# Trace Function Calls In A Program

## Compile Time

### Simple Tracing

If we are writing a new code or the code base is small, you can add the macro \_\_FUNCTION\_\_ and \_\_FILE\_\_  at the start and end of every  function to print the names of function.

For example:

#include <stdio.h>

#define ENTER\_FUNC  printf("Enter: %s - %s()\n", \_\_FILE\_\_, \_\_FUNCTION\_\_);

#define EXIT\_FUNC   printf("Exit: %s - %s()\n", \_\_FILE\_\_, \_\_FUNCTION\_\_);

void car()

{

    ENTER\_FUNC

    printf("This is a car\n");

    EXIT\_FUNC

}

void truck()

{

    ENTER\_FUNC

    printf("This is a truck\n");

    EXIT\_FUNC

}

int main(int argc, char \*\*argv)

{

    car();

    truck();

}

Output:

Enter: vehicle.c – car()

This is a car

Exit: vehicle.c – car()

Enter: vehicle.c – truck()

This is a truck

Exit: vehicle.c – truck()

**Explanation:**

|  |  |
| --- | --- |
| **Macro** | **Description** |
| \_\_FILE\_\_ | The current file name as a string literal. |
| \_\_FUNCTION\_\_  (or \_\_func\_\_) | The current function name as a string literal.  For example: getCarName |
| \_\_PRETTY\_FUNCTION\_\_ | The current class name + function name + return type + argument type (if have) as a string literal. Used in C++ only.  For example: void Vehicle::getCarName(int) |

### Hierarchical Tracing

Full code sampe: *Tutorials\C-C++\Code\_C\FunctionCallTrace\Linux\TreeTrace*

Output:

[2022:09:13 13:27:28.354] > (0) int main()

[2022:09:13 13:27:28.356] | > (1) void Vehicle::car(int)(6)

[2022:09:13 13:27:28.358] | > (1) void Vehicle::roadster()

[2022:09:13 13:27:28.360] | | > (2) void Vehicle::truck(int, double)(2, 10.000000)

[2022:09:13 13:27:28.362] | | | > (3) void Vehicle::car(int)(2)

## Run Time

The first approach (compile time) seems easy if we are writing new code. How about handling huge amounts of code that is already written? This is where *gcc* comes to rescue.

Most modern programming languages support the concept of *reflection*where in one can get the name of function or class during runtime very easily.

### Linux

Full code sampe: *Tutorials\C-C++\Code\_C\FunctionCallTrace\Linux\CygProfileTrace*

#### C

#include <stdio.h>

#ifndef \_\_USE\_GNU

    #define \_\_USE\_GNU

#endif

#include <dlfcn.h>

/\*

    Get file name + function name based on function address

/\*/

\_\_attribute\_\_((no\_instrument\_function)) const char\* getSymbolName(void \*addr)

{

    void \*hdl;

    Dl\_info dlInfo = {0};

    hdl = dlopen(NULL, 0);

    dladdr(addr, &dlInfo);

    if ((dlInfo.dli\_saddr != NULL) && (dlInfo.dli\_fname != NULL) && (dlInfo.dli\_sname != NULL)) {

        char name[1024] = {0};

        char\* p\_name = name;

        snprintf(name, sizeof(name), "[%s] %s()", dlInfo.dli\_fname, dlInfo.dli\_sname);

        return p\_name;

    }

    else {

        return "<unknown-symbol>";

    }

}

/\*

    This function is called after entering every other function in the program

\*/

\_\_attribute\_\_((no\_instrument\_function)) void \_\_cyg\_profile\_func\_enter(void \*this\_fn, void \*call\_site)

{

    printf("Enter: %s\n", getSymbolName(this\_fn));

}

/\*

    This function is called before exiting every other function in the program

\*/

\_\_attribute\_\_((no\_instrument\_function)) void \_\_cyg\_profile\_func\_exit(void \*this\_fn, void \*call\_site)

{

    printf("Exit: %s\n", getSymbolName(this\_fn));

}

void car()

{

    printf("This is a car\n");

}

void truck()

{

    printf("This is a truck\n");

}

int main(int argc, char \*\*argv)

{

    car();

    truck();

}

Compile:

$ gcc -g test\_trace.c -finstrument-functions -ldl -Wl,--export-dynamic -o test\_trace

$ ./test\_trace

Output:

Enter: [./test\_trace] main()

Enter: [./test\_trace] car()

This is a car

Exit: [./test\_trace] car()

Enter: [./test\_trace] truck()

This is a truck

Exit: [./test\_trace] truck()

Exit: [./test\_trace] main()

#### C++

Full code sampe: *Tutorials\C-C++\Code\_C++\FunctionCallTrace\Linux\CygProfileTrace*

