Procesy

Tento text sa zaoberá tým, ako systém riadi aplikácie. Začne vysvetlením, čo je to proces a ako systém vytvorí objekt jadra k riadeniu každého procesu. Následne ukáže ako manipulovať s procesom s využitím svojho objektu jadra. Budú prediskutované rôzne atribúty a vlastnosti procesu, tak ako aj niekoľko funkcií, ktoré ich umožňujú vyhľadávať a meniť. Taktiež sa bude venovať funkciám, ktoré dokážu vytvoriť dodatočné procesy v systéme. V závere sa venuje dôkladnému pohľadu na ukončenie procesov.

Proces je zvyčajne definovaný ako inštancia bežiaceho programu a pozostáva z dvoch zložiek:

- Objektu jadra, ktorý operačný systém používa k riadeniu procesu. Objekt jadra zároveň uchováva štatistické informácie o procese.
- Adresového priestoru, ktorý obsahuje kód a dáta všetkých spustiteľných aplikácií alebo DLL modulov. Obsahuje aj dynamicky pridelenú pamäť.

Procesy sú nečinné. Aby proces splnil akúkoľvek úlohu, musí mať vlákno, ktoré beží v jeho kontexte. Toto vlákno je zodpovedné za vykonanie kódu obsiahnutého v adresovom priestore procesu. V skutočnosti jeden proces môže obsahovať niekoľko vlákien, pričom každé vlákno vykonáva kód „súbežne“ v adresovom priestore procesu. Aby to bolo možné, každé z vlákien má svoju sadu CPU registrov a svoj vlastný zásobník. Každý proces má minimálne jedno vlákno, ktoré vykonáva kód v adresovom priestore procesu. Ak by neboli žiadne vlákna vykonávajúce kód v adresovom priestore procesu, nebol by ani žiadny dôvod, aby proces pokračoval a systém by automaticky zničil proces aj jeho adresový priestor.

Pri vytvorení procesu systém automaticky vytvorí aj jeho prvé vlákno, ktoré sa nazýva primárne vlákno. Toto vlákno môže potom vytvoriť ďalšie vlákna a tieto dokážu vytvoriť ešte ďalšie vlákna.

Vstupný bod procesu

Windows podporuje dva typy aplikácií: aplikácie založené na grafickom užívateľskom rozhraní (GUI) a aplikácie založené na konzolovom rozhraní (CUI). GUI aplikácie dokážu pracovať s oknami, majú menu, komunikujú s užívateľom pomocou dialógových okien, atď. Konzolové aplikácie sú založené na texte. Zvyčajne nevytvárajú okná a nevyžadujú GUI. Hranica medzi týmito typmi aplikácií je veľmi tenká. Je možné vytvoriť CUI aplikáciu, ktorá zobrazuje dialógové okná, ale aj GUI aplikáciu, ktorá ako výstup vypisuje textové reťazce na konzolu.

Pri vytváraní nového projektu vo Visual Studiu pomocou šablón patricich k Visual C++, sú v prostredí nastavené rôzne prepínače linkeru tak, aby linker vložil vhodný typ podsystému do výsledného programu. Tento prepínač má hodnotu /SUBSYSTEM:CONSOLE pre konzolové aplikácie a /SUBSYSTEM:WINDOWS pre GUI aplikácie. Keď užívateľ spustí aplikáciu, zavádzač operačného systému nazrie do zavádzacieho obrazu programu a vyberie túto hodnotu subsystému. Ak táto hodnota indikuje konzolovú aplikáciu, zavádzač sa automaticky uistí, že je pre aplikáciu vytvorené konzolové okno. V prípade, že hodnota indikuje GUI aplikáciu, zavádzač nevytvorí konzolové okno a jednoducho len načíta aplikáciu. V okamihu, keď je aplikácia spustená, operačný systém sa nestará o aké užívateľské rozhranie sa jedná.

