# 에러 핸들링

윈도우 함수가 실패하게 되면 내부적으로 함수를 호출한 스레드의 스레드 지역 저장소(thread-local storage)에 적절한 에러 코드를 저장해 둔다.

GetLastError() 저장된 스레드 지역 저장소로부터 **가장 최근**에 호출된 함수를 가져온다.

에러는 3가지 요소로 구성되어 있다 WinError.h 의 극히 일부분

1. MessageId: ex) ERROR\_SUCCESS
2. MessageText: ex) The operation completed successfully
3. Error Code: 0L

FormatMessage는 에러 코드를 메시지 텍스트로 변환해주는 함수다. 마소에서는 윈도우 개별 함수에 대해 모든 에러 코드를 제공하지는 않는다. 이러한 이유는 새로운 운영체제 등장 시 최신화가 어려울 뿐더러 호출된 함수가 내부적으로 또 다른 함수를 호출하게 되는 것을 반복하게 될 것이다.

FromatMessage FLAG  
FORMAT\_MESSAGE\_FROM\_SYSTEM – 운영체제가 정의하고 있는 에러 코드와 대응되는 메시지 텍스트를 얻겠다.  
FORMAT\_MESSAGE\_ALLOCATE\_BUFFER – 여러 메시지 텍스트를 저장할 수 있는 충분한 메모리 공간을 할당해 줄 것을 요청. 해당 메모리 블록 핸들은 hlocal변수를 통해 반환  
FORMAT\_MESSAGE\_IGNORE\_INSERT - %로 시작하는 자리 표시자를 실질적인 값으로 변경하지 않을 것을 지정. 상황에 맞는 추가 정보를 제공하기 위해 주로 사용

<https://docs.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-formatmessage>

|  |  |
| --- | --- |
| **Escape sequence** | **Meaning** |
| %0 | Terminates a message text line without a trailing new line character. This escape sequence can be used to build up long lines or to terminate the message itself without a trailing new line character. It is useful for prompt messages.  후행 줄 바꿈 문자없이 메시지 텍스트 행을 종료합니다. 이 이스케이프 시퀀스를 사용하면 긴 줄을 만들거나 후행 줄 바꿈 문자없이 메시지 자체를 종료 할 수 있습니다. 프롬프트 메시지에 유용합니다. |
| %*n*!*format string*! | Identifies an insert. The value of *n* can be in the range from 1 through 99. The format string (which must be surrounded by exclamation marks) is optional and defaults to !s! if not specified. For more information, see [Format Specification Fields](http://go.microsoft.com/fwlink/p/?linkid=83949).  The format string can include a width and precision specifier for strings and a width specifier for integers. Use an asterisk (*) to specify the width and precision. For example, %1!*.\*s! or %1!\*u!.  If you do not use the width and precision specifiers, the insert numbers correspond directly to the input arguments. For example, if the source string is "%1 %2 %1" and the input arguments are "Bill" and "Bob", the formatted output string is "Bill Bob Bill".  However, if you use a width and precision specifier, the insert numbers do not correspond directly to the input arguments. For example, the insert numbers for the previous example could change to "%1!\*.\*s! %4 %5!\*s!".  The insert numbers depend on whether you use an arguments array (**FORMAT\_MESSAGE\_ARGUMENT\_ARRAY**) or a **va\_list**. For an arguments array, the next insert number is *n+2* if the previous format string contained one asterisk and is *n+3* if two asterisks were specified. For a **va\_list**, the next insert number is *n+1* if the previous format string contained one asterisk and is *n+2* if two asterisks were specified.  If you want to repeat "Bill", as in the previous example, the arguments must include "Bill" twice. For example, if the source string is "%1!\*.\*s! %4 %5!\*s!", the arguments could be, 4, 2, Bill, Bob, 6, Bill (if using the **FORMAT\_MESSAGE\_ARGUMENT\_ARRAY** flag). The formatted string would then be "  Bi Bob   Bill".  Repeating insert numbers when the source string contains width and precision specifiers may not yield the intended results. If you replaced %5 with %1, the function would try to print a string at address 6 (likely resulting in an access violation).  Floating-point format specifiers—e, E, f, and g—are not supported. The workaround is to use the [StringCchPrintf](https://docs.microsoft.com/windows/desktop/api/strsafe/nf-strsafe-stringcchprintfa) function to format the floating-point number into a temporary buffer, then use that buffer as the insert string.  Inserts that use the I64 prefix are treated as two 32-bit arguments. They must be used before subsequent arguments are used. Note that it may be easier for you to use [StringCchPrintf](https://docs.microsoft.com/windows/desktop/api/strsafe/nf-strsafe-stringcchprintfa) instead of this prefix.  삽입을 식별합니다. n 값의 범위는 1-99입니다. 형식 문자열 (느낌표로 묶어야 함)은 선택 사항이며 기본적으로! s! 지정되지 않은 경우. 자세한 내용은 형식 사양 필드를 참조하십시오.  형식 문자열에는 문자열의 너비 및 정밀도 지정자와 정수의 너비 지정자가 포함될 수 있습니다. 별표 ()를 사용하여 너비와 정밀도를 지정하십시오. 예를 들어 % 1!. \* s! 또는 % 1! \* u !.  너비 및 정밀도 지정자를 사용하지 않으면 삽입 번호가 입력 인수와 직접 일치합니다. 예를 들어 소스 문자열이 "% 1 % 2 % 1"이고 입력 인수가 "Bill"및 "Bob"인 경우 형식화 된 출력 문자열은 "Bill Bob Bill"입니다.  그러나 너비 및 정밀도 지정자를 사용하는 경우 삽입 번호가 입력 인수와 직접 일치하지 않습니다. 예를 들어, 이전 예의 삽입 번호는 "% 1! \*. \* s! % 4 % 5! \* s!"로 변경 될 수 있습니다.  삽입 번호는 인수 배열 (FORMAT\_MESSAGE\_ARGUMENT\_ARRAY) 또는 va\_list를 사용하는지에 따라 다릅니다. 인수 배열의 경우, 이전 형식 문자열에 하나의 별표가 포함 된 경우 다음 삽입 번호는 n + 2이고 두 개의 별표가 지정된 경우 n + 3입니다. va\_list의 경우, 이전 형식 문자열에 하나의 별표가 포함 된 경우 다음 삽입 번호는 n + 1이고 두 개의 별표가 지정된 경우 n + 2입니다.  이전 예에서와 같이 "Bill"을 반복하려면 인수에 "Bill"이 두 번 포함되어야합니다. 예를 들어 소스 문자열이 "% 1! \*. \* s! % 4 % 5! \* s!"인 경우 인수는 4, 2, Bill, Bob, 6, Bill 일 수 있습니다 (FORMAT\_MESSAGE\_ARGUMENT\_ARRAY 플래그를 사용하는 경우). . 서식이 지정된 문자열은 "Bi Bob Bill"입니다.  소스 문자열에 폭이 있고 정밀도 지정자가 포함 된 경우 삽입 번호를 반복하면 원하는 결과를 얻지 못할 수 있습니다. % 5를 % 1로 바꾸면 함수는 주소 6에서 문자열을 인쇄하려고 시도합니다 (접근 위반이 발생할 수 있음).  부동 소수점 형식 지정자 (e, ​​E, f 및 g)는 지원되지 않습니다. 해결 방법은 StringCchPrintf 함수를 사용하여 부동 소수점 숫자를 임시 버퍼로 형식화 한 다음 해당 버퍼를 삽입 문자열로 사용하는 것입니다.  I64 접두사를 사용하는 삽입은 두 개의 32 비트 인수로 처리됩니다. 후속 인수를 사용하기 전에 사용해야합니다. 이 접두사 대신 StringCchPrintf를 사용하는 것이 더 쉬울 수 있습니다. |

