多线程

线程

- 程序 —— 磁盘上的可执行文件，属性是文件
- 进程 —— 可执行文件在内存中的映射，可执行文件执行所需要的资源的总和
  - 进程启动时，系统会为进程启动一个唯一**主线程**
- 线程 —— 操作系统分配CPU时间的最小单位
  - 进入时间片恢复环境`CONTEXT`，出时间片保存环境`CONTEXT`
  - 适合的线程数量：**CPU核心数 * 2**

1/10秒原则

在界面线程中，一般不要处理任务超过**1/10**秒

线程划分

- 带界面的线程 —— 用于创建窗口，建立消息循环，派发消息，一般都是主线程
- 不带界面的线程 —— 不处理消息，用来处理长时间需要解决的问题，一般是后台线程

创建线程

会创建线程内核对象，栈等资源

`DllMain`会被调用，`DLL_THREAD_ATTACH`参数被传入

CreateThread

```
HANDLE CreateThread(
  LPSECURITY_ATTRIBUTES   lpThreadAttributes,   // 安全属性
  SIZE_T                  dwStackSize,          // 栈大小，填0使用默认大小
  LPTHREAD_START_ROUTINE  lpStartAddress,       // 线程回调函数
  __drv_aliasesMem LPVOID lpParameter,          // 线程回调函数的参数
  DWORD                   dwCreationFlags,      // 创建标志
  LPDWORD                 lpThreadId            // 传出参数，线程ID
);
```

_beginthread 和 _beginthreadex

```
// 均属于CRT函数
uintptr_t _beginthread( // NATIVE CODE
   void( __cdecl *start_address )( void * ),
   unsigned stack_size,
   void *arglist
);
```

```
uintptr_t _beginthread( // MANAGED CODE
   void( __clrcall *start_address )( void * ),
   unsigned stack_size,
   void *arglist
);

uintptr_t _beginthreadex( // NATIVE CODE
   void *security,
   unsigned stack_size,
   unsigned ( __stdcall *start_address )( void * ),
   void *arglist,
   unsigned initflag,
   unsigned *thrdaddr
);

uintptr_t _beginthreadex( // MANAGED CODE
   void *security,
   unsigned stack_size,
   unsigned ( __clrcall *start_address )( void * ),
   void *arglist,
   unsigned initflag,
   unsigned *thrdaddr
);
```

挂起和恢复线程

Sleep放弃剩余的时间片

```
void Sleep(
  DWORD dwMilliseconds
);
```

SuspendThread、ResumeThread挂起和恢复线程

```
// 挂起线程，增加挂起次数
DWORD SuspendThread(
  HANDLE hThread
);

// 恢复挂起的线程，减少挂起次数
DWORD ResumeThread(
  HANDLE hThread
);
```

GetCurrentThread和GetCurrentProcess

`GetCurrentThread`永远返回`-2`即`0xfffffffe`，`GetCurrentProcess`永远返回`-1`即`0xffffffff`，这两个句柄都是伪句柄

要得到真正的句柄需要使用`DuplicateHandle`拷贝出真正的句柄，才能给其他线程/进程使用

```
// 伪进程句柄转真实进程句柄
HANDLE hProcess = NULL;
DuplicateHandle(GetCurrentProcess(), GetCurrentProcess(), GetCurrentProcess(),
                &hProcess, 0, FALSE, DUPLICATE_SAME_ACCESS);
//...
//不用的时候必须关闭 不然会有资源泄露
CloseHandle(hProcess);



// 伪线程句柄转真实线程句柄
HANDLE hThread = NULL;
DuplicateHandle(GetCurrentProcess(), GetCurrentThread(), GetCurrentProcess(),
                &hThread, 0, FALSE, DUPLICATE_SAME_ACCESS);
//..
//不用时也必须关闭
CloseHandle(hThread);
```

退出线程

从线程中return

```
// 线程回调函数
void thread_proc(void *arg)
{
    ...
    return ;
}
```

ExitThread

```
// 严禁在主线程中使用
void ExitThread(
  DWORD dwExitCode
);

// 获取退出码
BOOL GetExitCodeThread(
  HANDLE  hThread,
  LPDWORD lpExitCode
);
```

TerminateThread

```
BOOL TerminateThread(
  HANDLE hThread,
  DWORD  dwExitCode
);
```

_endthread 和 _endthreadex

```
void _endthread( void );

void _endthreadex(
   unsigned retval
);
```

资源释放情况

	DLL通知	线程栈	局部对象析构
从线程中return（自然死亡）	√	√	√
ExitThread	√	√	×
TerminateThread	×	√	×