线程的同步

Ring3下同步

只能在进程内使用，不可在进程间使用，效率比内核同步对象高出很多

原子操作

带有`InterlockedXXX`系列函数，比如

```
LONG InterlockedExchangeAdd(
  LONG volatile *Addend,
  LONG          Value
);
```

关键段

- 初始化关键段

```
void InitializeCriticalSection(
  LPCRITICAL_SECTION lpCriticalSection
);
```

- 进入关键段时上锁，获取所有权

```
// 若无法获取所有权，则阻塞
// 在同线程中多次调用会增加引用计数
void EnterCriticalSection(
  LPCRITICAL_SECTION lpCriticalSection
);

// 非阻塞，返回非0，获取所有权
BOOL TryEnterCriticalSection(
  LPCRITICAL_SECTION lpCriticalSection
);
```

- 出关键段时解锁，让出所有权

```
// 与 EnterCriticalSection 或 TryEnterCriticalSection 成对使用
void LeaveCriticalSection(
  LPCRITICAL_SECTION lpCriticalSection
);
```

- 释放关键段

```
void DeleteCriticalSection(
  LPCRITICAL_SECTION lpCriticalSection
);
```

内核同步对象

实现在Ring0，对Ring3提供了接口，可跨进程使用

内核对象的属性：

- 状态**state**
  - **signaled**，已触发 —— 开锁、结束等
  - **no-signal**，未触发 —— 关锁、运行等
  - 检测方式：
    1. `WaitForSingleObject`等待一个对象

```
DWORD WaitForSingleObject(
  HANDLE hHandle,
  DWORD  dwMilliseconds
);
```

    2. `WaitForMultipleObjects`等待多个对象

```
DWORD WaitForMultipleObjects(
  DWORD        nCount,
  const HANDLE *lpHandles,
  BOOL         bWaitAll,
  DWORD        dwMilliseconds
);
```

事件Event

- 创建事件对象

```
HANDLE CreateEventA(
  LPSECURITY_ATTRIBUTES lpEventAttributes,
  BOOL                  bManualReset,   // 是否手动复位
  BOOL                  bInitialState,  // TRUE - 已触发的，FALSE - 未触发
  LPCSTR                lpName      // 跨进程则使用
);
```

- 设置为触发状态

```
BOOL SetEvent(
  HANDLE hEvent
);
```

- 设置为未触发状态

```
BOOL ResetEvent(
  HANDLE hEvent
);
```

- 打开事件

```
HANDLE OpenEventA(
  DWORD  dwDesiredAccess,
  BOOL   bInheritHandle,
  LPCSTR lpName
);
```

信号

使用`WaitForSingleObject`等待信号量，等待成功计数减一。计数为**0**时，信号量处于**no-signal**状态

多用于处理多个线程共享多于一个资源的情况，常用于池技术

- 创建信号量

```
HANDLE CreateSemaphoreW(
  LPSECURITY_ATTRIBUTES lpSemaphoreAttributes,
  LONG                  lInitialCount,  // 初始计数个数
  LONG                  lMaximumCount,  // 最大计数个数
  LPCWSTR               lpName
);
```

- 打开信号量

```
HANDLE WINAPI OpenSemaphore(
  _In_ DWORD   dwDesiredAccess,
  _In_ BOOL    bInheritHandle,
  _In_ LPCTSTR lpName
);
```

- 释放信号量

```
BOOL ReleaseSemaphore(
  HANDLE hSemaphore,
  LONG   lReleaseCount,     // 释放的信号量的计数个数
  LPLONG lpPreviousCount
);
```

互斥体

使用`WaitForSingleObject`等待互斥体，等待成功获取互斥体使用权，并修改状态为**no-signal**

使用`ReleaseMutex`释放互斥体，修改状态为**signaled**

- 创建互斥体

```
HANDLE CreateMutexW(
  LPSECURITY_ATTRIBUTES lpMutexAttributes,
  BOOL                  bInitialOwner,  // 该线程是否拥有互斥体的使用权
  LPCWSTR               lpName
);
```

- 释放互斥体

```
BOOL ReleaseMutex(
  HANDLE hMutex
);
```