Vlákna

Vytvoření vláken

Linux

K vytvoření vlákna slouží funkce `pthread_create` z knihovny `pthread` vracející id nového procesu.

```
int pthread_create(pthread_t *restrict thread,
                   const pthread_attr_t *restrict attr,
                   void *(*start_routine)(void *),
                   void *restrict arg);
```

Nejdůležitější parametry této funkce jsou `thread`, což je ukazatel na identifikátor vlákna, `start_routine` a `arg`, což jsou ukazatele na funkci, kterou bude vlákno vykonávat, a argumenty této funkce.

Po zavolání je vlákno v běžícím stavu. Pro čekání na ukončení vlákna slouží funkce `pthread_join`. Pokud nepotřebujeme na dokončení čekat, je nutné přidat k inicializačním atributům `PTHREAD_CREATE_DETACHED`.

Windows

K vytvoření vlákna slouží funkce `CreateThread` z knihovny `processthreadsapi`, vracející `HANDLE` procesu.

```
HANDLE CreateThread(
  [in, optional]  LPSECURITY_ATTRIBUTES   lpThreadAttributes,
  [in]            SIZE_T                  dwStackSize,
  [in]            LPTHREAD_START_ROUTINE  lpStartAddress,
  [in, optional]  __drv_aliasesMem LPVOID lpParameter,
  [in]            DWORD                   dwCreationFlags,
  [out, optional] LPDWORD                 lpThreadId
);
```

Nejdůležitějšími parametry jsou pak `lpStartAddress`, což je ukazatel na funkci, která bude vykonána novým vláknem, a `lpParameter`, což je parametr vykonávané funkce. `lpThreadAttributes` a `lpThreadId`, pak mohou být `NULL` a `deStackSize` a `dwCreationFlags` 0.

Po zavolání `CreateThread` je vlákno v běžícím stavu. Pro čekání ukončení vláken se používají funkce `WaitForSingleObject` a `WaitForMultipleObjects` z `synchapi`. `WaitForSingleObject` příjmá jako parametry handle procesu a délka čekání (v milisekundách), `WaitForMultipleObjects` pak počet vláken, ukazatel na pole jejich handelů, zda-li se má čekat na všechny procesy (jinak stačí, aby skončil jeden) a opět délka čekání.

Vlákna také můžeme pozastavit, znovu spustit nebo násilně ukončit za pomoci `SuspendThread`, `ResumeThread` a `TerminateThread`.

Po ukončení práce s vláknem je vhodné zavolat funkci `CloseHandle`.

Základní synchronizační nástroje

Linux

Zámek

K vytvoření zámku se používá typ `pthread_mutex_t`. K jeho inicializaci můžeme použít funkci `pthread_mutex_init`, nebo pro zjednodušení makro `PTHREAD_MUTEX_INITIALIZER`. Funkce pro práci s mutexem jsou `pthread_mutex_lock` pro uzamčení a `pthread_mutex_unlock` pro uvolnění zámku. Po ukončení práce se zámkem ještě uvolníme alokované prostředky zavoláním `pthread_mutex_destroy`. Vše potřebné je obsaženo přímo z knihovně `pthread`.

Spinlock

V případech, kde je vhodné čekat na zámek aktivně, můžeme využít spinlocku. Práce s ním je velice podobná jako u zámku. K dispozici máme datový typ `pthread_spinlock_t` a funkce `pthread_spin_init`, `pthread_spin_lock`, `pthread_spin_unlock` a `pthread_spin_destroy`.

Read-write lock

Máme-li data, která chceme umožnit ostatním vláknům číst naráz, nebo jednomu vláknu zapisovat, máme dostupnou přímo implementaci read-write zámku. Zde jsou kromě typu `pthread_rwlock_t` a funkcí `pthread_rwlock_init` a `pthread_rwlock_destroy`, potřeba dvě různé uzamykací funkce, `pthread_rwlock_wrlock` a `pthread_rwlock_rdlock`, a odemykací funkce, `pthread_rwlock_unlock`.

Semafor

Knihovna `semaphore` poskytuje rozhraní pro práci se semafory. Objekt semaforu inicializujeme za pomoci `sem_init`, která bere ukazatel na `sem_t` objekt, počáteční hodnotu a maximální počet vláken. Čekací funkce se jmenuje `sem_wait` a signalizační `sem_post`. Pro uvolnění prostředků se volá `sem_destroy`.

Bariéra

K vytvoření bariéry se používá datový typ `pthread_barrier_t`. Inicializace bariéry probíhá pomocí funkce `pthread_barrier_init`, která přijímá ukazatel na bariéru, počet vláken, které jí musí dosáhnout, a volitelný parametr udávající maximální počet spinů. Vlákna čekají na bariéru pomocí funkce `pthread_barrier_wait`. Pokud je bariéra splněna, všechny čekající vlákna mohou pokračovat v práci. Pro uvolnění prostředků bariéry je k dispozici funkce `thread_barrier_destroy`.

