林炳文Evankaka的专栏

在程序的世界里遨游

- 目录视图
- 摘要视图
- 订阅

栈，堆，全局，文字常量，代码区总结

标签： 栈堆全局文字常量代码区总结
2015-03-19 14:55 10016人阅读 评论(8) 收藏 举报
分类：
C/C++学习笔记 *(16)*
作者同类文章*X*

目录(?) [+]

在C\C++中，通常可以把内存理解为4个分区：栈、堆、全局/静态存储区和常量存储区。下面我们分别简单地介绍一下各自的特点。

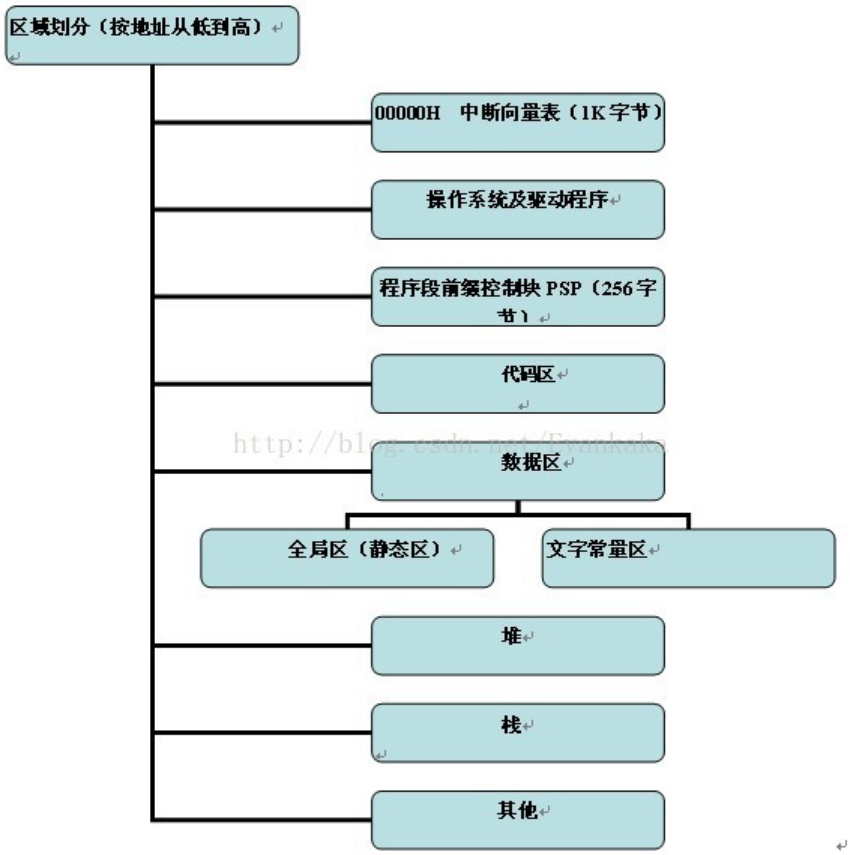
一. 区域划分

堆：　是大家共有的空间，分全局堆和局部堆。全局堆就是所有没有分配的空间，局部堆就是用户分配的空间。堆在操作系统对进程初始化的时候分配，运行过程中也可以向系统要额外的堆，但是记得用完了要还给操作系统，要不然就是内存泄漏。

栈：是个线程独有的，保存其运行状态和局部自动变量的。栈在线程开始的时候初始化，每个线程的栈互相独立，因此，栈是thread safe的。每个C　+ +对象的数据成员也存在在栈中，每个函数都有自己的栈，栈被用来在函数之间传递参数。操作系统在切换线程的时候会自动的切换栈，就是切换　S S / E S P寄存器。栈空间不需要在高级语言里面显式的分配和释放。

