DEVGURU

Sharing Events Between Kernel mode and User mode

이번 컬럼에서는 User mode 와 Kernel mode 사이에 event를 공유하는 방법을 알아보자. 공유(Shared) event 는 이름이 의미하듯이, 하나의 프로그램에서 어떤 종류의 event 가 일어났을 때 다른 프로그램에게 알리기 위한 방법으로 사용한다. 아마도 여러분의 드라이버가 애플리케이션의 주소공간에 맵핑되어 있는 공유 메모리 버퍼를 사용할 수 있다. 이 버퍼를 동기적으로 사용하기 위해 한 쌍의 event 을 사용할 수 있다. 먼저, 하나의 event 는 애플리케이션이 드라이버에게 버퍼로부터 데이터를 가지고 오라고 signal을 주기 위해 사용된다. 그리고, 다른 event 는 드라이버가 애플리케이션에게 버퍼로부터 데이타를 가져오라고 signal를 주기 위해 사용된다. 물론 이런 종류의 기능은 비동기적인 IRP를 기다리는 방법을 통해서 구현해도 된다. 그러나 event를 사용한 방법이 비동기 IRP 를 사용하는것보다는 overhead가 덜 든다.

event 를 공유하는 방법에는 몇가지가 있는데, 그중에서 다음과 같은 방법이 있다,

- user-mode 애플리케이션이 event 를 만들고, 그 event 의 handle 를 IOCTL 를 driver 에게 전달한다.
- driver 가 event 를 만들고, 그 event 의 handle 를 IOCTL 를 통해 user-mode 애플리케이션에게 전달한다.
- driver 가 미리 정한 이름으로 event 를 만들고, 애플리케이션이 그 event 를 open 한다.

이 방법들은 각각 장,단점과 잠재적인 문제점들이 있다. 그럼, 이제부터 위의 방법들에 대해서 설명해보겠다.

Sharing an Event by Handle

모든 사람들이 쉽게 이해하도록 구현하는 것은 user mode 에서 event의 handle 를 open 해서, driver 에게 IOCTL를 통해 handle를 전달하는 것이다. 그러면, driver 의 dispatch routine에서는 ObReferenceObjectByHandle(,,,)을 호출해서 event 에 reference count 를 증가시키고, (user 가 event 를 close 시켜도, delete 되지 않도록 하기위해서이다.) 그리고, user mode 애플리케이션의 event 를 signal 시키기 위해 사용할 KEVENT 변수에 대한 포인터를 리턴받는다. 그러나, 이 방법은 약간의 문제가 있다. 어떤 경우에서는 이 방법이 잘 작동할 수도 있지만, 어떤 경우에서는 실패할 수도 있다. User mode 에서의 handle 은 단지 process handle table 의 인덱스 일 뿐이다. 따라서, 애플리케이션의 process context 내에서 ObReferenceObjectHandle 를 호출할 경우에만 100 % 잘 작동한다고 확신할 수있다. 만약 여러분의 드라이버가 다른 드라이버 스택의 중간에 있게 되거나, 직접 named

device object 를 export 하지않는다면, 여러분의 드라이버 dispatch routine 이 실행되고 있는 context 가 호출한 애플리케이션의 context 인지 확신할 수 없다. 여러분이 IoRegisterDeviceInterface(,,,,) 를 사용하면, IRP 는 여러분의 device object 가 있는 stack 의 맨위로 갈 것이다. 그래서 만약 여러분이 작성한 드라이버가 디스크 port driver 의 위에 있는 filter driver 이고, 여러분의 드라이버에서 device interface 를 export 했다면, device interface로 보내진 요청은 stack 의 맨위로 가게 될것이다. 그리고, 여러분이 IOCTL 를 받게 될때는 arbitrary context 에 있을 수 있다.

argument 를 얘기를 하기 위하여, 다음과 같이 가정해보자. 여러분의 드라이버가 named device object 를 export 한 legacy device driver 이고, 이 드라이버는 적절한 context 에서 있고, dispatch routine 은 호출한 process의 context 에서 실행된다는 것을 확신할 수 있다. 이런 가정에서, 여러분이 확인할 필요가 있는 것은 ObReferenceObjectByHandle 함수를 올바르게 사용하고 있는가 이다. ObjectType 인자는 *ExEventObjectType로 지정해야만 한다. 이것은 악의적인 사용자가 event handle 과 다른 인자를 넘기는 것으로부터 막을 수 있다. 만약 handle 이 event 가 아니면, STATUS_OBJECT_TYPE_MISMATCH 를 리턴한다.

그리고, 여러분이 주의해야할 또 다른 인자는 AccessMode 인자이다. 이것은 **UserMode** 로 설정해야 한다. 호출하는 애플리케이션이 이 handle 에 충분히 접근할 수 있다는것을 확인하기 위해 체크한다. 그리고, 다른 이유는 절대적으로 호출 프로세스의 context 에 있다는 것을 확신시켜준다.