메시지 텍스트 내에서 메시지를 동적으로 형식화하기 위해 여러 이스케이프 시퀀스가 ​​지원됩니다. 이스케이프 시퀀스와 그 의미는 다음 표에 나와 있습니다. 모든 이스케이프 시퀀스는 퍼센트 문자 (%)로 시작합니다.

퍼센트 문자 뒤에 오는 다른 비 숫자 문자는 퍼센트 문자 없이 출력 메시지에서 형식화됩니다. 다음은 몇 가지 예입니다.

|  |  |
| --- | --- |
| **Format string** | **Resulting output** |
| %% | A single percent sign. - 단일 퍼센트 부호. |
| %*space* | A single space. This format string can be used to ensure the appropriate number of trailing spaces in a message text line.  단일 공간. 이 형식 문자열을 사용하여 메시지 텍스트 행에 적절한 수의 후행 공백을 확보 할 수 있습니다. |
| %. | A single period. This format string can be used to include a single period at the beginning of a line without terminating the message text definition.  단일 기간. 이 형식 문자열을 사용하면 메시지 텍스트 정의를 종료하지 않고 줄의 시작 부분에 단일 마침표를 포함시킬 수 있습니다. |
| %! | A single exclamation point. This format string can be used to include an exclamation point immediately after an insert without its being mistaken for the beginning of a format string.  느낌표 하나. 이 형식 문자열은 형식 문자열의 시작으로 오인되지 않고 삽입 직후 느낌표를 포함하는 데 사용할 수 있습니다. |
| %n | A hard line break when the format string occurs at the end of a line. This format string is useful when **FormatMessage** is supplying regular line breaks so the message fits in a certain width.  줄 끝에서 형식 문자열이 발생할 때 줄 바꿈. 이 형식 문자열은 FormatMessage가 규칙적인 줄 바꿈을 제공하여 메시지가 특정 너비에 맞을 때 유용합니다. |
| %r | A hard carriage return without a trailing newline character.  후행 줄 바꿈 문자가 없는 하드 캐리지 리턴 |
| %t | A single tab. |

# 문자와 문자열

UTF-8 – 하나의 문자를 나타내기 위해 1, 2, 3, 4 바이트로 인코딩을 수행한다.  
0x0000 ~ 0x0080 미만 – 1byte  
0x0080 ~ 0x07FF 미만 - 2byte  
0x0800 이상 - 3byte  
010000 ~ 10FFFF 4byte

