ISO/IEC JTC 1/SC 22/WG14 N1264

Potential Extensions For Inclusion In a Revision of ISO/IEC 9899

Introduction

In London, WG14 tasked me with coming up with a proposed list of Microsoft extensions that the committee could consider for standardization. This document contains that list. In general, I've provided a link to the product documentation rather than cutting and pasting relevant sections.

1. `__declspec` (http://msdn2.microsoft.com/en-us/library/dabb5z75(VS.80).aspx)

   `__declspec` is a general mechanism for applying an attribute to a declarator. In general, it behaves as a storage class specificer. There are several specific attributes that might make sense for standardization; they're described in subsequent sections.

   Examples:

   ```
   __declspec(attribute) int i;

   __declspec(attribute) void f();

   __declspec(attribute) struct X {  int m; };
   ```

2. `__declspec(noreturn)` (http://msdn2.microsoft.com/en-us/library/k6ktzx3s(VS.80).aspx)

   `noreturn` indicates that a function does not return, avoiding an error or warning message.

   Example:

   ```
   __declspec(noreturn) void fatal();

   int f(int arg)
   {
         if(arg > 5)
               return 1;
         else
               fatal();
   }
   ```

3. `__declspec(align(`*`alignment`*`))` (http://msdn2.microsoft.com/en-us/library/83ythb65(VS.80).aspx)

`align(alignment)` **specifies that data is to be aligned on a particular memory boundary.** `alignment` **is an integer power of 2 from 1 to 8192. There are fairly complex interactions with the compiler's default alignment and structure packing; see the link for more details.**

`Example:`

```
__declspec(align(8)) struct S2 { int a, b, c, d; };
```

4. `__declspec(thread)` (http://msdn2.microsoft.com/en-us/library/9w1sdazb(VS.71).aspx)

`thread` **specifies that data is thread-local. Note that in the Microsoft implementation, this can't be used with delay-loaded DLLs.**

`Example:`

```
__declspec(thread) int x;
```

5. **__declspec(deprecated) (http://msdn2.microsoft.com/en-us/library/044swk7y(VS.80).aspx)**

`deprecated` **provides a way for library writers to mark functions as deprecated, at translation time. In the Microsoft compiler, a warning is issued for the deprecation.**

Example:

```
__declspec(deprecated) void f();
```

6. **Structured exception handling (**`__try/__except, __try/__finally`**)** (http://msdn2.microsoft.com/en-us/library/swezty51(VS.71).aspx)

Nick Stoughton has covered this in some detail in N1229. try-except allows the application to gain control when an interrupt occurs. Its use is coupled with a few compiler intrinsics (`GetExceptionCode`, `GetExceptionInformation`) that allow decision making based on the type of interrupt.

Example:

```
__try
{
      *p = i;
}
/* handle the exception if it's an access violation; otherwise,
continue search for other exception handlers active on the stack */
__except(GetExceptionCode() == EXCEPTION_ACCESS_VIOLATION ?
EXCEPTION_EXECUTE_HANDLER : EXCEPTION_CONTINUE_SEARCH)
{
      /* handle the exception */
}
```

try-finally allows an application to guarantee execution of cleanup code when an interrupt occurs.

7. SAL (Structured Annotation Language) ((a) http://msdn2.microsoft.com/en-us/library/ms235402(VS.80).aspx, (b) http://blogs.msdn.com/michael_howard/attachment/602077.ashx)

SAL is a language to annotate functions in a way that describes how their parameters are related to each other. This allows static analysis tools to verify callsites of the functions to prevent common programming mistakes. SAL is usually used in a way that is focused on the prevention of security vulnerabilities (e.g. ensuring buffer lengths are passed with buffers).

A simple introduction is provided in link (b) above.

Example:

```
void FillString( __out_ecount(cchBuf) TCHAR* buf, size_t cchBuf,
char ch);
```

The annotation (in bold) informs the compiler as to the relationship between the parameters `buf` and `cchBuf`. The compiler can therefore issue warnings on this code:

```
TCHAR *b = (TCHAR*)malloc(200*sizeof(TCHAR));
FillString(b,210,'x');
```

Other relationships that SAL can describe include:

- **In parameters vs. out parameters**
- **Optional parameters**
- **Buffer sizes and counts**
- **Pointers to buffers of a certain size**
- **Required return value checks**

8. **`__unaligned` (http://msdn2.microsoft.com/en-us/library/ms253978(VS.80).aspx)**

   Used to declare data as unaligned. This can be useful on architectures where code generation must be done specially for unaligned data.