Každá Windows aplikácia musí mať tzv. vstupný bod. Jedná sa o funkciu, ktorá je volaná pri spustení aplikácie. Existujú 4 možnosti:

```
int WINAPI WinMain(
    HINSTANCE hinstExe,
    HINSTANCE /* hinstExePrev*/,
    PSTR cmdLine,
    int cmdShow
    );

int WINAPI wWinMain(
    HINSTANCE hinstExe,
    HINSTANCE /*hinstExePrev*/,
    PWSTR cmdLine,
    int cmdShow
    );

int __cdecl main(
    int argc,
    char *argv[],
    char *envp[]
    );

int __cdecl wmain(
    int argc,
    wchar_t *argv[],
    wchar_t *envp[]
    );
```

Operačný systém ale v skutočnosti nevolá vstupný bod, ktorý napíšete. Namiesto toho volá zavádzaciu funkciu C/C++ run-time (CRT) modulu. Táto funkcia inicializuje CRT knižnicu, aby bolo možné volať funkcie ako napr. *malloc* a *free*. Takisto zaručuje, že akékoľvek globálne a statické C++ objekty, ktoré boli deklarované, sú správne skonštruované pred samotným vykonaním kódu. Nasledujúca tabuľka upresňuje, ktorý vstupný bod implementovať vo vašom kóde a kedy.

Typ aplikácie	Vstupný bod	Zavádzacia funkcia CRT modulu
GUI aplikácia s využitím ANSI znakov a reťazcov	*WinMain*	*WinMainCRTStartup*
GUI aplikácia s využitím Unicode znakov a reťazcov	*wWinMain*	*wWinMainCRTStartup*
CUI aplikácia s využitím ANSI znakov a reťazcov	*main*	*mainCRTStartup*
CUI aplikácia s využitím Unicode znakov a reťazcov	*wmain*	*wmainCRTStartup*

Za výber správnej zavádzacej funkcie CRT modulu pri linkovaní vášho programu je zodpovedný linker. Ak je prepínač linkeru špecifikovaný ako /SUBSYSTEM:WINDOWS, linker očakáva, že nájde buď *WinMain*, alebo *wWinMain* funkciu. Ak nie je prítomná ani jedna z týchto funkcií, linker vráti error „*unresolved external symbol*“. Inak vyberá adekvátnu z *WinMainCRTStartup* a *wWinMainCRTStartup* funkcií. Analogicky to funguje pri hodnote prepínača /SUBSYSTEM:CONSOLE a funkciách *main* a *wmain*.

Málo známym faktom je, že je možné odstrániť /SUBSYSTEM prepínač z projektu. Po odstránení sa linker rozhoduje, ktorý podsystém a zavádzaciu funkciu CRT modulu použiť, na základe toho, aký typ funkcie je použitý v kóde ako vstupný bod.

Častou chybou je nedopatrením zvoliť zlý typ projektu pri zakladaní nového projektu. Napríklad ak zvolíte nový projekt typu *Win32 Application* a ako vstupný bod zvolíte `main`. Pri vytvorení programu dostanete *link error* pretože *Win32 Application* projekt nastaví prepínač na /SUBSYSTEM:WINDOWS, ale žiadna `WinMain` alebo `wWinMain` funkcia v programe neexistuje. V tomto prípade máte 4 možnosti:

- Zmeňte `main` funkciu na `WinMain` funkciu. Toto však zvyčajne nie je najlepšia voľba, keďže ste pravdepodobne chceli vytvoriť konzolovú aplikáciu.

- Vytvorte nový *Win32 Console Application* projekt a skopírujte doňho existujúce zdrojové kódy. Táto možnosť je však zdĺhavá, pretože máte pocit, že začínate od začiatku a musíte zmazať pôvodný projekt.

- Vo Vlastnostiach projektu zvoľte záložku *Linker* a v časti *System* zmeňte /SUBSYSTEM: WINDOWS na /SUBSYSTEM:CONSOLE.

- Vo Vlastnostiach projektu zvoľte záložku *Linker* a v časti *System* úplne odstráňte prepínač /SUBSYSTEM:WINDOWS.

