C++ Vtable Example

Revised 10 September 1999

[990910 IBM -- Brian] Added more examples, split out the two kinds of adjustments in Table 1a, and added a summary of the component counts for the two approaches.

Table 1a: Example Code and Call Semantics

Declarations	Call	Callee	Call-site Adjustment	Thunk/Entry-p Adjustmen

struct A {
  virtual void f ();
  virtual void g ();
  virtual void h ();
  int ia;
};
 	
A *pa;
	pa->f()	A::f()	none	none

	pa->g()	A::g()	none	none
	pa->h()	A::h()	none	none

struct B: public virtual A {
  void f ();
  void h ();
  int ib;
};
 	
B *pb;
A *pa_in_b = pb;
	pb->f()	B::f()	none	none

	pb->A::f()	A::f()	B => A	none
	pb->g()	A::g()	B => A	none
	pb->h()	B::h()	none	none
	pa_in_b->f()	B::f()	none	A => B
	pa_in_b->g()	A::g()	none	none
	pa_in_b->h()	B::h()	none	A => B
	pa_in_b->A::f()	A::f()	none	none

struct C: public virtual A {
  void g ();
  void h ();
  int ic;
};
 	
C *pc;
A *pa_in_c = pc;
	pc->f()	A::f()	C => A	none

	pc->g()	C::g()	none	none
	pc->A::g()	A::g()	C => A	none
	pc->h()	C::h()	none	none
	pa_in_c->f()	A::f()	none	none
	pa_in_c->g()	C::g()	none	A => C
	pa_in_c->h()	C::h()	none	A => C
	pa_in_c->A::g()	A::g()	none	none
	pd->f()	B::f()	none [D => B]	none

```
struct D: public B, public C {
  int id;
  void h();
};

D *pd;


A *pa_in_d = pd;
B *pb_in_d = pd;
C *pc_in_d = pd;


A *pa_in_b_in_d = pb_in_d;
A *pa_in_c_in_d = pc_in_d;
```

`pd->g()`	C::g()	D => C	none
`pd->h()`	D::h()	none	none
`pa_in_d->f()`	B::f()	none	A => B
`pa_in_d->g()`	C::g()	none	A => C
`pa_in_d->h()`	D::h()	none	A => D
`pb_in_d->f()`	B::f()	none	none
`pb_in_d->g()`	C::g()	B => A	A => C
`pb_in_d->h()`	D::h()	none	B => D
`pc_in_d->f()`	B::f()	C => A	A => B
`pc_in_d->g()`	C::g()	none	none
`pc_in_d->h()`	D::h()	none	C => D
`pa_in_b_in_d->f()`	same as for pa_in_d		
`pa_in_b_in_d->g()`			
`pa_in_b_in_d->h()`			
`pa_in_c_in_d->f()`			
`pa_in_c_in_d->g()`			
`pa_in_c_in_d->h()`			
`p...d->A::f()`	A::f()	... => A	none
`p...d->A::g()`	A::g()	... => A	none
`p...d->A::h()`	A::g()	... => A	none

```
struct X {
  int ix;
  virtual void x();
};

struct E : X, D {
  int ie;
  void f();
  void h();
};
```

`pe->f()`	E::f()	none	none
`pe->g()`	C::g()	E => C	none
pe->h()	E::h()	none	none
pe->x()	X::x()	none [E=>X]	none
`pa_in_e->f()`	E::f()	none	A => E
pa_in_e->g()	C::g()	none	A => C
pa_in_e->h()	E::h()	none	A => E
pb_in_e->f()	E::f()	none	B => E
pb_in_e->g()	C::g()	B => A	A => C
pb_in_e->h()	E::h()	none	B => E
pc_in_e->f()	E::f()	C => A	A => E
pc_in_e->g()	C::g()	none	none

	pc_in_e->h()	E::h()	none	C => E
	pd_in_e->f()	E::f()	none [D=>B]	B => E
	pd_in_e->g()	C::g()	D => C	none
	pd_in_e->h()	E::h()	none	D => E