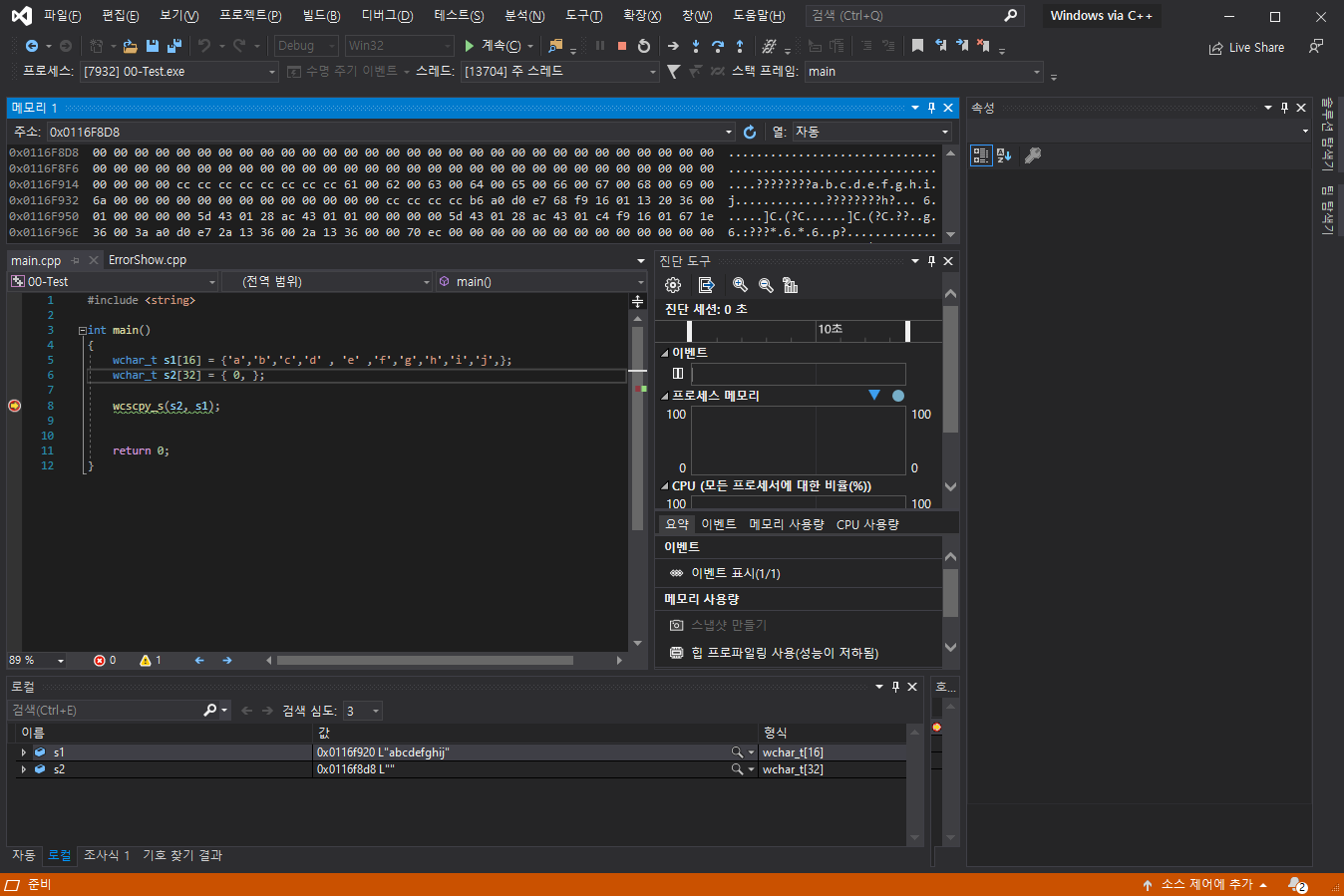
UTF-32 – 모든 문자를 4바이트로 인코딩, 전송 방식으로는 거의 사용 안 하며 프로그램 내부에서만 사용한다.

