

Multithreading Application in Win32

Chapter 5_Keeping Your Threads on a Leash

Doo_Hee, Lee

aa2591@mmlab.net

[Multimedia Lab.](#)

Dept. of Electrical and Computer Eng.

University of Seoul

Seoul, Korea

5.0 Contents

1. Stopping a Thread Cleanly.
2. Thread Priority.
3. Initializing a Thread.

5.1 Main Objectives

1. How to stop a running thread
2. How to understand and adjust thread priorities
3. How to initialize a new thread

5.2 Stopping a Thread Cleanly(1/8)

- *How do stop a current running thread from within another thread?*

1)With `TerminateThread()`.

2)Using Signals.

3)Throwing Exception across Threads.

4)Setting a Flag.

5.2 Stopping a Thread Cleanly(2/8)

1) Aborting a thread with TerminateThread()

- The first and most obvious solution to this problem is to use TerminateThread().

```
Bool TerminateThread(  
HANDLE hThread           : Handle to the thread to terminate  
                          The "target Thread".  
DWORD dwExitCode        : Exit code that should be reported  
                          by thread's handle.  
);
```

Return Value :
Succeed : TRUE.
Fail : FALSE.

5.2 Stopping a Thread Cleanly(3/8)

- “**TerminateThread** is a dangerous function that should be used in the **most extreme cases**”

Problem

- No chance to clean up after itself.
- Forcible abort a thread without allowing that thread to have any say in the matter.
- The target thread's stack is not deallocation causing a large memory leak.
- Any DLLs that are attached to the thread will not be notified that the thread is detaching.

Other more insidious Problem

- If that thread was in the middle of a critical section then the critical section will **stay permanently locked**.
- If the target thread was in the middle of updating a data-structure will be left permanently in an unstable state.

The Summary of this call

- *Stay away from it.*

2) Using Signals

IN Unix,

- signals are the standard way of sending notifications across processes.
- Ex)
 - SIGTERM : a request to "please wait".
 - SIGKILL : the rough equivalent of TerminateThread()

5.2 Stopping a Thread Cleanly(6/8)

In C run-time library,

- Supports the standard signals.
- Ex)
 - SIGABRT : Abnormal termination. The default action terminates the calling program with exit code 3.
 - SIGINT : CTRL+C interrupt. The default action issues INT 23H.
 - There is no call named kill() : Sends a signal in UNIX.
- There is call named raise(), but that only sends a signal *within the current thread*.
- There is *no native support* in Win32 for signals.

3) Throwing Exceptions across Threads

- There is *no standard supported way* in the Win32API to throw an exception into another thread

4) Setting a Flag

- The approved way in Win32 to cause a thread to terminate is to *set a flag in your code* that asks the thread to terminate itself.
 - Need some sort of *polling mechanism* to check and see if it should exit.
 - Use a *manual-reset event object*
-
- *Look at a example program*

5.3 Thread Priority(1/8)

- In Win32
 - Priority that determinates what thread gets to run next
 - The priority is a number that is calculated based on
 - 1) The process's priority class
 - 2) The priority of the thread
 - 3) Dynamic boost

5.3 Thread Priority(2/8)

Round Robin scheduling.

- *Same priority class & priority level*
- *Take turns running*

5.3 Thread Priority(3/8)

1) Priority Class

- ~ is an attribute of a process
- ~ represents how important this process is compared to other processes in the system.
- There is four priority class in Win32

Priority Class	Base Priority Level
HIGH_PRIORITY_CLASS	13
IDLE_PRIORITY_CLASS	4
<i>NORMAL_PRIORITY_CLASS</i>	<i>7 or 8</i>
REALTIME_PRIORITY_CLASS	24

5.3 Thread Priority(4/8)

2) Priority Level

- There are seven priority levels

Priority Level	Adjustment to Base
THREAD_PRIORITY_HIGHEST	+2
THREAD_PRIORITY_ABOVE_NORMAL	+1
THREAD_PRIORITY_NORMAL	0
THREAD_PRIORITY_BELOW_NORMAL	-1
THREAD_PRIORITY_LOWEST	-2
THREAD_PRIORITY_IDLE	Set to 1
THREAD_PRIORITY_TIME_CRITICAL	Set to 15

- SetThreadPriority()
- GetThreadPriority()

5.3 Thread Priority(5/8)

- **BOOL SetThreadPriority(
HANDLE hThread,
int nPriority
);**

-Parameters

hThread : Handle to the thread whose priority should be changed.

nPriority : A value as shown table.

-return value

Succeed : thread priority.

Fail : FALSE.

5.3 Thread Priority(6/8)

- **int GetThreadPriority(
HANDLE hThread
);**

-Parameters

hThread : Handle to the thread whose priority should be examined.

-Return value

Succeed : TRUE.

Fail : THREAD_PRIORITY_ERROR_RETURN.

5.3 Thread Priority(7/8)

3) Dynamic Boost

- ~ is an adjustment to current priority.

- 1) All thread in the foreground application. : +2
- 2) All thread in a process and happens in response to input from the user or from the disk. : +5
- 3) On a per-thread basis when a wait condition is satisfied.

5.3 Thread Priority(8/8)

- **Starvation**
 - Low Priority worker thread(or Process)may never execute
- **Aging**
 - As time progresses increase the priority of the thread(or process)

5.4 Initializing a Thread(1/5)

- Initializing a Thread
 - 1) Adjust a new thread's priority
 - 2) Set a thread's preferred processor on SMP systems. (SMP : Symmetric Multi Process : CPU)

Problem

if you call `CreateThread()` in its default form then the new thread is already running and it is too late to do initial setup.

Solution

fifth parameter of `CreateThread()` :
CREATE_SUSPENDED

5.4 Initializing a Thread(2/5)

```
HANDLE hThread;
```

```
DWORD threadId;
```

```
hThread = CreateThread(NULL,0,ThreadFunc,0
```

```
CREATE_SUSPEND,&threadId);
```

```
SetThreadPriority(hThread,THREAD_PRIORITY_IDLE);
```

- You can start the thread running with `ResumeThread()`.

5.4 Initializing a Thread(3/5)

- **BOOL ResumeThread(
HANDLE hThread
);**
- **Parameters**
 - hThread : Handle to the thread to restart.
- **Return value**
 - Succeed : the previous suspend count of the thread.
 - Fail : 0xFFFFFFFF

5.4 Initializing a Thread(4/5)

- `BOOL SuspendThread(
HANDLE hThread
);`
 - **Parameters**
 - `hThread` : Handle to the thread to suspend.
 - **Return value**
 - Succeed : the current suspend count of the thread.
 - Fail : `0xFFFFFFFF`.

5.4 Initializing a Thread(5/5)

- Suspending Thread
 - Companion Function to ResumeThread().
 - SuspendThread().
 - Allows the caller to put a thread to sleep.
 - Not walk until someone call ResumeThread().

- Problem

Deadlock