一个由C/C++编译的程序占用的内存分为以下几个部分
1、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。
2、堆区（heap） ： 一般由程序员分配释放， 若程序员不释放，程序结束时可能由OS回收 。注意它与数据结构中的堆是两回事，分配方式倒是类似于链表。
3、全局区（静态区）（static）：全局变量和静态变量的存储是放在一块的，初始化的全局变量和静态变量在一块区域， 未初始化的全局变量和未初始化的静态变量在相邻的另一块区域， 程序结束后有系统释放 。
4、文字常量区：常量字符串就是放在这里的， 程序结束后由系统释放 。
5、程序代码区：存放函数体的二进制代码。

图示1：可执行程序在存储器中的存放



二．示例代码

```
//main.cpp
int a = 0; 全局初始化区
char *p1; 全局未初始化区
main()
{
int b; 栈
char s[] = "abc"; 栈
char *p2; 栈
char *p3 = "123456"; 123456在常量区，p3在栈上。
static int c =0； 全局（静态）初始化区
p1 = (char *)malloc(10);
p2 = (char *)malloc(20);
分配得来得10和20字节的区域就在堆区。
strcpy(p1, "123456"); 123456放在常量区，编译器可能会将它与p3所指向的"123456"优化成一个地方。
}
```

三．堆和栈

3.1申请方式

stack: 遵循LIFO后进先出的规则，它的生长方向是向下的，是向着内存地址减小的方向增长，栈是系统提供的功能，特点是快速高效，缺点是有限制，数据不灵活。

由系统自动分配。 例如，声明在函数中一个局部变量 int b; 系统自动在栈中为b开辟空间

heap: 对于堆来讲，生长方向是向上的，也就是向着内存地址增加的方向。

需要程序员自己申请，并指明大小，在c中malloc函数

如p1 = (char *)malloc(10);

在C++中用new运算符

如p2 =new char[10];

但是注意p1、p2本身是在栈中的。

3.2 申请后系统的响应

栈：只要栈的剩余空间大于所申请空间，系统将为程序提供内存，否则将报异常提示栈溢出。

堆：首先应该知道操作系统有一个记录空闲内存地址的链表，当系统收到程序的申请时， 会遍历该链表，寻找第一个空间大于所

申请空间的堆结点，然后将该结点从空闲结点链表中删除，并将该结点的空间分配给程序，另外，对于大多数系统，会在这块内存空间中的首地址处记录本次分配的大小，这样，代码中的delete语句才能正确的释放本内存空间。另外，由于找到的堆结点的大小不一定正好等于申请的大小，系统会自动的将多余的那部分重新放入空闲链表中。

3.3申请大小的限制

栈：在Windows下,栈是向低地址扩展的数据结构，是一块连续的内存的区域，这句话的意思是栈顶的地址和栈的最大容量是系统预先规定好的，在 WINDOWS下，栈的大小是2M（也有的说是1M，总之是一个编译时就确定的常数），如果申请的空间超过栈的剩余空间时，将提示overflow。因此，能从栈获得的空间较小。

堆：堆是向高地址扩展的数据结构，是不连续的内存区域。这是由于系统是用链表来存储的空闲内存地址的，自然是不连续的，而链表的遍历方向是由低地址向高地址。堆的大小受限于计算机系统中有效的虚拟内存。由此可见，堆获得的空间比较灵活，也比较大。

3.4申请效率的比较：

栈：由系统自动分配，速度较快。但程序员是无法控制的。

堆：是由new分配的内存，一般速度比较慢，而且容易产生内存碎片,不过用起来最方便.

