线程篇（下）

七日成蝶——Windows API 编程入门

七日做茧，一朝成蝶！

主讲：袁春旭

个人博客：http://8413723.blog.51cto.com/

课程主页：http://edu.51cto.com/lecturer/8403723.html

可等待计时器WaitableTimer

可等待计时器WaitableTimer

创建可等待计时器：CreateWaitableTimer

创建可等待计时器：CreateWaitableTimerEx

打开可等待计时器：OpenWaitableTimer

设置可等待计时器：SetWaitableTimer

取消可等待计时器：CancelWaitableTimer

可等待计时器WaitableTimer

创建可等待计时器：

```
HANDLE  CreateWaitableTimer(
            LPSECURITY_ATTRIBUTES    lpTimerAttributes,
            BOOL                     bManualReset,
            LPCTSTR                  lpTimerName
);
```

lpTimerAttributes: 安全属性设置

bManualReset: 手动重置

lpTimerName: 计时器名称

可等待计时器WaitableTimer

创建可等待计时器：

```
HANDLE  CreateWaitableTimerEx(
        LPSECURITY_ATTRIBUTES    lpTimerAttributes,
        LPCTSTR                  lpTimerName,
        DWORD                    dwFlags,
        DWORD                    dwDesiredAccess
);
```

dwFlags: CREATE_WAITABLE_TIMER_MANUAL_RESET(0x00000001)手动重置

0x00000000 自动重置

可等待计时器WaitableTimer

dwDesiredAccess：

值	说明
DELETE (0x00010000L)	删除对象
READ_CONTROL (0x00020000L)	读取对象信息安全符信息
SYNCHRONIZE (0x00100000L)	对象同步
WRITE_DAC (0x00040000L)	修改DACL中的对象的安全描述符。
WRITE_OWNER (0x00080000L)	改变对象拥有者的安全描述符
MUTEX_ALL_ACCESS (0x1F0001)	互斥量对象所有权限
MUTEX_MODIFY_STATE (0x0001)	修改互斥量状态

可等待计时器WaitableTimer

dwDesiredAccess：

值	说明
EVENT_ALL_ACCESS (0x1F0003)	事件对象所有权限
EVENT_MODIFY_STATE (0x0002)	修改事件对象状态
TIMER_ALL_ACCESS (0x1F0003)	计时器对象所有权限
TIMER_QUERY_STATE (0x0001)	修改计时器状态
SEMAPHORE_ALL_ACCESS (0x1F0003)	信号量对象所有权限
SEMAPHORE_MODIFY_STATE (0x0002)	修改信号量状态

可等待计时器WaitableTimer

打开可等待计时器：

```
HANDLE  OpenWaitableTimer(
            DWORD       dwDesiredAccess,
            BOOL        bInheritHandle,
            LPCTSTR     lpTimerName
);
```

可等待计时器WaitableTimer

设置可等待计时器：

```
BOOL  SetWaitableTimer(
        HANDLE                  hTimer,
        const LARGE_INTEGER     *pDueTime,  //首次触发
        LONG                    lPeriod,        //触发频率间隔
        PTIMERAPCROUTINE        pfnCompletionRoutine,
        LPVOID                  lpArgToCompletionRoutine,
        BOOL                    fResume
);
```

LARGE_INTEGER说明：

```
typedef union _LARGE_INTEGER {
  struct {
    DWORD LowPart;
    LONG  HighPart;
  };
  struct {
    DWORD LowPart;
    LONG  HighPart;
  } u;
  LONGLONG QuadPart;
} LARGE_INTEGER, *PLARGE_INTEGER;
```

可等待计时器WaitableTimer

pDueTime：首次触发时间

指定首次触发时间点：

```
struct {
    DWORD LowPart;
    LONG  HighPart;
  };
```

指定启动后触发间隔：LONGLONG QuadPart; 单位：100ns

1秒 = 1000毫秒　　1毫秒 = 1000微秒　　1微秒 = 1000纳秒(ns)

1秒 = 10000000 * 100ns

可等待计时器WaitableTimer

lPeriod：触发频率间隔，单位：毫秒，0：不频率性触发

pfnCompletionRoutine：APC函数

APC (Asynchronous procedure call)异步过程调用函数

lpArgToCompletionRoutine：APC函数参数

可等待计时器WaitableTimer

取消可等待计时器：

```
BOOL  CancelWaitableTimer(
            HANDLE hTimer
);
```

可等待计时器WaitableTimer

```
int main()
{
        HANDLE hTimer = NULL;
        LARGE_INTEGER liDueTime;
        liDueTime.QuadPart = -50000000LL;
        hTimer = CreateWaitableTimer(NULL, TRUE, NULL);
        if (NULL == hTimer)
        {
                printf("CreateWaitableTimer failed (%d)\n", GetLastError());
                return 1;
        }
        printf("Waiting for 5 seconds...\n");
```

可等待计时器WaitableTimer

```
        if (!SetWaitableTimer(hTimer, &liDueTime, 0, NULL, NULL, 0))
        {
                printf("SetWaitableTimer failed (%d)\n", GetLastError());
                return 2;
        }
        if (WaitForSingleObject(hTimer, INFINITE) != WAIT_OBJECT_0)
                printf("WaitForSingleObject failed (%d)\n", GetLastError());
        else printf("Timer was signaled.\n");

        return 0;
}
```

与用户计时器的差别

与用户计时器的差别

用户计时器：需要在应用程序中使用大量的用户界面基础设施，消耗资源更多。WM_TIMER优先级最低，只有当线程对象没有任何其它消息的时候才会被处理。

可等待计时器：内核对象，可以多线程间共享，具备安全性。如果计时器被触发且线程正在等待，那么系统将唤醒线程。

编码实战

Thank You !