Synchronizační nástroje Windows určené pro vícevláknové aplikace

Cvičení III

Operační systém Microsoft Windows poskytuje ucelenou a konzistentní sadu nástrojů pro synchronizaci procesů a vláken. Zajímavým rysem této sady je, že se zde využívá i principů objektově orientovaného programování, na kterých je jádro Windows NT postaveno.

1 Obecné principy

1.1 Stavy objektů

Synchronizační objekty ve Windows se nachází ve dvou stavech. Buď je objekt (i) *signalizovaný* (signaled), nebo (ii) *nesignalizovaný* (non-signaled). Pokud se objekt nachází ve stavu *nesignalizovaný*, znamená to, že je nedostupný, a je potřeba čekat, dokud se nepřepne do stavu *signalizovaný*. Máme-li například zámek (mutex), který je odemčený, nachází se ve stavu signalizovaný, pokud je zámek uzamčený, je ve stavu nesignalizovaný. Podobně, máme-li semafor, který má nenulovou hodnotu, je ve stavu signalizovaný, pokud má semafor hodnotu nula, je ve stavu nesignalizovaný.

Takto obecně navržený synchronizační mechanismus má tu výhodu, že můžeme používat stejné funkce pro různé synchronizační objekty,[1] a můžeme používat širší škálu typů objektů a nemusíme se omezovat na základní synchronizační objekty jako je zámek nebo semafor. Například objekty reprezentující vlákno nebo proces jsou taktéž synchronizační objekty, pokud běží, nachází se v nesignalizovaném stavu, v momentě kdy skončí, přechází do signalizovaného stavu. Takto je možné čekat na dokončení vlákna nebo procesu.

Poznámka: Mezi další objekty, které lze použít k synchronizaci patří třeba vstup z konzole. Pokud je vstup prazdný, je objekt nesignalizovaný, jinak je signalizovaný. Podobně se dá sledovat i dostupnost nebo změny dalších zdrojů, jako je dostupná paměť nebo změny v adresářích.

1.2 Univerzální čekací funkce

Rozhraní Windows NT těží z toho, že všechny synchronizační objekty mají přibližně stejné rozhraní, a proto pro synchronizaci používají relativně malé množství funkcí, které zajišťují, že je běžící vlákno pozastaveno do doby, než je splněna předpokládaná podmínka pro synchronizaci.

[1] De facto se jedná o polymorfismus.

S jednou z těchto funkcí jsme se již setkali. Připomeňme práci s vlákny, kterou jsme si nastínili v předchozím semestru.

```
HANDLE *hThread = CreateThread(/* parametry */);
/* dalsi kod */
WaitForSingleObject(hThread, INFINITE);
```

Pro nás je zajímavý poslední řádek, kde je použita funkce `WaitForSingleObject`,[2] která čeká dokud objekt (v našem případě vlákno `hThread`) nepřejde z nesignalizovaného stavu do signalizovaného. Druhý argument funkce představuje maximální čas v milisekundách, po který se na daný objekt čeká, konstanta `INFINITE` indikuje, že se čeká nekonečně dlouho.

Další funkcí, která se používá pro čekaní, je funkce `WaitForMultipleObjects`,[3] která pracuje s více synchronizačními objekty současně a podle zvolených parametrů, buď čeká, než budou všechny předané objekty v signalizovaném stavu,[4] nebo čeká, dokud alespoň jeden objekt nebude v signalizovaném stavu, viz dokumentace.

Poslední užitečnou synchronizační funkcí, kterou si zmíníme je funkce `SignalObjectAndWait`,[5] tato funkce přepne objekt do signalizovaného stavu (např. odemče zámek) a čeká, dokud se jiný objekt nepřepne taktéž do signalizovaného stavu.

2 Tradiční zámky

2.1 Mutex

Základním synchronizačním nástrojem je zámek, který se podobně jako v knihovně `pthreads` označuje jako *mutex*. K vytvoření zámku slouží funkce:[6]

```
HANDLE CreateMutex(LPSECURITY_ATTRIBUTES lpMutexAttributes, BOOL bInitialOwner, LPCSTR lpName);
```

První argument udává vlastnosti zámku (může být `NULL`), druhý argument udává, zda je zámek při svém vytvoření uzamčen daným vláknem, a třetí argument udává jméno zámku (může být `NULL`). Funkce vrací odkaz typu `HANDLE` pro přístup k danému zámku.

