

Stack Overflow em Português é um site de perguntas e respostas para programadores profissionais e entusiastas. Leva apenas um minuto para se inscrever.

Qualquer pessoa pode fazer uma pergunta

Qualquer um pode responder

Inscreva-se para participar desta comunidade

As melhores respostas recebem votos positivos e sobem para os primeiros lugares

## O que são e onde estão a “stack” e “heap”?

Perguntada 7 anos, 3 meses atrás Ativa 2 dias atrás Vista 38mil vezes

- ▲ O que são esses tais de *stack* e *heap* que tanto se fala em gerenciamento de memória?
- 213 Isso realmente são porções da memória como algumas pessoas falam ou é apenas um conceito abstrato para facilitar o entendimento da forma como se gerencia a memória?
- ▼
- ★ 85 Algum deles é mais rápido que o outro? Se um é claramente mais rápido, por que existe o outro?
- 🕒 Faz diferença se estou usando, por exemplo, Assembly, C, Java, ou Javascript, Windows ou Linux? Isso é controlado pela "linguagem" ou pelo sistema operacional?

Enfim, queria entender melhor esse conceito que parece ser mal compreendido pelos programadores. Seria muito útil uma explicação para quem está começando ou aprendeu isso de forma errada.

memória gerenciamento-de-memória stack heap

compartilhar

editada 18/05/20 às 18:14

perguntada 2/02/14 às 19:19

melhorar esta pergunta seguir



Maniero ♦

433mil 76 1122  
1679

[translate.google.com/...](https://translate.google.com/...) Tradução automatica de um Curso de ASM68 que eu escrevi 30 anos atrás; Com a ajuda de um pouco de café, e paciencia isso vai permitiir vc entender como funciona tudo isso. :) – Peter 30/09/20 às 2:50

4 Respostas

Ativos Oldest Votos



218



+50



Uma *stack* (ou pilha), neste contexto, é uma forma otimizada para organizar dados na memória alocados em sequência e abandonados (sim, normalmente não há desalocação) em sequência invertida a da entrada.



Um *heap* (ou monte, ok, ninguém traduz isso) é a organização de memória mais flexível que permite o uso de qualquer área lógica disponível.



**De que pilha estamos falando?**

- Existe a **pilha de chamadas de funções**, que se confunde com o gerenciamento de memória, onde as funções vão sendo chamadas e empilhadas e quando sua execução termina, ela sai da pilha.
- Existe a [estrutura de dados genérica](#) que empilha dados diversos. [Exemplo em C#](#).

## Conceito abstrato

Os dois conceitos da pergunta são abstratos. Não existe fisicamente uma área da memória específica para a *stack* (e muito menos sua área é fisicamente empilhada) e não existe uma área reservada para o *heap*, pelo contrário, ele costuma ser bastante fragmentado. Usamos o conceito para entender melhor o funcionamento e suas implicações, principalmente no caso da pilha.

A maioria das arquiteturas de computadores modernos e populares não tem grandes facilidades para manipular essa *stack* de memória (costuma ter só o registrador de ponteiro de pilha), assim como o *heap*, apesar que neste caso, instruções que ajudam manipular memória virtual de uma certa forma ajudam a organizar o *heap*, mas isso vale pra toda memória, não só o *heap*.

## Ficando um pouco mais concreto

Já o sistema operacional está bem ciente destes conceitos e é fundamental que eles possuam alguma forma, mesmo que limitada, para manipular a memória das aplicações, principalmente nos sistemas modernos e de utilidade geral. Os sistemas modernos possuem um gerenciamento complexo através do que se convencionou chamar de memória virtual que também é um conceito abstrato, muitas vezes mal compreendido.

## Onde mexemos diretamente

Em Assembly ou C é muito comum ter contato com esse gerenciamento de memória. Em Assembly é comum manipular a *stack* quase diretamente e em ambas linguagens pelo menos a alocação e desalocação do *heap* devem ser feitas manualmente através da API do sistema operacional. Em C a *stack* é gerenciado pelo compilador, salvo alguma operação incomum que seja necessária.

Nada impede que se use alguma biblioteca que abstraia essa manipulação, mas isso só é comum em linguagens de mais alto nível. De fato é muito comum que outras linguagens

compressão e acesso.

## Pilha

### Alocação

Em condições normais, **a *stack* é alocado no início da execução da aplicação**, mais precisamente no início da *thread*, mesmo que a aplicação só tenha a *thread* principal.

A *stack* é uma porção contígua de memória reservada para empilhar os dados necessários durante a execução de blocos de código.

Cada necessidade de alocação é um trecho da *stack* que vai sendo usado sempre em sequência determinado por um marcador, ou seja, um apontador, um **ponteiro**, se "movimenta" para indicar que uma nova parte na sequência desta porção reservada está comprometida.

Quando algo reservado para um segmento não é mais necessário, este marcador se movimenta em direção contrária a sequência de dados indicando que alguns desses dados podem ser descartados (sobrepostos com novos dados).

A alocação de cada trecho da memória não existe na *stack*, é apenas o movimento deste ponteiro indicando que aquela área será usada por algum dado.

A grosso modo podemos dizer que a aplicação tem total controle sobre a *\*stack*, exceto quando acaba o espaço disponível nele.