Všetky zavádzacie funkcie CRT modulu robia prakticky to isté. Rozdiel spočíva v tom, či spracovávajú ANSI alebo Unicode reťazce a ktorý vstupný bod volajú po inicialnizácii CRT knižnice. Pri inicializácii CRT modulu sa najskôr získa ukazateľ na príkazový riadok nového procesu. Následne sa získa ukazateľ na premenné prostredia nového procesu. Inicializujú sa globálne premenné CRT modulu a halda používaná alokačnými funkciami CRT modulu (`malloc, calloc`) a ďalšie vstupno-výstupné rutiny na najnižšej úrovni. Na záver sa volajú konštruktory pre všetky globálne a statické objekty C++ tried.

Po tejto inicializácii zavádzacia funkcia CRT modulu volá vami vytvorený vstupný bod. Volanie funkcie `wWinMain`:

```
GetStartupInfo(&StartupInfo);
int mainRetVal = wWinMain(
                          GetModuleHandle(NULL),
                          NULL,
                          commandLineUnicode,
                          (StartupInfo.dwFlags & STARTF_USESHOWWINDOW)
                            ? StartupInfo.wShowWindow : SW_SHOWDEFAULT
                          );
```

Volanie funkcie `WinMain`:

```
GetStartupInfo(&StartupInfo);
int mainRetVal = WinMain(
                          GetModuleHandle(NULL),
                          NULL,
                          commandLineAnsi,
                          (StartupInfo.dwFlags & STARTF_USESHOWWINDOW)
                            ? StartupInfo.wShowWindow : SW_SHOWDEFAULT
                          );
```

Volanie funkcie *wmain*:

```
int mainRetVal = wmain(__argc, __wargv, _wenviron);
```

Volanie funkcie *main*:

```
int mainRetVal = wmain(__argc, __wargv, _wenviron);
```

Po návrate z vášho vstupného bodu, zavádzacia funkcia volá *exit* funkciu CRT modulu s návratovou hodnotou vášho vstupného bodu (*mainRetVal*) ako parametrom. Funkcia *exit* vykoná nasledovné:

- Zavolá všetky funkcie registrované volaniami funkcie *_onexit*.
- Zavolá deštruktory pre všetky globálne a statické objekty C++ tried.
- Zavolá funkciu operačného systému *ExitProcess* a odovzdá jej *mainRetValue*. Toto spôsobí, že operačný systém ukončí váš proces a nastaví jeho exit kód.

Proces a jeho *instance handle*

Každému EXE alebo DLL súboru načítanému do adresového priestoru procesu je priradená unikátna *instance handle.* Hodnotu *instance handle* vašej aplikácie obsahuje prvý parameter *(w)WinMain* funkcie, *hinstExe*. Typicky je táto hodnota potrebná pri volaniach, ktoré pracujú so zdrojmi vášho programu. Mnoho aplikácii ukladá *hinstExe* parameter *(w)WinMain* funkcie do globálnych premenných, aby bol ľahko prístupný v celom kóde programu.

Aktuálna hodnota *hinstExe* parametru je adresa pamäte, kde systém zaviedol obraz spustiteľného súboru do adresového priestoru procesu. Túto adresu volí linker, prípadne je možné ju nastaviť pomocou prepínača /BASE:address. V tomto prípade je nutné voliť adresu 0x00400000 alebo vyššiu.

Funkcia *GetModuleHandle* vracia *handle*/adresu, kde je spustiteľný súbor, prípadne DLL súbor zavedený do adresového priestoru procesu:

```
HMODULE GetModuleHandle(PCTSTR module);
```

Pri volaní tejto funkcie sa ako parameter odovzdáva nulou ukončený reťazec, ktorý špecifikuje meno EXE, prípadne DLL súboru zavedeného do adresového priestoru volajúceho procesu. Ak systém nájde špecifikovaný súbor, vráti už spomínanú adresu. V prípade, že specifikovaný súbor nenájde, vracia `NULL`. Ako parameter *module* je možné odovzdať funkcii aj hodnotu `NULL`. V tomto prípade funkcia vráti základnú adresu volajúceho programu. Toto robí aj zavádzacia funkcia CRT modulu pri volaní vašej *(w)WinMain* funkcie.