윈도우는 UTF-16을 채용

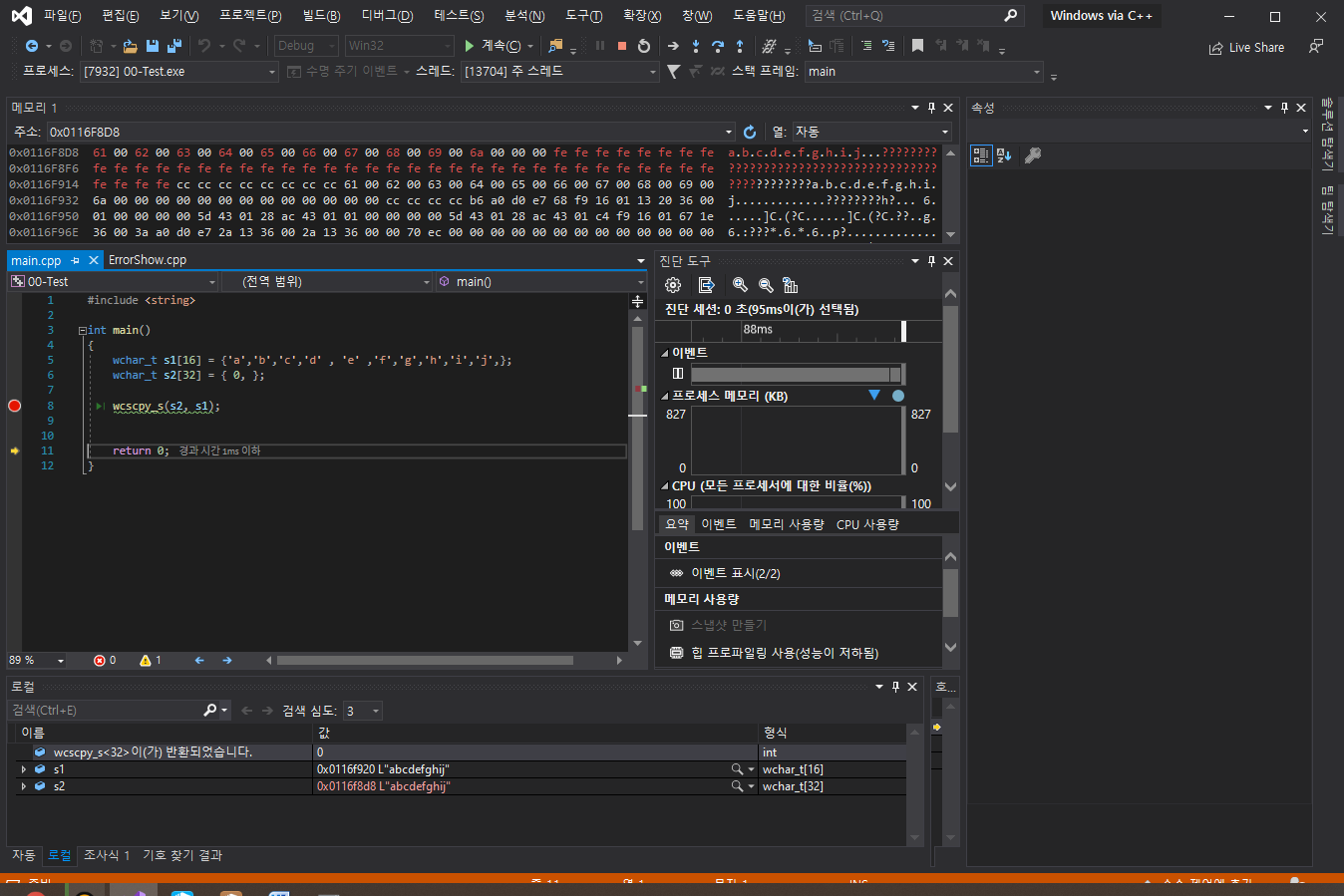
윈도우에서는 ANSI(char) 문자열을 사용한 함수는 되도록 사용하지 말 것. 내부적으로 WideCharacter 변환이 일어날 가능성이 높다.

C 런타임 라이브러리 내의 유니코드 함수와 ANSI 함수는 각각 상호 간 변경을 하는 함수가 실행되진 않으며, 이를 보완하기 위해선 tchar.h 헤더를 참조해야 한다

String 초기화 시 초기화 되지 않은 영역은 cc 로 전환되며, 복사나 여타 오버런 할 수 있는 함수 실행시 메모리가 fe로 초기화 된다. 교제에서는 fd로 소개하였으나, 2019 버전에서는 fe로 되더라… 이 같은 메모리 변환은 런타임 확인 옵션 /RTCc, /RTCu, 또는 /RTC1 덕분이다. 이 플래그를 지정하지 않으면, 메모리가 서로 다른 값을 가지게 될 것이다. 디버그를 위해선 해당 옵션을 사용하는 것이 좋다.



S2 의 주소에서 00으로 초기화 된 모습이 보인다.



해당 연산을 마친 후에는 여분의 공간이 0xfe로 초기화 된 모습이 보인다.

핵심은 0xfd냐 0xfe냐라는 것이 아니고 일정 수로 나머지 공간이 채워진다는 것이 이 글의 핵심이다.

CompareString 은 지역을 지정해야하며, CompareStringOrdinal은 필요없다. 그냥 쌩으로 비교함. 되도록 뒤에껄 쓰는 걸 추천한다.

WideCharToMultiByte, MultiByteToWideChar – 문자 변환 사용법 링크 추가바람…

IsTextUnicode 를 활용해 Unicode인지 아닌지를 확인하라. 문제는 이 함수가 그리 썩 믿음직한 함수는 아님 .- 잘못된 결과를 반환할 수 있음(과학적인 방법이 아니라 책에서 설명)

