

# memória ram e suas segmentações

lucas pereira  
nº Usp: 12543930

Junho 2021

## 1 A Stack e a Heap

Tanto a Stack quanto a Heap são duas formas de segmentação da memória do computador, sendo a Stack um espaço na memória que o sistema operacional atribui ao seu programa assim que este é iniciado, enquanto a Heap possibilita o acesso a toda memória RAM disponível na máquina em que o sistema estiver sendo executado.

### 1.1 Como são organizadas

A memória Stack é organizada em formato de pilha, alocando os dados do programa como funções e variáveis que são automaticamente desalocadas assim que a função é finalizada. Já a Heap, é um espaço de alocação dinâmica que armazena os dados de forma não sequencial, que podem ser acessados a qualquer momento e precisam ser desalocados manualmente.

### 1.2 Tamanho

A Stack possui um tamanho que varia de acordo com o sistema operacional que geralmente consiste em algo em torno de 3mb, o que é extremamente limitado para os sistemas modernos. Enquanto a Heap, por sua vez, pode utilizar toda memória disponível no sistema, o que pode representar um limite de aproximadamente 3,12 gb em um sistema de 32 bits e até 512gb em um sistema operacional moderno de 64bits (Windows 10 Pro 64 bit).

## **1.3 Em quais casos utilizar cada tipo**

### **1.3.1 A Stack**

A memória Stack é ideal para ser utilizada em variáveis que tenham uma entrada previsível e tamanho limitado como: datas, números de telefone, peso ou altura de uma pessoa, pois já temos uma boa ideia do quanto de memória será necessária para alocar este tipo de dado e que esta quantidade não é significativamente grande. Fora que também não será necessário desalocar a memória depois de utilizá-la e correr o risco de vazamento de memória (Memory Leak) caso esqueçamos de desalocar.

### **1.3.2 A Heap**

A Heap deve ser utilizada em casos de alocação de variáveis como nomes, textos, dados em grandes quantidades no geral e entradas que não podemos prever a quantidade de espaço necessário, pois ao utilizarmos a memória Heap, temos muito mais espaço e a capacidade de alterar a quantidade de memória que será necessária para cada variável em tempo de execução, embora precisemos ter mais responsabilidade ao utilizá-la para evitar o memory Leak.

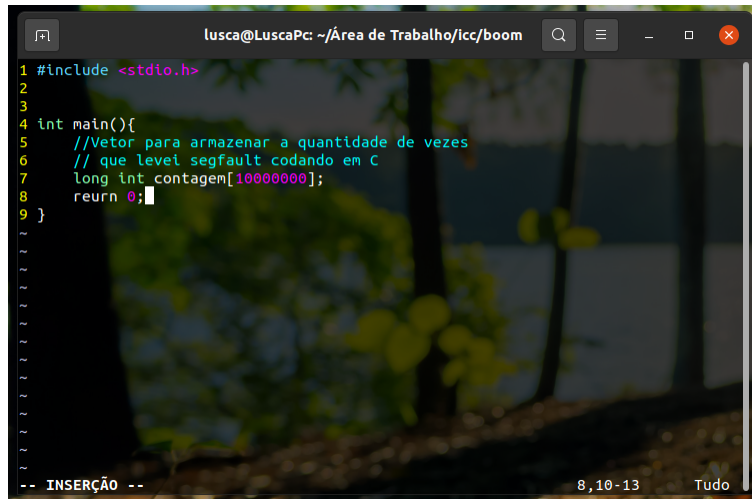
## **1.4 O que acontece se abusar da memória stack?**

Como a stack é limitada por um tamanho bem pequeno, caso você abuse dela declarando muitas variáveis, funções recursivas sem condição de saída ou vetores com dez milhões de posições, o espaço disponível para a sua Stack pode acabar e para evitar que suas próximas declarações utilizem endereços que não foram atribuídos ao seu programa, o sistema operacional o encerra, retornando o erro de Falha de segmentação (Segmentation fault).

## 2 Exemplos

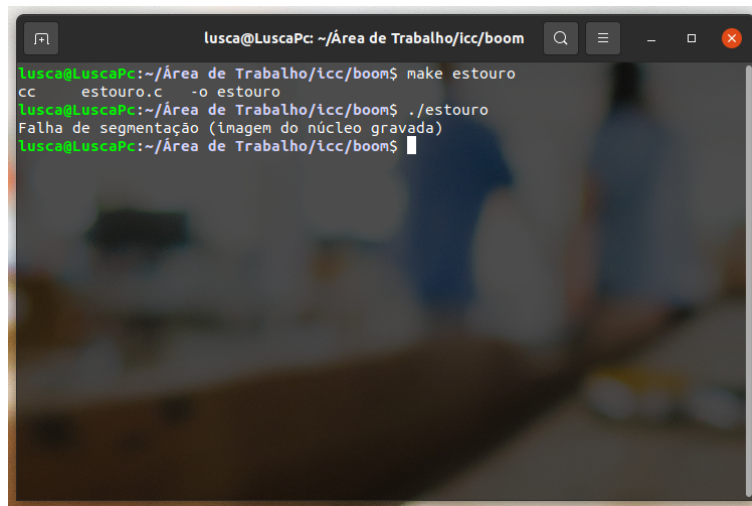
Aqui separei alguns casos em que podem resultar em erros:

