I/O zariadení a medzivláknová komunikácia

Tento text pokrýva dve témy, ktoré sú základné pri vytváraní vysoko výkonných a škálovateľných aplikácií: I/O zariadení a medzivláknovú komunikáciu. Škálovateľná aplikácia zvláda rovnako efektívne veľké aj malé množstvo súbežných operácií.

Vlákna sú najlepším prostriedkom k deleniu práce vo Windows aplikáciách. Každé vlákno je pridelené procesoru, čo umožňuje multiprocesorovým počítačom vykonávať niekoľko operácií súbežne, čím sa zvyšuje výkon aplikácie. Keď vlákno vydá požiadavok na I/O nejakého zariadenia, je dočasne pozastavené a čaká, kým zariadenie skončí I/O požiadavok. Toto pozastavenie znižuje výkon, pretože vlákno nemôže robiť žiadnu užitočnú činnosť. Vy však chcete, aby vlákno vykonávalo užitočnú činnosť po celý čas.

Aby ste vlákna udržali zamestnané, musíte ich donútiť spolu komunikovať. K tomu slúžia rôzne komunikačné mechanizmy, ktorých použitím vlákna dosiahnu výrazný výkon zapisovaním a čítaním zo zariadení bez čakania na odpoveď zariadení.

Otváranie a zatváranie zariadení

Windows podporuje veľké množstvo zariadení. V tomto kontexte sa bude ako zariadenie chápať hocičo, čo umožňuje komunikáciu. V nasledujúcej tabuľke je zoznam niektorých zariadení a ich najčastejších využití:

Zariadenie	Najčastejšie využitie
Súbor	Permanentné úložište rôznych dát.
Adresár	Nastavenia atribútov a kompresií súborov.
Logická disková mechanika	Formátovanie mechaniky.
Fyzická disková mechanika	Prístup k *partition table*.
Sériový port	Prenos dát cez telefónnu linku.
Paralelný port	Prenos dát k tlačiarni.
Mailslot	One-to-many prenos dát, zvyčajne cez sieť k počítaču s Windows.
Named pipe	One-to-one prenos dát, zvyčajne cez sieť k počítaču s Windows.
Anonymous pipe	One-to-one prenos dát na tom istom počítači (nikdy nie cez sieť).
Socket	Datagramový alebo streamový prenos dát, zvyčajne cez sieť k počítaču podporujúcemu sockety.
Konzola	Buffer textového okna.

Windows sa snaží čo najviac skryť rozdiely medzi jednotlivými zariadeniami pred vývojármi. To znamená, že v okamihu, keď otvoríte niektoré zariadenie, Windows funkcie pre zápis a čítanie z neho sú tie isté, nezávisle na zariadení, s ktorým komunikujete. Asi najbežnejším zariadením sú súbory.

Aby ste mohli previesť ktorúkoľvek z I/O operácií, musíte najskôr otvoriť požadované zariadenie a získať k nemu *handle*. Nasledujúca tabuľka uvádza, ako získať *handle* k jednotlivým zariadeniam:

Zariadenie	Funkcia použitá k otvoreniu zariadenia
Súbor	`CreateFile`(`pszName` je cesta k súboru)
Adresár	`CreateFile`(`pszName` je názov adresára). Adresár otvoríte špecifikovaním príznaku `FILE_FLAG_BACKUP_SEMANTICS`.

Logická disková mechanika	*CreateFile*(*pszName* je "\\.\x:"), kde x je označenie mechaniky (\\.\C:).
Fyzická disková mechanika	*CreateFile*(*pszName* je "\\.\PHYSICALDRIVEx"), kde x je číslo fyzického disku.
Sériový port	*CreateFile*(*pszName* je "COMx")
Paralelný port	*CreateFile*(*pszName* je "LPTx")
Mailslot server	*CreateMailslot*(*pszName* je "\\.\mailslot*mailslotname*")
Mailslot klient	*CreateFile*(*pszName* je "*servername*\mailslot*mailslotname*")
Named pipe server	*CreateNamedPipe*(*pszName* je "\\.\pipe*pipename*")
Named pipe klient	*CreateFile*(*pszName* je "*servername*\pipe*pipename*")
Anonymous pipe	*CreatePipe* klient a server
Socket	*socket, accept*, alebo *AcceptEx*
Konzola	*CreateConsoleScreenBuffer* alebo *GetStdHandle*