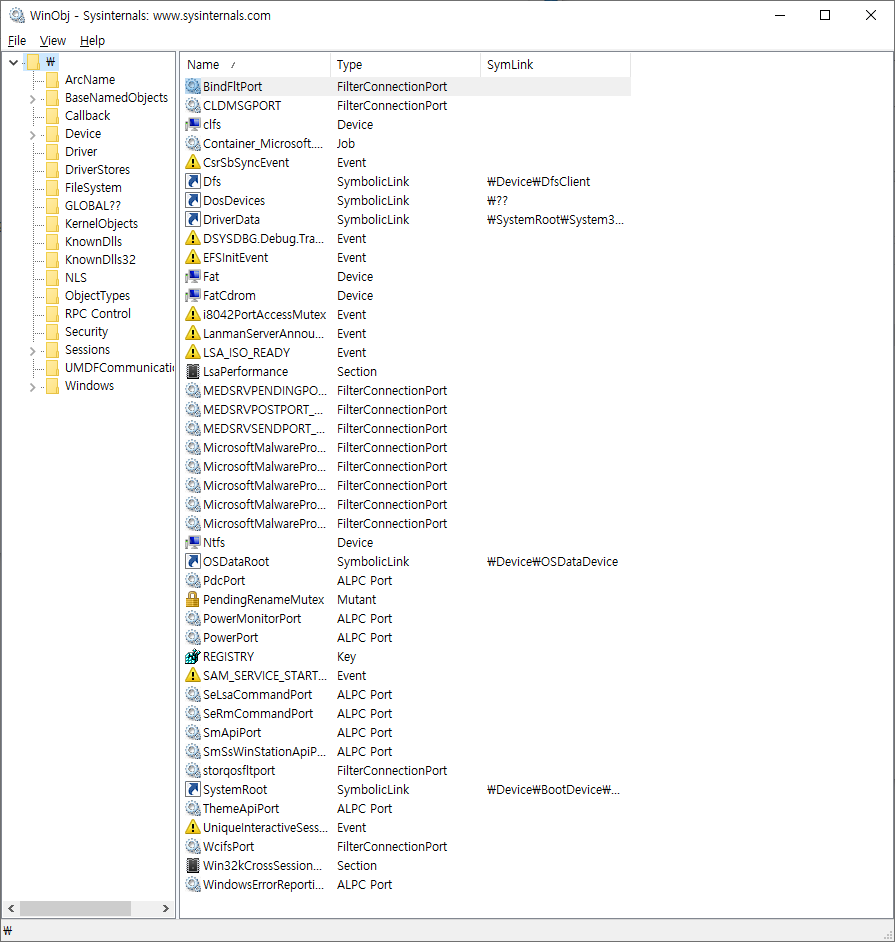
# 커널 오브젝트

<https://docs.microsoft.com/en-us/windows/win32/sysinfo/kernel-objects>

숙달된 윈도우 소프트웨어 개발자가 되기 위해서는 반드시 커널 오브젝트에 대해 완벽하게 알고 있어야 한다고 함.

액세스 토큰 오브젝트, 이벤트 오브젝트, 파일 오브젝트, 파일-매핑 오브젝트, I/O 콤플리션 포트 오브젝트, 잡, 메일슬롯, 뮤텍스, 파이프, 프로세스, 세마포어, 스레드, 대기 타이머, 스레드 풀 워커 팩토리 등 다양한 형태의 커널 오브젝트를 생성하고 조작한다.

<https://docs.microsoft.com/ko-kr/sysinternals/downloads/winobj> 사이트에서 모든 커널 오브젝트 타입을 나열하고 확인할 수 있다.

