Checking available stack size in C

Asked 16 years, 3 months ago Modified 1 year, 1 month ago

Viewed 90k times

46

I'm using MinGW with GCC 3.4.5 (mingw-special vista r3).

My C application uses a lot of stack so I was wondering is there any way I can tell programatically how much stack is remaining so I can cleanly handle the situation if I find that I'm about to run out.

If not what other ways would you work around the problem of potentially running out of stack space?

I've no idea what size of stack I'll start with so would need to identify that programatically also.

c stack mingw

Share

Improve this question

Follow

edited Sep 10, 2008 at 12:15

asked Sep 10, 2008 at 11:53

Paul Hargreaves

1,797 ● 3 ● 23 ● 23

9 Answers

Sorted by: Highest score (default)

36

The getrusage function gets you the current usage . (see man getrusage).

The `getrlimit` in Linux would help fetching the stack size with the `RLIMIT_STACK` parameter.

```
#include <sys/resource.h>
int main (void)
{
  struct rlimit limit;

  getrlimit (RLIMIT_STACK, &limit);
  printf ("\nStack Limit = %ld and %ld max\n", limit.r
}
```

Please give a look at man getrlimit. The same information could be fetched by `ulimit -s` or `ulimit -a` stack size row. Also have a look at `setrlimit` function which would allow to set the limits. But as the mentioned in the other answers if you need to adjust stack then probably you should re consider your design. If you want a big array why not take the memory from the heap ?

Share Improve this answer Follow

edited Jun 3, 2016 at 10:38
Zheng Qu
799 ● 7 ● 25

answered May 3, 2011 at 3:37
phoxis
61.8k ● 14 ● 83 ● 118

9 `getrusage()` doesn't work for stack size on Linux. "`ru_isrss (unmaintained) This field is currently unused on Linux.`" (linux.die.net/man/2/getrusage). I don't know when it came to be so, but it's true for kernel 2.6.28. – atzz Jan 13, 2012 at 11:46

6 @phoxis : `getrlimit (RLIMIT_STACK, &limit)` seems to give total stack size, not the remaining free stack size. – user2284570 Jul 19, 2016 at 17:32

@user2284570 : From `man getrlimit` I can see it's written "The maximum size of the process stack, in bytes." . Can you elaborate on what makes you think that it may be the remaining stack size? – phoxis Jul 21, 2016 at 12:55

2 @phoxis : This is what I'm telling. It's the total stack size. And in the case of the OP only getting the remaining is useful. – user2284570 Jul 22, 2016 at 13:20

20

Taking the address of a local variable off the stack would work. Then in a more nested call you can subtract the address of another local to find the difference between them

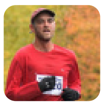
```
size_t top_of_stack;

void Main()
{
  int x=0;
  top_of_stack = (size_t) &x;

  do_something_very_recursive(....)
}
```

```
size_t SizeOfStack()
{
  int x=0;
  return top_of_stack - (size_t) &x;
}
```

If you code is multi-threaded then you need to deal with storing the top_of_stack variable on a per-thread basis.

Share Improve this answer Follow

answered Sep 10, 2008 at 12:02



Rob Walker
47.4k ● 15 ● 100 ● 137

6 I like this answer but without knowing the size of stack up front I've got no way of telling if I'm about to blow it up. – Paul Hargreaves Sep 10, 2008 at 12:18

There will be a default size of the stack for threads on your system. In Windows this is 1MB of address space. You can control this if you are creating your own threads. Though as Skizz points out it would be best not to have to worry about the exact limit! – Rob Walker Sep 10, 2008 at 13:16

4 This might be fine on MinGW in particular. In general, the stack for a program is not guaranteed to be contiguous. It's legal for an implementation (one which doesn't have virtual memory, for example) to allocate stack blocks as required and chain them together. Of course if your platform does that, then there might not even be a default max stack size for a program: you could just keep going until you run out of free memory. But a good reason to have a limit anyway is to prevent runaway recursion from taking down the whole system by exhausting memory. – Steve Jessop Nov 2, 2009 at 12:15