Každá z funkcií uvedených v predchádzajúcej tabuľke vracia *handle* k danému zariadeniu. Túto *handle* môžete odovzdať ako parameter rôznym funkciám pracujúcim so zariadeniami. Keď ste skončili manipuláciu s daným zariadením, musíte ho zatvoriť. To sa deje pomocou volania funkcie *CloseHandle*:

```
BOOL CloseHandle (HANDLE hObject);
```

Ak je však zariadením *socket*, musíte volať miesto toho funkciu *closesocket*:

```
int closesocket(SOCKET s);
```

Pomocou *handle* k zariadeniu zistíte, o aké zariadenie sa jedná volaním funkcie *GetFileType*:

```
DWORD GetFileType(HANDLE hDevice);
```

Túto funkciu voláte s *handle* zariadenia ako parametrom. Návratová hodnota funkcie je jedna z nasledovných:

- **FILE_TYPE_UNKNOWN** – typ špecifikovaného súboru je neznámy.
- **FILE_TYPE_DISK** – špecifikovaný súbor je diskový súbor.
- **FILE_TYPE_CHAR** – špecifikovaný súbor je znakový súbor, typicky LPT zariadenie alebo konzola.
- **FILE_TYPE_PIPE** – špecifikovaný súbor je buď *named*, alebo *anonymous pipe*.

Detailný pohľad na funkciu *CreateFile*

Funkcia *CreateFile*, pochopiteľne, vytvára a otvára súbory, ale nenechajte sa zmiasť jej názvom – otvára aj mnoho iných zariadení:

```
HANDLE CreateFile(
    PCTSTR pszName,
    DWORD desiredAccess,
    DWORD shareMode,
    PSECURITY_ATTRIBUTES psa,
    DWORD creationDistribution,
    DWORD flagsAndAttrs,
    HANDLE hFileTemplate
    );
```

Ako si môžete všimnúť, funkcia *CreateFile* má celkom veľa parametrov. To umožňuje veľkú flexibilitu pri otváraní zariadenia. Hneď prvý parameter *pszName* identifikuje typ zariadenia, prípadne špecifickej inštancie zariadenia.

Parametrom *desiredAccess* určíte za akým účelom chcete k zariadeniu pristupovať. Máte 4 možnosti:

- **0** – Zo zariadenia nechcete čítať, ani doňho zapisovať. Túto hodnotu zadajte, ak chcete zmeniť konfiguračné nastavenia zariadenia.

- **GENERIC_READ** – Umožňuje *read-only* prístup k zariadeniu.

- **GENERIC_WRITE** – Umožňuje *write-only* prístup k zariadeniu.

- **GENERIC_READ | GENERIC_WRITE** – Umožňuje zápis aj čítanie.

Parameter *shareMode* špecifikuje privilégiá zdieľania daného zariadenia. Je pravdepodobné, že k jednému zariadeniu môže a aj bude pristupovať v tom istom čase niekoľko počítačov (v sieti) alebo niekoľko procesov (vo viacvláknovom prostredí). To znamená, že musíte myslieť na to, či máte obmedziť ostatné počítače a procesy v prístupe k dátam zariadenia a ako to urobiť. Parameter *shareMode* môže nadobúdať nasledujúce hodnoty:

- **0** – Žiadny iný proces nebude mať prístup k danému zariadeniu. Ak je zariadenie otvorené iným procesom, volanie vašej *CreateFile* funkcie zlyhá. Ak zariadenie úspešne otvoríte, zlyhá volanie funkcie *CreateFile* inými procesmi.

- **FILE_SHARE_READ** – Požadujete, aby žiadny iný proces nemohol zapisovať do zariadenia. Ak je zariadenie otvorené iným procesom pre zápis alebo výlučný prístup, volanie vašej *CreateFile* funkcie zlyhá. Ak zariadenie úspešne otvoríte, zlyhá volanie funkcie *CreateFile* inými procesmi, ak požadujú GENERIC_WRITE prístup.

- **FILE_SHARE_WRITE** - Požadujete, aby žiadny iný proces nemohol čítať zo zariadenia. Ak je zariadenie otvorené iným procesom pre čítanie alebo výlučný prístup, volanie vašej *CreateFile* funkcie zlyhá. Ak zariadenie úspešne otvoríte, zlyhá volanie funkcie *CreateFile* inými procesmi, ak požadujú GENERIC_READ prístup.

- **FILE_SHARE_READ | FILE_SHARE_WRITE** – Nezáleží na tom, či zo zariadenia číta, alebo doňho zapisuje nejaký iný proces. Ak je zariadenie otvorené iným procesom pre výlučný prístup, volanie vašej *CreateFile* funkcie zlyhá. Ak zariadenie úspešne otvoríte, zlyhá volanie funkcie *CreateFile* inými procesmi, ak požadujú výlučné čítanie, zápis alebo čítanie/zápis.