Proces a jeho pôvodná *instance handle*

Ako ste si mohli všimnúť, parameter *hinstExePrev* funkcie *(w)WinMain* pri volaní zavádzacím kódom CRT modulu je vždy `NULL`. Tento parameter bol používaný v 16–bitovom Windowse a zostal formálnym parametrom *(w)WinMain* funkcie. V kóde by ste sa nikdy nemali odkazovať na tento parameter. Z tohto dôvodu je dobré vždy používať nasledovný zápis funkcie *(w)WinMain*:

```
int WINAPI WinMain(
    HINSTANCE hinstExe,
    HINSTANCE,
    PSTR cmdLine,
    int cmdShow);
```

Vzhľadom na to, že nie je uvedené žiadne meno druhého parametru, prekladač nevyhodí upozornenie „*parameter not referenced*“.

Proces a jeho príkazový riadok

Keď je vytvorený nový proces, je mu priradený príkazový riadok. Príkazový riadok nie je takmer nikdy prázdny. Prinajmenšom obsahuje názov programu, ktorý nový proces vytvoril. Keď sa začne vykonávať zavádzací kód CRT modulu, získa príkazový riadok , preskočí názov programu a odovzdá ukazateľ na zvyšok príkazového riadku *cmdLine* parametru funkcie *WinMain*.

Aplikácia môže analyzovať a interpretovať reťazec príkazového riadku ako chce. Dokonca je možné zapisovať do pamäťového bufferu, na ktorý ukazuje parameter *cmdLine*. Za žiadnych okolností by sa však nemalo zapisovať za koniec bufferu. Keď je nutné zmeniť príkazový riadok, je dobré spraviť si lokálnu kópiu a meniť túto kópiu.

Kompletný príkazový riadok môžete získať aj volaním funkcie *GetCommandLine*:

```
PTSTR GetCommandLine();
```

Táto funkcia vracia ukazateľ na buffer obsahujúci úplný príkazový riadok, vrátane celej cesty k spustiteľnému súboru.

Mnoho aplikácií uprednostňuje mať príkazový riadok rozdelený na jednotlivé tokeny. Aplikácia môže získať prístup k jednotlivým komponentám príkazového riadku použitím globálnych premenných *__argc* a *__argv*. Nasledujúca funkcia rozdelí akýkoľvek Unicode reťazec na jednotlivé tokeny:

```
PWSTR CommandLineToArgvW(
    PWSTR cmdLine,
    int* numArgs
    );
```

Ako naznačuje W na konci mena funkcie, táto funkcia existuje iba v Unicode verzii (W ako *wide*). Parameter *cmdLine* ukazuje na reťazec príkazového riadku. Väčšinou sa jedná o návratovú hodnotu skôr volanej funkcie *GetCommandLineW*. Parameter *numArgs* je nastavený na počet argumentov v príkazovom riadku. Funkcia *CommandLineToArgvW* vracia adresu poľa ukazateľov na Unicode reťazce.

Pri vykonávaní funkcie *CommandLineToArgvW* je interne alokovaná pamäť. Väčšina aplikácii túto pamäť neuvoľňuje a spolieha sa, že bude uvoľnená operačným systémom pri ukončení procesu. Tento spôsob je úplne v poriadku. Keby ste aj napriek tomu chceli uvoľniť túto pamäť sami, správny spôsob, ako to urobiť je volaním funkcie *HeapFree* nasledovne:

```
int numArgs;
PWSTR *argv = CommandLineToArgvW(GetCommandLineW(), &numArgs);
      // použitie argumentov
if (argv[1] == L”x”)
{
      …
}
      // uvoľnenie bloku pamäte
HeapFree(GetProcessHeap(), 0, argv);
```

Funkcia *CreateProcess*

Proces vytvoríte volaním funkcie *CreateProcess*:

```
BOOL CreateProcess(
    PCTSTR applicationName,
    PTSTR commandLine,
    PSECURITY_ATTRIBUTES processAttributes,
    PSECURITY_ATTRIBUTES threadAttributes,
    BOOL inheritHandles,
    DWORD creationFlags,
    PVOID environment,
    PCTSTR curDir,
    PSTARTUPINFO startInfo,
    PPROCESS_INFORMATION procInfo
    );
```

Keď vlákno volá funkciu *CreateProcess*, systém vytvorí objekt jadra procesu s iniciálnou hodnotou *usage count* 1. Tento objekt jadra procesu nie je procesom samotným, ale jedná sa o malú datovú štruktúru, ktorú OS využíva na správu procesu – jej obsah pozostáva zo štatistických údajov o procese. Systém potom vytvorí virtuálny adresový priestor pre nový proces a načíta kód, dáta programu a prípadné DLL súbory do adresového priestoru procesu.