Table 1b: Example Data Layout

Declarations	Size	Offset	Member

`struct A {`
`  virtual void f ();`
`  virtual void g ();`
`  virtual void h ();`
`  int ia;`
`};`
	16	0	*A::vptr*

		8	ia

`struct B: public virtual A {`
`  void f ();`
`  void h ();`
`  int ib;`
`};`
	32	0	*B::vptr*

		8	ib
		16	*A::vptr*
		24	ia

`struct C: public virtual A {`
`  void g ();`
`  void h ();`
`  int ic;`
`};`
	32	0	*C::vptr*

		8	ic
		16	*A::vptr*
		24	ia

`struct D: public B, public C {`
`  void h ();`
`  int id;`
`};`
	48	0	*D/B::vptr*

		8	ib
		16	*C::vptr*
		24	ic
		28	id
		32	*A::vptr*
		40	ia

`struct X {`
`  int ix;`
`  virtual void x();`
`};`
 	
`struct E : X, D {`
`  void f ();`
`  void h ();`
	64	0	*X/E::vptr*

		8	ix
		16	*D/B::vptr*
		24	ib
		32	*C::vptr*
		40	ic

`int ie;` `};`		48	id
		56	*A::vptr*
		64	ia

Table 1c: Example Vtable Layout

Declarations	Vtable (HP) [1,2,3]	Vtable (Cyg

`struct A {`
`  virtual void f ();`
`  virtual void g ();`
`  virtual void h ();`
`  int ia;`
`};`
	`A::offset_to_top (0)`
	`A::rtti`
	`-- A vtable address --`
	`A::f() []`
	`A::g() []`
	`A::h() []`
		`A::offset_`
		`A::rtti`
		`-- A vtabl`
		`A::f() []`
		`A::g() []`
		`A::h() []`

`struct B: public virtual A {`
`  void f ();`
`  void h ();`
`  int ib;`
`};`
	`B::offset_to_A (16)`
	`B::offset_to_top (0)`
	`B::rtti`
	`-- B vtable address --`
	`B::f() []`
	`B::h() []`
	
	`A::offset_to_top (-16)`
	`A::rtti`
	`-- A-in-B vtable address --`
	`B::f() [[-72] B::offset_to_A : thunk]`
	`A::g() []`
	`B::h() [[-72] B::offset_to_A : thunk]`
		`B::offset_`
		`B::offset_`
		`B::rtti`
		`-- B vtabl`
		`B::f() []`
		`B::h() []`
		
		`A::offset_`
		`A::offset_`
		`A::offset_`
		`A::offset_`
		`A::rtti`
		`-- A-in-B`
		`B::f() [[-`
		`A::g() []`
		`B::h() [[-`

`struct C: public virtual A {`
`  void g ();`
`  void h ();`
`  int ic;`
`};`
	`C::offset_to_A (16)`
	`C::offset_to_top (0)`
	`C::rtti`
	`-- C vtable address --`
	`C::g() []`
	`C::h() []`
	
	`A::offset_to_top (-16)`
	`A::rtti`
	`-- A-in-C vtable address --`
	`A::f() []`
	`C::g() [[-72] C::offset_to_A : thunk]`
	`C::h() [[-72] C::offset_to_A : thunk]`
	
	`total size 15*8 = 120 bytes`
		`C::offset_`
		`C::offset_`
		`C::rtti`
		`C vtable a`
		`C::g() []`
		`C::h() []`
		
		`A::offset_`
		`A::offset_`
		`A::offset_`
		`A::offset_`
		`A::rtti`
		`A-in-C vta`
		`A::f() []`
		`C::g() [[-`
		`C::h() [[-`
		
		`total size`

	`D::offset_to_C (16)`
	`D::offset_to_A (32)`
	`D::offset_to_top (0)`
		`D::offset_`
		`D::offset_`
		`D::rtti`
		`-- D, B-in`

<table>
<tr>
<td>struct D: public B, public C {
  void h ();
  int id;
};</td>
<td>D::rtti
-- D, B-in-D vtable address --
B::f() []
D::h() []