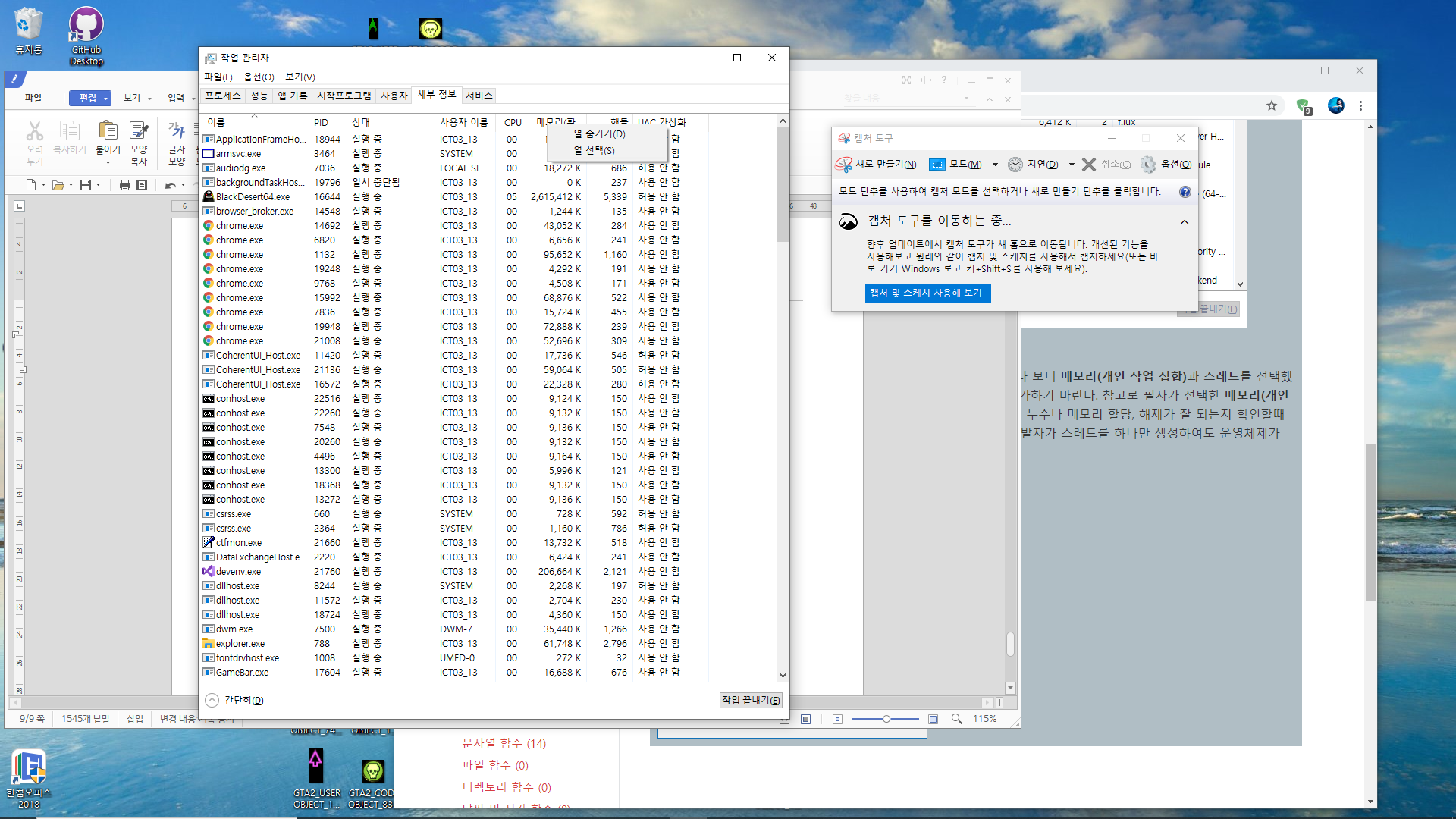


이러한 오브젝트들은 다양한 종류의 함수로 제작된다. 각 커널 오브젝트는 커널에 의해 할당된 간단한 메모리 블록이다. 그리고 이 메모리 블록은 커널에 의해서만 접근이 가능한 구조체로 구성, 세부정보들을 저장하고 있다.

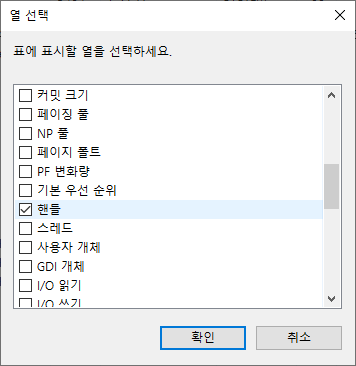
따라서 직접 메모리에 접근해 변경하는 것은 불가능하다. 커널오브젝트 생성시 그 오브젝트를 구분하기 위한 반환 값을 핸들로 준다. 32환경에선 32비트 64환경에선 64비트이다.

커널 오브젝트는 프로세스가 아니라 커널에 의해 소유된다. 즉, 프로세스가 종료되더라도 커널오브젝트가 함께 종료되는 것이 아니다. 삭제되지 않고 남아있을 수 있다는 것이다. 오브젝트는 내부 카운트(usage count)를 가지며, 프로세스가 그 값을 증감시킨다. 사용하는 곳이 없어 그 값이 0이 되면 비로소 삭제가 된다.