Systém následne vytvorí objekt jadra primárneho vlákna nového procesu (*usage count* 1). Objekt jadra vlákna je tiež malá datová štruktúra používaná OS na správu vlákna. Toto primárne vlákno začína vykonávaním počiatočného kódu CRT modulu, ktorý následne volá vašu *(w)WinMain*, resp. *(w)main* funkciu. Ak systém úspešne vytvorí nový proces a jeho primárne vlákno, funkcia *CreateProcess* vráti `TRUE`.

Poznámka:
Funkcia `CreateProcess` *vráti hodnotu TRUE predtým, než je proces plne inicializovaný. To znamená, že zavádzač OS sa ešte nepokúsil lokalizovať všetky potrebné DLL súbory. Ak DLL súbor nemôže byť lokalizovaný, prípadne sa správne neinicializuje, proces sa ukončí. Keďže však funkcia* `CreateProcess` *vrátila hodnotu TRUE, rodičovský proces nie je upozornený na problémy s inicializáciou .*

Bližšie informácie o jednotlivých parametroch funkcie *CreateProcess* sú dostupné na adrese http://msdn.microsoft.com/en-us/library/ms682425%28VS.85%29.aspx

Ukončenie procesu

Proces je možné ukončiť štyrmi spôsobmi:

- Vstupný bod primárneho procesu sa vráti. (Táto možnosť je veľmi odporúčaná.)
- Jedno vlákno procesu zavolá funkciu *ExitProcess*. (Vyhnite sa tejto metóde.)
- Vlákno v inom procese zavolá funkciu *TerminateProcess*. (Vyhnite sa tejto metóde.)
- Všetky vlákna v procese sa ukončia sami od seba. (Táto situácia takmer nikdy nenastane.)

Vstupný bod primárneho procesu sa vráti

Pri vytváraní aplikácie by sa vždy malo dbať na to, aby jej proces bol ukončený až po návrate vstupného bodu primárneho vlákna. Toto je jediný spôsob, ktorý zaručuje, že všetky zdroje primárneho vlákna budú náležite uvoľnené. Návrat vstupého bodu primárneho vlákna zaručuje nasledujúce:

- Každý C++ objekt vytvorený týmto vláknom bude náležite zničený s použitím deštruktora.
- Operačný systém vhodne uvoľní pamäť používanú zásobníkom vlákna.
- Systém nastaví exit kód procesu ako návratovú hodnotu vášho vstupného bodu.
- Systém dekrementuje *usage count* objektu jadra procesu.

Funkcia *ExitProcess*

Proces je ukončený, keď jedno z vlákien procesu zavolá funkciu *ExitProcess*:

```
VOID ExitProcess(UINT exitCode);
```

Táto funkcia ukončí proces a nastaví exit kód procesu na hodnotu *exitCode*. Funkcia *ExitProcess* nevracia žiadnu hodnotu, pretože po jej zavolaní je proces ukončený a akýkoľvek kód umiestnený za volanie tejto funkcie sa nikdy nevykoná.

Pri návrate vstupného bodu primárneho vlákna do počiatočného kódu CRT modulu, tento kód náležite uprace všetky zdroje CRT modulu používané procesom. Následne je volaná funkcia *ExitProcess* s hodnotou parametru odpovedajúcou návratovej hodnote funkcie vstupného bodu. To vysvetľuje, prečo je po návrate vstupného bodu primárneho vlákna ukončený celý proces. Všetky ďalšie vlákna, ktoré bežia v procese v tom čase, sú ukončené spolu s procesom.

Funkcia *TerminateProcess*

Proces je možné ukončiť aj použitím funkcie *TerminateProcess*:

```
BOOL TerminateProcess(
    HANDLE processHandle,
    UINT exitCode
    );
```

Táto funkcia sa od funkcie *ExitProcess* výrazne líši v jednej veci: akékoľvek vlákno môže zavolať funkciu *TerminateProcess* a ukončiť tak svoj vlastný alebo iný proces. Parameter *processHandle* identifikuje proces, ktorý má byť ukončený a parameter *exitCode* špecifikuje exit kód ukončeného procesu.

