面试官：Linux是如何进行函数调用的？

程序员面试 1周前

以下文章来源于程序喵大人，作者程序喵大人

程序喵大人
分享计算机基础及高端知识，主要有C++、java、Android、音视频、数据结构、操作…

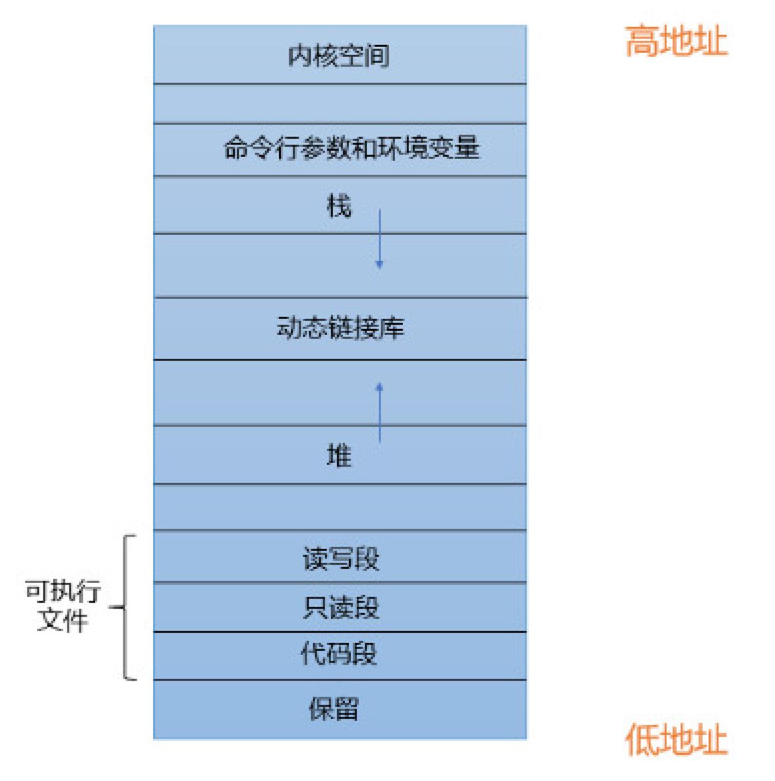
> 先抛出几个问题：

- 进程虚拟地址空间是如何分布的？
- 函数调用的栈帧结构是什么样子？
- 函数调用涉及到的寄存器都起了什么作用？
- 函数参数是如何传递的？传递顺序如何？
- 函数的返回值是如何传递的？