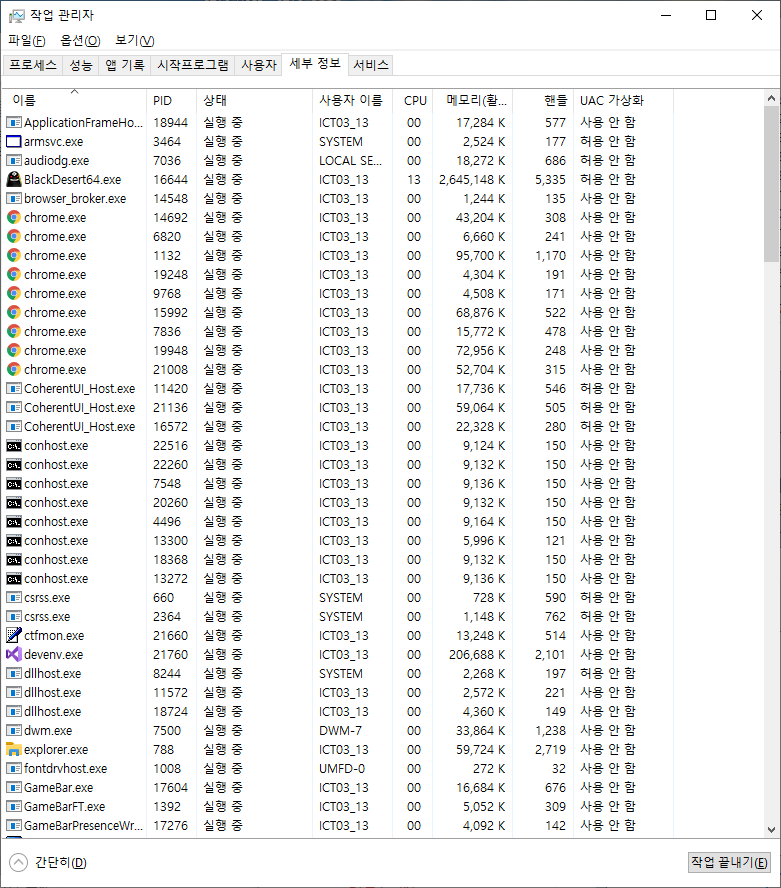
작업관리자에서 핸들(Handle)보기



세부 정부 - 오른쪽 클릭 - 열 선택

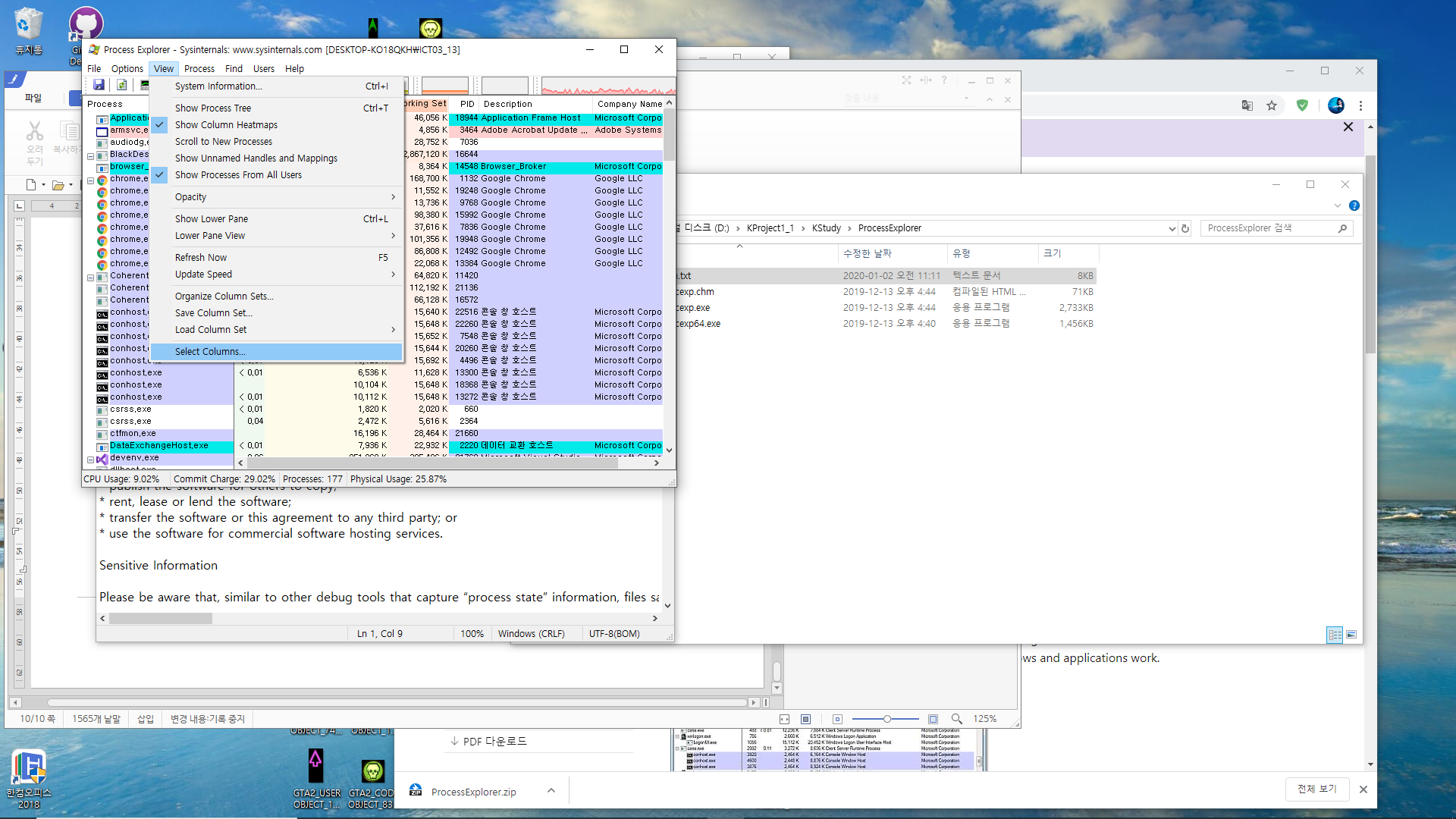


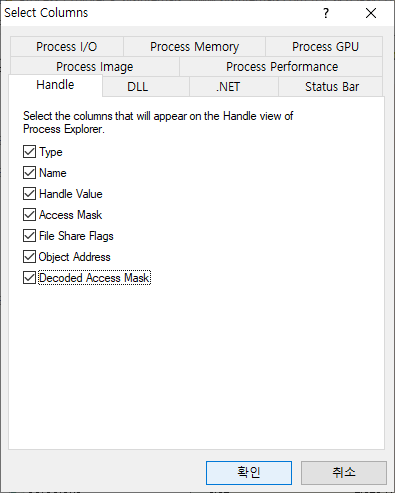
아래에 핸들 찾기

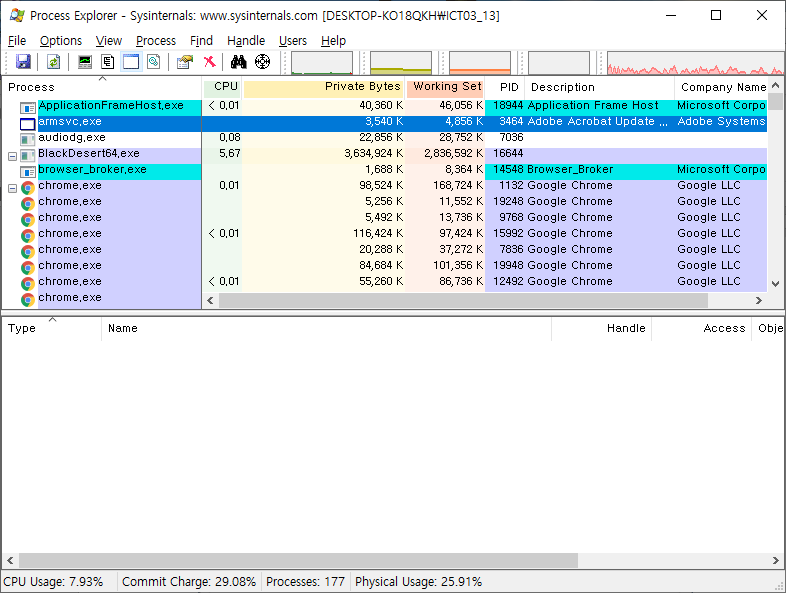
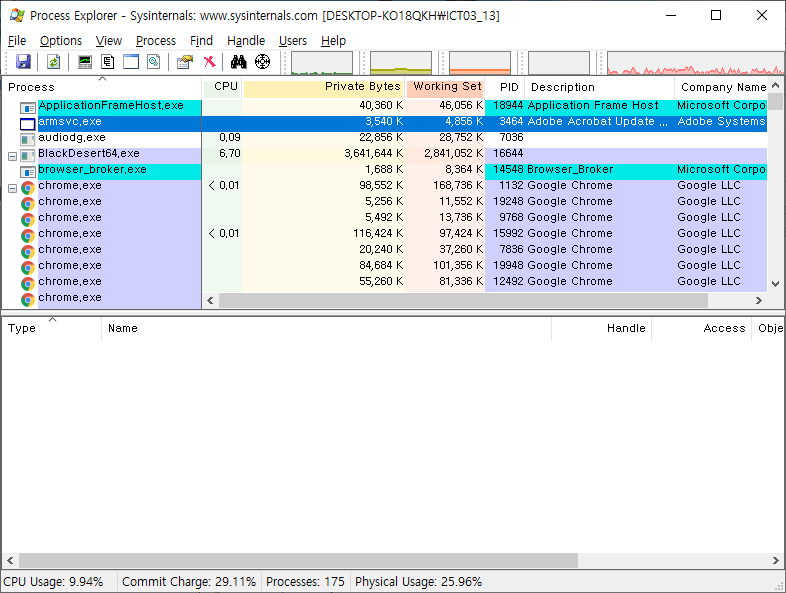
  
핸들 보기

핸들 보는 프로그램 - Process Explorer

<https://docs.microsoft.com/ko-kr/sysinternals/>





Shown Lower Pane 아이콘 클릭 시 모니터링 가능

프로세스간 핸들 공유

핸들은 고유한 값으로 배정을 받는데 안정성과 보안성 때문이다. 전자는 전역적이 값이라면 오동작 가능성을 가지고 있으며, 후자는 커널 오브젝트 자체가 권한 획득을 요구하므로 접근 권한을 제어할 수 있다는 점이다.

이러한 커널 오브젝트를 안전하게 공유할 수 있는 방법이 있다.

1. 오브젝트 핸들의 상속을 이용

공유하고자 하는 프로세스들이 상속관계일 때 가능. 정확히 말하면 오브젝트 핸들의 상속관계일 때를 말한다.