Parameter *psa* ukazuje na štruktúru SECURITY_ATTRIBUTES, ktorá umožňuje špecifikovať bezpečnostné informácie a či chcete dediť *handle*, ktorú vráti funkcia *CreateFile*. Väčšinou je tento parameter NULL.

Parameter *creationDistribution* má význam pri otváraní súborov. Hodnoty tohto parametru môžu byť nasledovné:

- **CREATE_NEW** – Funkcia vytvorí nový súbor. Ak súbor s tým istým menom už existuje, funkcia zlyhá.

- **`CREATE_ALWAYS`** – Funkcia vždy vytvorí nový súbor nezávisle na tom, či súbor s daným menom už existuje. Ak existuje, funkcia *`CreateFile`* ho prepíše.

- **`OPEN_EXISTING`** – Funkcia otvorí existujúci súbor alebo zariadenie. Funkcia zlyhá, ak súbor /zariadenie neexistuje.

- **`OPEN_ALWAYS`** – Funkcia otvorí existujúci súbor, prípadne vytvorí nový, ak daný súbor neexistuje.

- **`TRUNCATE_EXISTING`** – Funkcia otvorí existujúci súbor a skráti jeho veľkosť na 0 bytov. Ak súbor neexistuje, funkcia zlyhá.

Poznámka:
Ak voláte funkciu `CreateFile` na otvorenie iného zariadenia ako súboru, ako parameter `creation-Distribution` musíte odovzdať hodnotu `OPEN_EXISTING`.

Parameter *`flagsAndAttrs`* má dva účely: umožňuje nastaviť príznaky vylepšujúce komunikáciu so zariadením a ak je zariadením súbor, môžete nastaviť jeho atribúty. Väčšina týchto komunikačných príznakov sú signály, ktoré povedia systému, ako chcete pristupovať k zariadeniu. Systém môže následne optimalizovať *caching* algoritmy, aby vaša aplikácia pracovala efektívnejšie.

Cache príznaky funkcie *`CreateFile`*

`FILE_FLAG_NO_BUFFERING` Špecifikovaním tohto príznaku hovoríte cache manažérovi, že nechcete, aby vytváral buffer pre žiadne dáta – zodpovednosť za to preberáte sami. V závislosti na tom, čo budete robiť, tento príznak môže vylepšiť rýchlosť a využitie pamäti vašej aplikácie. Pretože ovládač súborového systému má zapisovať dáta priamo do bufferov, ktoré mu poskytnete, musíte dodržiavať určité pravidlá:

- K súboru musíte vždy pristúpiť použitím offsetov, ktoré majú presný násobok veľkosti sektoru disku. (Veľkosť sektoru disku zistíte volaním funkcie *`GetDiskFreeSpace`*.)
- Musíte vždy prečítať/zapísať počet bytov, ktorý je presným násobkom veľkosti sektoru.
- Musíte sa uistiť, že buffer vo vašom adresovom priestore procesu začína na adrese, ktorá je celočíselne deliteľná veľkosťou sektoru.

`FILE_FLAG_SEQUENTIAL_SCAN a FILE_FLAG_RANDOM_ACCESS` Tieto príznaky sú užitočné iba ak dovolíte systému, aby načítal dáta do cache pamäti. Obidva tieto príznaky budú ignorované, ak budete špecifikovať príznak `FILE_FLAG_NO_BUFFERING`. Pri použití príznaku `FILE_FLAG_SEQUENTIAL_SCAN` si systém bude myslieť, že pristupujete k súboru sekvenčne. Pri čítaní dát zo súboru systém v skutočnosti načíta viac dát, než je požadované. Predpokladá totiž, že budete chcieť súbor čítať aj ďalej a znižuje tak počet prístupov priamo na disk. Ak však nepostupujete sekvenčne, je lepsie použiť príznak `FILE_FLAG-_RANDOM_ACCESS`. Pri extrémne veľkých súboroch je nutné použiť príznak `FILE_FLAG_NO_BUFFERING`.

`FILE_FLAG_WRITE_THROUGH` Pri použití tohto príznaku systém zapíše všetky zmeny v súbore priamo na disk. Systém však stále spravuje vnútornú cache pre dáta zo súboru. Čítanie dát uprednostňuje dáta načítané do cache, než dáta priamo z disku. Ak je tento príznak použitý pri otvorení súboru na serveri, Windows funkcia zapisujúca do súboru sa nevráti, až kým dáta nie sú zapísane na disk serveru.

