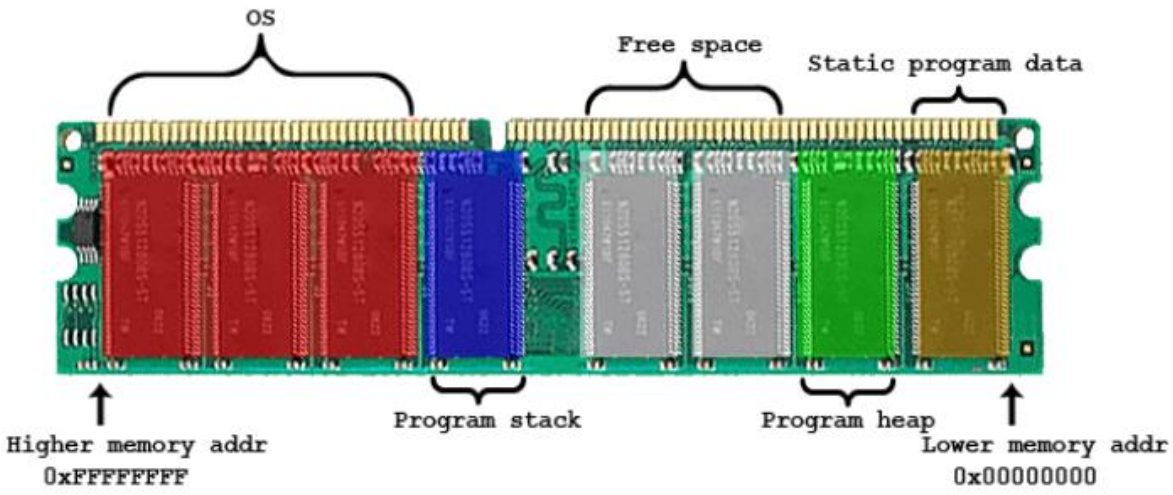
Heap and Stack memory

05/23/94

Introduction •

Virtual memory in a computer is a large long array of bits and these bits are divided into blocks called bytes (every 8 .bits = 1 byte) and each byte is assigned an address for access

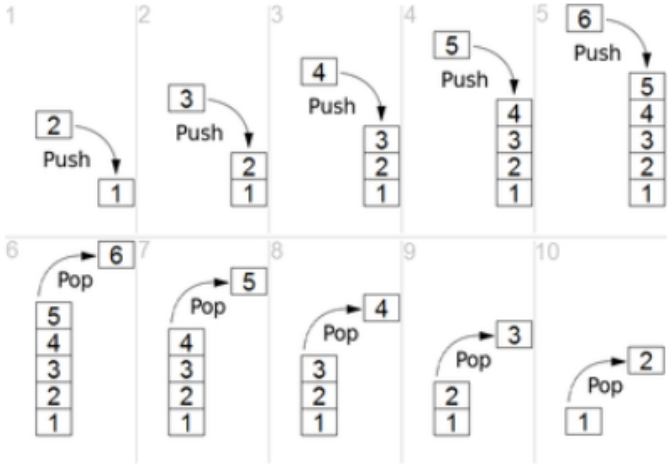
In programming languages, when you work with non-physical data (such as variables, objects, functions, etc.), the value and address of these data are stored in **virtual memory .** At a lower level, when you define and use a local variable or a function, their values and addresses are placed in the **Stack section of the virtual memory.** But by creating the object or allocating the memory manually (Dynamic), their value and address are placed in **the Heap part of the virtual memory.** And finally, all of these are arranged in the RAM cells of the computer hardware in an .orderly manner



:Table of contents

Stack memory •

Stack memory is like a bunch of plates that are placed on top of each other... whenever you want to put a plate on the pile of plates (push), you put it on top. In the same way, whenever you want to remove a plate from this category . (Pop), you always remove the top plate... This category is called Stack or **LIFO** or **LastIn-FirstOut**



The Stack memory is located in the user-space part of the memory and is automatically managed by **the CPU .** This memory is the storage place for non-static local variables, function parameters and function return addresses, which are accumulated inside this memory as LIFO (Last In First Out) and are known as **Frame** . When a function is called and executed, the function, its parameters and all its internal non-static variables are placed in the Stack memory. By calling the new function, this function is placed **on top of** the previous function, and when the work of one of these functions is finished, that function is removed from the Stack memory along with all the relevant variables... and the .work continues in the same way