Pro zamčení zámku se používá některá z čekacích funkcí, např. `WaitForSingleObject`, k odemčení slouží funkce `ReleaseMutex`[7] a k uvolnění objektu z paměti se používá funkce `CloseHandle`,[8] která uvolní daný objekt, pokud neexistuje na daný objekt žádný odkaz v podobě používané hodnoty typu `HANDLE`.

Práci se zámky ilustruje následující kód:

```
HANDLE hMutex = CreateMutex(NULL, FALSE, NULL); // otevreny zamek s vychozim nastavenim
WaitForSingleObject(hMutex, INFINITE); // zamceni zamku
```

[2] `https://learn.microsoft.com/en-us/windows/win32/api/synchapi/nf-synchapi-waitforsingleobject`
[3] `https://learn.microsoft.com/en-us/windows/win32/api/synchapi/nf-synchapi-waitformultipleobjects`
[4] Může sloužit jako bariéra.
[5] https://learn.microsoft.com/en-us/windows/win32/api/synchapi/nf-synchapi-signalobjectandwait
[6] `https://learn.microsoft.com/en-us/windows/win32/api/synchapi/nf-synchapi-createmutexa`
[7] `https://learn.microsoft.com/en-us/windows/win32/api/synchapi/nf-synchapi-releasemutex`
[8] `https://learn.microsoft.com/en-us/windows/win32/api/handleapi/nf-handleapi-closehandle`

```
/* kod kriticke sekce */
ReleaseMutex(hMutex); // odemceni zamku
CloseHandle(hMutex); // uvolneni objektu zamku
```

Úkol č. 1: Vezměte zdrojové kódy z předchozích cvičení a převeďte je na platformu Windows. Kompatibilitu s Linuxem není nutné zachovávat, pro synchronizaci použijte objekt(y) typu mutex.

Poznámka: Windows bohužel nemají v základu ekvivalent nástroje `nl`, kterým bychom mohli otestovat, zda jsou opravdu vypsány všechny řádky. Můžeme si ale pomoci PowerShellem a následujícím kouskem kódu.

```
$i = 1; (.\queue-test.exe).ForEach({ "$i $_i"; $i = $i + 1; });
```

Tento kousek kódu si udržuje proměnnou `$i` obsahující počet vypsaných řádků a pro každý vypsaný řádek programu `queue-test.exe` sestaví řetězec ve tvaru `"$i řádek"`.

Poznámka: Jeden z argumentů, který jsme při vytváření zámku mohli předat, je jméno objektu. Podle tohoto jména může jiný proces, pomocí funkce `OpenMutex`[9] získat přístup k tomuto synchronizačnímu objektu, a je tak možné synchronizovat vlákna napříč procesy.

2.2 Kritická sekce

Objekt mutex je navržený a implementovaný jako obecný synchronizační nástroj pro vlákna, který je možné sdílet i mezi procesy, a to může mít dopad na výkon. Proto Windows obsahují ještě jeden nástroj, označovaný jako *kritická sekce* (anglicky *critical section*), který umožňuje synchronizovat pouze vlákna v rámci jednoho procesu, ale měl by být efektivnější.

S kritickou sekcí se pracuje mírně jinak než s mutexy. Jednak synchronizační objekty jsou reprezentovány jako hodnoty datového typu `CRITICAL_SECTION`.[10] Objekt je inicializován pomocí funkce `InitializeCriticalSection`[11], uzamčen funkcí `EnterCriticalSection`[12], odemčen funkcí `LeaveCriticalSection`[13] a uvolněn funkcí `DeleteCriticalSection`,[14] kde všechny tyto funkce mají jedinný parametr, kterým je odkaz na hodnotu typu `CRITICAL_SECTION`.

Použití je následující:

```
CRITICAL_SECTION lock; // alokace objektu zamku
InitializeCriticalSection(&lock); // inicializace zamku
EnterCriticalSection(&lock); // zamceni zamku
LeaveCriticalSection(&lock); // odemceni zamku
DeleteCriticalSection(&lock); // uvolneni zamku
```

Úkol č. 2: Upravte kód tak, aby místo zámku typu *mutex* používal *kritickou sekci*.

