C printf() in interrupt handler?

I heard printf() in C is not supposed to be used in ISR. Is it because it's a blocking call, or is it because it's not re-entrant?

If printf() is not re-entrant, then wouldn't it means that it can not be used for multi-thread program as well, unless it's 'synchronized' in some way?

Thanks,

c multithreading interrupt-handling

asked Oct 3 '12 at 7:58
user1559625
559 4 16

Here is an excellent article about how you should write ISRs. – Lundin Oct 3 '12 at 8:52

5 Answers

I think it might be all of those, and more. Typical `printf()` implementations can do dynamic (heap) memory allocation, which is generally not the fastest thing to be doing, and might also have issues with being non-re-entrant. The fastness thing can matter since you're typically not supposed to spend too much time in an interrupt service routine.

See this answer for a discussion about `printf()` and `malloc()`.

answered Oct 3 '12 at 8:00
unwind
200k 30 270 387

I'm going to assume that you mean interrupts, even though interrupt handlers in kernels usually have much more special limitations. The same argument applies to signal handlers, but it's usually simpler than the special restrictions on interrupt handlers. In case my assumption is wrong just replace "interrupt" with "signal" in the answer and it will apply.

Functions can be thread-safe without being signal/interrupt safe. If the function protects its internal state with a lock and then holds that lock when getting an interrupt there is no way for the interrupt handler to acquire that lock since the execution path that holds the lock is blocked by the interrupt. To release the lock you'd have to exit from the interrupt handler, resume execution of the thread until the lock is released and then go back to the interrupt handler. This is typically not really doable unless your kernel has implemented interrupt handlers as threads that can yield execution when waiting for locks.

A normal way to make a function both interrupt and thread safe is to block interrupts while holding the lock, but this is quite expensive and isn't done unless it's very necessary.

edited Oct 3 '12 at 11:53

answered Oct 3 '12 at 8:33
Art
6,744 4 19

Why are you making strange assumptions? The question is about ISRs, not signals. There is nothing in the original post suggesting what kind of application this is. The OP could be writing the kernel, or more likely, they could be writing an embedded application. – Lundin Oct 3 '12 at 8:43

@Lundin "multi-threaded program" is a hint. Also, it's a question about the general behavior of printf which usually has very specific semantics, different from libc, inside kernels if they have printf at all.If you'd actually read my answer instead of having an itchy trigger finger, you'd see that I'm saying that the same argument applies to interrupt handlers and the only reason I chose to use the word "signal" is because I had to choose between "signal" and "interrupt" and given the earlier assumptions, I chose "signal". – Art Oct 3 '12 at 9:01

If you couldn't write multi-threaded applications without an OS, how do you think the OS itself turned multi-threaded? In its most simple level, a multi-threaded program could simply be a CPU with 2 cores, running one thread each, but sharing data between them. – Lundin Oct 3 '12 at 9:10

@Lundin Changed the words from signals to interrupts. Happier? – Art Oct 3 '12 at 11:55

It shouldn't be in an ISR because it is not re-entrant nor thread-safe, but *mainly* because it is an **extremely huge function** which will lock up the whole program if you call it from an ISR, creating extreme interrupt jitter and instantly killing every hint of real-time performance in your program.

Huge, bombastic functions should not be in ISRs, no matter if they are thread-safe or not!

answered Oct 3 '12 at 8:40

Lundin
25.3k 5 34 77

> I heard printf() in C is not supposed to be used in ISR. Is it because it's a blocking call, or is it because it's not re-entrant?

More precisely because `printf()` is not a async-signal-safe function. See the list of async-signal-safe at the bottom of Signal Concepts.

answered Oct 3 '12 at 8:50

Maxim Egorushkin
35.1k 4 40 74

Why are you assuming that the OP's application sits on top of an OS? – Lundin Oct 3 '12 at 8:53

@Lundin - 'Is it because it's a blocking call' - a good clue:) – Martin James Oct 3 '12 at 9:41

If you call printf() from there, it may well work, maybe once or twice.. If it tried to block, that's pretty much a disaster since interrupt-handlers have no thread context.

If you link in multithreaded libraries on embedded stuff, printf() will get a mutex-style lock to ensure it's safe to call from multiple threads.

As the other posters say, just don't call such stuff from interrupt-handlers. Signaling semaphore units is always safe, IME. Other stuff only if specifically noted as such in the OS docs.

answered Oct 3 '12 at 9:40

Martin James
16.2k 3 14 33