.**Note:** Another name for this type of memory allocation is Static memory allocation

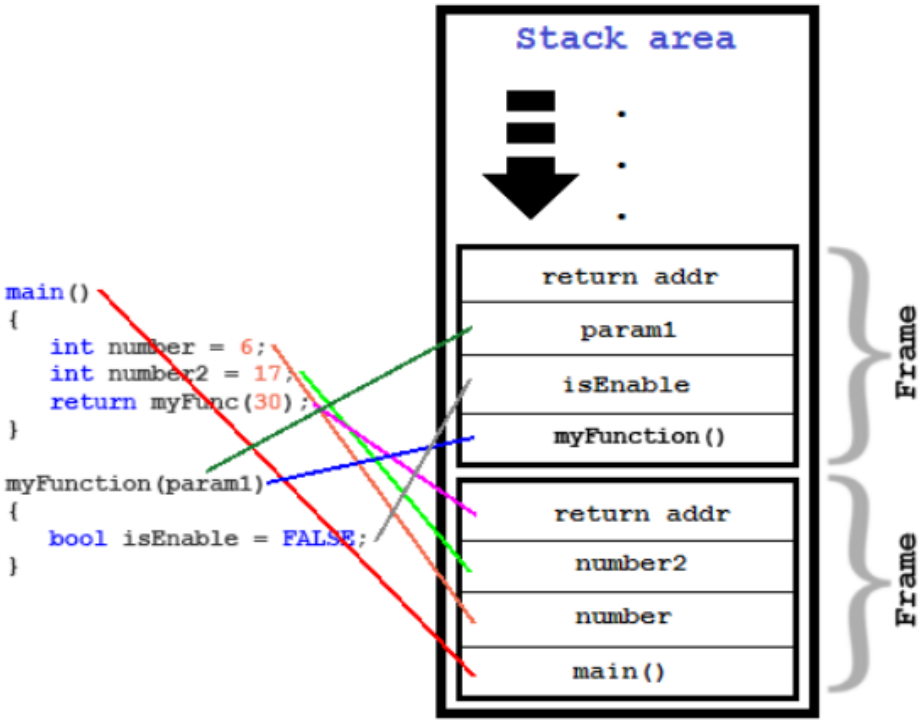
Note 2: The memory that is talked about in Memory Firewall software, this part of Stack and Heap is virtual memory .and not physical memory/RAM

Note 3: In some programming languages such as Cpp, you can store the object in the Stack memory as well, .provided that it is inside the function and you do not use new

:Example

```
function myFunction
    ( param1 , param2 ) // To Stack memory {
   _testVar = param1 + param2 ; // To stack memory return _testVar ; // To Stack mem
```

:Image example



- **Stack memory rules**

.In the virtual memory of the device, the Stack section is located .1

It is the place to store local variables (non-static), parameters of functions, return addresses of functions. *(also* .2
tracking nested functions)

.The stack size of the program is determined and allocated during compilation .3

. It is responsive to the problem of Stackoverflow .4

.It is automatically managed by the CPU .5

When the function is called, it is pushed into the Stack memory and when the function is finished, it is Popped .6
.from the Stack memory

.In terms of speed in occupying space, it is faster than Heap memory .7

. language , classes and structures are stored in Stack memory if pointers are not used In C/Cpp .8

.A stackoverflow error can occur during unwise allocation/occupation or over capacity .9

Each program has a main thread and each thread has a private Stack memory. So, as long as the thread is not .10
closed, the stack is still there. (closing here means exit and not terminate)

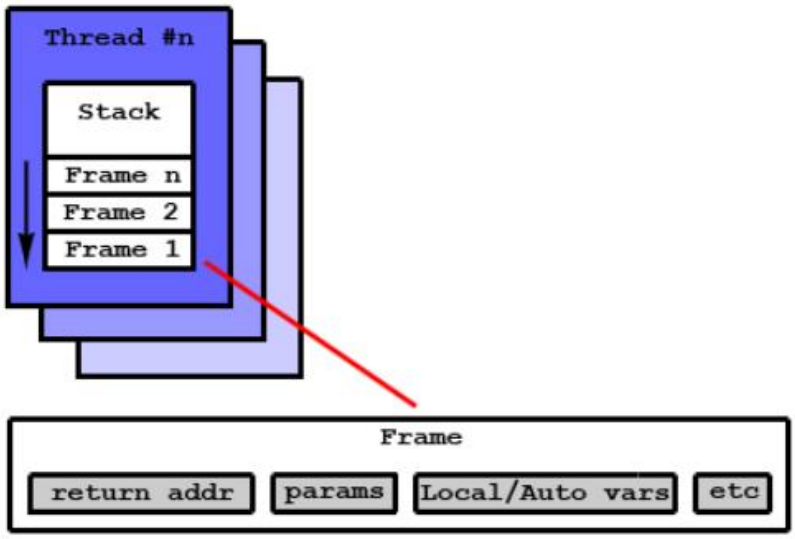
.The Stack memory area of the program is attached to the thread of that program .11

