1551020 Vo Tran Thanh Luong

Virtual Table

What is virtual table ?

A virtual table is is a mechanism used in a programming language to support dynamic dispatch (or run-time method binding).

So first , let's talk about binding.

Each line of machine language is given its own unique sequential address. This is no different for functions -- when a function is encountered, it is converted into machine language and given the next available address. Thus, each function ends up with a unique address. Binding refers to the process that is used to convert identifiers (such as variable and function names) into addresses. Although binding is used for both variables and functions, in this lesson we're going to focus on function binding.

In C++ we have two kinds of binding: Static binding and Dynamic Binding. Static binding means reference are resolved in compile time, whereas dynamic binding means that references are resolved in run time.

Static binding:

```
1. #include
2. int Add(int x, int y)
3. {
4. return x + y;
```

```
}
int main()
{
printf("%d", Add(1, 2));
return 0;
    }
```

Dynamic binding:

```
#include <stdio.h>

int Add(int x, int y)
{
return x + y;
}

int Subtract(int x, int y)
{
    return x - y;
    }

    int main()
    {
    int x = 0;
    int (*ptf_Operator)(int, int);
    printf("Enter an operator (1 = add, 2 = subtract): ");
    scanf_s("%d", &x);
    if (x == 1)
    {
    ptf_Operator = Add;
    }
    else
    {
    ptf_Operator = Subtract;
    }
    printf("%d", ptf_Operator(1,2));
    return 0;
    }
```

Then what is virtual table? The virtual table is a lookup table of functions used to resolve function calls in a dynamic/late binding manner. Every class that owns or inherits a virtual function owns a virtual table. This table is simply a static array

that the compiler sets up at compile time. A virtual table contains one entry for each virtual function that can be called by objects of the class. Each entry in this table is simply a function pointer that points to the most-derived function accessible by that class. The compiler also adds a hidden pointer to the base class, which we will call *__vptr. *__vptr is set (automatically) when a class instance is created so that it points to the virtual table for that class. Unlike the *this pointer, which is actually a function parameter used by the compiler to resolve self-references, *__vptr is a real pointer. Consequently, it makes each class object allocated bigger by the size of one pointer. It also means that *__vptr is inherited by derived classes, which is important.