Podmíněná proměnná

Pro případ, kdy jedno vlákno čeká na splnění podmínky slouží podmínková proměnná. Na linuxu používán typ `pthread_cond_t`. Inicializace podmíněné proměnné se provádí funkcí `pthread_cond_init`, která přijímá ukazatel na objekt podmíněné proměnné a volitelné atributy. K synchronizaci s podmíněnou proměnnou používáme funkce `pthread_cond_wait`, `pthread_cond_signal` a `pthread_cond_broadcast`. `pthread_cond_wait` blokuje vlákno, dokud není podmíněná proměnná signálována, `pthread_cond_signal` signálizuje jedno čekající vlákno

a `pthread_cond_broadcast` signálizuje všechna čekající vlákna. Pro uvolnění prostředků podmíněné proměnné je k dispozici funkce `pthread_cond_destroy`.

Windows

Na platformě Windows platí, že se synchronizační objekty nachází bud v *signalizovaném*, nebo *nesignalizovaném* stavu. Díky tomu můžeme používat jednu čekací funkci pro více primitiv. Tato funkce počká, dokud objekt není v signalizovaném stavu, a pak změní jeho stav na nesignalizovaný. V sekci o vytvoření vlákna jsme ji již použili, je jí `WaitForSingleObject` (případně `WaitForMultipleObjects`). Vlákna jsou totiž v nesignalizovaném stavu, když běží, a v signalizovaném po ukončení. Připomeňme si, že tyto funkce přijmají vlákna ve formě jejich handlu. Tak tomu bude tedy u synchronizačních objektů.

Obecně platí, že bychom práci s objekty měli ukončit voláním `CloseHandle`, nebo jiné příslušné funkce.

Zámek

Pro vytvoření zámku se používá funkce `CreateMutex` z knihovny `synchapi`, která přijímá tři parametry, kde první definuje vlastnosti zámku a může být `NULL`, druhý říká, zda-li bude po vytvoření zámek uzamčen aktuálním vláknem, a třetí udává jméno zámku a opět může být `NULL`.

Uzamykání zámku je řešeno funkcí `WaitForSingleObject`, pro odemykání máme dostupnou funkci `ReleaseMutex` přijímající handle zámku.

Kritická sekce

Windows poskytuje efektivnější alternativu pro zámku, pokud jsou synchronizovaná vlákna v rámci jednoho procesu. Touto alternativou je objekt typu `CRITICAL_SECTION`. Inicializuje se voláním `InitializeCriticalSection`, uvolňuje se zase voláním `DeleteCriticalSection`.
Pro vstoupení pak máme funkce `EnterCriticalSection` a `TryEnterCriticalSection`. Nakonec kritickou sekci opustíme zavoláním `LeaveCriticalSection`.

Semafor

Objekt semaforu vytvoříme zavoláním funkce `CreateSemaphore` s vlastnostmi semaforu, maximálním počtem vláken, počáteční hodnotou a jménem. Navíc je pak k dispozici funkce `ReleaseSemaphore`, které udáme jaký semafor chceme o kolikrát zasignalizovat, případně můžeme ještě získat předchozí hodnotu semaforu.

Bariéra

Bariéry vytvoříme deklarací `SYNCHRONIZATION_BARRIER` a zavoláním `InitializeSynchronizationBarrier` s ukazatelem na bariéru, počtem vláken a číslem určující, kolik spinů má vlákno provést, když čeká na ostatní (pro hodnotu -1 je zvolena výchozí hodnota). Vlákna pak začnou čekat po zavolání `EnterSynchronizationBarrier`.

Podmněná proměnná

I na Windows je možnost synchronizace pomocí podmíněné proměnné, na rozdíl však od systému Linux se v kombinaci s podmíněnou proměnnou nepoužívá zámek, ale kritická sekce.

K vytvoření podmíněné proměnné nám slouží funkce `InitializeConditionVariable` přijímající ukazatel na objekt typu `CONDITION_VARIABLE`.

Pokud vlákno musí vlákno čekat na splnění podmínky, volá funkci `SleepConditionVariableCS`. Pokud jiné vlákno chce signalizovat, že by podmínka mohla být splněna, zavolá `WakeConditionVariable`, která na rozdíl od `SleepConditionVariableCS` příjmá kromě ukazatele na proměnnou také ukazatel na kritickou sekci a délku čekání v milisekundách.

Události

Jako poslední se podíváme na synchronizaci pomocí událostí. Tento nástroj umožňuje čekání na obecnou událost, která je, jak již víme, buď v signalizovaném, nebo nesignalizovaném stavu. Na začátku se vytvoří událost zavoláním fuknce `CreateEvent` vracející handle na objekt. Tato funkce přijímá čtyři argumenty, prvním jsou vlastnosti události, druhým je bool určující, zda musíme událost manuálně resetovat do nesignalizovaného stavu, nebo bude stav automaticky resetován po uvolnění jednoho vlákna, které na událost čekalo, třetím je počáteční stav, a posledním je jméno. Dále nám už postačí funkce na změnu stavu události - `SetEvent` a `ResetEvent`.