**《Windows核心编程》读书笔记**

# 0概述

作者：keenjin（金才）

时间：2017/11/1

计划阅读：**2**个月

截止日期：2017/1/1

# 1内核对象

# 2进程

# 3线程、调度及同步

## 3.1线程使用注意事项

划重点：

1. 伪线程句柄永远表示当前线程所在句柄，有歧义，在传递句柄时候，尽可能使用DuplicateHandle将GetCurrentThread转换成真正句柄。这个只有在父子线程传递的时候才需要用到。
2. \_beginthread一定不要用，而用\_beginthreadex，因为\_endthread内部会调用CloseHandle，外界如果再次调用CloseHandle会导致句柄重复释放
3. 线程栈默认占用虚拟内存1M，占用物理内存1K。如果物理内存1K用完，请求会触发页面异常中断；但是如果线程栈占用空间超过1M，则会导致栈溢出。

## 3.2线程上下文

|  |
| --- |
| typedef struct \_CONTEXT {  //  // The flags values within this flag control the contents of  // a CONTEXT record.  //  // If the context record is used as an input parameter, then  // for each portion of the context record controlled by a flag  // whose value is set, it is assumed that that portion of the  // context record contains valid context. If the context record  // is being used to modify a threads context, then only that  // portion of the threads context will be modified.  //  // If the context record is used as an IN OUT parameter to capture  // the context of a thread, then only those portions of the thread's  // context corresponding to set flags will be returned.  //  // The context record is never used as an OUT only parameter.  //  DWORD ContextFlags;  //  // This section is specified/returned if CONTEXT\_DEBUG\_REGISTERS is  // set in ContextFlags. Note that CONTEXT\_DEBUG\_REGISTERS is NOT  // included in CONTEXT\_FULL.  //  DWORD Dr0;  DWORD Dr1;  DWORD Dr2;  DWORD Dr3;  DWORD Dr6;  DWORD Dr7;  //  // This section is specified/returned if the  // ContextFlags word contians the flag CONTEXT\_FLOATING\_POINT.  //  FLOATING\_SAVE\_AREA FloatSave;  //  // This section is specified/returned if the  // ContextFlags word contians the flag CONTEXT\_SEGMENTS.  //  DWORD SegGs;  DWORD SegFs;  DWORD SegEs;  DWORD SegDs;  //  // This section is specified/returned if the  // ContextFlags word contians the flag CONTEXT\_INTEGER.  //  DWORD Edi;  DWORD Esi;  DWORD Ebx;  DWORD Edx;  DWORD Ecx;  DWORD Eax;  //  // This section is specified/returned if the  // ContextFlags word contians the flag CONTEXT\_CONTROL.  //  DWORD Ebp;  DWORD Eip;  DWORD SegCs; // MUST BE SANITIZED  DWORD EFlags; // MUST BE SANITIZED  DWORD Esp;  DWORD SegSs;  //  // This section is specified/returned if the ContextFlags word  // contains the flag CONTEXT\_EXTENDED\_REGISTERS.  // The format and contexts are processor specific  //  BYTE ExtendedRegisters[MAXIMUM\_SUPPORTED\_EXTENSION];  } CONTEXT; |

通过BOOL WINAPI GetThreadContext(\_\_in HANDLE hThread, \_\_in\_out LPCONTEXT lpContext);

可以获取到线程句柄对应的线程上下文

其中，FS寄存器，记录了TEB结构地址

## 3.3线程调度

（1）大约每隔20ms（GetSystemTimeAdjustment第二个参数的返回），Windows会查看当前所有存在的线程内核对象，查找可调度的（未被挂起的线程），载入CPU寄存器