The data in the Stack memory is sequentially placed on top of each other. (with the LIFO rule) .12

.As stack memory consumption increases, less memory is left for heap .13

.This type of memory is readable and writable .14

...And .15



Stackoverflow problem •

The word stackoverflow literally means stack overflow. Among the reasons for the stackoverflow error, we can mention the depth of nested and chain functions, bulky local variables, excessive stack size, destruction or corruption of part of .the memory, wrong use of Native API

.**Note:** Stackoverflow problems are errors and not exceptions

:By running the following codes, you will get a better understanding of Stack memory and the Stackoverflow problem

:PHP example

```
<? php
 function myInfiniteRecursion () {
   myInfiniteRecursion (); }


myInfiniteRecursion ();
```

```
PHP Fatal error: Allowed memory size of 134217728 bytes(134MB) exhausted (tried to
allocate 130968 bytes) in develop.php on line 7.
```

In this example, **every time** the function is called `myInfiniteRecursion()`, a **Stack Frame** is created for that function in the Stack memory, and this process continues as long as the PHP engine (memory_limit value in php.ini) allows... Finally, the allowed capacity of the engine's Stack memory PHP fills up and the script tries to access beyond the capacity of the Stack memory, but the system does not allow it and stops the script with the .stackoverflow/0xC00000FD error

:Java example

```
class MyClass { private MyClass myCls = new MyClass ();

    public static void main ( String [] args ) { new MyClass (); } } // javac MyClas
```

```
Exception in thread "main" java.lang.StackOverflowError
```

:C# example

```
class Program { static void Recursive () { Recursive (); }

   static void Main () { Recursive (); } }
```

System. **StackOverflow** Exception: 'Exception of type '' was thrown.'

:Example C

```c
int main () { int large [ 10000000 ] = { 0 }; return 0 ; }
```

```
Linux: Segmentation fault.
```

```
Windows: Unhandled exception in develop.exe: 0xC00000FD: Stackoverflow.
```

;You can see that the words stack and overflow are common in all errors

Windows tip: You can change the stack size of executable programs with the **editbin program available in VS .and nasm32**

.**Note:** The buffer overflow error is not related to the stack overflow error and these two are different from each other

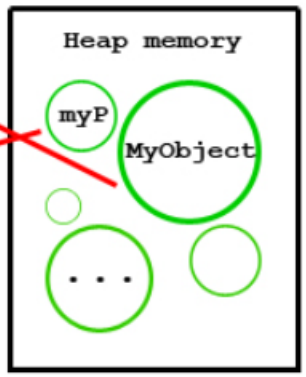
Heap memory •

Heap memory is located in the user-space part of the virtual memory and is manually managed **by the programmer . In fact, data is stored in this memory by the programmer and must** be emptied/destroyed by the programmer himself . The data stored in the Heap memory will not be lost when the call is completed and will remain until the programmer deletes this data from the Heap memory and even cause a memory leak or OutOfMemory. .Unless they are detected and destroyed by Garbage Collector

Note: Here we mean data, objects that `new`are created by, variables that `calloc/alloc/malloc` are defined by, . and values that are moved by pointers

.**Note 2:** Another name for this type of memory allocation is Dynamic memory allocation

```
int myfunction()
{
   int i;

   MyObject object = new MyObject();

   int *myP = (int*)malloc(10);

   return i
}
```



Heap memory size is determined relative to the size of the total data (mentioned above) and the operating system .gives heap space to the program as much as it can and the user is not limited and has free space in RAM

Note: In scripting languages, Heap memory management is done by **the interpreter** and in VMs by the runtime .language