如果您对上述问题有些困惑，请继续往下看吧！

进程的内存布局

如图：



高地址的一部分空间会分配给内核，称为内核空间，剩下的内存空间给用户使用，称为用户空间。

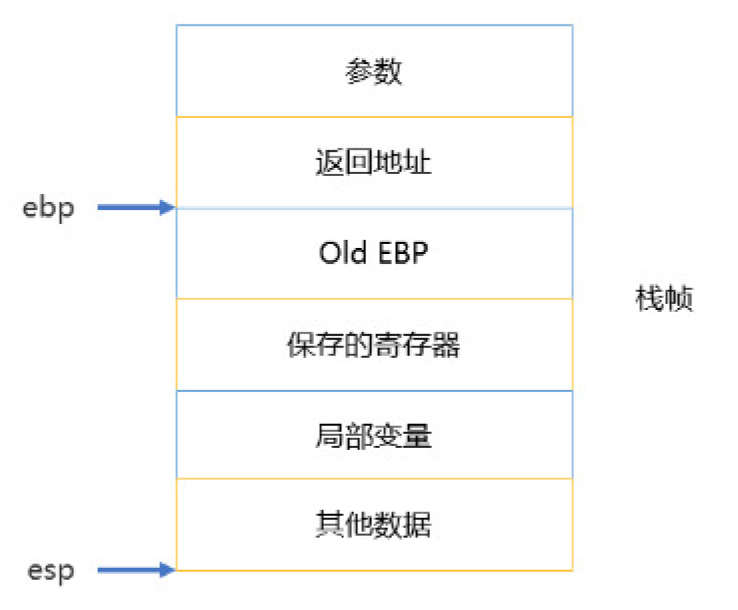
用户空间中有几个**主要的内存区域**：

- **栈**：用于维护函数调用的上下文，离开了栈，函数调用就没法实现，栈通常在用户空间的最高地址处分配，通常有数兆字节的大小。
- **堆**：堆用来容纳程序动态分配的内存区域，程序中malloc或new分配的内存就来自堆里。堆通常存在于栈的下方（低地址方向），在某些时候，堆也可能没有固定统一的存储区域，堆一般比栈大很多，可以有百兆甚至几G的大小。
- **动态链接库映射区**：这个区域用于映射装载的动态链接库，Linux下如果可执行文件依赖其它共享库，那系统就会在这个区域分配相应空间，并将共享库装入该空间。
- **可执行文件映像**：存储着可执行文件在内存里的映像，由装载器在装载时将可执行文件的内存读取或映射到这里。
- **保留区**：保留区并不是一个单一的内存区域，而是堆内存中受到保护而禁止访问的内存区域的总称，例如在大多数操作系统里，极小的地址通常都是不允许访问的，如NULL，通常C语言将无效地址赋值为0也是出于这个考虑，因为0地址正常情况下不可能有有效的可访问数据。

函数调用的栈帧结构

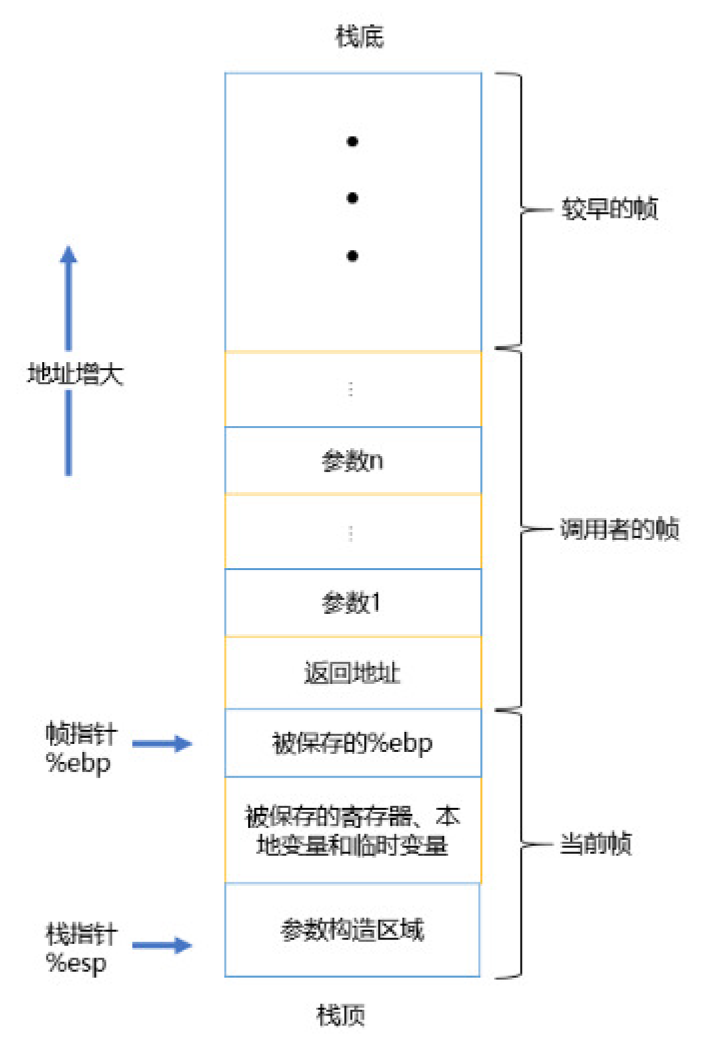
我们都知道函数调用都是以栈帧为单位，机器通常用栈来传递函数参数、保存返回地址、保存寄存器（即函数调用的上下文）及存储本地局部变量等。