이 구현방법을 반대로해서, kernel mode 에서 event 를 열고, user mode 에서 사용할 핸들을 넘겨주는 것은 아주 잘못된 생각이다. 여러분이 이것을 구현하려한다면, 처음 만나게 될 장애물은 IoCreateNotificationEvent(,,,) 에서 리턴된 KEVENT object 의 pointer 를 애플리케이션에서 사용할 적절한 reference count 를 가진 HANDLE 로 바꿀려고 할 것이다. 그러나, IoCreateNotificationEvent() 에서 리턴된 핸들은 사용할수 없다. 한가지 이유는 리턴된 핸들은 단지 kernel mode 에서만 유효한 handle 이기 때문이다. 물론, 여러분이 노력을 통해서 적절한 핸들을 얻을 수는 있지만, 필요한 API 함수는 문서화되지 않았다.

Sharing Event by Name

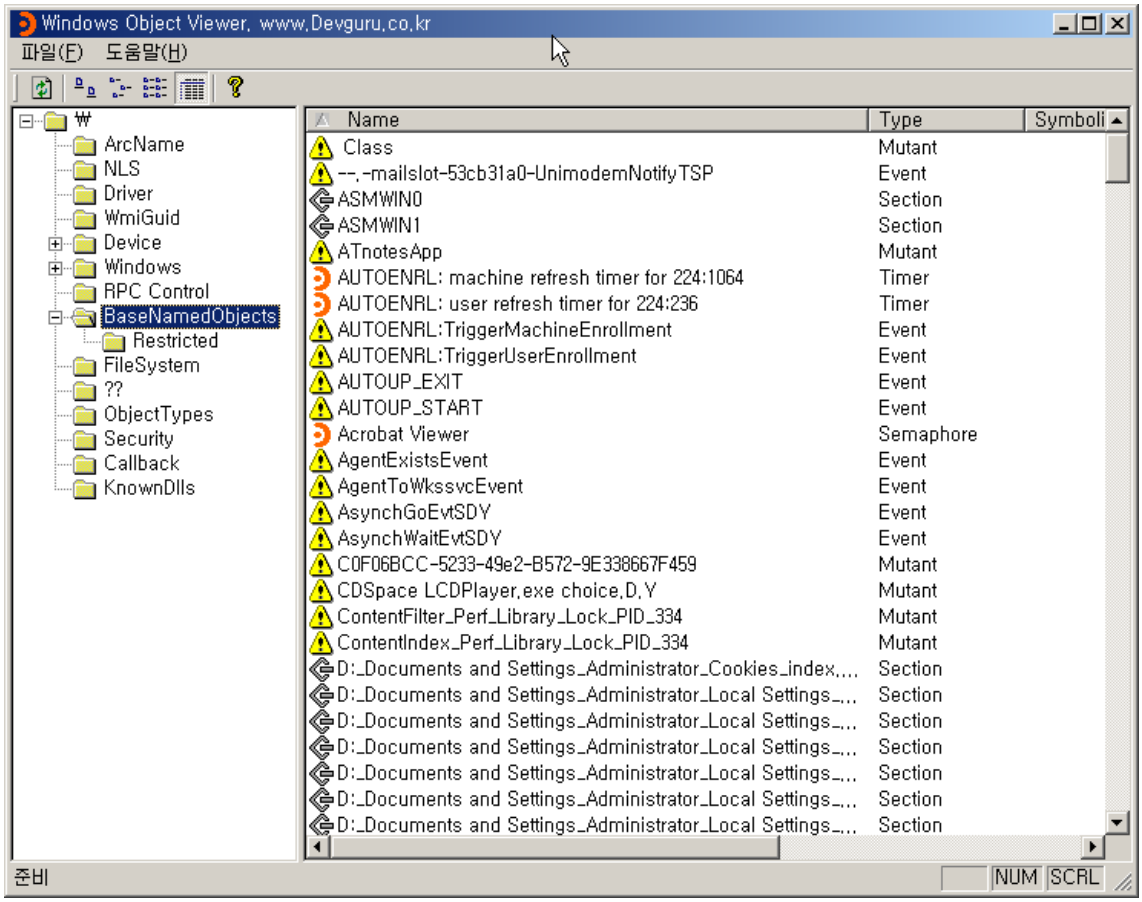
만약 내가 핸들을 공유할 수 없다면, 어떻게 나의 애플리케이션과 드라이버가 event 를 공유할 수 있는가? 실제로 그건 아주 간단하다. 핸들은 매우 휘발성 강해서, 일반적으로 신뢰할 수 없다. 그러나, 그것이 여러분의 드라이버에서 호출할 수 있는 named event 를 만들 수 없는 이유는 아니다. 이 구현 방법은 더 이상 context 가 문제가 되지않기 때문에 전에 있던 구현 방법의 제약조건 들은 문제가 되지 않는다. Object manager 내의 Named Object 는 모든 process context 에서 사용할 수 있다. 그리고, 그것들과 관련된 모든 security 속성을 가진다. 그래서 더 이상 호출한 애플리케이션의 context 내에 있는지 permission 을 검사할 필요가 없다.