### 2.1 Tentando alocar um vetor muito grande na Stack



```
1 #include <stdio.h>
2
3
4 int main(){
5     //Vetor para armazenar a quantidade de vezes
6     // que levei segfault codando em C
7     long int contagem[10000000];
8     return 0;
9 }
~
~
~
~
~
~
~
~
~
~
-- INSCRIÇÃO -- 8,10-13 Tudo
```

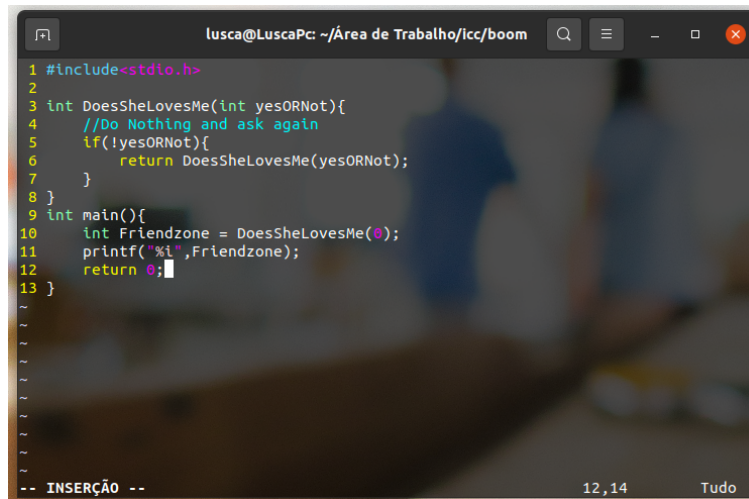
Figure 1: Código utilizado



```
lusca@LuscaPc: ~/Área de Trabalho/icc/boom$ make estouro
cc estouro.c -o estouro
lusca@LuscaPc:~/Área de Trabalho/icc/boom$ ./estouro
Falha de segmentação (imagem do núcleo gravada)
lusca@LuscaPc:~/Área de Trabalho/icc/boom$
```

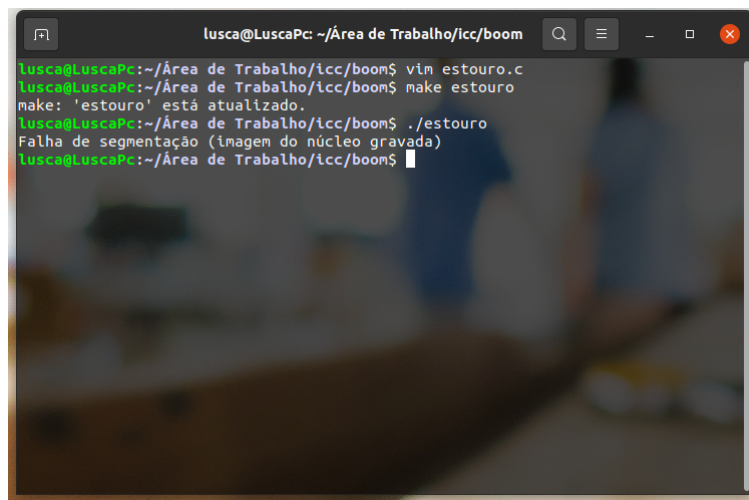
Figure 2: Resultado Obtido

## 2.2 Criando uma callstack enorme até estourar



```
1 #include<stdio.h>
2
3 int DoesSheLovesMe(int yesORNot){
4     //Do Nothing and ask again
5     if(!yesORNot){
6         return DoesSheLovesMe(yesORNot);
7     }
8 }
9 int main(){
10     int Friendzone = DoesSheLovesMe(0);
11     printf("%i", Friendzone);
12     return 0;
13 }
```

