# Linux C程序基本构成

## 1.1 未运行的C程序构成

下图为一个C语言可执行程序的基本情况。

**[**root@localhost Ctest**]**# ls test **-**l     //test为一个可执行程序

**-**rwxr**-**xr**-**x  1 root root 4868 Mar 26 08**:**10 test

**[**root@localhost Ctest**]**# file test //此文件基本情况

test**:** ELF 32**-**bit LSB executable**,** Intel 80386**,** version 1 **(**SYSV**),**

**for** GNU**/**Linux 2.2.5**,** dynamically linked **(**uses shared libs**),** **not** stripped

**[**root@localhost Ctest**]**# size test  //此二进制可执行文件结构情况

//代码区静态数据/全局初始化数据区　未初始化数据区　十进制总和　十六进制总和 文件名

text   data         bss    dec    hex filename

906    284          4    1194    4aa test

可以看出，此可执行程序在存储时（没有调入到内存）分为代码区（text）、数据区（data）和未初始化数据区（bss）3个部分。

**(1)代码区（text segment）**。存放CPU执行的机器指令（machine instructions）。通常，代码区是可共享的（即另外的执行程序可以调用它），因为对于频繁被执行的程序，只需要在内存中有一份代码即可。代码区通常是只读的，使其只读的原因是防止程序意外地修改了它的指令。另外，代码区还规划了局部变量的相关信息。

**(2)全局初始化数据区/静态数据区**（initialized data segment/data segment）。该区包含了在程序中明确被初始化的全局变量、静态变量（包括全局静态变量和局部静态变量）和常量数据（如字符串常量）。

例如，一个不在任何函数内的声明（全局数据）：

int   maxcount **=** 99**;**

使得变量maxcount根据其初始值被存储到初始化数据区中。

static mincount**=**100**;**

这声明了一个静态数据，如果是在任何函数体外声明，则表示其为一个全局静态变量，如果在函数体内（局部），则表示其为一个局部静态变量。另外，如果在函数名前加上static，则表示此函数只能在当前文件中被调用。

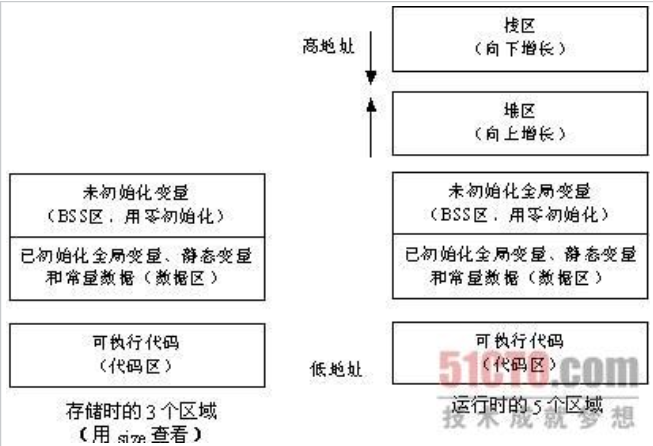
**(3)未初始化数据区**。亦称BSS区（uninitialized data segment），存入的是全局未初始化变量。BSS这个叫法是根据一个早期的汇编运算符而来，这个汇编运算符标志着一个块的开始。BSS区的数据在程序开始执行之前被内核初始化为0或者空指针（NULL）。例如一个不在任何函数内的声明：

long sum**[**1000**];**

将变量sum存储到未初始化数据区。

## 1.2 运行的C程序基本构成

下图所示为可执行代码存储时结构和运行时结构的对照图。一个正在运行着的C编译程序占用的内存分为代码区、初始化数据区、未初始化数据区、堆区和栈区5个部分。



**(1)代码区（text segment）**。代码区指令根据程序设计流程依次执行，对于顺序指令，则只会执行一次（每个进程），如果反复，则需要使用跳转指令，如果进行递归，则需要借助栈来实现。

代码区的指令中包括操作码和要操作的对象（或对象地址引用）。如果是立即数（即具体的数值，如5），将直接包含在代码中；如果是局部数据，将在栈区分配空间，然后引用该数据地址；如果是BSS区和数据区，在代码中同样将引用该数据地址。

**(2)全局初始化数据区/静态数据区**（Data Segment）。只初始化一次。

**(3)未初始化数据区（BSS）**。在运行时改变其值。

**(4)栈区（stack）**。由编译器自动分配释放，存放函数的参数值、局部变量的值等。其操作方式类似于数据结构中的栈。每当一个函数被调用，该函数返回地址和一些关于调用的信息，比如某些寄存器的内容，被存储到栈区。然后这个被调用的函数再为它的自动变量和临时变量在栈区上分配空间，这就是C实现函数递归调用的方法。每执行一次递归函数调用，一个新的栈框架就会被使用，这样这个新实例栈里的变量就不会和该函数的另一个实例栈里面的变量混淆。

**(5)堆区（heap）**。用于动态内存分配。堆在内存中位于bss区和栈区之间。一般由程序员分配和释放，若程序员不释放，程序结束时有可能由OS回收。

之所以分成这么多个区域，主要基于以下考虑：

一个进程在运行过程中，代码是根据流程依次执行的，只需要访问一次，当然跳转和递归有可能使代码执行多次，而数据一般都需要访问多次，因此单独开辟空间以方便访问和节约空间。

临时数据及需要再次使用的代码在运行时放入栈区中，生命周期短。

全局数据和静态数据有可能在整个程序执行过程中都需要访问，因此单独存储管理。

堆区由用户自由分配，以便管理。

下面通过一段简单的代码来查看C程序执行时的内存分配情况。相关数据在运行时的位置如注释所述。

//main.cpp

int a **=** 0**;**    //a在全局已初始化数据区

char **\***p1**;**    //p1在BSS区（未初始化全局变量）

main**()**

**{**

int b**;**    //b在栈区

char s**[]** **=** "abc"**;** //s为数组变量，存储在栈区，

//"abc"为字符串常量，存储在已初始化数据区

char **\***p1，p2**;**  //p1、p2在栈区

char **\***p3 **=** "123456"**;** //123456\0在已初始化数据区，p3在栈区

static int c **=**0；  //C为全局（静态）数据，存在于已初始化数据区

//另外，静态数据会自动初始化

p1 **=** **(**char **\*)**malloc**(**10**);**//分配得来的10个字节的区域在堆区

p2 **=** **(**char **\*)**malloc**(**20**);**//分配得来的20个字节的区域在堆区

free**(**p1**);**

free**(**p2**);**

**}**