https://docs.microsoft.com/en-us/previous-versions/windows/desktop/legacy/aa379560(v%3Dvs.85)

SECURITY\_ATTRIBUTES 초기화 사용 - 변수의 bInheritHandle 을 TRUE값으로 지정 -> 내부적으로 NULL경우는 상속 여부를 0으로, TRUE일 경우는 플래그 비트를 1로 지정한다. 이는 각각 인덱스를 1과 3으로 지정해 상속가능 여부를 파악한다.

이러한 핸들은 프로세스 생성 시 OS에서 추가적인 작업을 하는데 바로 부모 핸들을 조사해 상속 가능한 테이블을 복사한다. 이는 부모 핸들의 위치와 정확히 일치한다. 이로서 부모와 자식 프로세스가 동일한 값을 이용할 수 있는 것. 이를 통해 커널 오브젝트 내 사용 카운트를 증가 시키며, 종료 하려면 양 쪽 모두에서 CloseHandle을 호출하거나 각각의 프로세스를 종료시키면 된다. 여기서, 부모나 자식이 종료되어야 하는 순서는 정해지지 않는다.

커널 오브젝트 내용은 커널 주소 공간 상에 저장된다. - 13장 내용

자식 프로세스는 어떤 핸들을 상속된 것인지 알 수 없는 특징을 가지고 있다. \_stscanf\_s를 사용하여 핸들 값을 얻어낼 수 있다. (명령행 인자를 분석하는 방식)

GetEnvironmentVariable 함수를 호출하면 간단하게 상속된 오브젝트 핸들 값을 얻을 수 있다.