另外，在WINDOWS下，最好的方式是用VirtualAlloc分配内存，他不是在堆，也不是在栈是直接在进程的地址空间中保留一快内存，虽然用起来最不方便。但是速度快，也最灵活。

3.5堆和栈中的存储内容

栈： 在函数调用时，第一个进栈的是主函数中后的下一条指令（函数调用语句的下一条可执行语句）的地址，然后是函数的各个参数，在大多数的C编译器中，参数是由右往左入栈的，然后是函数中的局部变量。注意静态变量是不入栈的。

当本次函数调用结束后，局部变量先出栈，然后是参数，最后栈顶指针指向最开始存的地址，也就是主函数中的下一条指令，程序由该点继续运行。

堆：一般是在堆的头部用一个字节存放堆的大小。堆中的具体内容有程序员安排。

3.6存取效率的比较

```
char s1[] = "aaaaaaaaaaaaaaa";
char *s2 = "bbbbbbbbbbbbbbbbb";
aaaaaaaaaaa是在运行时刻赋值的；
而bbbbbbbbbbb是在编译时就确定的；
但是，在以后的存取中，在栈上的数组比指针所指向的字符串(例如堆)快。
比如：
#include
void main()
{
char a = 1;
char c[] = "1234567890";
char *p ="1234567890";
a = c[1];
a = p[1];
return;
}
对应的汇编代码
10: a = c[1];
00401067 8A 4D F1 mov cl,byte ptr [ebp-0Fh]
0040106A 88 4D FC mov byte ptr [ebp-4],cl
11: a = p[1];
0040106D 8B 55 EC mov edx,dword ptr [ebp-14h]
00401070 8A 42 01 mov al,byte ptr [edx+1]
00401073 88 45 FC mov byte ptr [ebp-4],al
第一种在读取时直接就把字符串中的元素读到寄存器cl中，而第二种则要先把指针值读到edx中，在根据edx读取字符，显然慢了。
```

3.7小结：

堆和栈的区别可以用如下的比喻来看出：

使用栈就象我们去饭馆里吃饭，只管点菜（发出申请）、付钱、和吃（使用），吃饱了就走，不必理会切菜、洗菜等准备工作和洗碗、刷锅等扫尾工作，他的好处是快捷，但是自由度小。

使用堆就象是自己动手做喜欢吃的菜肴，比较麻烦，但是比较符合自己的口味，而且自由度大。

1、内存分配方面：

堆：一般由程序员分配释放， 若程序员不释放，程序结束时可能由OS回收 。注意它与数据结构中的堆是两回事，分配方式是类似于链表。可能用到的关键字如下：new、malloc、delete、free等等。

栈：由编译器(Compiler)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。

2、申请方式方面：

堆：需要程序员自己申请，并指明大小。在c中malloc函数如p1 = (char *)malloc(10)；在C++中用new运算符，但是注意p1、p2本身是在栈中的。因为他们还是可以认为是局部变量。

栈：由系统自动分配。 例如，声明在函数中一个局部变量 int b；系统自动在栈中为b开辟空间。

3、系统响应方面：

堆：操作系统有一个记录空闲内存地址的链表，当系统收到程序的申请时，会遍历该链表，寻找第一个空间大于所申请空间的堆结点，然后将该结点从空闲结点链表中删除，并将该结点的空间分配给程序，另外，对于大多数系统，会在这块内存空间中的首地址处记录本次分配的大小，这样代码中的delete语句才能正确的释放本内存空间。另外由于找到的堆结点的大小不一定正好等于申请的大小，系统会自动的将多余的那部分重新放入空闲链表中。

栈：只要栈的剩余空间大于所申请空间，系统将为程序提供内存，否则将报异常提示栈溢出。

4、大小限制方面：

堆：是向高地址扩展的数据结构，是不连续的内存区域。这是由于系统是用链表来存储的空闲内存地址的，自然是不连续的，而链表的遍历方向是由低地址向高地址。堆的大小受限于计算机系统中有效的虚拟内存。由此可见，堆获得的空间比较灵活，也比较大。

栈：在Windows下,栈是向低地址扩展的数据结构，是一块连续的内存的区域。这句话的意思是栈顶的地址和栈的最大容量是系统预先规定好的，在WINDOWS下，栈的大小是固定的（是一个编译时就确定的常数），如果申请的空间超过栈的剩余空间时，将提示overflow。因此，能从栈获得的空间较小。

