Be Careful Using Unsigned

Ashwin Menon

December 15, 2024

There are some important pitfalls associated with using unsigned numeric types, and in general it's better to not use them. After all, why would you even need an unsigned type? Perhaps you think that in a certain context (e.g. a string's length) only unsigned numbers make sense, since there's no such thing as a negative length. This is fine, but the problem comes when you mix signed and unsigned numbers. Consider this buffer copying function which takes an unsigned length parameter, since lengths can't be negative:

```
1 void copy_buffer(const char *src, char *dest, unsigned int length);
2
3 int len = get_buffer_length();
4 copy_buffer(src, dest, len);
```

The problem here is that *len* is a signed integer. If it ever becomes negative (perhaps it gets it's value over a network, which may be getting malicious data), then it gets cast into an unsigned number of the same bit width, which results in a very large unsigned value. Then *copy_buffer* will likely access memory it's not supposed to.

Of course, I didn't have to invent the *copy_buffer* function above to illustrate the point. Many standard library functions (*read*, *strcpy*, *memcpy*, *memset*, etc.) take in the length as a *size_t* type, which is unsigned, and is therefore subject to the same pitfall.

Maybe you want to use an unsigned type to increase the range of the positive values you can use? In that case, simply use a wider data type of the signed variety, and avoid the need to think of pathological cases like the example above.

So is an unsigned type of no use at all? Well, not really. You could use it in cases where what you're representing is not a number, but some kind of bit pattern; perhaps the contents of a register inside a hardware device. In that case, it would be odd to store that value in a signed numeric type, and an unsigned type would feel more natural.