Monkey Weekend

Unleash the monkey within you.

Win32 Mutex, HANDLEs and WinDbg !handle extension.

Posted on July 26, 2014March 8, 2016 by kiewic

This blog moved to kiewic.com (http://kiewic.com)

A **mutex** is like a lock that is shared between processes. So if we run two instances of the following program at the same time, the second instance will hang/wait 10 seconds before acquiring the mutex.

```
#include "stdafx.h"
#include <Windows.h>
#include <synchapi.h>

int main(int argc, _TCHAR* argv[])
{
    WCHAR* cacheMutexName = L"HelloWorldMutex";
    HANDLE handle = ::CreateMutex(nullptr, FALSE, cacheMutexName);

    DWORD result = WaitForSingleObject(handle, INFINITE);

    switch (result)
    {
    case WAIT_OBJECT_0:
        printf("The thread got ownership of the mutex.\r\n");
        break;
    case WAIT_ABANDONED:
        printf("The thread got ownership of an abandoned mutex.\r\n");
        break;
    }

    // Countdown.
    for (int i = 10; i > 0; i--)
    {
        wprintf(L"%d\r\n", i);
        Sleep(1000);
    }
    wprintf(L"0\r\n");

    ReleaseMutex(handle);
    printf("Mutex released.\r\n");

    return 0;
}
```

In Win32, once you get the mutex, you can distinguish between two different cases. The first case is `WAIT_OBJECT_0` and it means the previous owner released the mutex properly by calling `ReleaseMutex()` and the second case is `WAIT_ABANDONED` and it means the previous owner terminated without calling `ReleaseMutex()`.

To reproduce the `WAIT_ABANDONED` case with the sample program, press **CTRL + C** in the first instance before the countdown hits zero.

When using WinDbg, during live debugging or during dump analysis, the **!handle** extension comes very handy.

Just get the handle value:

```
0:000> dv
 argc = 0n1
 argv = 0x010f6f28
 handle = 0x00000038
 result = 0xcccccccc
 cacheMutexName = 0x003f5858 "HelloWorldMutex"
```

And print all the handle info:

```
0:000> !handle 0x00000038 f
Handle 38
 Type Mutant
 Attributes 0
 GrantedAccess 0x1f0001:
 Delete,ReadControl,WriteDac,WriteOwner,Synch
 QueryState
 HandleCount 3
 PointerCount 98306
 Name \Sessions\1\BaseNamedObjects\HelloWorldMutex
 Object Specific Information
 Mutex is Owned
 Mutant Owner 11ac.1628
```

Now we know **11ac.1628** is the owner of the mutex:

If we print the call stack for each thread in the second instance of the program:

```
0:000> ~* kcn

.  0  Id: 29f8.778 Suspend: 1 Teb: 7f15c000 Unfrozen
 #
00 ntdll!NtWaitForSingleObject
01 KERNELBASE!WaitForSingleObjectEx
02 KERNELBASE!WaitForSingleObject
03 MutexExample!main
04 MutexExample!__tmainCRTStartup
05 MutexExample!mainCRTStartup
06 KERNEL32!BaseThreadInitThunk
07 ntdll!__RtlUserThreadStart
08 ntdll!_RtlUserThreadStart

#  1  Id: 29f8.ec4 Suspend: 1 Teb: 7f159000 Unfrozen
 #
00 ntdll!DbgBreakPoint
01 ntdll!DbgUiRemoteBreakin
02 KERNEL32!BaseThreadInitThunk
03 ntdll!__RtlUserThreadStart
04 ntdll!_RtlUserThreadStart
```

We do not see any **11ac.1628** thread.

But if we print the call stacks of the first process intance:

```
0:000> ~* kcn

.  0  Id: 11ac.1628 Suspend: 1 Teb: 7fe7d000 Unfrozen
 #
00 ntdll!NtDelayExecution
01 KERNELBASE!SleepEx
02 KERNELBASE!Sleep
03 MutexExample!main
04 MutexExample!__tmainCRTStartup
05 MutexExample!mainCRTStartup
06 KERNEL32!BaseThreadInitThunk
07 ntdll!__RtlUserThreadStart
08 ntdll!_RtlUserThreadStart

#  1  Id: 11ac.1c48 Suspend: 1 Teb: 7fe7a000 Unfrozen
 #
00 ntdll!DbgBreakPoint
01 ntdll!DbgUiRemoteBreakin
02 KERNEL32!BaseThreadInitThunk
03 ntdll!__RtlUserThreadStart
04 ntdll!_RtlUserThreadStart
```

Then we see the **11ac.1628** thread.

In fact 0x11ac and 0x29f8 match the PID of the processes:

```
C:\>tlist

4524 MutexExample.exe
10744 MutexExample.exe
```

Happy handle debugging!

SECURE YOUR DEVICES ONLINE WITH
UP TO 40% OFF*
NORTON™ SECURITY
Norton by Symantec
*First year introductory offer. See subscription details.

Tagged C++, MSDN, Win32, WinDbg