Systémové programování Windows

Synchronizace vláken

Obsah

- Uživatelský režim
  - Interlocked…
  - CriticalSection
- Objekty jádra
  - Event
  - Mutex
  - Semaphore

Špatně

```
//
//TAKTO NIKDY NE
//
void Lock( void )
{
   //cekani na zamek
   while (_locked)
   {
      Sleep(100);
   }

   //uzamceni
   _locked = true;
}
```

Interlocked…

```
LONG __cdecl InterlockedIncrement(
       _Inout_ LONG volatile *Addend );

LONG __cdecl InterlockedDecrement(
       _Inout_ LONG volatile *Addend );

LONG __cdecl InterlockedCompareExchange(
       _Inout_ LONG volatile *Destination,
       _In_ LONG Exchange,
       _In_ LONG Comparand );
```

CriticalSection

```
VOID InitializeCriticalSection(PCRITICAL_SECTION cs);

VOID DeleteCriticalSection(PCRITICAL_SECTION cs);

VOID EnterCriticalSection(PCRITICAL_SECTION cs);

VOID LeaveCriticalSection(PCRITICAL_SECTION cs);
```

WaitForSingleObject

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

Návratové hodnoty:

- WAIT_OBJECT_0
- WAIT_TIMEOUT
- WAIT_FAILED

Event

```
HANDLE CreateEvent(
        PSECURITY_ATTRIBUTES sa,
        BOOL manualReset,
        BOOL initialState, //true = signaled
        PCTSTR name );

BOOL SetEvent(HANDLE eventHandle);

BOOL ResetEvent(HANDLE eventHandle);
```

Mutex

```
HANDLE CreateMutex(
        PSECURITY_ATTRIBUTES sa,
        BOOL initialOwner,
        PCTSTR name );

BOOL ReleaseMutex(HANDLE mutexHandle);
```

Semaphore

```
HANDLE CreateSemaphore(
        PSECURITY_ATTRIBUTE sa,
        LONG initialCount,
        LONG maximumCount,
        PCTSTR name );


BOOL ReleaseSemaphore(
        HANDLE semaphoreHandle,
        LONG releaseCount,
        PLONG previousCount );
```

C++11 Mutexes

- std::mutex - non-recursive, no timeout support
- std::recursive_mutex - recursive, no timeout support

- std::timed_mutex - non-recursive, timeout support
- std::recursive_timed_mutex - recursive, timeout support

C++11 Locks

- std::lock_guard

```
std::mutex m; // mutex object
{
std::lock_guard<std::mutex> L(m); // lock m
… // critical section
} // unlock m
```

- std::unique_lock

```
using RCM = std::recursive_timed_mutex; // typedef
RCM m; // mutex object
{
std::unique_lock<RCM> L(m); // lock m
… // critical section
L.unlock(); // unlock m
…
} // nothing happens
```

C++11 condition_variable

```
std::atomic<bool> readyFlag(false);
std::mutex m;
std::condition_variable cv;
{
  std::unique_lock<std::mutex> lock(m);

  //1
  while (!readyFlag) // loop for spurious wakeups
    cv.wait(lock); // wait for notification

  //2
  cv.wait(lock, []{ return readyFlag; }); // library loops

  … // critical section
}

{
   readyFlag = true;
   cv.notify_one();
}
```

A mnohem více …

- Slim Reader/Writer (SRW) Locks
- Waitable Timer Objects
- Synchronization Barrier

Díky za pozornost