- **Heap memory rules**

1. The Heap section is stored in the device's virtual memory.
2. It is the place to store allocated objects and data.
3. Heap memory is responsible for memory leak and OutOfMemory problems.
4. In terms of speed in occupying space, it is slower than Stack memory because it is managed by pointers.
5. In Cpp, the data stored in the Heap memory is called by the pointer and can be reoccupied.
6. **OutOfMemory can occur when allocation/reservation of memory is unwise.**
7. The data in this space are randomly placed together.
8. Heap memory area depends on the program runtime. That is, it is occupied by starting the program and emptied by closing the program.
9. Heap memory size of the program is reserved/allocated during runtime/starting the program.
10. all threads use only one Heap memory (in most languages).
11. As the Heap memory consumption increases, less memory is left for the stack.
12. When the program is closed, the allocated space in Heap is emptied by the program.
13. And...

Note: The problem of OutOfMemory belongs to the category of Errors and not Exceptions.

PHP example:

```
$objTest = new MyObject (); // MyObject to Heap memory
```

Java and C# example:

```
MyObject objTest = new MyObject (); // to Heap memory
```

Example C:

```
int * ptr = new int ; // 4 bytes in the Heap memory
```

In the first and second examples, by creating the MyObject object, we occupied some Heap memory, in fact, we put the MyObject object into the **Heap** memory and not its variable. In the third example, we put 4 bytes with the variable `ptr` into `malloc` Heap memory **,** which **we have to** free and destroy later by coding . But this is only possible in Cpp/C and low-level languages, and not in Java, PHP and high-level languages!

So what is the task of memory emptying in high-level languages? Because the memory has a maximum capacity and only a small part of it is available to our program...

The answer to this problem is **Garbage Collector** ! The garbage collection system is designed for high-level languages to cover this deficiency to some extent.

Windows tip: You can change the heap size of executable programs with the **editbin program available in VS and nasm32.**

- **Garbage Collector / GC**

Garbage collector system (abbreviated GC) is a feature that [usually] runs automatically and starts checking objects that are no longer used by the program (they are dead) and no references point to them (unreferenced **)** . Then he .destroys them

.This is the same **2-phase process** that **every garbage collector** performs to empty the memory

:Garbage collector usually comes into action when

.The read/write operation is done in the system memory .1
.The program/script is finished and wants to be closed .2
.Manually mute the sound (such as IDEs Netbeans, Eclipse, Java language, PHP5.3, etc.) .3

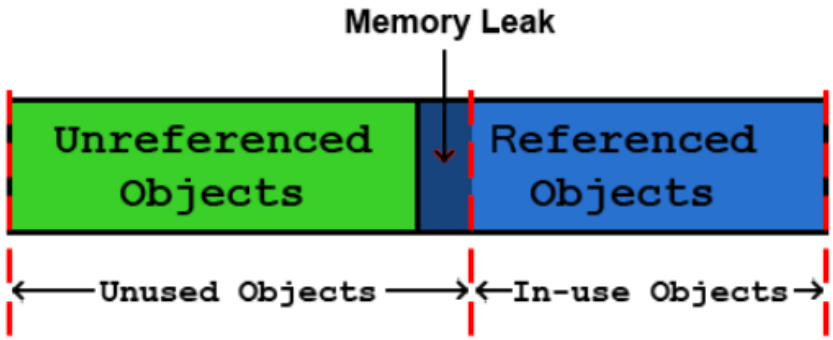
.**Note:** Whether or not objects are collected by the garbage collector has **nothing to do** with the issue of scope

As mentioned, VMs and interpreters store objects created by the running application in the heap memory. Like objects that are made by new. When the Heap is slow, the VM or the interpreter starts the garbage collection system; Then the garbage collection process is executed in a circular manner and deletes the objects and data that can be collected .from the Heap memory... which, of course, slows down the system a little until its operation is finished

Memory leak •

In short, it means: the objects are no longer used by the program, they are referenced somewhere, and the Garbage .Collector is not able to release (destroy) them

In other words: keeping built objects busy (in use) **for no reason , which is equal to** being ignored by the garbage collector . (According to GC law)



.**Note:** memory leaks are critical bugs and app-vulnerabilities, which originate from programmer's wrong coding

:Other definitions

> a failure in a program to release discarded memory, causing impaired performance or failure.
> Failure to release unreachable memory, which can no longer be allocated again by any process during the execution of allocating process.
> Definition: Failure to release memory after allocation.

