In [computer programming](https://en.m.wikipedia.org/wiki/Computer_programming), the term **hooking** covers a range of techniques used to alter or augment the behaviour of an [operating system](https://en.m.wikipedia.org/wiki/Operating_system), of [applications](https://en.m.wikipedia.org/wiki/Application_software), or of other software components by intercepting [function calls](https://en.m.wikipedia.org/wiki/Subroutine) or [messages](https://en.m.wikipedia.org/wiki/Message_passing) or [events](https://en.m.wikipedia.org/wiki/Event_(computing)) passed between [software components](https://en.m.wikipedia.org/wiki/Modular_programming). Code that handles such intercepted function calls, events or messages is called a **hook**.

**Hooking** is used for many purposes, including debugging and extending functionality. Examples might include intercepting **keyboard** or mouse event messages before they reach an application, or intercepting operating system calls in order to monitor behavior or modify the function of an application or other component.

Typically hooks are inserted while software is already running, but hooking is a tactic that can also be employed prior to the application being started. Both these techniques are described in greater detail below.

**Source modification**

By modifying the source of the [executable](https://en.m.wikipedia.org/wiki/Executable) or [library](https://en.m.wikipedia.org/wiki/Library_(computing)) before an application is running, through techniques of [reverse engineering](https://en.m.wikipedia.org/wiki/Reverse_engineering), you can also achieve hooking. This is typically used to intercept function calls to either monitor or replace them entirely.

An alternative method for achieving function hooking is by intercepting function calls through a [wrapper library](https://en.m.wikipedia.org/wiki/Wrapper_library). When creating a wrapper, you make your own version of a library that an application loads, with all the same functionality of the original library that it will replace. That is, all the functions that are accessible are essentially the same between the original and the replacement. This wrapper library can be designed to call any of the functionality from the original library, or replace it with an entirely new set of logic.

**Runtime modification**

Operating systems and software may provide the means to easily insert event hooks at [runtime](https://en.m.wikipedia.org/wiki/Run_time_(program_lifecycle_phase)). It is available provided that the [process](https://en.m.wikipedia.org/wiki/Process_(computing)" \o "Process (computing))inserting the hook is granted enough permission to do so. Microsoft Windows for example, allows you to insert hooks that can be used to process or modify system [events](https://en.m.wikipedia.org/wiki/Event_(computing)) and application events for [dialogs](https://en.m.wikipedia.org/wiki/Dialog_box), [scrollbars](https://en.m.wikipedia.org/wiki/Scrollbar), and [menus](https://en.m.wikipedia.org/wiki/Menu_(computing)) as well as other items. It also allows a hook to insert, remove, process or modify [keyboard](https://en.m.wikipedia.org/wiki/Keyboard_(computing)) and [mouse](https://en.m.wikipedia.org/wiki/Mouse_(computing)) events. Linux provides another example where hooks can be used in a similar manner to process network events within the [kernel](https://en.m.wikipedia.org/wiki/Kernel_(computing)) through [NetFilter](https://en.m.wikipedia.org/wiki/NetFilter" \o "NetFilter).

**Virtual Method Table hooking**[**Edit**](https://en.m.wikipedia.org/w/index.php?title=Hooking&action=edit&section=5)

Whenever a class defines/inherits a [virtual function](https://en.m.wikipedia.org/wiki/Virtual_function) (or method), compilers add a hidden member variable to the class which points to a [virtual method table](https://en.m.wikipedia.org/wiki/Virtual_method_table) (VMT or Vtable). Most compilers place the hidden VMT pointer at the first 4 bytes of every instance of the class. A VMT is basically an array of pointers to all the virtual functions that instances of the class may call. At runtime these pointers are set to point to the right functions, because at compile time, it is not yet known if the base function is to be called or if an overridden version of the function from a derived class is to be called (thereby allowing for [polymorphism](https://en.m.wikipedia.org/wiki/Polymorphism_(computer_science))). Therefore, virtual functions can be hooked by replacing the pointers to them within any VMT that they appear. The code below shows an example of a typical VMT hook in Microsoft Windows, written in C++

#include *<iostream>*

#include *"windows.h"*

**using** **namespace** std;

**class** **VirtualClass**

{

**public**:

int number;

**virtual** void VirtualFn1() *//This is the virtual function that will be hooked.*

{

cout << "VirtualFn1 called " << number++ << "**\n\n**";

}

};

**using** VirtualFn1\_t = void(\_\_thiscall\*)(void\* thisptr);

VirtualFn1\_t orig\_VirtualFn1;

void **\_\_fastcall** hkVirtualFn1(void\* thisptr, int edx) *//This is our hook function which we will cause the program to call instead of the original VirtualFn1 function after hooking is done.*

{

cout << "Hook function called" << "**\n**";

orig\_VirtualFn1(thisptr); *//Call the original function.*

}

int main()

{

VirtualClass\* myClass = **new** VirtualClass(); *//Create a pointer to a dynamically allocated instance of VirtualClass.*

void\*\* vTablePtr = \***reinterpret\_cast**<void\*\*\*>(myClass); *//Find the address that points to the base of VirtualClass' VMT (which then points to VirtualFn1) and store it in vTablePtr.*

DWORD oldProtection;

VirtualProtect(vTablePtr, 4, PAGE\_EXECUTE\_READWRITE, &oldProtection); *//Removes page protection at the start of the VMT so we can overwrite its first pointer.*

orig\_VirtualFn1 = **reinterpret\_cast**<VirtualFn1\_t>(\*vTablePtr); *//Stores the pointer to VirtualFn1 from the VMT in a global variable so that it can be accessed again later after its entry in the VMT has been*

*//overwritten with our hook function.*

\*vTablePtr = &hkVirtualFn1; *//Overwrite the pointer to VirtualFn1 within the virtual table to a pointer to our hook function (hkVirtualFn1).*

VirtualProtect(vTablePtr, 4, oldProtection, 0); *//Restore old page protection.*

myClass->VirtualFn1(); *//Call the virtual function from our class instance. Because it is now hooked, this will actually call our hook function (hkVirtualFn1).*

myClass->VirtualFn1();

myClass->VirtualFn1();

**delete** myClass;

**return** 0;

}

<https://en.m.wikipedia.org/wiki/Hooking>