Rôzne príznaky funkcie *CreateFile*

FILE_FLAG_DELETE_ON_CLOSE Systém zmaže súbor po zatvorení všetkých k nemu otvorených *handles*.

FILE_FLAG_BACKUP_SEMANTICS Tento príznak použite pri zálohovacom a obnovovacom software. Pred otvorením alebo vytvorením súboru systém zistí, či proces, ktorý sa o túto akciu pokúša, má požadované prístupové privilégiá. Pomocou tohto príznaku je možné aj získať *handle* k adresáru.

FILE_FLAG_POSIX_SEMANTICS Tento príznak spôsobí, že funkcia *CreateFile* použije *case-sensitive* hľadanie názvu súboru pri vytváraní/otváraní súboru.

FILE_FLAG_OPEN_REPARSE_POINT Tento príznak hovorí systému, aby ignoroval *reparse* atribúty súboru – ak nejaké existujú. *Reparse* atribúty umožňujú filtru súborového systému zmeniť chovanie otrvárania, čítania, písania a zatvárania súboru. Väčšinou je zmenené chovanie žiadané, takže použitie tohto príznaku nie je odporúčané.

FILE_FLAG_OPEN_NO_RECALL Tento príznak hovorí systému, aby neukladal obsah súboru z offline úložišťa (napr. páska) naspäť do online úložišťa (napr. pevný disk).

FILE_FLAG_OVERLAPPED Tento príznak hovorí systému, že chcete k zariadeniu pristupovať asynchrónne. Implicitne sa k zariadeniam pristupuje synchrónne. Keď čítate dáta zo súboru, vaše vlákno je pozastavené, čakajúce na prečítané informácie. Po prečítaní dát opäť získava kontrolu a pokračuje. Pretože I/O operácie sú v porovnaní s inými operáciami pomalé, môžete zvážiť možnosť komunikovať s nejakým zariadením asynchrónne. Pri asynchrónnom prístupe ide o to, že zavoláte funkciu na načítanie, alebo zapísanie dát, ale namiesto čakania na ukončenie I/O sa vaše volanie okamžite vráti a operačný systém dokončí I/O vo vašom mene použitím svojho vlastného vlákna. Keď systém skončí vami vyžiadanú I/O operáciu, ste na to upozornení. Asynchrónne I/O operácie sú kľúčom k vytváraniu vysokovýkonných služieb. Windows ponúka niekoľko rôznych metód asynchrónneho I/O, ktoré budú rozobrané neskôr v tomto texte.

Príznaky atribútov súboru

Jedná sa o príznaky atribútov odovzdávané funkcii *CreateFile* ako parameter *flagsAndAttrs*. Tieto príznaky sú úplne ignorované systémom pokiaľ nevytvárate úplne nový súbor alebo nedáte funkcii *CreateFile* NULL ako parameter *hFileTemplate*. Jednotlivé príznaky sú nasledujúce:

- FILE_ATTRIBUTE_ARCHIVE – súbor je archívom.
- FILE_ATTRIBUTE_ENCRYPTED – súbor je zakódovaný.
- FILE_ATTRIBUTE_HIDDEN – súbor je skrytý. Nebude zahrnutý v obyčajnom zobrazení adresára.
- FILE_ATTRIBUTE_NORMAL – súbor nemá iné atribúty. Tento príznak je platný len ak je použitý samostatne.
- FILE_ATTRIBUTE_NOT_CONTENT_INDEXED – súbor nebude indexovaný *content indexing* službou.
- FILE_ATTRIBUTE_OFFLINE – súbor existuje, ale jeho dáta boli presunuté do offline úložišťa.
- FILE_ATTRIBUTE_READONLY – súbor je read-only. Aplikácie z neho môžu čítať, ale nesmú doňho zapisovať, ani ho zmazať.
- FILE_ATTRIBUTE_SYSTEM – súbor je časťou operačného systému alebo je používaný výlučne operačným systémom.
- FILE_ATTRIBUTE_TEMPORARY – dáta súboru budú používané iba krátku dobu. Súborový systém sa snaží držať dáta tohto súboru v RAM namiesto disku, aby bol prístupový čas čo najnižší.

Popri všetkých týchto príznakoch ešte existujú príznaky špecifické pri použití funkcie *CreateFile* na otvorenie *named pipe*. Tieto príznaky však už nebudú rozoberané v tomto texte.