一个单独的栈帧结构如图所示：



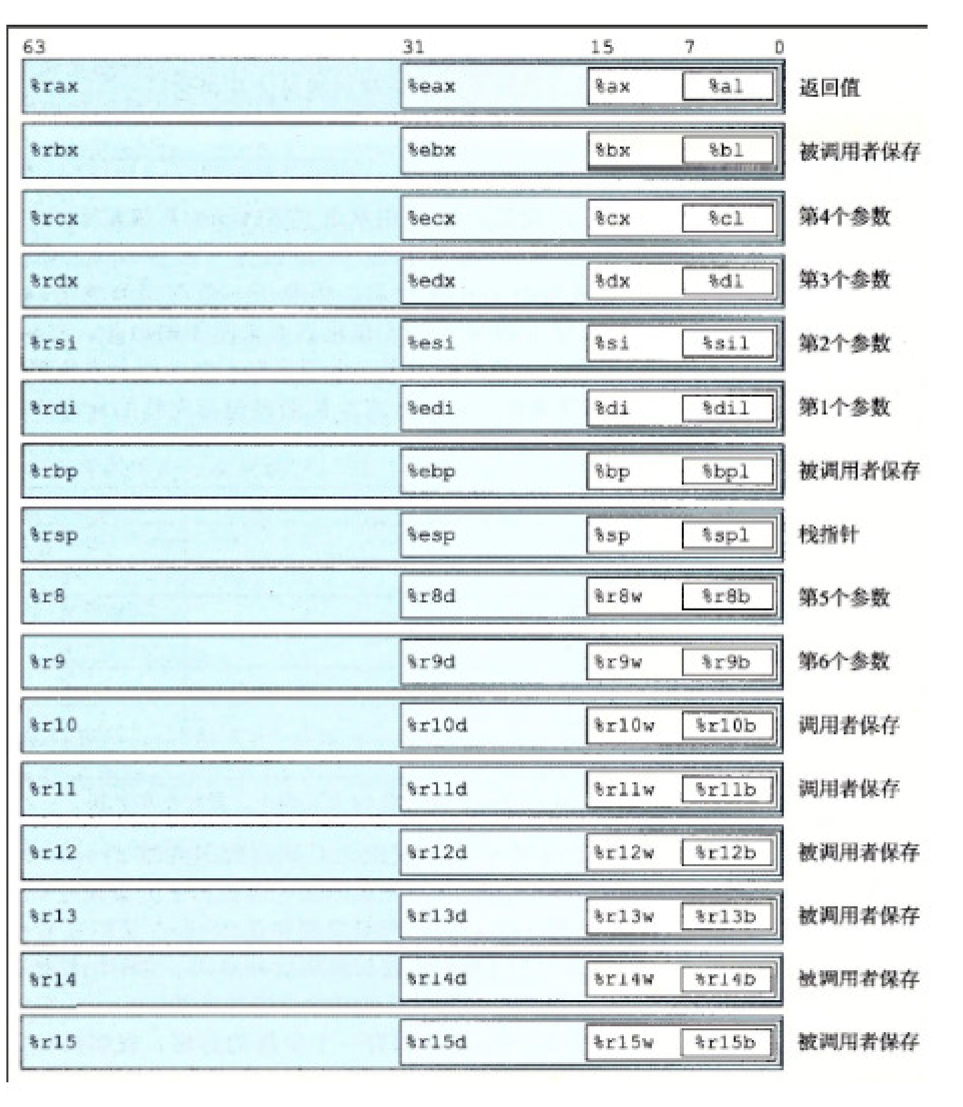
为单个函数调用分配的那部分栈称为栈帧，栈帧的边界由两个指针界定：寄存器%ebp为帧指针，指向当前栈帧的起始处，通常较为固定；寄存器%esp为栈指针，指向当前栈帧的栈顶位置，当程序执行时，栈指针可以移动，因此大多数数据的访问都是相对于帧指针的。

一次函数调用的栈帧图如下：



寄存器使用约定

直接看图：



图片来源于网络，侵权删

上图表达的应该已经很清楚啦，简单示例解释一下，函数调用需要传递参数时，第一个参数存到%edi里，第二个参数会存到%esi里，如果有返回值会存到%eax里，这里如果是64位的返回值，会使用%rax。

函数的调用约定

这里主要涉及三种约定：

- 函数参数的传递顺序和方式：这里可以有很多参数传递方式，栈传递和寄存器传递，函数的调用方将参数压入栈中，函数自己再从栈中将参数取出，需要规定压栈的顺序，是从左到右，还是从右到左，有的也使用寄存器传递，这都需要约定好。
- 栈的维护方式：在函数将参数压栈后，函数体会被调用，此后需要将被压入栈中的参数全部弹出，以使得栈在函数调用前后保持一致，这个弹出的工作可以是由函数的调用方完成还是函数本身来完成需要约定好。
- 名字修饰策略：为了链接的时候对调用约定进行区分，需要对函数本身的名字进行修饰，不同的调用约定有不同的名字修饰策略。一般都是前面加个下划线。

C语言默认的调用约定是cdecl方式，可以通过__attribute__((cdecl))标明使用cdecl约定，其实还有其它一些调用约定，如图：

调用惯例	出栈方	参数传递	名字修饰
cdecl	函数调用方	从左至右的顺序压参数入栈	下划线+函数名
stdcall	函数本身	从右至左的顺序压参数入栈	下划线+函数名+@+参数的字节数，如函数int func(int a, double b)的修饰名是func@12
fastcall	函数本身	头两个DWORD(4字节)类型或者占更少字节的参数被放入寄存器，其他剩下的参数按从右至左的顺序压入栈	@+函数名+@+参数的字节数
pascal	函数本身	从左至右的顺序压参数入栈	较为复杂，参见pascal文档

