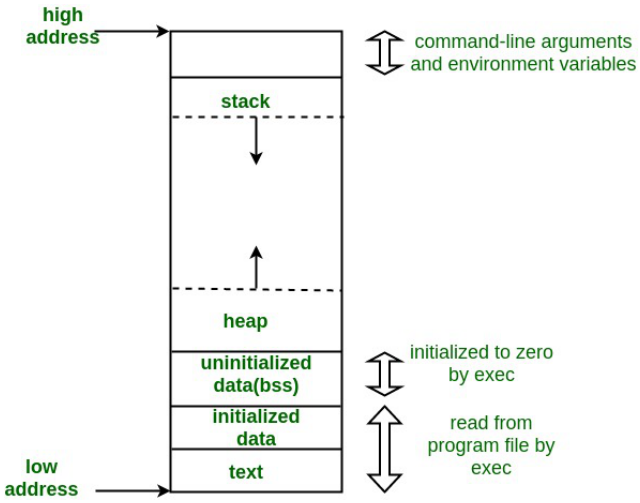
perm_identity

Memory Layout of C Programs

A typical memory representation of C program consists of following sections.

1. Text segment
2. Initialized data segment
3. Uninitialized data segment
4. Stack
5. Heap



A typical memory layout of a running process

1. Text Segment:

A text segment , also known as a code segment or simply as text, is one of the

sections of a program in an object file or in memory, which contains executable instructions.

As a memory region, a text segment may be placed below the heap or stack in order to prevent heaps and stack overflows from overwriting it.

Usually, the text segment is sharable so that only a single copy needs to be in memory for frequently executed programs, such as text editors, the C compiler, the shells, and so on. Also, the text segment is often read-only, to prevent a program from accidentally modifying its instructions.

2. Initialized Data Segment:
Initialized data segment, usually called simply the Data Segment. A data segment is a portion of virtual address space of a program, which contains the global variables and static variables that are initialized by the programmer.

Note that, data segment is not read-only, since the values of the variables can be altered at run time.

This segment can be further classified into initialized read-only area and initialized read-write area.

For instance the global string defined by char s[] = “hello world” in C and a C statement like int debug=1 outside the main (i.e. global) would be stored in initialized read-write area. And a global C statement like const char*

calloc() versus malloc()

How does free() know the size of memory to be deallocated?

Use of realloc()

Check if given Preorder, Inorder and Postorder traversals are of same tree | Set 2

Difference between pointer to an array and array of pointers

Count substrings that contain all vowels | SET 2

C program to sort an array using pointers

Basic Code Optimizations in C

string = “hello world” makes the string literal “hello world” to be stored in initialized read-only area and the character pointer variable string in initialized read-write area.

Ex: static int i = 10 will be stored in data segment and global int i = 10 will also be stored in data segment

3. Uninitialized Data Segment:
Uninitialized data segment, often called the “bss” segment, named after an ancient assembler operator that stood for “block started by symbol.” Data in this segment is initialized by the kernel to arithmetic 0 before the program starts executing

uninitialized data starts at the end of the data segment and contains all global variables and static variables that are initialized to zero or do not have explicit initialization in source code.

For instance a variable declared static int i; would be contained in the BSS segment.
For instance a global variable declared int j; would be contained in the BSS segment.

4. Stack:
The stack area traditionally adjoined the heap area and grew the opposite direction; when the stack pointer met the heap pointer, free memory was exhausted. (With modern large address spaces and virtual memory techniques they may be placed almost anywhere, but they still typically grow opposite directions.)

The stack area contains the program stack, a LIFO structure, typically located in the higher

parts of memory. On the standard PC x86 computer architecture it grows toward address zero; on some other architectures it grows the opposite direction. A “stack pointer” register tracks the top of the stack; it is adjusted each time a value is “pushed” onto the stack. The set of values pushed for one function call is termed a “stack frame”; A stack frame consists at minimum of a return address.

Stack, where automatic variables are stored, along with information that is saved each time a function is called. Each time a function is called, the address of where to return to and certain information about the caller’s environment, such as some of the machine registers, are saved on the stack. The newly called function then allocates room on the stack for its automatic and temporary variables. This is how recursive functions in C can work. Each time a recursive function calls itself, a new stack frame is used, so one set of variables doesn’t interfere with the variables from another instance of the function.

5. Heap:
Heap is the segment where dynamic memory allocation usually takes place.

The heap area begins at the end of the BSS segment and grows to larger addresses from there.The Heap area is managed by malloc, realloc, and free, which may use the brk and sbrk system calls to adjust its size (note that the use of brk/sbrk and a single “heap area”

is not required to fulfill the contract of malloc/realloc/free; they may also be implemented using mmap to reserve potentially non-contiguous regions of virtual memory into the process' virtual address space). The Heap area is shared by all shared libraries and dynamically loaded modules in a process.