Posledným parametrom *CreateFile* funkcie je *hFileTemplate*. Tento parameter identifikuje *handle* otvoreného súboru, alebo je NULL. Ak obsahuje *handle*, funkcia ignoruje príznaky atribútov uvedené vo *flagsAndAttrs* parametri a používa atribúty asociované so súborom identifikovaným s *hFileTemplate*. Tento súbor musí byť otvorený ako GENERIC_READ. Ak však funkcia *CreateFile* otvára už existujúci súbor, parameter *hFileTemplate* je ignorovaný.

Ak funkcia *CreateFile* uspeje, návratová hodnota je *handle* k zariadeniu. Ak zlyhá, návratová hodnota je INVALID_HANDLE_VALUE!!

Práca so súbormi

Prvá vec na ktorú si musíte dať pozor je, že Windows bol navrhnutý tak, aby pracoval s extrémne veľkými súbormi. Namiesto reprezentovania veľkosti súboru 32-bitovou hodnotou, Windows používa 64-bitovú hodnotu. To znamená, že súbor môže teoreticky dosiahnuť veľkosť 16EB.

Pracovať so 64-bitovými hodnotami v 32-bitovom operačnom systéme je trochu nepríjemné, nakoľko mnoho Windows funkcií vyžaduje zadať 64-bitovú hodnotu ako dve 32-bitové hodnoty. V každodennom živote však väčšinou nepracujete so súbormi väčšími ako 4GB, čo v praxi znamená, že hodnota *HighPart* 64-bitovej veľkosti súboru bude 0.

Získanie veľkosti súboru

Pri práci so súbormi je často nutné získať veľkosť súboru. To je možné volaním funkcie *GetFileSizeEx*:

```
BOOL GetFileSizeEx(
    HANDLE hFile,
    PLARGE_INTEGER fileSize
    );
```

Prvý parameter je *handle* k otvorenému súboru a *fileSize* parameter je adresa LARGE_INTEGER unionu. Tento union umožňuje vyjadriť 64-bitovú znamienkovú hodnotu pomocou dvoch 32-bitových hodnôt alebo ako jednu 64-bitovú hodnotu:

```
typedef union _LARGE_INTEGER{
    struct {
       DWORD LowPart;      // Low 32-bit unsigned value
       LONG HighPart;      // High 32-bit signed value
    };
    LONGLONG QuadPart;     // Full 64-bit signed value
} LARGE_INTEGER, *PLARGE_INTEGER;
```

Ďalšou užitočnou funkciou pre získanie veľkosti súboru je funkcia *GetCompressedFileSize*:

```
DWORD GetCompressedFileSize(
    PCTSTR filename,
    PDWORD fileSizeHigh
    );
```

Táto funkcia vráti fyzickú veľkosť súboru, kým funkcia *GetFileSizeEx* vracia logickú veľkosť súboru. Ďalšou odlišnosťou funkcie *GetCompressedFileSize* je spôsob vrátenia veľkosti súboru:

LowPart hodnoty je návratová hodnota funkcie a *HighPart* hodnoty je reprezentovaná parametrom *fileSizeHigh*.

Nastavenie ukazateľa súboru

Volaním funkcie *CreateFile* systém vytvorí objekt jadra súboru, ktorý spravuje operácie so súborom. Vo vnútri tohto objektu sa nachádza ukazateľ súboru, ktorý indikuje 64-bitový offset v súbore, kde má byť vykonané najbližšie čítanie alebo zápis. Spočiatku je tento pointer nastavený na 0, takže ak zavoláte funkciu *ReadFile* hneď po *CreateFile*, začnete súbor čítať z offsetu 0. Ak načítate 10 bytov súboru do pamäte, systém aktualizuje pointer asociovaný s *handle* súboru, takže ďalšie volanie funkcie *ReadFile* začne načítavať na desiatom byte súboru.

Pokiaľ potrebujete pristupovať k súboru náhodne, budete potrebovať zmeniť pointer súboru asociovaný s jeho objektom jadra. Pointer zmeníte volaním funkcie *SetFilePointerEx*:

```
BOOL SetFilePointerEx(
    HANDLE hFile,
    LARGE_INTEGER distanceToMove,
    PLARGE_INTEGER newFilePointer,
    DWORD moveMethod
    );
```

Parameter *hFile* identifikuje súbor, ktorého pointer chcete zmeniť. Parameter *distanceToMove* hovorí systému, o koľko bytov chcete posunúť pointer. Číslo, ktoré špecifikujete je pridané k aktuálnej hodnote ukazateľa, takže záporné číslo posunie pointer späť v súbore. Posledný parameter funkcie hovorí funkcii *SetFilePointerEx* ako má interpretovať parameter *distanceToMove*. Nadobúda jednu z nasledujúcich hodnôt:

- FILE_BEGIN – pointer súboru je nastavený na hodnotu špecifikovanú parametrom *distanceToMove*.
- FILE_CURRENT – hodnota *distanceToMove* je pridaná k aktuálnej hodnote ukazateľa.
- FILE_END – pointer je nastavený na logickú veľkosť súboru + hodnota parametra *distanceToMove*.