C::offset_to_A (16)
C::offset_to_top (-16)
C::rtti
-- C-in-D vtable address --
C::g() []
D::h() [[-88] D::offset_to_C]

A::offset_to_top (-32)
A::rtti
-- A-in-D vtable address --
B::f() [[-128] D::offset_to_A : thunk]
C::g() [[-72] C::offset_to_A : thunk]
D::h() [[-128] D::offset_to_A : thunk]

total size 23*8 = 184 bytes</td>
<td>B::f() []
D::h() []

C::offset_
C::offset_
C::rtti
-- C-in-D
C::g() []
D::h() [-1

A::offset_
A::offset_
A::offset_
A::offset_
A::rtti
-- A-in-D
B::f() [[-
C::g() [[-
D::h() [[-

total size</td>
</tr>
<tr>
<td>struct X {
  int ix;
  virtual void x();
};
struct E : X, D {
  int ie;
  void f();
  void h ();
};</td>
<td>E::offset_to_D (16)
not used
not used
not used
not used
E::offset_to_C (32)
E::offset_to_A (56)
E::offset_to_top (0)
E::rtti
-- E, X-in-E vtable address --
X::x() []
E::f() []
E::h() []

D::offset_to_A (40)
D::offset_to_top (-16)
D::rtti
-- D, B-in-E vtable address --
E::f() [[-144] E::offset_to_D]
E::h() [[-144] E::offset_to_D]

C::offset_to_A (24)
C::offset_to_top (-32)
C::rtti
-- C-in-E vtable address --
C::g() []
E::h() [[-144] E::offset_to_C]

A::offset_to_top (-56)
A::rtti
-- A-in-E vtable address --
E::f() [[-200] E::offset_to_A : thunk]
C::g() [[-72] C::offset_to_A : thunk]
E::h() [[-200] E::offset_to_A : thunk]

total size 37*8 = 296 bytes</td>
<td>E::offset_
E::offset_
E::rtti
-- E, X-in
X::x() []
E::f() []
E::h() []

D::offset_
D::offset_
D::rtti
-- D, B-in
E::f() [-1
E::h() [-1

C::offset_
C::offset_
C::rtti
-- C-in-E
C::g() []
E::h() [-3

A::offset_
A::offset_
A::offset_
A::offset_
A::rtti
-- A-in-E
E::f() [[-
C::g() [[-
E::h() [[-

total size</td>
</tr>
</table>

1. Numbers in parentheses after offset_to_top entries are actual values.
2. Class prefixes for functions identify class where defined.
3. Information in square brackets after function pointer entries indicates entry-point adjustment:
   [] no adjustment required, use primary entry point
   [n] use adjusting entry point that adds "n" to *this*
   [[n] blurb] use adjusting entry point that dereferences *vptr+n* and subtracts (HP) or adds (Cygnus/IBM) that value to *this. blurb* is the name of the accessed field
   [[n] blub : thunk] use adjusting 3rd party thunk that dereferences *vptr+n* and subtracts that value from *this*

Notes: 1) Each function descriptor in the vtable is 16 bytes but the offset and data pointers are only 8, the earlier versions of this table didn't take that into account
2) In the HP column for struct E, I have omitted the D::offset_to_C field because the overrides in E render it unnecessary. However, if maintaining navigability inside the nonvirtual parts of the vtable is important then this "cleanup" can only be done for direct nonvirtual bases and not for more deeply nested ones.
3) I have taken Christophe at his word that thunks are used for adjusting vtable entries in virtual bases in the HP proposal. Some of them could be done with entry points though.

When all is said and done we have

x/y/z
x = # direct secondary entries
y = # "reach back" secondary entries
z = # 3rd-party thunks

Function	HP	Cygnus/IBM
A::f	0/0/0	0/0/0
A::g	0/0/0	0/0/0
A::h	0/0/0	0/0/0
B::f	0/0/2	0/1/0
B::h	0/0/1	0/1/0
C::g	0/0/1	0/1/0
C::h	0/0/1	0/1/0
D::h	0/1/1	1/1/0
E::f	0/1/1	1/1/0
E::h	0/1/1	2/1/0