Funkcia *TerminateProcess* by mala byť použitá iba v prípade, že nie je možné ukončiť proces iným spôsobom. Proces, ktorý má byť takýmto spôsobom ukončený, totiž vôbec nie je upozornený na to, že končí – aplikácia nestihne náležite upratať a nemôže sa ukončeniu brániť.

Kým však proces nedostane žiadnu šancu na vlastné upratanie, operačný systém po procese kompletne uprace, takže pamäť používaná procesom je uvoľnená, otvorené súbory sa zatvoria, *usage count* všetkých objektov jadra je dekrementovaný a všetky User a GDI objekty sú zničené.

Všetky vlákna v procese sa ukončili

Ak dôjde k ukončeniu všetkých vlákien v procese (či už volaním funkcie *ExitThread*, alebo boli ukončené funkciou *TerminateThread*), operačný systém usúdi, že už nie je žiadny dôvod udržiavať adresový priestor procesu. Nakoľko už žiadne vlákno nevykonáva v danom adresovom priestore žiadny kód,

je táto úvaha správna. Po zistení, že žiadne vlákno v procese už nebeží, systém proces ukončí. V takomto prípade exit kód procesu je ten istý ako exit kód posledného ukončeného vlákna.

Keď sa proces končí...

...prebiehajú nasledujúce akcie:

- Všetky zostávajúce vlákna v procese sú ukončené.
- Všetky User a GDI objekty alokované procesom sú uvoľnené a všetky objekty jadra sú ukončené – pokiaľ nie sú používané iným procesom.
- Exit kód procesu sa zmení zo `STILL_ACTIVE` na kód odovzdaný funkcii *`ExitProcess`* alebo *`TerminateProcess`*.
- Stav objektu jadra procesu je signalizovaný. Ostatné vlákna v systéme sa do ukončenia procesu môžu pozastaviť.
- *Usage count* objektu jadra procesu je dekrementovaný o 1.

Všimnite si, že objekt jadra procesu vždy žije aspoň tak dlho ako proces samotný. Objekt jadra procesu však môže žiť aj po skončení procesu. Pri skončení procesu je automaticky dekrementovaný *usage count* objektu jadra procesu. Pokiaľ iný proces v systéme má otvorenú *handle* objektu jadra končiaceho procesu, *usage count* tohto objektu nebude 0. Toto zvyčajne nastane, keď rodičovské procesy zabudnú zavrieť *handle* child procesu. Je to však výhoda a nie chyba. Objekt jadra procesu totižto udržiava štatistické informácie o procese a tieto informácie môžu byť užitočné aj po ukončení procesu. Napríklad môžete chcieť zistiť koľko času CPU proces vyžaduje.

Child procesy

Pri vytváraní aplikácií sa môžete stretnúť so situáciou, kedy chcete, aby prácu vykonal iný blok kódu. Takúto prácu väčšinou priradíte nejakým funkciám alebo podprogramom. Keď však voláte funkciu, váš kód nemôže pokračovať v práci, kým sa funkcia neskončí. V mnohých situáciách je toto konanie žiadúce. Ďalší možný spôsob je vytvoriť v rámci procesu ďalšie vlákno. To umožní vášmu kódu pokračovať, kým druhé vlákno vykoná požadovanú prácu. Táto technika je užitočná, ale vytvára synchronizačné problémy v okamihu, keď pôvodné vlákno potrebuje výsledky nového vlákna.

Ďalšou možnosťou je vytvorenie nového procesu – tzv. child procesu. Je to aj jeden zo spôsobov ako ochrániť adresový priestor pôvodného procesu – pri vytvorení iba nového vlákna v rámci procesu môže dôjsť k prepísaniu niečoho dôležitého v adresovom priestore, čo môže viesť k nepredvídaným výsledkom. Po vytvorení nového procesu môžete čakať, kým sa tento proces ukončí a až potom pokračovať v ďalšej práci, alebo môžete pokračovať v práci aj za behu nového procesu.