Po zavolaní funkcie sa nová hodnota ukazateľa súboru vráti v parametri *newFilePointer*. Ak vás táto hodnota nezaujíma, nastavte parameter na NULL.

Niekoľko faktov o funkcii *SetFilePointerEx*:

- Nastavenie ukazateľa za koniec súboru je možné. Neznamená to však, že sa zmení veľkosť súboru, až kým na toto miesto nezapisujete, prípadne nezavoláte funkciu *SetEndOfFile*.
- Pri volaní funkcie *SetFilePointerEx* na súbor vytvorený s príznakom FILE_FLAG_NO_BUFFERING, pointer môže byť nastavený len na hranice sektoru.
- Aktuálnu hodnotu ukazateľa súboru získate nasledovne:

```
LARGE_INTEGER currentPosition = { 0 };
SetFilePointerEx(hFile, currentPosition, &currentPosition, FILE_CURRENT);
```

Nastavenie konca súboru

Zvyčajne sa o nastavenie konca súboru stará systém, pri zatváraní súboru. Každopádne, môže sa stať, že chcete donútiť súbor, aby bol menší, alebo väčší. Vtedy zavolajte funkciu *SetEndOfFile*:

```
BOOL SetEndOfFile(HANDLE hFile);
```

Táto funkcia zväčší, alebo zmenší veľkosť súboru na veľkosť určenú ukazateľom objektu súboru. Ak napríklad chcete donútiť súbor, aby mal veľkosť 1024 bytov, použite funkciu *SetEndOfFile* nasledovne:

```
HANDLE hFile = CreateFile(...);
LARGE_INTEGER distanceToMove;
distanceToMove.QuadPart = 1024;
SetFilePointerEx(hFile, distanceToMove, NULL, FILE_BEGIN);
SetEndOfFile(hFile);
CloseHandle(hFile);
```

Po prevedení tohto kódu bude mať daný súbor veľkosť 1024 bytov.

Synchrónne I/O operácie zariadenia

Táto časť sa venuje Windows funkciám zabezpečujúcim synchrónne I/O operácie zariadení. Je dôležité pamätať na to, že zariadením môžu byť súbory, mailsloty, sockety,... Nezávisle na type zariadenia sú však funkcie pre I/O operácie tie isté.

Najjednoduchšími a najčastejšie používanými funkciami pre čítanie a zápis sú *ReadFile* a *WriteFile*:

```
BOOL ReadFile(
    HANDLE hFile,
    PVOID buffer,
    DWORD numBytesToRead,
    PDWORD numBytes,
    OVERLAPPED* overlapped
    );

BOOL WriteFile(
    HANDLE hFile,
    CONST VOID *buffer,
    DWORD numBytesToWrite,
    PDWORD numBytes,
    OVERLAPPED* overlapped
    );
```

Parameter *hFile* je *handle* k zariadeniu, ku ktorému chcete pristupovať. Pri otváraní zariadenia nesmie byť špecifikovaný príznak `FILE_FLAG_OVERLAPPED`, pretože inak by si systém myslel, že chcete pracovať s asynchrónnymi I/O operáciami. Parameter *buffer* ukazuje na buffer, do ktorého by sa mali načítať dáta zo zariadenia, resp. obsahuje dáta, ktoré sa majú do zariadenia zapísať. Parametre *numBytesToRead* a *numBytesToWrite* určujú, koľko bytov sa má zo zariadenia prečítať, prípadne doňho zapísať. Parameter *numBytes* funkcia naplní počtom bytov, ktoré boli úspešne prenesené z/do zariadenia. Posledný parameter, *overlapped*, by mal byť v prípade synchrónnych I/O operácií `NULL`.

Obe funkcie v prípade úspechu vrátia `TRUE`. Funkcia *ReadFile* môže byť použitá iba pre zariadenie otvorené s príznakom `GENERIC_READ`, funkcia *WriteFile* obdobne so zariadením s príznakom `GENERIC_WRITE`.

