how can I know if a function in C is a system call or not [closed]

I really didn't understand this topic. For example, functions like

- `printf()`
- `strlen()`
- `malloc()`

Which are a system call and which are not? How do I know it when a function is given to me? Didn't find anything on Google.

c call system

edited Mar 18 at 10:11
unwind
260k 40 340 465

asked Mar 18 at 10:01
Niv Gabso
22 2

closed as too broad by Martin James, Alexander O'Mara, Magisch, Mogsdad, JAL Mar 24 at 19:19

There are either too many possible answers, or good answers would be too long for this format. Please add details to narrow the answer set or to isolate an issue that can be answered in a few paragraphs.

If this question can be reworded to fit the rules in the help center, please edit the question.

1 This is highly implementation and platform dependent. I wonder why you'd want to know up front? – rubenvb Mar 18 at 10:15

2 What is "system call" for you and why need to know this? – i486 Mar 18 at 10:25

The most important question is - why do you need this information? – zoska Mar 18 at 10:27

If you dont understand the topic you should probably start in the correct end and start by understanding the concept of "system call" on a higher level. Then you should ask why you would require to know whether something is a system call or not - the answer is very platform dependent and in most cases is completely meaningless. – skyking Mar 18 at 10:30

Test in operating systems. – Niv Gabso Mar 18 at 10:31

4 Answers

That's system dependent. There is a man page `syscalls` that lists the system calls on Linux, for example: http://man7.org/linux/man-pages/man2/syscalls.2.html

This list is also version dependent: new system calls are occasionally added to Linux kernel.

None of the functions in your list are system calls. However, their implementations may use system calls: `printf` uses `write` system call, `malloc` uses `mmap` or `mmap2` or `brk` system calls. In contrast, a typical implementation of `strlen` would NOT use system calls.

answered Mar 18 at 10:13
kfx
3,063 2 7 24

System calls are defined by the type of system or your architecture. Mostly you can expect a system call to have some effect on your program or system itself, like printing to the terminal, communication between processes, dynamic memory allocation and many, many, more.

Most of the commonly used system calls are wrapped by standard library, like `open`, `send`, `clone`, etc. To know which functions out of standard library are system calls you need to check manual pages. Moreover, you can check what kind of system calls does your program call using strace tool. Some more complicated functions are not equivalent to system calls, but use them underneath - like your `printf` and `malloc`.

Some of system calls are not wrapped by standard library and can be only called explicitly: like `gettid` can be only used with `syscall(SYS_gettid)`.

edited Mar 18 at 11:05

answered Mar 18 at 10:26
zoska
1,050 4 18

None of those functions you listed are system calls. Those are all functions in the C Standard Library and I don't think system calls are in it. I think you would have to look up the man page or something equivalent on Google for example for every function if you want to determine whether that function is a system call or not.

answered Mar 18 at 10:08

MacSual
42 ● 4

1 both `printf` and `malloc` use system calls underneath – zoska Mar 18 at 10:13

@zoska Since printf uses a system call "underneath", does it make printf() a system call then? – P.P. Mar 18 at 10:25

@zoska That too is platform dependent. I've seen cases where they do not. – skyking Mar 18 at 10:26

no, it does not – zoska Mar 18 at 10:28

2 @l3x My interpretation is that system calls in the context of C and Systems Programming are the C wrapper functions that encapsulate "true" system calls and that more importantly, their intended purpose is to **explicitly** invoke a specific system call. I don't know if the common and correct consensus is that a function that indirectly invokes a system call as a side effect of its implementation is itself a system call, but I am not of that opinion. – MacSual Mar 18 at 10:43

There are a few concepts here:

- system calls
- standard library

A *system call* connects the program to the particular operating system. These calls perform some standard functionality whose implementation is operating system dependent. For eample, opening a file: `open` is available on all operating systems but its implementation is OS dependent.

In addition, some operating systems provide special functionality that other operating systems don't. Invoking this functionality uses system calls that only exist on that OS. Programs using this functionality are non-portable.

the *standard library* provides many functions that many programs need, for example formating a string using `sprintf`. Often these function also use system calls, for example `printf` uses calls to print to `stdout`. The functions of the standard library are available on all operating systems that support the language.

edited Mar 18 at 10:38

answered Mar 18 at 10:33

Paul Ogilvie
8,631 ● 1 ● 8 ● 22

Your last change was not entirely right. On all C implementations `fopen` exists, but it's implementation differs. While `open` is quite widely available you should not take for granted that it's available in all implementations. – skyking Mar 18 at 10:41

@skyking well, it's available on "most" systems according to Wikipedia: en.wikipedia.org/wiki/Open_%28system_call%29 Bringing in `fopen` in the discussion is confusing because it typically does some user-space stuff besides calling `open`. – kfx Mar 18 at 10:47

@kfx First of all it doesn't claim that. Second on some OSes there will be some user-space stuff being done before issuing the system call, and some user-space stuff being done after it as well. The most known example of such an OS is probably windows. – skyking Mar 18 at 10:54