Note 2: Garbage collector system is not an insurance to fix such vandalism and wrong coding, but it is only an .auxiliary system

:Example of memory leak in PHP

```
class MyClass { // ... }



$obj = new MyClass ;
$obj -> self = $obj ; // Memory leak by $obj
```

:Example of memory leak in C

```
int main () { char * myVar = new char [ 5 ]; // Got 5 bytes in Heap memory. return 0
```

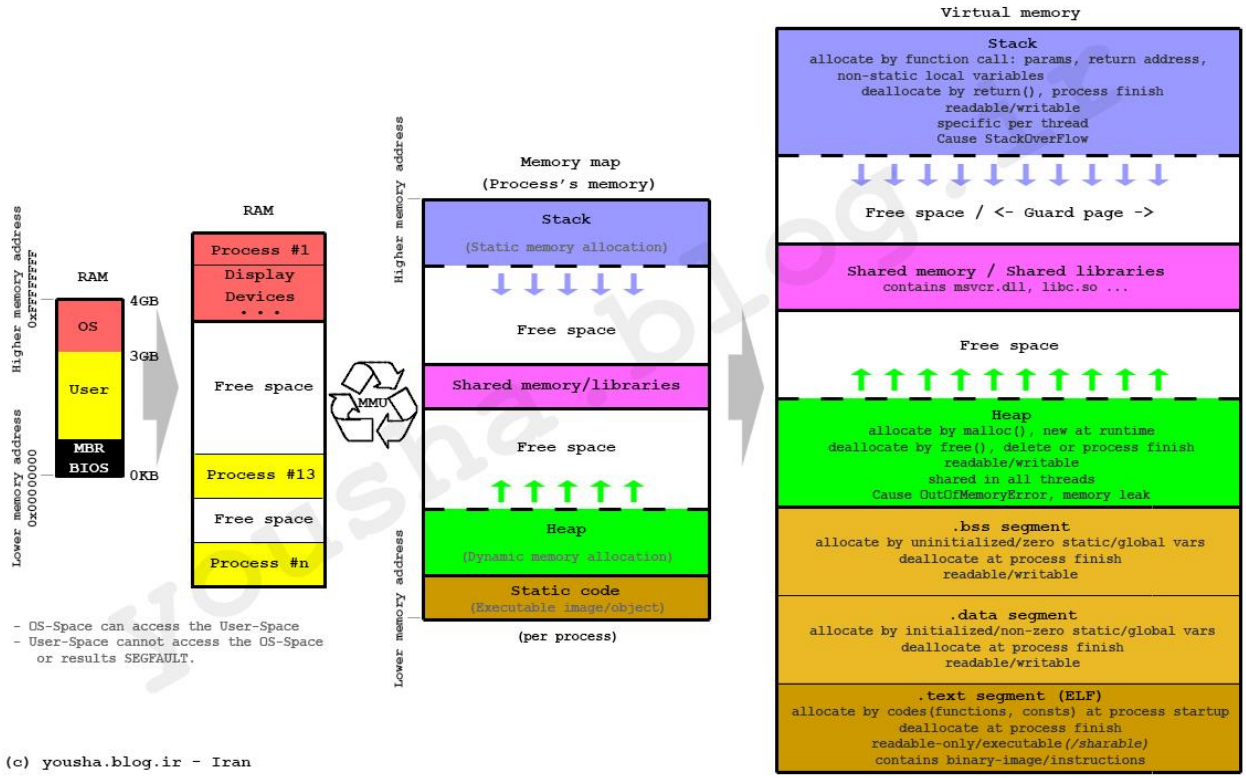
:Example of memory leak in Android Java

```
@Override protected void onCreate ( Bundle state ) { super . onCreate ( state ); Tex
    backgroundImage = getDrawable ( R . drawable . large_bitmap );
    myTextView . setBackgroundDrawable ( backgroundImage );
    setContentView ( myTextView );



        // Memory leak }
```

In this Java example, memory leak occurs 2 times, 1- in the current state because the myTextView object variable is kept busy by the activity, 2- in the state that the orientation of the device changes, so the onCreate method is called .again

.Of course, the most sensitive memory leak is in systems that are facing limited resources or service type programs

Full description of stack, heap, data of each process in virtual memory



Please note, these are images from the website that is made for ease share. The original author of the above post is Yusha Al-Ayoub.

Link for the website: https://yousha.blog.ir/post/45