On Linux, you can get the stack size with `ulimit -a`. – Mark Lakata Jun 29, 2018 at 0:35

2 Warnigs: Some platforms (particularly embedded systems) do not allocate data on the stack (only function return addresses are stored on the stack). In this case, the address of local variables are meaningless. – Mark Lakata Jun 29, 2018 at 0:37

9

check if your compiler supports stackavail()

Share Improve this answer Follow

answered Sep 10, 2008 at 13:40

dmityugov
4,480 ● 24 ● 18

6

Assuming you know the size of the full stack you could probably add some assembly code to read ESP.
If you read ESP and save it aside in the main function you can compare the current ESP to the ESP you have in main and see how much ESP has changed. That'll give you an indication of how much stack you're used.

Share Improve this answer Follow

answered Sep 10, 2008 at 12:01

Nathan Fellman
127k ● 104 ● 264 ● 326

This is a problem I have given up on. With a lot of hacking and (mostly) praying, you can get a solution that

6

works at a given time on a given machine. But in general there seems to be no decent way to do this.

You will have to obtain the stack position and size from outside your program (on Linux you might get it from `/proc/<pid>/maps`). In your program you must somehow test where you are at the stack. Using local variables is possible, but there is no real guarantee that they are actually on the stack. You can also try to get the value from the stack pointer register with some assembly.

So now you have the location of the stack, its size and the current position and you assume you know in which direction the stack grows. When are you going in stack-overflow mode? You better not do it close to the end because your estimation (i.e. address of local variable or value from stack pointer) is probably a bit too optimistic; it's not uncommon to address memory beyond the stack pointer. Also, you have no clue about how much room on the stack any given function (and the functions it calls) need. So you'll have to leave quite some room at the end.

I can only advice you not do get into this mess and try to avoid very deep recursion. You might also want to increase your stack size; on Windows you have to compile this into the executable, I believe.

Share Improve this answer Follow

answered Sep 10, 2008 at 12:47

mweerden
14k ● 5 ● 34 ● 32

4

maybe this will help for Windows platform only:

in the PE header (IMAGE_NT_HEADERS) of your exe there are some records such as:

```
typedef struct _IMAGE_NT_HEADERS {
    DWORD Signature;
    IMAGE_FILE_HEADER FileHeader;
    IMAGE_OPTIONAL_HEADER32 OptionalHeader;
} IMAGE_NT_HEADERS32, *PIMAGE_NT_HEADERS32;

typedef struct _IMAGE_OPTIONAL_HEADER {
    ...
    DWORD   SizeOfStackReserve;
    DWORD   SizeOfStackCommit;
    ...
}
```

There is a simple way to obtain these values: using GetModuleHandle(NULL) will give you the imagebase (handle) of your module, address where you'll find a IMAGE_DOS_HEADER structure which will help you to find the IMAGE_NT_HEADERS structure (imagebase+IMAGE_DOS_HEADER.e_lfanew) -> IMAGE_NT_HEADERS, and there you'll find those fields: **SizeOfStackReserve** and **SizeOfStackCommit**.

The maximum amount of space that the OS will allocate for your stack is SizeOfStackReserve.

If you consider trying this, let me know and I will assist you. There is a way to obtain the size of stack used in a certain point.

Share Improve this answer Follow

answered Sep 11, 2008 at 13:55

botismarius
3,035 ● 4 ● 31 ● 30

The values in the PE header are only used for the entry thread. Every other thread can be created with caller's choice of either the process-wide default or any other size.
– Ben Voigt Apr 14, 2021 at 22:00