5、效率方面：

堆：是由new分配的内存，一般速度比较慢，而且容易产生内存碎片，不过用起来最方便，另外，在WINDOWS下，最好的方式是用VirtualAlloc分配内存，他不是在堆，也不是在栈是直接在进程的地址空间中保留一快内存，虽然用起来最不方便。但是速度快，也最灵活。

栈：由系统自动分配，速度较快。但程序员是无法控制的。

6、存放内容方面：

堆：一般是在堆的头部用一个字节存放堆的大小。堆中的具体内容有程序员安排。

栈：在函数调用时第一个进栈的是主函数中后的下一条指令（函数调用语句的下一条可执行语句）的地址然后是函数的各个参数，在大多数的C编译器中，参数是由右往左入栈，然后是函数中的局部变量。 注意: 静态变量是不入栈的。当本次函数调用结束后，局部变量先出栈，然后是参数，最后栈顶指针指向最开始存的地址，也就是主函数中的下一条指令，程序由该点继续运行。

7、存取效率方面：

堆：char *s1 = "Hellow Word"；是在编译时就确定的；

栈：char s1[] = "Hellow Word"； 是在运行时赋值的；用数组比用指针速度要快一些，因为指针在底层汇编中需要用edx寄存器中转一下，而数组在栈上直接读取。

在C++中，内存分成5个区，他们分别是堆、栈、自由存储区、全局/静态存储区和常量存储区。 栈，就是那些由编译器在需要的时候分配，在不需要的时候自动清楚的变量的存储区。里面的变量通常是局部变量、函数参数等。 堆，就是那些由new分配的内存块，他们的释放编译器不去管，由我们的应用程序去控制，一般一个new就要对应一个delete。如果程序员没有释放掉，那么在程序结束后，操作系统会自动回收。 自由存储区，就是那些由malloc等分配的内存块，他和堆是十分相似的，不过它是用free来结束自己的生命的。 全局/静态存储区，全局变量和静态变量被分配到同一块内存中，在以前的C语言中，全局变量又分为初始化的和未初始化的，在C++里面没有这个区分了，他们共同占用同一块内存区。 常量存储区，这是一块比较特殊的存储区，他们里面存放的是常量，不允许修改（当然，你要通过非正当手段也可以修改，而且方法很多）

四 总结

根据上面的内容，分别将栈和堆、全局/静态存储区和常量存储区进行对比，结果如下。

表1 栈和堆的对比

	栈	堆
存储内容	局部变量	变量
作用域	函数作用域、语句块作用域	函数作用域、语句块作用域
编译期间大小是否确定	是	否
大小	1MB	4GB
内存分配方式	地址由高向低减少	地址由低向高增加
内容是否可以修改	是	是