（2）CreateThread时，线程挂起计数为1，直到初始化结束，会检查是否传入CREATE\_SUSPENDED，没有，挂起计数递减为0

（3）SuspendThread要非常小心，因为线程如果正在分配堆中的内存，线程将锁定堆，将它挂起，当前进程的其他线程就会一直等待这个线程恢复，才能使用堆

（4）Sleep传入非0，会导致当前线程在这个时间段内不被调度，不存在线程上下文切换。传入0，只是主动放弃主调线程剩余的时间片，下次任意时间依旧可以被调度。

（5）Sleep（0）和SwitchToThread功能类似，只不过前者重新回到调用Sleep的线程，后者可以选择到饥饿线程

（6）调用GetThreadContext前，应该先调用SuspendThread

## 3.4线程优先级

（1）每个线程都被赋予0（最低）~31（最高）的优先级数，高优先级线程会优先被执行，再逐步检查低优先级的

（2）20ms检查线程状态的时候，较高优先级的线程总是会抢占较低优先级的线程（抢占式操作系统），即使较低优先级的线程的时间片依旧存在。

（3）Windows提供了6个大的进程优先级类，如下：

idle（IDLE\_PRIORITY\_CLASS）：系统空闲时运行，例如屏保程序、后台使用程序和统计数据收集软件

below normal（BELOW\_NORMAL\_PRIORITY\_CLASS）

normal（NORMAL\_PRIORITY\_CLASS）：99%的应用程序都在这一层

above normal（ABOVE\_NORMAL\_PRIORITY\_CLASS）

high（HIGH\_PRIORITY\_CLASS）：此进程中的线程必须立即响应事件，例如任务管理器

real-time（REALTIME\_PRIORITY\_CLASS）：会抢占操作系统组件的CPU时间

设置进程优先级的方法有两种

方法一：由父进程创建子进程时，传递参数，如下：

BOOL WINAPI CreateProcess(

……

// NORMAL\_PRIORITY\_CLASS、IDLE\_PRIORITY\_CLASS、BELOW\_NORMAL\_PRIORITY\_CLASS

**DWORD dwCreationFlags,**

……

);

方法二：调用SetPriorityClass提权

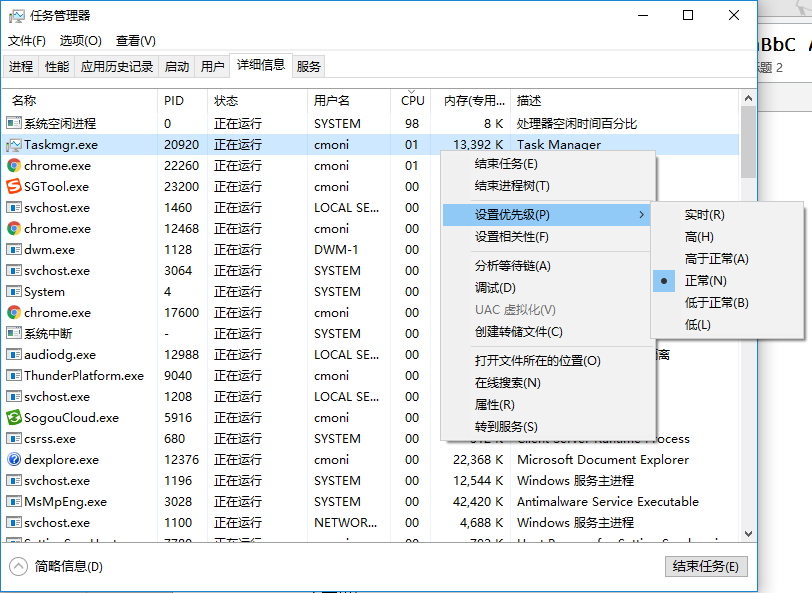
BOOL WINAPI SetPriorityClass(

\_\_in HANDLE hProcess,

\_\_in DWORD dwPriorityClass

);

DWORD WINAPI GetPriorityClass(HANDLE hProcess);



（4）Windows提供了7个大的相对线程优先级类，如下：

idle（THREAD\_PRIORITY\_IDLE）

lowest（THREAD\_PRIORITY\_LOWEST）

below normal（THREAD\_PRIORITY\_BELOW\_NORMAL）

normal（THREAD\_PRIORITY\_NORMAL）

above normal（THREAD\_PRIORITY\_ABOVE\_NORMAL）

highest（THREAD\_PRIORITY\_HIGHEST）

time-critical（THREAD\_PRIORITY\_TIME\_CRITICAL）

线程优先级只有一种方式可以改变，如下：

BOOL WINAPI SetThreadPriority(

\_\_in HANDLE hThread,

\_\_in int nPriority

);

int WINAPI GetThreadPriority(HANDLE hThread);

（5）CreateThread总是创建相对线程优先级为normal的线程，要创建idle的线程，需要传入CREATE\_SUSPENDED标记，然后设置线程优先级为idle，然后再ResumeThread

（6）从Vista开始，可以进行I/O请求时设置优先级SetThreadPriority（THREAD\_MODE\_BACKGROUND\_BEGIN），提升性能，同时还能针对文件维度，设置优先级（SetFileInformationByHandle）

## 3.5用户模式线程同步方式

4设备IO及线程池

5内存、内存映射文件及堆

6动态链接库

7异常