Základy asynchrónnych I/O operácií zariadenia

V porovnaní s inými operáciami, ktoré počítač musí vykonávať, sú I/O operácie zariadenia jedny z najpomalších a najmenej predvídateľných. Vďaka asynchrónnym I/O operáciám však môžete lepšie využívať zdroje a vytvárať tak efektívnejšie aplikácie.

Vezmite si ako príklad vlákno, ktoré má na nejaké zariadenie asynchrónnu I/O požiadavku. Táto požiadavka je prenesená na ovládač zariadenia, ktorý bude niesť zodpovednosť za spracovanie I/O. Zatiaľ čo ovládač zariadenia čaká, kým zariadenie spracuje požadovanú I/O operáciu, vlákno nie je pozastavené a nečaká na výsledok I/O operácie. Namiesto toho vlákno pokračuje vo vykonávaní iných úloh.

V istom bode potom zariadenie ukončí požadovanú I/O operáciu a musí upozorniť aplikáciu, že dáta boli odoslané, prijaté alebo došlo k chybe. Skôr než sa však budeme zaoberať prijatím upozornení, pozrieme sa na to, ako sa I/O požiadavky zaraďujú do fronty.

Aby ste mohli k zariadeniu pristupovať asynchrónne musíte najskôr dané zariadenie otvoriť pomocou funkcie *CreateFile* a špecifikovaním príznaku FILE_FLAG_OVERLAPPED. Tento príznak upozorní systém, že si želáte k zariadeniu pristupovať asynchrónne.

Pre zaradenie I/O žiadosti na dané zariadenie do fronty sa používajú už spomínané funkcie *ReadFile* a *WriteFile*. Keď je volaná jedna z týchto funkcií, funkcia skontroluje, či zariadenie identifikované parametrom *hFile* bolo otvorené s príznakom FILE_FLAG_OVERLAPPED. Ak áno, funkcia prevedie asynchrónnu I/O operáciu. Nakoľko sa očakáva, že funkcia sa vráti skôr, než bude hotová I/O operácia, je zbytočné používať parameter *numBytes* – tento parameter bude NULL.

Štruktúra OVERLAPPED

Pri prevádzaní asynchrónnych I/O operácií zariadenia musíte ako parameter *overlapped* funkcii *ReadFile/WriteFile* odovzdať adresu inicializovanej OVERLAPPED štruktúry. Samotná štruktúra vyzerá nasledovne:

```
typedef struct _OVERLAPPED {
    DWORD Internal;       // [out] Error code
    DWORD InternalHigh;   // [out] Number of bytes transferred
    DWORD Offset;         // [in] Low 32-bit file offset
    DWORD OffsetHigh;     // [in] High 32-bit file offset
    DWORD hEventl;        // [in] Event handle or data
} OVERLAPPED, *LPOVERLAPPED;
```

Ako je možné vidieť, táto štruktúra má 5 členov. Členy *Offset, OffsetHigh* a *hEvent* musia byť inicializované pred volaním funkcie *ReadFile/WriteFile*. Ostatné dva členy sú nastavené ovládačom zariadenia a môžu byť vyšetrené po skončení I/O operácie. Jednotlivé členy štruktúry:

- *Offset* a *OffsetHigh* – tieto dva členy špecifikujú 64-bitový offset v súbore, kde chcete, aby začala I/O operácia. Majte na pamäti, že tieto dva členy pre zariadenia, ktoré nie sú súbormi, nie sú ignorované, preto ich pre iné zariadenia vždy musíte nastaviť na 0, inak I/O žiadosť zlyhá a funkcia *GetLastError* vráti chybu ERROR_INVALID_PARAMETER.
- *hEvent* – tento člen je používaný funkciami, ktoré prijímajú upozornenia o ukončení I/O operácii (*WaitForSingleObject*,...).
- *Internal* – tento člen obsahuje chybový kód spracovávanej I/O operácie. Po začatí operácie je tento člen nastavený na STATUS_PENDING.
- *InternalHigh* – po skončení asynchrónnej I/O operácie obsahuje tento člen počet prenesených bajtov.

Je dôležité neuvoľniť parameter *buffer* *ReadFile/WriteFile* funkcií a `OVERLAPPED` štruktúru skôr, ako skončí samotná I/O operácia. Takisto musíte alokovať a inicializovať `OVERLAPPED` štruktúru zvlášť pre každú I/O žiadosť.