#include <iostream>     // cout

#include <sstream>      // stringstream

#ifndef \_\_USE\_GNU

    #define \_\_USE\_GNU

#endif

#include <dlfcn.h>

extern "C"

{

    const char\* getSymbolName(void \*addr) \_\_attribute\_\_((no\_instrument\_function));

    void \_\_cyg\_profile\_func\_enter(void \*this\_fn, void \*call\_site) \_\_attribute\_\_((no\_instrument\_function));

    void \_\_cyg\_profile\_func\_exit(void \*this\_fn, void \*call\_site) \_\_attribute\_\_((no\_instrument\_function));

    /\*

        Get function name based on its address

    \*/

    const char\* getSymbolName(void \*addr)

    {

        void \*hdl;

        Dl\_info dlInfo = {0};

        hdl = dlopen(NULL, 0);

        dladdr(addr, &dlInfo);

        if ((dlInfo.dli\_saddr != NULL) && (dlInfo.dli\_fname != NULL) && (dlInfo.dli\_sname != NULL)) {

            char name[1024] = {0};

            char\* p\_name = name;

            snprintf(name, sizeof(name), "[%s] %s", dlInfo.dli\_fname, dlInfo.dli\_sname);

            return p\_name;

        }

        else {

            return "<unknown-symbol>";

        }

    }

    /\*

        This function is called after entering every other function in the program

    \*/

    void \_\_cyg\_profile\_func\_enter(void \*this\_fn, void \*call\_site)

    {

        printf("Enter: %s\n", getSymbolName(this\_fn));

    }

    /\*

        This function is called before exiting every other function in the program

    \*/

    void \_\_cyg\_profile\_func\_exit(void \*this\_fn, void \*call\_site)

    {

        printf("Exit: %s\n", getSymbolName(this\_fn));

    }

}

class Vehicle

{

public:

    void car()

    {

        std::cout << "This is a car\n";

    }

    void truck()

    {

        std::cout << "This is a truck\n";

    }

};

int main()

{

    Vehicle vehicle;

    vehicle.car();

    vehicle.truck();

    return 0;

}

Compile:

$ g++ -g test\_trace.cpp -finstrument-functions -ldl -Wl,--export-dynamic -o test\_trace

$ ./test\_trace | c++filt

Output:

Enter: <unknown-symbol>

Enter: <unknown-symbol>

Exit: <unknown-symbol>

Exit: <unknown-symbol>

Enter: [./test\_trace] main

Enter: [./test\_trace] Vehicle::car()

This is a car

Exit: [./test\_trace] Vehicle::car()

Enter: [./test\_trace] Vehicle::truck()

This is a truck

Exit: [./test\_trace] Vehicle::truck()

Exit: [./test\_trace] main

**Explanation:**

GCC/G++ has a special option [-finstrument-functions](https://gcc.gnu.org/onlinedocs/gcc-4.4.0/gcc/Code-Gen-Options.html#index-finstrument_002dfunctions-1976). We need to implement the two special functions \_\_cyg\_profile\_func\_enter and \_\_cyg\_profile\_func\_exit. The first is called when the program enters a function and the second is called when the program exits the function.

To skip the instrumentation, we add \_\_attribute\_\_((no\_instrument\_function)). Otherwise, the program will be in an infinite loop of entry->entry->entry …

Now we have the mechanism to trace the functions, the next step is to print the function names. The dl library will come to our aid here. You needs to pass on -g -ldl -Wl,–export-dynamic to the compiler:

* -g: Add the symbol tables to the executable
* –export-dynamic: Export all the dynamic symbols in the executable
* ldl: Link to the dl library, so dlopen() and dladdr() functions can be used to get the function name from the address.

Don’t forget to add #define \_\_USE\_GNU, as the dlopen() is a GNU extension.

In C++, the resulting names are mangled, for readability, [c++filt](https://sourceware.org/binutils/docs/binutils/c_002b_002bfilt.html) is used for demangling C++ names.

**Tips:**

* To ignore certain code files, add option: -finstrument-functions-exclude-file-list=path/to/file/
* To know how this feature works in different contexts, check [it](https://fekir.info/post/tracing-with-gcc/).

### Windows

#### C

#### C++

Full code sampe: *Tutorials\C-C++\Code\_C++\FunctionCallTrace\Windows*

/Gh (Enable \_penter hook function): This function is called when the program enters a function.

/GH (Enable \_pexit hook function): This function is called when the program exits a function.

Ref:

<https://www.codeproject.com/Articles/800172/A-Simple-Cplusplus-Profiler-on-x>

<http://www.fengyuan.com/article/profile.html>

<https://www.google.com/search?q=Visualize+function+calls+with+Graphviz&oq=Visualize+function+calls+with+Graphviz&aqs=edge..69i57j69i60l2&sourceid=chrome&ie=UTF-8>