Nový proces však pravdepodobne bude potrebovať pracovať aj s dátami obsiahnutými v adresovom priestore rodičovského procesu. V tomto prípade je dobré, ak child proces beží vo svojom vlastnom adresovom priestore a jednoducho mu je povolený prístup k relevantným dátam v adresovom priestore rodičovského procesu.Týmto spôsobom sú chránené všetky dáta, ktoré pre child proces nie sú relevantné. Windows ponúka niekoľko metód prenosu dát medzi rôznymi procesmi: *dynamic data exchange* (DDE), OLE, *pipes, mailslots*,... Jednou z najvhodnejších metód ako zdieľať dáta je použitie tzv. *memory-mapped* súborov.

Pokiaľ chcete vytvoriť nový proces, ktorý má niečo spraviť a chcete počkať na výsledok, môžete použiť štruktúru kódu podobnú tejto:

```
PROCESS_INFORMATION pi;
DWORD exitCode;
```

```
// vytvorenie child procesu
BOOL success = CreateProcess(…, &pi);
if (success)
{
      // zatvorte handle vlákna hneď ako nie je viac potrebná!
    CloseHandle(pi.Thread);

      // suspenduj naše spracovávanie, kým sa child proces neukončí
    WaitForSingleObject(pi.hProcess, INFINITE);

      // child process bol ukončený, získaj jeho exit kód
    GetExitCodeProcess(pi.hProcess, &exitCode);

      // zatvorte handle procesu hneď ako nie je viac potrebná!
    CloseHandle(pi.hProcess);
}
```

V danom kóde je vytvorený nový proces. Ak je vytvorenie úspešné, volá sa funkcia *WaitForSingleObject:*

```
DWORD WaitForSingleObject(HANDLE objectHandle, DWORD timeout);
```

Táto funkcia čaká, kým objekt identifikovaný parametrom *objectHandle* nie je signalizovaný. V tomto prípade je tento objekt signalizovaný v okamihu ukončenia child procesu. V praxi to znamená, že funkcia *WaitForSingleObject* pozastaví rodičovský proces až do doby, kedy je ukončený child proces. Následne je možné zistiť exit kód child procesu.

Volania funkcie *CloseHandle* v predchádzajúcej časti kódu spôsobia, že systém dekrementuje *usage count* objektov vlákna a procesu na 0, čím umožní ich uvoľnenie z pamäte. *Handle* primárneho vlákna child procesu je zatvorená hneď po návrate funkcie *CreateProcess*. *Usage count* objektu primárneho vlákna child procesu je tým dekrementovaný na 0, ale samotné vlákno nie je ukončené. To znamená, že v okamihu, keď je ukončené vlákno, bude uvoľnený aj objekt primárneho vlákna child procesu z pamäte – nie je nutné čakať, kým rodičovské vlákno uvoľní *handle* child objektu vlákna.

Samostatné child procesy

Vo väčšine prípadov aplikácia spustí ďalší proces ako samostatný proces. To znamená, že po tom, ako je nový proces vytvorený a spustený, rodičovský proces s ním viac nepotrebuje komunikovať alebo nevyžaduje, aby bol ukončený skôr, než rodičovský proces bude pokračovať.

Pre prerušenie všetkých väzieb k child procesu je nutné zatvoriť *handle* nového procesu a jeho primárneho vlákna. Nasledujúca ukážka kódu je príklad ako vytvoriť nový proces a nechať ho bežať ako proces samostatný:

```
PROCESS_INFORMATION pi;
// vytvorenie child procesu
BOOL success = CreateProcess(…, &pi);
if (success)
{
      // dovoľte systému zničiť objekty jadra procesu a vlákna
      // hneď ako sa child proces ukončí!!
    CloseHandle(pi.Thread);
    CloseHandle(pi.hProcess);
}
```

Použitá literatúra:

RICHTER, Jeffrey. CLARK, Jason D. *Programming Server-Side Applications for Microsoft Windows 2000*. 1. vyd. 2000. ISBN 0-7356-0753-2

Microsoft Developer Network, domovská www stránka, dostupná na URL http://msdn.microsoft.com