

Re: why does this work ?

Source: http://coding.derkeiler.com/Archive/C_CPP/comp.lang.c/2003-10/1819.html

From: Barry Schwarz (*schwarzb_at_deloz.net*)

Date: 10/11/03

Date: 11 Oct 2003 17:17:28 GMT

On Sat, 11 Oct 2003 12:53:07 +0200, Sidney Cadot <sidney@jigsaw.nl> wrote:

>Joona I Palaste wrote:

>

>> <snip>

>> *There is no requirement for implementations to *have* a stack in the first place.*

>

>*This is the second time I see this posted over the last couple of days,*

>*and you're surely right. But it does beg the following question:*

>

>-----

>

>`#include <stdio.h>`

>

>`/* returns n! modulo 2^(number of bits in an unsigned long) */`

>`unsigned long f(unsigned long n)`

>`{`

>`return (n==0) ? 1 : f(n-1)*n;`

>`}`

>

>`int main(void)`

>`{`

>`unsigned long z;`

>`for(z=1;z!=0;z*=2)`

>`{`

>`printf("%lu %lu\n", z, f(z));`

>`fflush(stdout);`

>`}`

>`return 0;`

>`}`

>

>-----

>

>*As far as I can see, this is a perfectly valid C program that should*

>*reach the 'return 0' statement always, were it not for the fact that in*

comp.lang.c: Re: why does this work ?

It is valid in the theoretical sense but it executes in the real world.

>*all compilers I tried (one, actually) it terminates with a segmentation
>fault of some sort, due to limited stack size.*
>
>*Is this 'incorrect behavior' of all these compilers, or is there some
>wording in the Standard that covers for machines with a finite stack
>(much to my dismay, this covers all machines I have access to)?*
>
>*If so, is there a minimum depth of function calls that I can rely on to
>be executed properly? I would hate to rewrite all my programs to do
>everything within main() without function calls, for the ultimate
>portability :-)*

All systems have limited resources. Even virtual memory is backed up in some kind of file. I could not find in the standard a minimum for the number of recursive function calls an implementation was required to support. It does state that a function can be recursively called at least once.

Do you know how much data your system must save when a function is re-called? Usually, this includes information about the state of your task (such as the address to return to) and all automatic variables (those defined in your code and any generated by the compiler to hold intermediate results). There is probably more that I cannot think of at the moment. What would you accept as a reasonable estimate? 100 bytes just to make the arithmetic easier?

ULONG_MAX is required to be greater than 4,000,000,000 (that's billion on my side of the pond but perhaps milliard on yours, $4 \cdot 10^9$ in any case). What will happen when z is 10,000,000 (a mere 10 million)? f will be recursively called 10 million times, requiring 1 gigabyte of storage. Is your task authorized to consume that much of your system resources? Does your system even have that much available? Even if you can support this, can you support z at 100 million requiring 10 gigabytes? At 4 billion (milliard) requiring 400 gigabytes? What happens if long is 64 bits on your system?

Would you be asking your question if your program simply issued malloc requests until no more memory was available? The only difference is that malloc fails "politely" by returning NULL. There doesn't seem to be any way for a recursion failure to be "polite." Both fail for exactly the same reason. Only the symptoms of that failure are different.

Welcome to the limitations of practicality.

<<Remove the del for email>>