C

CASTING THE NULL POINTER

Consider the following call to the variadic function *execl()*:

```
execl("ls", "ls", "-l", (char *) NULL);
```

A *variadic function* is one that takes a variable number of arguments or arguments of varying types.

Whether the cast is required before the NULL in cases like this is the source of some confusion. While we can often get away without the cast, the C standards require it; failure to include it may lead an application to break on some systems.

NULL is typically defined as either 0 or as *(void *) 0*. (The C standards allow other definitions, but they are essentially equivalent to one of these two possibilities.) The main reason casts are needed is that NULL is allowed to be defined as 0, so this is the case we examine first.

The C preprocessor translates NULL to 0 before the source code is passed to the compiler. The C standards specify that the integer constant 0 may be used in any context where a pointer may be used, and the compiler will ensure that this value is treated as a null pointer. In most cases, everything is fine, and we don't need to worry about casts. We can, for example, write code such as the following:

```
int *p;

p = 0;                                  /* Assign null pointer to 'p' */
p = NULL;                               /* Same as 'p = 0' */
```

The above assignments work because the compiler can determine that a pointer value is required on the right-hand side of the assignment, and it will convert the value 0 to a null pointer.

Similarly, for functions with prototypes specifying a fixed argument list, we can specify either 0 or NULL for a pointer argument, to indicate that a null pointer should be passed to the function:

```
sigaction(SIGINT, &sa, 0);
sigaction(SIGINT, &sa, NULL);          /* Equivalent to the preceding */
```

If we are passing a null pointer to an old-style, nonprototyped C function, then all of the arguments given here about the need to appropriately cast 0 or NULL also apply, regardless of whether the argument is part of a variadic argument list.

Because casting is not required in any of the above examples, one might conclude that it is never required. But this is wrong. The need for casting arises when specifying a null pointer as one of the varying arguments in a call to a variadic function such as *execl()*. To realize why this is necessary, we need to be aware of the following:

- The compiler can't determine the expected types of the varying arguments of a variadic function.
- The C standards don't require that a null pointer is actually represented in the same way as the integer constant 0. (In theory, a null pointer could be represented by any bit pattern that wasn't the same as a valid pointer.) Nor do the standards even require that a null pointer is the same size as the integer constant 0. All that the standards require is that when the integer constant 0 is found in a context where a pointer is expected, the 0 should be interpreted as a null pointer.

Consequently, it is wrong to write either of the following:

```
execl(prog, arg, 0);
execl(prog, arg, NULL);
```

This is an error because the compiler will pass the integer constant 0 to *execl()*, and there is no guarantee that this is equivalent to a null pointer.

In practice, we can often get away without the cast, since, on many C implementations (e.g., Linux/x86-32), the representations of the integer (*int*) constant 0 and the null pointer are the same. However, there are implementations where they are not—for example, where the size of a null pointer is larger than the size of the integer constant 0—so that in the above examples, *execl()* is likely to receive some random bits adjacent to the integer 0, and the resulting value will be interpreted as a random (nonnull) pointer. Omitting the cast leads to programs breaking when ported to such implementations. (On some of the aforementioned implementations, NULL is defined as the *long* integer constant *0L*, and *long* and *void ** have the same size, which may save wrongly constructed programs that use the second of the *execl()* calls above.) Therefore, we should rewrite the above *execl()* calls in the following ways:

```
execl(prog, arg, (char *) 0);
execl(prog, arg, (char *) NULL);
```

Casting NULL in the manner of the last call above is generally required, even on implementations where NULL is defined as *(void *) 0*. This is because, although the C standards require that null pointers of different types should test true for comparisons on equality, they don't require that pointers of different types have the same internal representation (although on most implementations they do). And, as before, in a variadic function, the compiler can't cast *(void *) 0* to a null pointer of the appropriate type.

> The C standards make one exception to the rule that pointers of different types need not have the same representation: pointers of the types *char ** and *void ** are required to have the same internal representation. This means that passing *(void *) 0* instead of *(char *) 0* would not be a problem in the example case of *execl()*, but, in the general case, a cast is needed.