Examples.

The size(1) command reports the sizes (in bytes) of the text, data, and bss segments. (for more details please refer man page of size(1))

1. Check the following simple C program

filter_none
edit
play_arrow
brightness_4

```
#include <stdio.h>

int main(void)
{
    return 0;
}
```

```
[narendra@CentOS]$ gcc memory-
layout.c -o memory-layout
[narendra@CentOS]$ size memory-
layout
text       data        bss
dec        hex    filename
960        248          8       1216
4c0    memory-layout
```

2. Let us add one global variable in program,

now check the size of bss (highlighted in red color).

filter_none
edit
play_arrow
brightness_4

```
#include <stdio.h>

int global; /* Uninitialized variable
stored in bss*/

int main(void)
{
    return 0;
}
```

```
[narendra@CentOS]$ gcc memory-
layout.c -o memory-layout
[narendra@CentOS]$ size memory-
layout
text       data        bss
dec        hex    filename
 960        248         12
1220        4c4    memory-layout
```

3. Let us add one static variable which is also stored in bss.

filter_none
edit
play_arrow
brightness_4

```
#include <stdio.h>

int global; /* Uninitialized variable
stored in bss*/

int main(void)
```

```
{
    static int i; /* Uninitialized
static variable stored in bss */
    return 0;
}
```

```
[narendra@CentOS]$ gcc memory-
layout.c -o memory-layout
[narendra@CentOS]$ size memory-
layout
text       data        bss
dec        hex    filename
 960        248         16
1224        4c8    memory-layout
```

4. Let us initialize the static variable which will then be stored in Data Segment (DS)

filter_none

edit

play_arrow

brightness_4

```
#include <stdio.h>

int global; /* Uninitialized variable
stored in bss*/

int main(void)
{
    static int i = 100; /* Initialized
static variable stored in DS*/
    return 0;
}
```

```
[narendra@CentOS]$ gcc memory-
layout.c -o memory-layout
[narendra@CentOS]$ size memory-
layout
text       data        bss
dec        hex    filename
960         252         12
```

```
1224        4c8    memory-layout
```

5. Let us initialize the global variable which will then be stored in Data Segment (DS)

filter_none
edit
play_arrow
brightness_4

```
#include <stdio.h>

int global = 10; /* initialized global
variable stored in DS*/

int main(void)
{
    static int i = 100; /* Initialized
static variable stored in DS*/
    return 0;
}
```

```
[narendra@CentOS]$ gcc memory-
layout.c -o memory-layout
[narendra@CentOS]$ size memory-
layout
text       data        bss
dec        hex    filename
960         256          8
1224        4c8    memory-layout
```

This article is compiled by **Narendra Kangralkar**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Source:

http://en.wikipedia.org/wiki/Data_segment

http://en.wikipedia.org/wiki/Code_segment

http://en.wikipedia.org/wiki/.bss

http://www.amazon.com/Advanced-Programming-UNIX-Environment-

2nd/dp/0201433079

Recommended Posts:

Common Memory/Pointer Related bug in C Programs

Output of C programs | Set 63

C/C++ Tricky Programs

Error Handling in C programs

Output of C programs | Set 30 (Switch Case)

Programs to print Interesting Patterns

How to Compile and Run C/C++/Java Programs in Linux

8085 programs to find 2's compliment with carry | Set 2

IPC through shared memory

Facts and Question related to Style of writing programs in C/C++

Memory leak in C++ and How to avoid it?

What is Memory Leak? How can we avoid?

How to deallocate memory without using free() in C?

C | Dynamic Memory Allocation | Question 6

C | Dynamic Memory Allocation | Question 5

Article Tags : C C-Dynamic Memory Allocation system-programming

Practice Tags : C

thumb_up

76

2.6

To-do Done Based on **171** vote(s)

Feedback/ Suggest Improvement

Add Notes Improve Article

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Previous
first_page Hiding of all overloaded methods with same name in base class

Next
Templates and Default Arguments last_page

Most popular in C

- OpenMP | Hello World program
- Types of Literals in C/C++ with Examples
- C Program to count the Number of Characters in a File

dot (.) operator in C/C++

Arrow operator -> in C/C++ with Examples

More related articles in C

Difference between C and C#

Difference between while and do-while loop in C, C++, Java

AKTU (UPTU) Previous Year Solved Papers | C Programming

OpenMP | Introduction with Installation Guide

Logical Not ! operator in C with Examples