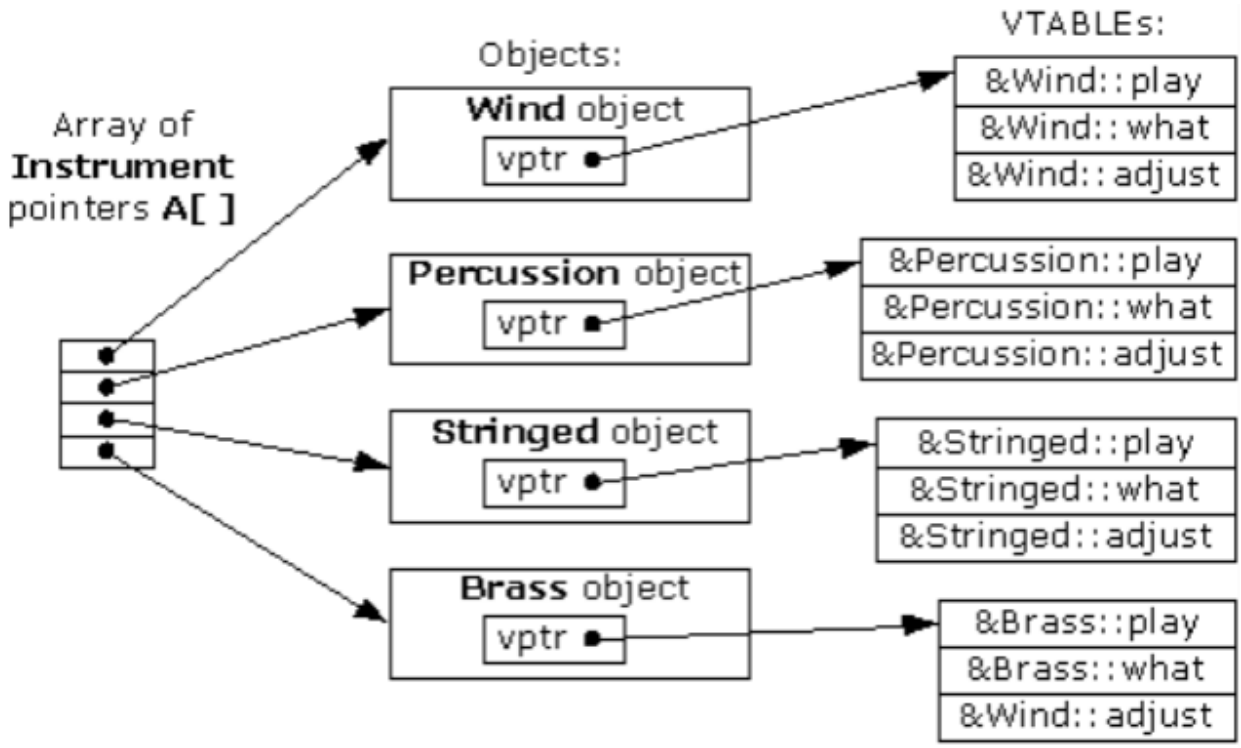
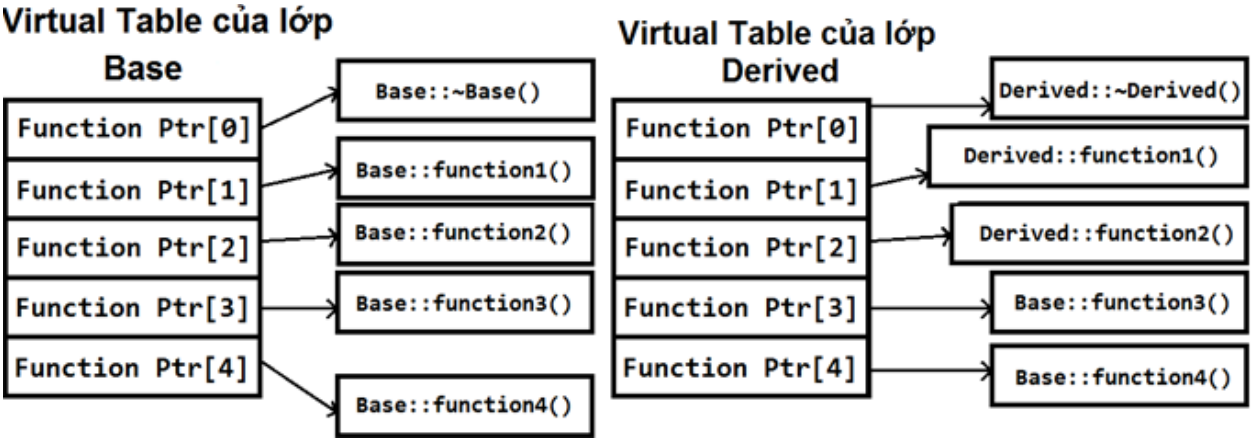


Image example of virtual table.



By using these tables, the compiler and program are able to ensure function calls resolve to the appropriate virtual function, even if you're only using a pointer or reference to a base class!

When working with virtual functions in C++, it's the vtable that's being used behind the scenes to help achieve polymorphism.

Calling a virtual function is slower than calling a non-virtual function for a couple of reasons: First, we have to use the *__vptr to get to the appropriate virtual table. Second, we have to index the virtual table to find the correct function to call. Only then can we call the function. As a result, we have to do 3 operations to find the function to call, as opposed to 2 operations for a normal indirect function call, or one operation for a direct function call. However, with modern computers, this added time is usually fairly insignificant.