이름에 의해 event 를 공유하는 예를 보자. user mode 에서 event 를 만들기 위해 SDK 함수인 CreateEvent(,,,) 를 사용한다.

```
SharedEvent = CreateEvent( NULL, TRUE, FALSE, "SharedEvent" );
```

CreateEvent 에 대한 호출이 성공적이면, 우리는 object manager 내에서 우리의 named evnet를 볼 수 있다. 여러분들이 user mode 에서 event 를 만들때, 그것은 WBaseNamedObjects 디렉토리에 놓는다.

여러분은 그정보를 DevObj (http://www.devguru.co.kr/pds01.asp?tg=pds&pds_part=ut) 를 통해서 볼 수 있다.



어떤 애플리케이션에서는 단지 named event 만 만드는 것은 가능하다. 다른것으로는, 우리는 driver 에서 이벤트가 만들어졌고, 오픈할 준비가되었다고 알 필요가 있다. 우리 구현에서 이것은 DeviceIoControl(,,,) 표준 함수에 의해 수행된다.

```
DeviceIoControl( DeviceDriver,
                 IOCTL_OPEN_SHARE_EVNET,
                 NULL, 0,
```

```
                    NULL, 0,
                    &bytesReturened,
                    NULL );
```

드라이버에서 코드는 간단하다. 대신 event 는 WBaseNamedObjects 에 있다는 것을 잊지 말아라. 여러분이 user mode 에 있다면 자동으로 추가되지만, kernel mode 에서 evnet 를 open 하려고 하면 명시적으로 추가를 시켜줘야만 한다.

```
HANDLE  SharedEvnetHandle = NULL;
PKEVNET SharedEvent = NULL;
UNICODE_STRING EventName;

RtlInitUnicodeString( &EventName, L"WWBaseNamedObjectsWWSharedEvent" );
SharedEvent = IoCreateNotificationEvent( &EventName, &SharedEventHandle );
if ( SharedEvent != NULL ){
        status = STATUS_SUCCESS;
}
else {
    status = STATUS_UNSUCCESSFUL;
}
```

여러분은 SharedEvent 변수를 KeSetEvent(,,,) 에 인자로 사용할수 있다. 이 함수를 호출하면 여러분의 user mode 에 event handle 를 signal 시킨다.

Kernel mode 에서 Event 를 만들기 ?

물어볼 한가지 질문이 있다. 마지막 구현한 것을 반대로 할 수 있나? - kernel mode 에서 named event 를 만들고, user mode 에서 그것을 open 하기 - 절대적으로 불가능한 생각인가 ? 이 질문에 대한 정확한 답변은 우선 여러분은 event 를 WBaseNamedObjects 디렉토리 아래에 만들었다는 것을 확신해야 하고, 우리는 IoCreateNotificationEvent 함수를 호출해 event 가 이미 존재하지 않는다면, event를 새로 만들 것이다. 만약 여러분의 드라이버가 부팅 시점에 로딩된다면,그것은 BaseNamedObjects 디렉토리를 만드는 Win32 전에 로딩된다. 그러면 Win32 초기화전에 드라이버의 DriveEntry routine 에서 event 를 만들려고 할 것이고, 이것은 아마도 원하는 결과로 리턴되지 않을 것이다. 그리고, 보안의 문제도 있다. IoCreateNotifiactionEvent 은 event 에 보안 속성을 지정할 인자를 가지지 않기 때문에, event에 적용될 security는 현재 쓰레드의 것이 된다. 그것은 여러분이 event를 만들때 system process 의 context 내에서 실행되고 있다는 것을 의미한다.(DriveEntry, AddDevice 에 있는것처럼) 애플리케이션은 event 를 접근할 수 있기 위해서는 관리자 계정

에서 실행될 필요가 있다.

앞에서 Event 를 공유하기 위한 두가지 방법을 살펴보았다. 두가지 방법이 각각 장,단점을 가지고 있다. 하지만 뒤에 소개한 방법을 더 사용하길 권한다. 특정 process context 에서 발생할 문제가 없기 때문에 사용하는데 제약이 더 없기 때문이다.
여러분들도 한번 각각의 방법들을 사용해 보고 문제점이 무엇인지, 어떻게 사용해야 할지 공부해 보도록 하자.

그럼, 다음 칼럼에서 또 봅시다.