3

For windows: I've done this before using the VirtualQuery function from Kernel32.dll. I only have an example in C# but it demonstrates the technique:

```
public static class StackManagement
    {
        [StructLayout(LayoutKind.Sequential)]
        struct MEMORY_BASIC_INFORMATION
        {
            public UIntPtr BaseAddress;
            public UIntPtr AllocationBase;
            public uint AllocationProtect;
            public UIntPtr RegionSize;
            public uint State;
            public uint Protect;
            public uint Type;
        };

        private const long STACK_RESERVED_SPACE = 4096

        public unsafe static bool CheckForSufficientSt
        {
            MEMORY_BASIC_INFORMATION stackInfo = new
MEMORY_BASIC_INFORMATION();
            UIntPtr currentAddr = new UIntPtr(&stackIn
            VirtualQuery(currentAddr, ref stackInfo,
sizeof(MEMORY_BASIC_INFORMATION));
```

```
            UInt64 stackBytesLeft = currentAddr.ToUInt
stackInfo.AllocationBase.ToUInt64();

            return stackBytesLeft > (bytes + STACK_RES
        }

        [DllImport("kernel32.dll")]
        private static extern int VirtualQuery(UIntPtr
MEMORY_BASIC_INFORMATION lpBuffer, int dwLength);
    }
```

BTW: This code can also be found on StackOverflow on another question which I asked when I was trying to fix a bug in the code: [Arithmetic operation resulted in an overflow in unsafe C#enter link description here]

Share Improve this answer

Follow

edited May 23, 2017 at 10:31

Community Bot
1 ● 1

answered Jan 3, 2012 at 17:44

Daniel James Bryars
4,601 ● 3 ● 45 ● 58

2

Raymond Chen (The Old New Thing) has a good answer to this sort of question:

> If you have to ask, you're probably doing something wrong.

Here's some Win32 details on stack allocation: MSDN.

If you think you might be limited by stack space, you will almost certainly be limited by available virtual memory, in which case, you will need to find a different solution.

What exactly are you trying to do?

Share Improve this answer Follow

edited Jan 3, 2012 at 17:26
user142162

answered Sep 10, 2008 at 12:38
Skizz
71k ● 10 ● 74 ● 109

2 A (not great) example would be: void subroutine(int i) { char foo[20000]; i++; if (i < 1000) subroutine(i); }
– Paul Hargreaves Sep 10, 2008 at 12:57

You're right, that's not a good example. What I really wanted to know was what you were doing with the 20k array. – Skizz Sep 10, 2008 at 13:39

49 Although if you've ever tried to write really, truly portable code, you learn that "you always have to ask, and you're always doing something wrong, because there is no portable concept of "stack usage" and yet it's the programmer's responsibility not to use too much stack. So it's best just to join the conspiracy of silence, write a functional test that you hope consumes as much stack as your program ever will in practice, and leave it to the platform integrator to worry about". – Steve Jessop Nov 2, 2009 at 12:10

13 Question is not "Should I check the stack size?" it is "How do I check the stack size?" – Justicle May 20, 2018 at 18:25

1 I just want to query stack size to answer the question "are file scoped arrays put onto the stack?" I assume no, but by

making two simple programs that query stack size, one with, and one without a file scoped array, I can answer the question myself. – KANJICODER Apr 17, 2020 at 21:49

2

On Linux, you would call `getrusage` and check the returned `struct rusage`'s `ru_isrss` member (integral unshared stack size).

From the MinGW site and its SourceForge site's tracking of patches, I see that in May of 2008 there was some patching done around `getrusage` and it looks like it's been generally supported for quite a while. You should check carefully for any caveats in terms of how much of the typical Linux functionality is supported by MinGW.

Share Improve this answer Follow

edited Oct 29, 2023 at 12:59
ib.
28.9k ● 12 ● 85 ● 105

answered Sep 12, 2008 at 17:34
Thomas Kammeyer
4,507 ● 23 ● 31

This is the best way to do it, unless your doing some insane statically allocated mapping. Not to say all statically allocated mapping is insane, but it usually is :) – user50049 Apr 26, 2011 at 11:24

7 `getrusage()` doesn't work for stack size on Linux. "`ru_isrss (unmaintained)    This field is currently unused on Linux.`" (linux.die.net/man/2/getrusage). I don't know when it came to

be so, but it's true for kernel 2.6.28. – atzz Jan 13, 2012 at 11:43