Zrušenie zaradenej I/O žiadosti

Môže sa stať, že budete chcieť zrušiť nejakú I/O žiadosť predtým, než ju ovládač zariadenia vykoná. Existuje niekoľko možností, ako to urobiť:

- Volaním funkcie *CancelIo* zrušíte všetky I/O žiadosti zaradené vláknom pre špecifikovanú *handle*:

  ```
  BOOL CancelIo(HANDLE hFile);
  ```

- Zatvorením *handle* k zariadeniu zrušíte aj všetky I/O požiadavky k tomuto zariadeniu.

- Keď je ukončené vlákno, systém automaticky zruší všetky I/O žiadosti tohto vlákna.

Nie je možné ukončiť iba jednu, konkrétnu I/O žiadosť. Zrušené I/O žiadosti skončia s chybou `ERROR_OPERATION_ABORTED`.

Prijímanie správ o ukončení I/O žiadostí

Windows ponúka 4 metódy prijímania správ o ukončení asynchrónnych I/O žiadostí:

- *Signalizovanie objektu jadra zariadenia* – neumožňuje viacero súbežných I/O žiadostí k tomu istému zariadeniu. Umožňuje jednému vláknu vydať I/O žiadosť a ďalšiemu túto žiadosť spracovať.
- *Signalizovanie objektu jadra udalosti* – umožňuje viacero súbežných I/O žiadostí k tomu istému zariadeniu. Taktiež umožňuje jednému vláknu vydať I/O žiadosť a ďalšiemu túto žiadosť spracovať.
- *Použitie alertable I/O* - umožňuje viacero súbežných I/O žiadostí k tomu istému zariadeniu. Vlákno, ktoré žiadosť vydá, ju musí aj spracovať.
- *Použitie I/O completion portov* - umožňuje viacero súbežných I/O žiadostí k tomu istému zariadeniu. Taktiež umožňuje jednému vláknu vydať I/O žiadosť a ďalšiemu túto žiadosť spracovať. Táto technika je najzložitejšia a najviac flexibilná.

V rámci tohto predmetu sa budeme zaoberať len signalizovaním objektov jadra zariadenia a udalosti. Keď vlákno vydá asynchrónnu I/O žiadosť, pokračuje ďalej v užitočnej práci. Nakoniec sa však dostane do bodu, kedy sa bude musieť zosynchronizovať s ukončením I/O operácie – bez ukončenia tejto operácie nemôže ďalej pokračovať.

Nakoľko je objekt zariadenia objektom jadra, môže byť použitý pre synchronizáciu vlákien – môže byť v signalizovanom, alebo nesignalizovanom stave. Funkcia *ReadFile/WriteFile*, skôr než zaradí I/O žiadosť do fronty, nastaví stav objektu zariadenia na nesignalizovaný. Keď zariadenie ukončí žiadosť, zariadenie nastaví stav objektu zariadenia na signalizovaný. Vlákno následne môže zistiť, či bola žiadaná operácia ukončená, volaním funkcií *WaitForSingleObject*, príp. *WaitForMultipleObjects*.

Táto metóda je veľmi jednoduchá a priama. Nerieši však problém s viacerými I/O žiadosťami na jedno zariadenie. Objekt zariadenia je totižto signalizovaný hneď ako niektorá z operácií skončí – vlákno však na základe toho nevie, ktorá operácia skončila.

Riešením tohto problému je použitie člena `hEvent` štruktúry `OVERLAPPED`. Tento objekt udalosti vytvoríte volaním funkcie `CreateEvent`. Keď končí asynchrónna I/O žiadosť, ovládač zariadenia skontroluje, či je `hEvent` `NULL`. Ak nie, ovládač ho signalizuje volaním funkcie `SetEvent`. Ovládač zariadenia takisto signalizuje aj objekt jadra zariadenia. Ak však používate udalosti na rozhodnutie sa, či operácia zariadenia skončila, nemali by ste čakať na objekt zariadenia, ale na objekt udalosti.

Ak chcete previesť súčasne niekoľko asynchrónnych I/O žiadostí na jednom zariadení , musíte vytvoriť samostatnú udalosť pre každú žiadosť, inicializovať `hEvent` člen každej `OVERLAPPED` štruktúry a potom zavolať funkciu `ReadFile/WriteFile`. Keď potom v kóde dosiahnete bod, kedy je nutné zosynchronizovať vlákno s ukončením I/O žiadosti, jednoducho zavolajte funkciu `WaitForMultipleObjects` a odovzdajte jej *handle* udalostí spojených s daným zariadením. Takto môžete ľahko a spoľahlivo previesť niekoľko asynchrónnych I/O operácií súbežne a používať pritom ten istý objekt zariadenia.

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