

Re: Virtual function and multiple inheritance

Source: <http://www.tech-archive.net/Archive/VC/microsoft.public.vc.language/2008-02/msg00045.html>

- *From:* "Bo Persson" <bop@xxxxxx>
 - *Date:* Sat, 2 Feb 2008 12:32:09 +0100
-

George wrote:

Thanks Alf,

Your reply is great! I think you mean,

1.

containing two `__vfptr` is for the simple reason to maintain the same memory structure as sub-object;

It could be used in place of either a Foo or a Goo object, for example when passed by reference to a function. To do this, it probably needs two different vtables, at least in this implementation.

Note that the language as such doesn't say anything about how virtual functions are to be implemented, just how they should behave.

2.

`__vfptr` for Foo contains Derived overridden virtual methods for Foo, not Foo itself's virtual methods implementation, and `__vfptr` for Goo contains Derived overridden virtual methods for Goo, not Goo itself's virtual methods implementation. Right?

I'll make a guess that these two vtables are specific for Derived. If you were to define two separate objects of the Foo and Goo classes, I bet there will be one or two separate vtables for those. As Alf suggests, the implementation just might share (or overlay) the vtables for Foo and Derived, especially if Derived is the only class deriving from Foo.

As the Foo and Goo subobjects cannot possibly have the same offset from the start of Derived, one vtable might be shared, but not both.

Re: Virtual function and multiple inheritance

That's likely why you found two vtable pointers in Derived.

3.

Where is class Derived's own `__vfptr` (points to its own virtual methods)? From debugging, I can not find it out by adding a new method in Derived which is not in Foo and Goo.

I believe you have actually found Derived's `__vfptr`. As there aren't any separate Foo or Goo objects, their own vtable pointers need not be present in the resulting executable. Or perhaps one of them is?

Bo Persson

regards,
George

"Alf P. Steinbach" wrote:

* George "the mysterious BOT":

(this question is posted to `vc.language` newsgroup)

Hello everyone,

In the following multiple inheritance sample code, I have tested in class Derived, there are two `__vfptr`, pointing to the virtual function table for Foo and Goo respectively -- i.e. 8 bytes, 2 pointer on 32-bit machine.

My questions,

1. Why two `__vfptr` is needed? Why not just one?
2. class Derived has its own virtual method `func2`, why it does not have its own virtual function table pointer?

[Code]
class Foo {
public:

Re: Virtual function and multiple inheritance

```
virtual int func() {return 1;};
};

class Goo {
public:
virtual int func() {return 2;};
};

class Derived: Foo, Goo {
public:
virtual int func2() {return 3;};
int increase() {return -1;};
int decrease() {return -2;};
};

int main()
{
Derived d;
int size;

size = sizeof (d); // size is 8, two __vfptr?

return 0;
}
[/Code]
```

Consider

```
Derived* pD = &d;
Foo* pF = pD;
Goo* pG = pD;
```

Here pF is a pointer to an object with the same memory layout as a Foo (it is, typewise, a Foo) and pG is a pointer to an object with the same memory layout as a Goo (ditto).

That implies that a Derived /contains/ a Foo and a Goo object; in the standard they're called "sub-objects".

However, in a vtable-based implementation the Foo sub-object's vtable pointer does not necessarily point to the Foo vtable, but in the general case to a copy with the function pointers corresponding to overrides in Derived, replaced with pointers to Derived's overrides. And ditto for the Goo sub-object.

Derived does not technically need a separate vtable and vtable pointer because it can just extend (the copy of) the Foo or Goo vtable.

Re: Virtual function and multiple inheritance

How that is done is an implementation detail.

A consequence of the above is that multiple inheritance can have an $O(n^2)$ memory cost where n is the total number of class derivations overriding base virtual functions. I haven't thought more deeply about it but I don't think it can be worse. Some programmers erroneously think that this strongly contra-indicates using multiple inheritance; however, relative to usual memory consumption for any program (even "Hello, world!") the memory cost is in practice vanishingly small.

Cheers, & hth.,

– Alf