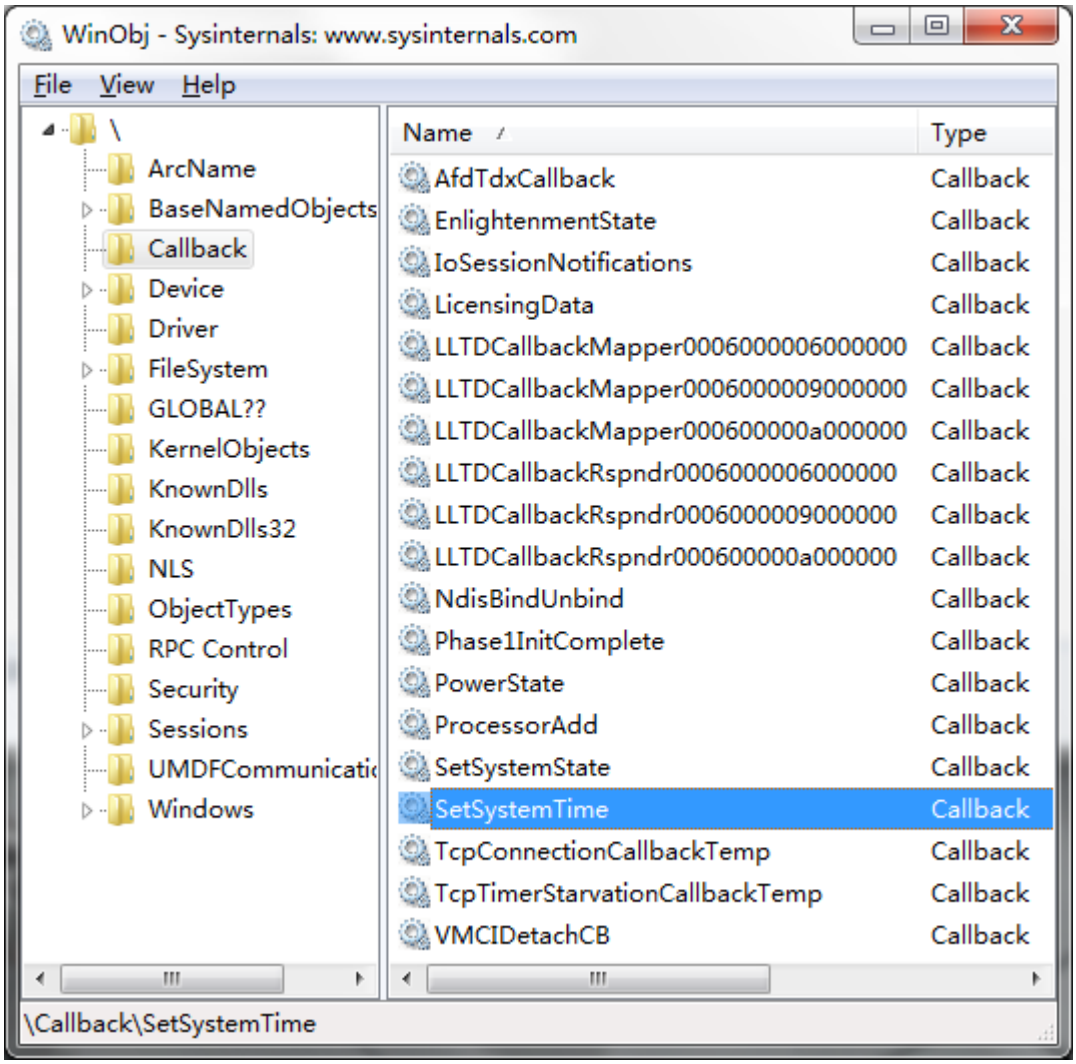
在 WIN32 时代，通过 HOOK NtSetSystemTime 可以拦截对时间日期的修改。不过在 WIN64 里，实现内核级别的监控已经不可能了，但实现监视还是没问题的（注意“监控”和“监视”这两个词语的区别）。

在讲监视时间修改之前，必须说一种神奇的内核对象，叫做“回调”。如果你注册一个回调函数，就会在系统发生指定变化的时候通知你，不过属于“事后诸葛亮”类型的，就是说，当你知道某个事情已经发生的时候，已经无法阻止了。既然说它是一个内核对象，也就是说任何驱动都可以自行注册一个回调类型（注意“注册回调函数”和“注册回调类型”的区别）。系统中有什么“回调”可以通过 WINOBJ 查看：




从图中可见这些回调的名字乱七八糟的，很明显大多是第三方驱动注册的。不过也有几个是系统自带的，比如 ProcessorAdd、SetSystemState、SetSytemTime 等。这次我们使用的，是 SetSytemTime。注册回调函数非常简单，先使用 ExCreateCallback 打开一个回调类型（这货跟 CreateFile 的德行一样，可以作为 Create 和 Open 两用），然后使用 ExRegisterCallback 把你的回调函数“登记在册”。如果不用了，就使用 ExUnregisterCallback 注销即可。实现监视时间日期改变的代码非常简单：

```
PVOID g_CbRegistration;
```

```
VOID SetSystemTimeNotify
(
	IN PVOID CallbackContext,
	IN PVOID Argument1,
	IN PVOID Argument2
)
{
	//此回调函数的 IRQL 为 2，大部分内核函数不能在里面使用！否则 BSOD！
	DbgPrint("[SetSystemTimeNotify]IRQL: %ld\n",KeGetCurrentIrql());
}

NTSTATUS RegisterSetSystemTimeNotify(BOOLEAN IsUndo)
{
	if(!IsUndo)
	{
		PCALLBACK_OBJECT pCallBackObj;
		OBJECT_ATTRIBUTES oa ;
		UNICODE_STRING callbackname;
		NTSTATUS status;
		//打开回调类型
		RtlInitUnicodeString(&callbackname,L"\\Callback\\SetSystemTime");
		InitializeObjectAttributes(&oa,&callbackname,OBJ_CASE_INSENSITIVE,0,0);
		status = ExCreateCallback(&pCallBackObj,&oa,TRUE,FALSE);
		if(!NT_SUCCESS( status ))
			return status;
		//注册回调函数
		g_CbRegistration = ExRegisterCallback(pCallBackObj,SetSystemTimeNotify ,NULL);
		if(g_CbRegistration == NULL)
			return STATUS_UNSUCCESSFUL;
		ObDereferenceObject(pCallBackObj);
		return STATUS_SUCCESS;
	}
	else
	{
		if(g_CbRegistration == NULL)
			return STATUS_UNSUCCESSFUL;
		//反注册回调函数
		ExUnregisterCallback(g_CbRegistration);
		return STATUS_SUCCESS;
	}
}
```

不过回调函数里的环境非常坑爹，它的 IRQL 竟然为 2。也就是说，在回调函数里，大多数的内核 API 不能使用，否则将会引发 IRQL_NOT_LESS_OR_EQUAL 蓝屏。代码的执行效果如

下（其实也真没啥效果了，只能打印点东西，因为调用其它函数必然蓝屏）：

```
7.18475962      [SetSystemTimeNotify]IRQL: 2
7.18520498      [SetSystemTimeNotify]IRQL: 2
```

总体来说，这类“回调”的用途不大。如果非要实现监控时间改变，必须增加非常麻烦的异步处理函数。