表2 全局/静态存储区和常量存储区的对比

	全局/静态存储区	常量存储区
存储内容	全局变量、静态变量	常量
编译期间大小是否确定	是	是
内容是否可以修改	是	否

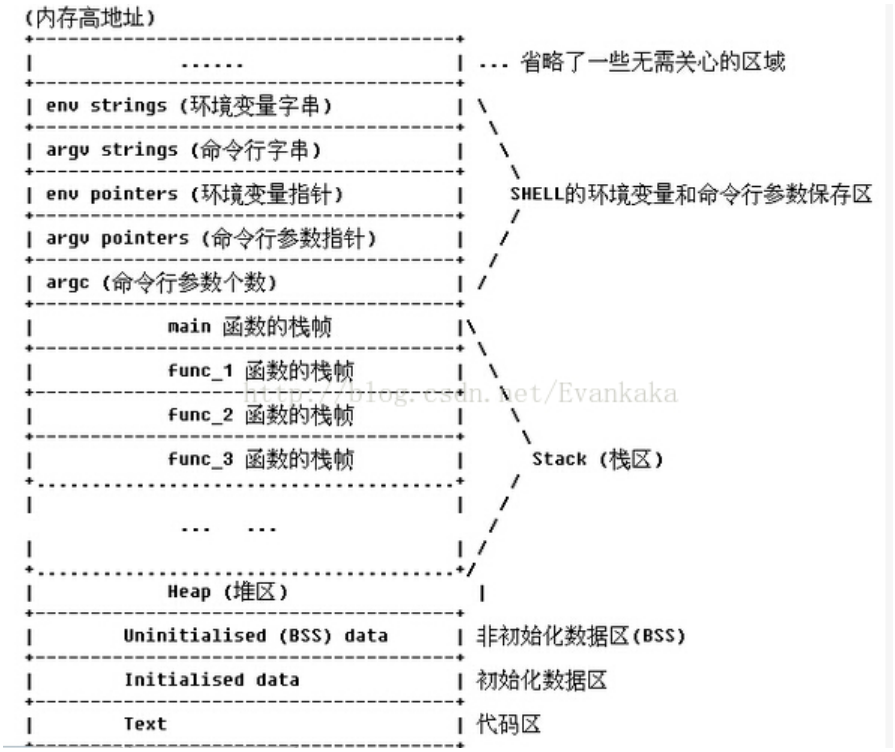
五、使用范例

1、一条进程在内存中的映射

假设现在有一个程序，它的函数调用顺序如下：

main(...) ->; func_1(...) ->; func_2(...) ->; func_3(...)，即：主函数main调用函数func_1; 函数func_1调用函数func_2; 函数func_2调用函数func_3。

当一个程序被操作系统调入内存运行, 其对应的进程在内存中的映射如下图所示：



注意：

- 随着函数调用层数的增加，函数栈帧是一块块地向内存低地址方向延伸的；
- 随着进程中函数调用层数的减少（即各函数调用的返回），栈帧会一块块地被遗弃而向内存的高址方向回缩；
- 各函数的栈帧大小随着函数的性质的不同而不等，由函数的局部变量的数目决定。

- 未初始化数据区(BSS)：用于存放程序的静态变量，这部分内存都是被初始化为零的；而初始化数据区用于存放可执行文件里的初始化数据。这两个区统称为数据区。

- Text(代码区)：是个只读区，存放了程序的代码。任何尝试对该区的写操作会导致段违法出错。代码区是被多个运行该可执行文件的进程所共享的。

- 进程对内存的动态申请是发生在Heap(堆)里的。随着系统动态分配给进程的内存数量的增加，Heap(堆)有可能向高址或低址延伸，这依赖于不同CPU的实现，但一般来说是向内存的高地址方向增长的。

- 在未初始化数据区（BSS）或者Stack(栈区)的增长耗尽了系统分配给进程的自由内存的情况下，进程将会被阻塞，重新被操作系统用更大的内存模块来调度运行。

- 函数的栈帧：包含了函数的参数(至于被调用函数的参数是放在调用函数的栈帧还是被调用函数栈帧，则依赖于不同系统的实现)。函数的栈帧中的局部变量以及恢复该函数的主调函数的栈帧(即前一个栈帧)所需要的数据,包含了主调函数的下一条执行指令的地址。

2、 函数的栈帧

函数调用时所建立的栈帧包含下面的信息：

1）函数的返回地址。返回地址是存放在主调函数的栈帧还是被调用函数的栈帧里，取决于不同系统的实现；
2）主调函数的栈帧信息，即栈顶和栈底；
3）为函数的局部变量分配的栈空间；
4）为被调用函数的参数分配的空间取决于不同系统的实现。

注意：

- BSS区（未初始化数据段）：并不给该段的数据分配空间，仅仅是记录了数据所需空间的大小。
- DATA（初始化的数据段）：为数据分配空间，数据保存在目标文件中。

- 上一篇进程和线程的定义及区别、线程同步、进程通讯方式总结
- 下一篇玩转Android之Activity详细剖析

查看评论

4楼 大梦方觉醒 2016-01-29 11:03发表 [回复]