使用平台是Ubuntu10.0 ，GCC编译器

我们先看下面这段程序，

1. #include <stdio.h>
2. #include <string.h>
4. **int** mian(){
5. **char** str1[] = "abcd" ;
6. **char** str2[2];
7. printf("address\_str1= %p\n",str1);
8. printf("address\_str2= %p\n",str2);
10. strcpy(str2,"123");
12. printf("str1=%s\n",str1);
13. printf("str2=%s\n",str2);
15. **return** 0;
16. }

程序执行结果为：

address\_str1=0xbfe5fbb7

address\_str2=0xbfe5fbb5

str1=3

str2=123

我们没有对字符数组str1进行操作，str1[]的值不是abcd而是3 ，这是为什么呢？

要解决这个问题，我们首先要解决两个问题：

1.strcpy函数的使用方法

2.局部变量在栈中是如何存储的

一 . strcpy 函数

这是strcnp函数的原型

char \*strcpy(char \*dest , const char \*src );

char \*strncpy(char \*dest , const char \*src , size\_t n );

使用strcpy有个缺点：strcpy进行复制时，不进行数组越界的检查。

str2数组只有2个字节，但是"123"却有4个字节，这是很明显的数组越界，但是GCC编译器编译中没有报错，也就是ctrcpy复制是不进行数组越界的检查。要避免这个问题，可以使用strncpy ，因为strncpy有参数n来限定复制的字节数。

二.栈的知识

1、栈区（stack）—   由编译器在程序执行过程中自动分配，在函数返回或程序结束时释放  ，存放函数的参数值，局部变量的值等，

2、栈有栈顶和栈底，只能在栈顶进行入栈和出栈的操作。

3、栈的分类：栈分为满栈、空栈、递增栈（向上增长型的）和递减栈（向下增长型的）

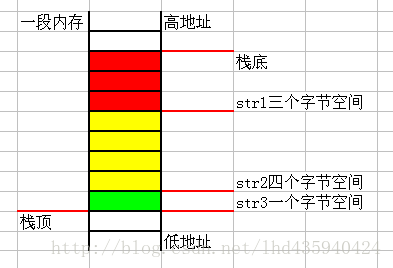
4、对于8086来说，它的栈的生长方向是从高地址到地址的，因此，X86的栈是一个向下增长型的栈 （这点很重要，涉及到局部变量的存储），即栈底为高地址，栈顶为低地址

三.局部变量在栈中的存储

下面通过举例来说明变量在栈中的存储。

假如现在在main函数中定义三个数组：char str1[3] ,str2[4] ,str3[1]  ;

那么系统会在程序执行过程中为这个三个变量在栈中分配存储空间：如下内存图



系统会先分配三个字节来存入数组str1，然后再分配四个字节的空间来存入数组str2，最后分配一个字节空间来存入数组str3 。 这就是变量入栈的顺序：str1 --> str2 --> str3

说这么多，其实还是没有解决str1=3这个问题。那么下面通过对栈的具体分析，就能获得答案的。

我们打印str1[]的地址为0xbfe5fbb7  ，str2[2]的地址为0xbfe5fbb5  ,那么程序执行时在栈中先把str1入栈，并为str1分配5个字节 ，然后再把str2入栈，并为str2分配2个字节。内存图如下：



执行语句strcpy(str2,"123")时，strcpy不会进行数组越界检查，因此会把3和\0存入str1的内存空间中，覆盖原来存入的a 和 b 。

由于printf是行缓存，输出遇见\0就结束。所以执行printf语句时，str1=3 ；str2 = 123