Existem recursos para alterar manualmente o tamanho da *stack*, mas isso é incomum.

## Funcionamento

A pilha funciona usando uma forma LIFO (Last in First Out) ou UEPS (Último a entrar, primeiro a sair).

O escopo de uma variável costuma definir o tempo de alocação na *stack*. Os dados



Deu para entender que cada *thread* tem sua própria *stack*, certo? E o tamanho da *stack* de cada *thread* criada pode ter seu tamanho definido antes da criação. Um valor *default* costuma ser usado.

A *stack* é considerado uma forma **automática** de alocação (muitas vezes confundida com estática que é alocação que ocorre junto ao execução logo na sua carga. Tecnicamente existe outra área da memória que realmente seja estática, que seja alocada antes do início da execução. A área efetivamente estática não pode ser manipulada, não pode ser escrita (pelo menos não deveria poder). A *stack* em si é estático, apesar de que seus dados não sejam, afinal eles vão sendo colocados e abandonados conforme o seu uso, o seu gerenciamento é automático.

## Decisão sobre onde alocar

Assim como no *heap*, não é possível alocar dados na *stack* antes de saber seu tamanho

# Heap

## Alocação

O *heap*, ao contrário da *stack*, não impõe um modelo, um padrão de alocação de memória. Isso não é muito eficiente mas é bastante flexível.

O *heap* é considerado **dinâmico**. Em geral você aloca ou desaloca pequenos trechos de memória, só para a necessidade do dado. Esta alocação pode ocorrer fisicamente em qualquer parte livre da memória disponível para seu processo.

O gerenciamento de memória virtual do sistema operacional, auxiliado por instruções do processador, ajudam a organizar isto.

De certa forma podemos dizer que a *stack* como um todo é o primeiro objeto alocado no *heap*.

Efetivamente estas alocações reais costumam ocorrer em blocos de tamanho fixo chamados de páginas. Isso evita a aplicação fazer dezenas ou centenas de pequenas alocações que fragmentaria a memória de forma extrema e evita chamadas ao sistema operacional que troca contexto e costuma ser bem mais lento. Em geral todo sistema de alocação da memória aloca mais do que precisa e vai dando acesso à aplicação conforme ela precisa, em alguns casos, ele quase simula uma *stack*, por algum tempo, ou fazer reorganização da memória (através de um GC compactador).

seja escrita novamente.

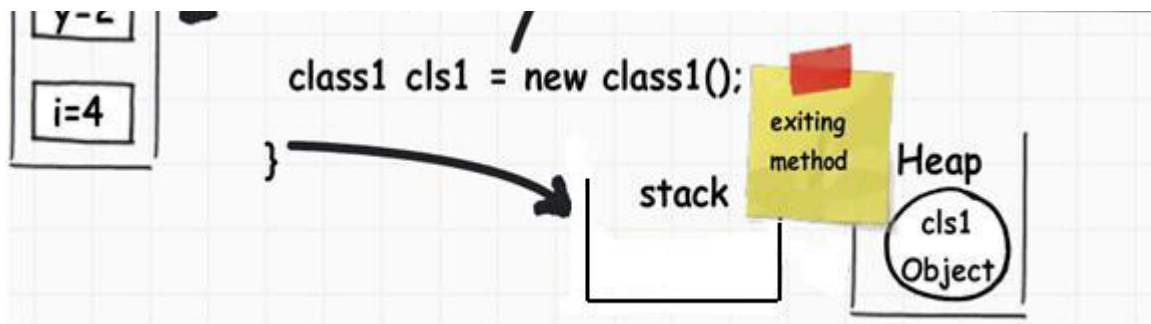
## Custo do *heap*

A **alocação no *heap* "custa" caro**. Muitas tarefas devem ser realizados pelo sistema operacional para garantir a perfeita alocação de uma área para um trecho dele, principalmente em ambientes concorrentes, muito comuns hoje em dia, e mesmo quando não precisa do SO ainda tem um algoritmo complexo para alocar. Desalocar, ou disponibilizar de volta uma área também tem seu custo, em alguns casos para a alocação custar mais barata a liberação custa bem cara (ironicamente pode ser controlada por várias pilhas).

Até existem formas de evitar as chamadas ao sistema operacional para cada alocação necessária, mas ainda assim o "custo" de processamento disto é considerado alto. Manter listas (em alguns casos ligadas) de áreas ou páginas alocadas não é algo trivial para o processador, pelo menos comparando com o movimento do ponteiro que é necessário na *stack*.

## Funcionamento






Note que no exemplo, um objeto do tipo `class1` é alocado na *heap*. Mas há uma referência para este objeto, que é alocada na *stack* (em alguns casos poderia não estar).

Esta alocação é necessária porque o tamanho do objeto pode ser muito grande para caber na *stack* (ou pelo menos ocupar uma parte considerável), ou porque ele pode sobreviver por mais tempo do que a função que criou ele.



---

"Em Assembly é comum manipular o stack quase diretamente" porque "quase"? `move.w d1,-(sp)`  
`bsr my_function addq.l #2,sp` Vc edita diretamente o "stack pointer". Não tem outro jeito. – [Peter](#)  
30/09/20 às 2:31 

---