Figure 3: Código utilizado

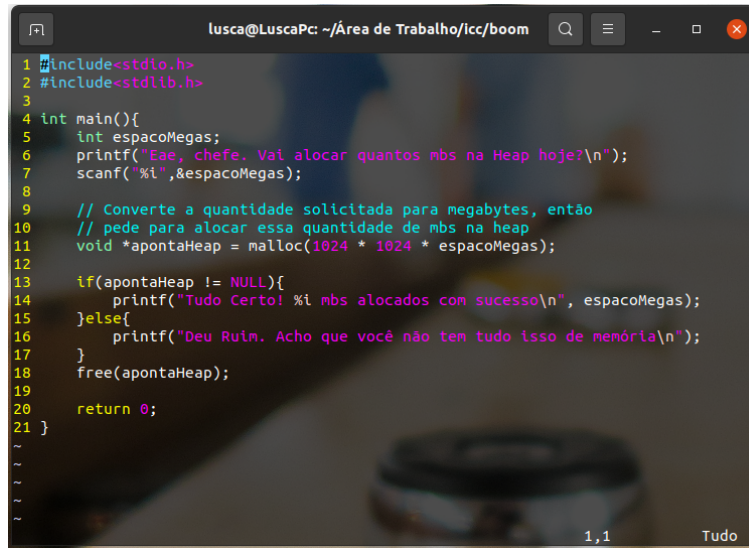


```
lusca@LuscaPc:~/Área de Trabalho/icc/boom$ vim estouro.c
lusca@LuscaPc:~/Área de Trabalho/icc/boom$ make estouro
make: 'estouro' está atualizado.
lusca@LuscaPc:~/Área de Trabalho/icc/boom$ ./estouro
Falha de segmentação (imagem do núcleo gravada)
lusca@LuscaPc:~/Área de Trabalho/icc/boom$
```

Figure 4: Resultado Obtido

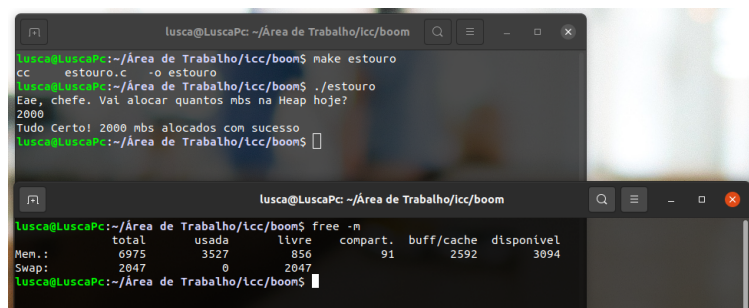
## 2.3 Testando a Heap

Para este caso, utilizei uma máquina com 8gbs de memória RAM e 3gbs disponíveis.



```
1 #include<stdio.h>
2 #include<stdlib.h>
3
4 int main(){
5     int espacoMegas;
6     printf("Eae, chefe. Vai alocar quantos mbs na Heap hoje?\n");
7     scanf("%i",&espacoMegas);
8
9     // Converte a quantidade solicitada para megabytes, então
10    // pede para alocar essa quantidade de mbs na heap
11    void *apontaHeap = malloc(1024 * 1024 * espacoMegas);
12
13    if(apontaHeap != NULL){
14        printf("Tudo Certo! %i mbs alocados com sucesso\n", espacoMegas);
15    }else{
16        printf("Deu Ruim. Acho que você não tem tudo isso de memória\n");
17    }
18    free(apontaHeap);
19
20    return 0;
21 }
```

Figure 5: Código utilizado



```
lusca@LuscaPc: ~/Área de Trabalho/lcc/boom
lusca@LuscaPc:~/Área de Trabalho/lcc/boom$ make estouro
cc  estouro.c  -o estouro
lusca@LuscaPc:~/Área de Trabalho/lcc/boom$ ./estouro
Eae, chefe. Vai alocar quantos mbs na Heap hoje?
2000
Tudo Certo! 2000 mbs alocados com sucesso
lusca@LuscaPc:~/Área de Trabalho/lcc/boom$
```

```
lusca@LuscaPc: ~/Área de Trabalho/lcc/boom
lusca@LuscaPc:~/Área de Trabalho/lcc/boom$ free -m
             total        usada       livre      compart.  buff/cache  disponível
Mem.:          6975          3527         856           91         2592         3094
Swap:          2047           0         2047
```

Figure 6: Alocando 2gbs

```
lusca@LuscaPc: ~/Área de Trabalho/icc/boom
lusca@LuscaPc:~/Área de Trabalho/icc/boom$ ./estouro
Eae, chefe. Vai alocar quantos mbs na Heap hoje?
8000
Deu Ruim. Acho que você não tem tudo isso de memória
lusca@LuscaPc:~/Área de Trabalho/icc/boom$

lusca@LuscaPc: ~/Área de Trabalho/icc/boom
lusca@LuscaPc:~/Área de Trabalho/icc/boom$ free -m
Men.:      total      usada      livre   compart.  buff/cache  disponível
Swap:      2047         0        2047
lusca@LuscaPc:~/Área de Trabalho/icc/boom$ free -m
Men.:      total      usada      livre   compart.  buff/cache  disponível
Swap:      2047         0        2047
```

Figure 7: Tentando alocar todos os 8gbs