[9] `https://learn.microsoft.com/en-us/windows/win32/api/synchapi/nf-synchapi-openmutexw`
[10] Všimněme si, že se nejedná o `HANDLE`.
[11] `https://learn.microsoft.com/en-us/windows/win32/api/synchapi/nf-synchapi-initializecriticalsection`
[12] `https://learn.microsoft.com/en-us/windows/win32/api/synchapi/nf-synchapi-entercriticalsection`
[13] `https://learn.microsoft.com/en-us/windows/win32/api/synchapi/nf-synchapi-leavecriticalsection`
[14] `https://learn.microsoft.com/en-us/windows/win32/api/synchapi/nf-synchapi-deletecriticalsection`

3 Semafor

Vedle jednoduchých zámků můžeme ve Windows pro synchronizaci používat i semafory. Práce s nimi je podobná jako s mutexy. Objekt semaforu vytvoříme pomocí funkce `CreateSemaphore`,[15] kde specifikujeme atributy semaforu, počáteční a maximální hodnotu semaforu a případně název. Ke snížení hodnoty semaforu slouží čekací funkce, např. `WaitForSingleObject`, ke zvýšení hodnoty slouží funkce `ReleaseSemaphore`,[16] kde specifikujeme, o kolik se má zvednout hodnota semaforu a je možné získat i předchozí hodnotu, viz dokumentace.

Úkol č. 3: Upravte kód tak, aby místo zámku typu *mutex* používal *semafor*.

4 Další synchronizační nástroje

Mezi další nástroje, které operační systém Windows nabízí patří bariéry. Práce s nimi je velmi podobná tomu, co jsme viděli v Linuxu, a rozhraní pro práci s nimi je velmi podobné rozhraní, které má objekt kritické sekce, viz dokumentace.[17] Analogicky fungují i podmínkové proměnné.[18]

Úkol č. 4: Upravte kód tak, aby se vlákna producentů při dosažení poloviny vygenerovaných čísel se synchronizovala. Tj. aby vlákno, které dosáhne poloviny vygenerováných čísel, počkalo, dokud druhé vlákno nevygeneruje taktéž polovinu čísel.

4.1 Čekání na obecnou událost

Operační systém Windows nabízí praktický nástroj, který umožňuje čekat na obecnou událost. Slouží k tomu objekt *event*, který se podle svého nastavení nachází v signalizovaném nebo nesignalizovaném stavu.

K vytvoření objektu slouží funkce `CreateEvent`,[19] kde podobně jako v případě práce s mutexem specifikujeme atributy a název objektu a vedle toho specifikujeme, zda se objekt po vytvoření nachází v signalizovaném nebo nesignalizovaném stavu, a zda je potřeba manuálně objekt resetovat do nesignalizovaného stavu, nebo se tak bude dít automaticky.

S objektem události se pracuje pomocí funkcí `SetEvent`,[20] která přepne objekt do signalizovaného stavu, `ResetEvent`,[21] která přepne objekt do nesignalizovaného stavu, a pomocí některé z čekacích funkcí, např. `WaitForSingleObject`. Pokud jsme při vytváření objektu nastavili, že se má automaticky resetovat do nesignalizovaného stavu, čekací funkce jej přepne do nesignalizovaného stavu v momentě, kdy některé z vláken projde přes funkci `WaitForSingleObject`.

Úkol č. 5: Rozšiřte předchozí kód o další vlákno, které bude s pomocí objektu event čekat, než některé z vláken producentů dosáhne tří čtvrtin zpracovaných hodnot, a vypíše o tom informaci na standardní výstup.

[15] `https://learn.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-createsemaphorea`
[16] `https://learn.microsoft.com/en-us/windows/win32/api/synchapi/nf-synchapi-releasesemaphore`
[17] `https://learn.microsoft.com/en-us/windows/win32/sync/synchronization-barriers`
[18] `https://learn.microsoft.com/en-us/windows/win32/sync/condition-variables`
[19] `https://learn.microsoft.com/en-us/windows/win32/api/synchapi/nf-synchapi-createeventa`
[20] `https://learn.microsoft.com/en-us/windows/win32/api/synchapi/nf-synchapi-setevent`
[21] `https://learn.microsoft.com/en-us/windows/win32/api/synchapi/nf-synchapi-resetevent`

Bonusový úkol: Nahraďte bariéru z úkolu č. 4 objektem (nebo objekty) typu event.