Threadsafe vs re-entrant

Recently, I asked a question, with title as "Is malloc thread safe?", and inside that I asked, "Is malloc re-entrant?"

I was under the impression that all re-entrant are thread-safe.

Is this assumption wrong?

c thread-safety reentrancy

edited Jul 6 '09 at 16:46
Piotr Dobrogost
16k 9 87 162

asked May 13 '09 at 8:43
Alphaneo
3,520 8 43 70

4 Answers

Re-entrant functions do not rely on global variables that are exposed in the C library headers .. take strtok() vs strtok_r() for example in C.

Some functions need a place to store a 'work in progress' , re-entrant functions allow you to specify this pointer within the thread's own storage, not in a global.

errno, however, is a slightly different case on POSIX systems :)

In short, reentrant *often* means thread safe (as in "use the reentrant version of that function if you're using threads"), but thread safe does not always mean re-entrant. Some functions do not rely on some exposed global variable that other threads could clobber.

malloc() has no need to be reentrant, it does not depend on anything out of the scope of the entry point for any given thread.

Functions that return statically allocated values are *not* thread safe without the use of a mutex, futex, or other atomic locking mechanism. Yet, they don't need to be reentrant.

i.e.:

```
static char *foo(unsigned int flags)
{
  static char ret[2] = { 0 };

  if (flags & FOO_BAR)
    ret[0] = 'c';
  else if (flags & BAR_FOO)
    ret[0] = 'd';
  else
    ret[0] = 'e';

  ret[1] = 'A';

  return ret;
}
```

So, as you can see, having multiple threads use that without some kind of locking would be a disaster .. but it has no purpose being re-entrant. You'll run into that when dynamically allocated memory is taboo on some embedded platform.

In purely functional programming, reentrant often *doesn't* imply thread safe, it would depend on the behavior of defined or anonymous functions passed to the function entry point, recursion, etc.

A better way to put 'thread safe' is ***safe for concurrent access*** , which better illustrates the

need.

edited Jul 28 '13 at 1:46

answered May 13 '09 at 8:55

Tim Post ♦
22.4k 9 71 129

@tinketim, Appreciate your clear and detailed response, thanks. – Alphaneo May 15 '09 at 2:13

1 Reentrant does not imply thread-safe. Pure functions imply thread-safety. – Julio Guerra May 30 '13 at 13:36

Great answer Tim. Just to clarify, my understanding from your "often" is that thread-safe doesn't imply reentrant, but also reentrant doesn't imply thread-safe. Would you be able to find an example of a reentrant function which is *not* thread-safe? – Riccardo Jun 9 '14 at 8:28

This is the definition that Qt uses:

> - A *thread-safe* function can be called simultaneously from multiple threads, even when the invocations use shared data, because all references to the shared data are serialized.
> - A *reentrant* function can also be called simultaneously from multiple threads, but only if each invocation uses its own data.
>
> Hence, a *thread-safe* function is always reentrant, but a *reentrant* function is not always thread-safe.
>
> By extension, a class is said to be *reentrant* if its member functions can be called safely from multiple threads, as long as each thread uses a different instance of the class. The class is *thread-safe* if its member functions can be called safely from multiple threads, even if all the threads use the same instance of the class.

edited Oct 2 '14 at 9:10

answered May 13 '09 at 9:08

Georg Schölly
67.7k 24 144 212

The link no longer seems to be working. – Shafik Yaghmour Oct 1 '14 at 13:15

This definition of reentrant is too strong. – bl4ck5un Oct 25 '14 at 2:33

All re-entrant code is thread-safe. However, not all thread-safe code is re-entrant, for example a function that synchronizes access with a critical section is not re-entrant (because it cannot be entered by multiple threads) but it is thread-safe (because calling it from multiple threads will not cause any problems)

answered May 13 '09 at 8:49

immibis
8,706 1 12 30

9 Code *can* be reentrant and still be *not* thread-safe. See the Wikipedia article which Brian Rasmussen links to. – A.H. Dec 22 '11 at 10:13

In addition to previous answers: two re-entrant functions can be not thread-safe if they handle shared data without locking.

answered May 13 '09 at 8:55

stefaanv
7,635 1 13 29

Since re-entrant code shouldn't handle shared data, I stand corrected. re-entrant code is thread-safe as already stated by others. – stefaanv May 13 '09 at 13:10