLE` se používá i pro práci s dalšími objekty, např. procesy, vlákny, synchronizačními objekty atd. Technicky se jedná o typ `void *`, ale správně bychom tuto vlastnost neměli předjímat.

Pokud operace `CreateFile` z nějakého důvodu selže, je jako návratová hodnota vrácena konstanta `INVALID_HANDLE_VALUE`. Podrobnosti o chybě můžeme zjistit pomocí funkce `GetLastError()`.

V ukázkovém kódu je na řádku 21 ukázán zápis do souboru pomocí funkce `WriteFile`[5]. Svým chováním se funkce příliš neliší od funkce `fwrite` ze standardní knihovny jazyka C, má navíc dva argumenty. Jedním je možné získat počet zapsaných bytů (předáváme ukazatel na hodnotu, kam se výsledek zapíše), druhým je struktura umožňující asynchronní práci se souborem, což nebudeme využívat.

Práci se souborem je nutné ukončit jeho uzavřením, k tomu slouží funkcei `CloseHandle`. Tato funkce je obecná a slouží k práci i s dalšími objekty.

Úkoly

1. Spusťte program a podívejte se do vytvořeného souboru, ideálně s textovým editorem nebo prohlí-

[5] `https://docs.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-writefile`

žečem, který umí soubor zobrazit v hexadecimální podobě (např. Total Commander).

2. Změnte nastavení znaků v projektu a opakujte krok 1.

3. Upravte program tak, aby přečetl obsah vámi zvoleného souboru a vypsal jej na terminál. Použijte funkci `ReadFile`[6].

4 Vytvoření nového procesu

Vytvoření procesu je ve WinAPI přímočaré. Funkci `CreateProcess`[7] předáme název (resp. cestu) ke spouštěnému souboru, argumenty, a další příznaky či odkazy na struktury, kterými se vytvoření souboru má řídit. Důležité jsou dvě struktury `STARTUPINFO` a `PROCESS_INFORMATION`. První struktura slouží k předání doplňujících informací pro spouštěný proces (např. pozice okna). Druhá struktura slouží k předání informací o vytvořeném procesu, např. handle na vytvořený proces. Při inicializaci struktury `STARTUPINFO` je nutné do atributu `cb` zadat velikost struktury v bytech.

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

int _tmain(int argc, TCHAR* argv[])
{
        STARTUPINFO si;
        PROCESS_INFORMATION pi;

        ZeroMemory(&si, sizeof(si));
        si.cb = sizeof(si);
        ZeroMemory(&pi, sizeof(pi));

        // vytvori novy proces
        if (!CreateProcess(
        _T("C:\\WINDOWS\\system32\\notepad.exe"),          // cesta k programu
        _T("C:\\WINDOWS\\system32\\notepad.exe foo.txt"),  // uplny seznam argumentu
        NULL,           // nastaveni bezpecnostnich atributu
        NULL,           // nastaveni bezpecnostnich atributu
        FALSE,          // zakaz dedeni handlu meyi procesz
        0,              // priznaky pro vytvoreni procesu
        NULL,           // definice prostredi (pouzije se prostredi aktualniho procesu)
        NULL,           // pracovni adresar (pouzije se adresar aktualniho procesu)
        &si,            // ukazatel na strukturu s informacemi ke spoustenumu procesu
        &pi)            // ukazatel na strukturu, kterou jsou predany informace o vzniklem procesu
        ) {
                _tprintf(_T("CreateProcess failed (%d).\n"), GetLastError());
```

[6] https://docs.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-readfile

[7] https://docs.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-createprocessa

```
            return 1;
        }

        // ceka na ukonceni procesu
        WaitForSingleObject(pi.hProcess, INFINITE);

        // uzavre proces
        CloseHandle(pi.hProcess);
        CloseHandle(pi.hThread);
        return 0;
}
```

Úkol:

4. Rozšiřte ukázkový příklad o volání funkce `GetExitCodeProcess`[8], která vrací návratový kód ukončeného procesu, a tento kód vypište.

V rámci operačního systému má každý proces přiřazené číslo (*process id* nebo zkráceně *pid*). Pomocí funkce `OpenProcess`[9] můžeme získat handle na existující proces na základě tohoto identifikátoru a to například následovně:

```
HANDLE hProcess = OpenProcess(PROCESS_ALL_ACCESS, FALSE, pid);
```

Pokud máme dostatečná oprávnění, můžeme s procesem dále pracovat, například zjistit o něm informace nebo jej ukončit.

Úkol:

5. Spusťte vhodnou aplikaci (notepad, kalkulačku), v TaskManageru zjistěte jeho pid.
6. Pomocí funkce `GetProcessImageFileName`[10] zjistěte cestu ke spuštěnému souboru.
7. Pomocí funkce `TerminateProcess`[11] jej ukončete.

[8] https://docs.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-getexitcodeprocess
[9] `https://docs.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-openprocess`
[10] https://docs.microsoft.com/en-us/windows/win32/api/psapi/nf-psapi-getprocessimagefilenamea
[11] https://docs.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-terminateprocess