Inside STL: The shared_ptr constructor vs make_shared

devblogs.microsoft.com/oldnewthing/20230815-00

August 15, 2023

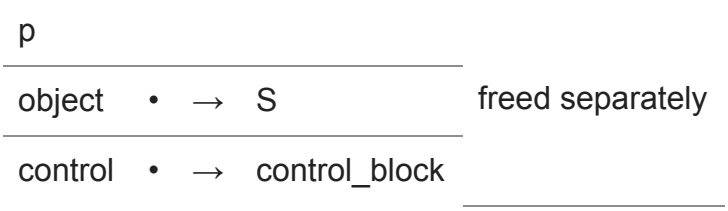
Raymond Chen

There are two ways to create a new object that is controlled by a `shared_ptr`.

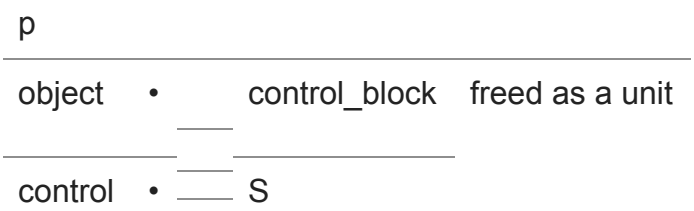
```
// From a raw pointer
auto p = std::shared_ptr<S>(new S());

// Via make_shared
auto p = std::make_shared<S>();
```

They result in two different memory layouts.

In the first case, you manually created a new `S` object, and then passed a pointer to it to the `shared_ptr` constructor. The `shared_ptr` adopts the raw pointer and creates a control block to monitor its lifetime. When the last shared pointer destructs, the `Dispose()` method deletes the pointer you passed in.[1] When the last shared or weak pointer destructs, the `Delete()` method deletes the control block.



In the second case, you let the `make_shared` function create the `S` object, and in practice, what it does is create a single memory allocation that consists of a control block stacked on top of an `S` object. This time, when the last shared pointer destructs, the `Dispose()` method runs the `S` destructor, but the memory isn't freed yet. Only when the last shared or weak pointer destructs does the `Delete()` method get called to free the entire memory block.



The two memory layouts have their own pros and cons.

	Two allocations	**Single allocation**
Last shared pointer destructs	Object destructs Object memory freed	Object destructs Object memory not freed
Last shared or weak pointer destructs	Control block destructs Control block freed	Control block destructs Combo block freed
Locality	Worse	Better
Straggler weak pointer	Control block lingers (Object memory freed already)	Entire combo block lingers

The single-allocation version has better memory locality since the control block is kept right next to the managed object.

On the other hand, with the single-allocation version, a straggler weak pointer (a weak pointer which lives for a long time after the last shared pointer has destructed) prevents the entire combo memory block from being freed. By comparison, only the control block remains in memory with the two-allocation version.

If your weak pointers exist to break circular references, then you won't have stragglers because they will go away when the object graph destructs. Similarly, if your weak pointers are in event handlers, then those weak pointers won't be stragglers if you are careful to unregister the event handlers at destruction. The stragglers come into play if you retain weak references in, say, a cache or other long-lifetime storage, and even then, they cause a problem only if `sizeof(S)` is significant or if you have a lot of them.

Next time, we'll look at `make_shared`'s close friend, `std::enable_shared_from_this`.

[1] More specifically, the `Deleter` object deletes the pointer you passed in. The default deleter uses the `delete` operator to delete the pointer.