

空间里。不过，长一些的字符串就会导致 `gets` 覆盖栈上存储的某些信息。随着字符串变长，下面的信息会被破坏：

输入的字符数量	附加的被破坏的状态
0~7	无
9~23	未被使用的栈空间
24~31	返回地址
32+	caller 中保存的状态

字符串到 23 个字符之前都没有严重的后果，但是超过以后，返回指针的值以及更多可能的保存状态会被破坏。如果存储的返回地址的值被破坏了，那么 `ret` 指令(第 8 行)会导致程序跳转到一个完全意想不到的位置。如果只看 C 代码，根本就不可能看出会有上面这些行为。只有通过研究机器代码级别的程序才能理解像 `gets` 这样的函数进行的内存越界写的影响。

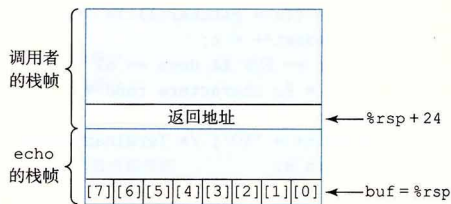



图 3-40 echo 函数的栈组织。字符数组 `buf` 就在保存的状态下面。对 `buf` 的越界写会破坏程序的状态

我们的 `echo` 代码很简单，但是有点太随意了。更好一点的版本是使用 `fgets` 函数，它包括一个参数，限制待读入的最大字节数。家庭作业 3.71 要求你写出一个能处理任意长度输入字符串的 `echo` 函数。通常，使用 `gets` 或其他任何能导致存储溢出的函数，都是不好的编程习惯。不幸的是，很多常用的库函数，包括 `strcpy`、`strcat` 和 `sprintf`，都有一个属性——不需要告诉它们目标缓冲区的大小，就产生一个字节序列[97]。这样的情况就会导致缓冲区溢出漏洞。

 **练习题 3.46** 图 3-41 是一个函数的(不太好的)实现，这个函数从标准输入读入一行，将字符串复制到新分配的存储中，并返回一个指向结果的指针。

考虑下面这样的场景。调用过程 `get_line`，返回地址等于 `0x400076`，寄存器 `%rbx` 等于 `0x0123456789ABCDEF`。输入的字符串为“0123456789012345678901234”。程序会因为段错误(segmentation fault)而中止。运行 GDB，确定错误是在执行 `get_line` 的 `ret` 指令时发生的。

- A. 填写下图，尽可能多地说明在执行完反汇编代码中第 3 行指令后栈的相关信息。在右边标注出存储在栈中的数字含意(例如“返回地址”)，在方框中写出它们的十六进制值(如果知道的话)。每个方框都代表 8 个字节。指出 `%rsp` 的位置。记住，字符 0~9 的 ASCII 代码是 `0x3~0x39`。

00 00 00 00 00 40 00 76	返回地址

- B. 修改你的图，展现调用 `gets` 的影响(第 5 行)。  
C. 程序应该试图返回到什么地址？