Memory Layout of a C Program

Historically a C program has been composed of the following pieces:

- Text segment. This is the machine instructions that are executed by the CPU. Usually the text segment is sharable so that only a single copy needs to be in memory for frequently executed programs (text editors, the C compiler, the shells, etc.). Also, the text segment is often read-only, to prevent a program from accidentally modifying its instructions.
- Initialized data segment. This is usually just called the data segment and it contains variables that are specifically initialized in the program. For example, the C declaration

```
int    maxcount = 99;
```

  appearing outside any function causes this variable to be stored in the initialized data segment with its initial value.
- Uninitialized data segment. This segment is 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 0 before the program starts executing. The C declaration

```
long   sum[1000];
```

  appearing outside any function causes this variable to be stored in the uninitialized data segment.
- Stack. This is 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) is saved on the stack. The newly called function then allocates room on the stack for its automatic and temporary variables. By utilizing a stack in this fashion, C functions can be recursive.
- Heap. Dynamic memory allocation usually takes place on the heap. Historically the heap has been located between the top of the uninitialized data and the bottom of the stack.

Figure shows the typical arrangement of these segments. This is a logical picture of how a program looks—there is no requirement that a given implementation arrange its memory in this fashion. Nevertheless, this gives us a typical arrangement to describe.

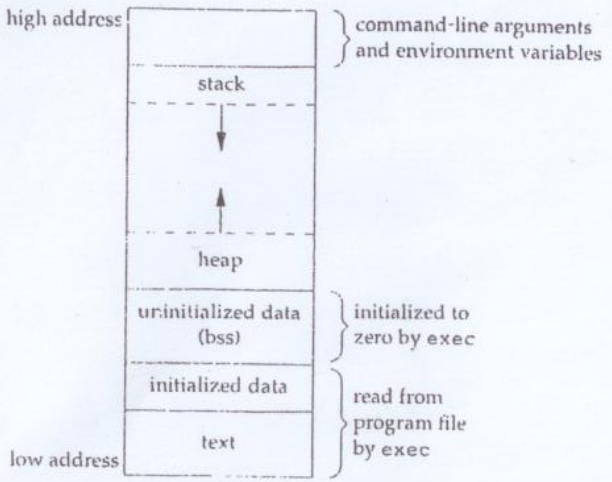

Figure Typical memory arrangement.

With 4.3+BSD on a VAX, the text segment starts at location 0 and the top of the stack starts just below `0x7fffffff`. On the VAX the unused virtual address space between the top of the heap and the bottom of the stack is large.

Note from Figure that the contents of the uninitialized data segment are not stored in the program file on disk. This is because the kernel sets it to 0 before the program starts running. The only portions of the program that need to be saved in the program file are the text segment and the initialized data.

The `size`(1) command reports the sizes in bytes of the text, data, and bss segments. For example

```
$ size /bin/cc /bin/sh
text    data    bss     dec     hex
81920   16384   664     98968   18298   /bin/cc
90112   16384   0       106496  1a000   /bin/sh
```

The fourth and fifth columns are the total of the sizes in decimal and hexadecimal.

<Source: Advanced Programming in the UNIX Environment, W. Richard Stevens
Addison Wesley, 1992, PP 167-168