Stack Memory vs Heap Memory [duplicate]

Possible Duplicate:
What and where are the stack and heap

I am programming in C++ and I am always wondering what exactly is stack memory vs heap memory. All I know is when I call new, I would get memory from heap. If if create local variables, I would get memory from stack. After some research on internet, the most common answer is stack memory is temporary and heap memory is permanent.

Is stack and heap memory model a concept of operating system or computer architecture? So some of it might not follow stack and heap memory model or all of them follow it?

Stack and heap memory is the abstraction over the memory model of the virtual memory (which might swap memory between disk and RAM). So both stack and heap memory physically might be RAM or the disk? Then what is the reason where heap allocation seems to be slower than the stack counterpart?

Also, the main program would be run in the stack or a heap?

Also, what would happen if a process run out of the stack memory or heap memory allocated?

Thanks

c++ memory

edited Apr 29 '11 at 19:11

asked Apr 29 '11 at 19:09
Steveng
303 3 8 13

marked as duplicate by meagar ♦, Fred Foo, Bo Persson, Prasoon Saurav, YOU Apr 30 '11 at 11:50

This question has been asked before and already has an answer. If those answers do not fully address your question, please ask a new question.

3 @meagar This does raise some questions not mentioned in the question you linked to. – Maxpm Apr 29 '11 at 19:35

3 Answers

Stack memory is specifically the range of memory that is accessible via the Stack register of the CPU. The Stack was used as a way to implement the "Jump-Subroutine"-"Return" code pattern in assembly language, and also as a means to implement hardware-level interrupt handling. For instance, during an interrupt, the Stack was used to store various CPU registers, including Status (which indicates the results of an operation) and Program Counter (where was the CPU in the program when the interrupt occurred).

Stack memory is very much the consequence of usual CPU design. The speed of its allocation/deallocation is fast because it is strictly a last-in/first-out design. It is a simple matter of a move operation and a decrement/increment operation on the Stack register.

Heap memory was simply the memory that was left over after the program was loaded and the Stack memory was allocated. It may (or may not) include global variable space (it's a matter of convention).

Modern pre-emptive multitasking OS's with virtual memory and memory-mapped devices make the actual situation more complicated, but that's Stack vs Heap in a nutshell.

edited Apr 29 '11 at 19:30

answered Apr 29 '11 at 19:24

rskar
2,944 13 17

This is wrong in so many ways. There is typically no difference at all between "stack memory" and "heap memory", on any architecture I'm aware of. Both can be accessed by stack pointer, index registers, or whatever. Both the stack pointer and index registers can be changed arbitrarily to access any memory available to the process. – Neil Butterworth Apr 29 '11 at 20:06

@unapersson: The RAM is the RAM, and the SP can be change and assigned of course. "Stack Memory" is more a term of intent or role - not necessarily of a specialize piece of hardware. BTW, in the case of the 6502, your stack was indeed stuck at Page 1 (bytes 256 thru 511). – rskar Apr 29 '11 at 20:11

@rskar Yes indeed. I wrote an implementation of the KERMIT protocol, with full VT100 terminal emulation on the BBC Micro in 6502 assembler (not a happy experience for a Z80 programmer) back in 1984, so I do know of what I speak here. – Neil Butterworth Apr 29 '11 at 20:15

3 @rskar For most architectures, there is no difference between stack memory and ordinary memory. Even on the 6502 you could (and sometimes had to) implement your own stack in "ordinary" memory - you just could not use it via call and return opcodes. Even on the 6502 there was nothing special about stack memory, only about how it was accessed (via the 6502 stack page) You seem to be suggesting that there is actually a physical difference between stack and other types of memory. – Neil Butterworth Apr 29 '11 at 20:29

4 @unapersson: My apologies - I never meant to suggest that there is actually a physical difference between stack and other types of memory. – rskar Apr 29 '11 at 20:39

In C++ the stack memory is where local variables get stored/constructed. The stack is also used to hold parameters passed to functions.

The stack is like a very like the std::stack class, you push parameters onto it and then call a function. The function then knows the parameters it expects can be found on the end of the stack. Likewise the function can push locals onto the stack and pop them off it before returning from the function. (caveat- compiler optimizations and calling conventions all mean things aren't this simple)

The stack is really best understood from a low level and I'd recommend this link Art of Assembly - Passing Parameters on the Stack. Rarely if ever would you consider any sort of manual stack manipulation from C++.

Generally speaking the stack is preferred as it is usually in the CPU cache, so operations involving objects stored on it tend to be faster. However the stack is a limited resource, and shouldn't be used for anything large. Running out of stack memory is called a Stack buffer overflow. It's a serious thing to encounter, but you really shouldn't come across one unless you have a crazy recursive function or something similar.

Heap memory is much as rskar says. Generally speaking in C++ objects allocated with new, or blocks of memory allocated with the likes of malloc ends up on the heap. Heap memory almost always must be manually freed, though you should reallly use a smart pointer class or similar to avoid needing to remember to do so. Running out of heap memory can (will?) result in a std::bad_alloc.

edited Sep 2 '14 at 1:25

Edmond Burnett
949 1 8 16

answered Apr 29 '11 at 19:34

Eoin
770 4 16

2 "Stack memory almost always must be manually freed." You meant heap memory. Also, it's worth mentioning that heap memory is "global", when you run out of heap other applications will also run out of heap. The stack though, has local scope. – Spidey Mar 16 '12 at 14:08

It's a language abstraction - some languages have both, some one, some neither.

In the case of C++, the code is not run in either the stack or the heap. You can test what happens if you run out of heap memory by repeatingly calling `new` to allocate memory in a loop without calling `delete` to free it it. **But make a system backup before doing this**.

answered Apr 29 '11 at 19:14

Neil Butterworth
21.7k 4 40 71

15 "Make a system backup" before trying to fill the heap? What platform are you on, MS-DOS? – Fred Foo Apr 29 '11 at 19:33

1 @larsman No, I've had this crap out expensively on Unix systems where user limits were not set. And it will probably cause problems even on "modern" operating systems - try it (with my proviso). – Neil Butterworth

Apr 29 '11 at 19:36

@unapersson: I've run out of memory on Unix and Linux systems before. I never experienced any great problems; usually just a long slow-down until everything was back in order, sometimes the necessity to reboot. – Fred Foo Apr 29 '11 at 19:41

@larsman But did you remove the limits on user processes? And have you tried it on, say, Windows? And most for most of us Unix folks, the "necessity to reboot" is the next step to getting the sack. – Neil Butterworth Apr 29 '11 at 19:46

@unapersson: all systems (Linux, Mac OS X, OpenBSD) had default limits. And no, I never try anything heavy on Windows, I hate cleaning up broken glass ;) – Fred Foo Apr 29 '11 at 19:48