函数的返回值传递

这里有几种情况：

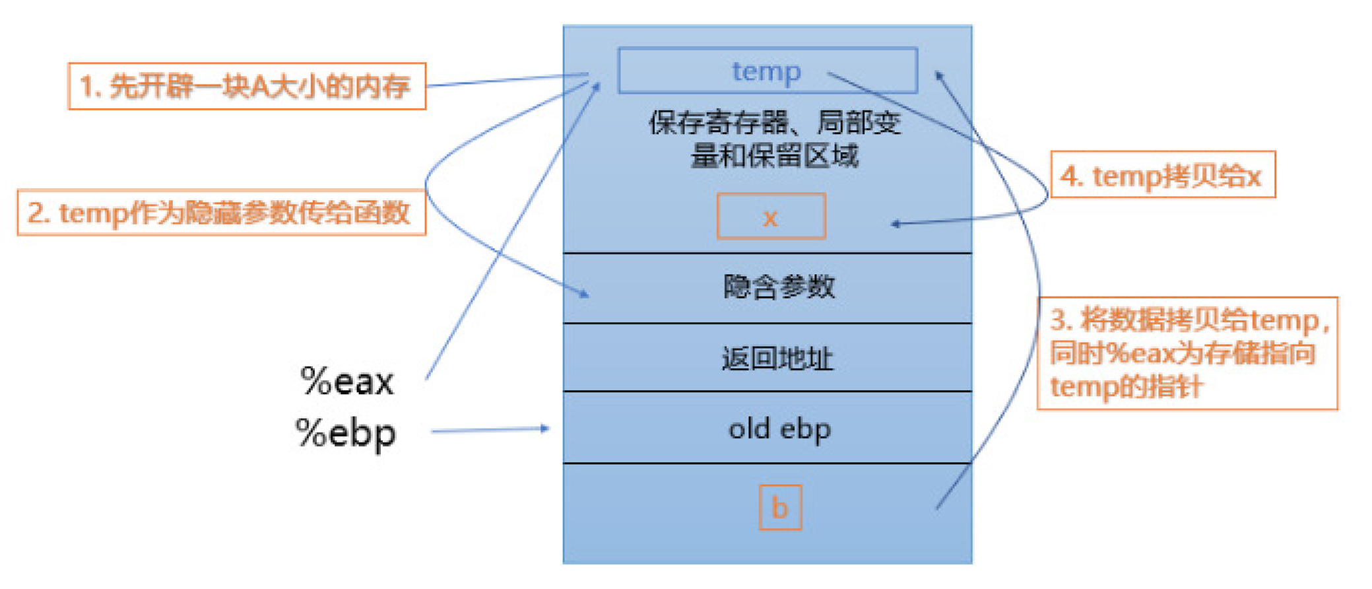
4字节：当函数返回值是4个字节会通过%eax寄存器作为通道，函数将返回值存储在%eax中，返回后函数的调用方再读取%eax。

5-8个字节：通过rax寄存器作为通道。

大于8个字节：以如下代码举例：

```
struct A {
// ...大于8字节
};
A func() {
A b;
return b;
}
A x = func();
```

返回值传递方式如图：



1. 调用函数首先在栈上额外开辟一片空间，作为临时对象（temp）
2. temp作为隐藏参数传递给被调用函数
3. 函数将数据拷贝给temp，同时%eax为指向temp的指针
4. 返回返回后将%eax指向的temp拷贝回被赋予的对象

返回值类型的尺寸太大导致函数返回时，会开辟一段区域作为中介，返回值对象会被拷贝两次，而C++在有些情况下会做返回值优化，减少拷贝的次数。

参考资料

https://blog.csdn.net/slvher/article/details/8831885

https://blog.csdn.net/slvher/article/details/8831983

https://www.cnblogs.com/alantu2018/p/8465904.html

https://mp.weixin.qq.com/s/fpf4qRRLN3wVDUrWka3HfQ

https://mp.weixin.qq.com/s/j7SKtrMCmYs6g8yH75OH4A

https://www.sec4.fun/2018/05/29/stack/

https://murphypei.github.io/blog/2019/01/linux-heap

https://cloud.tencent.com/developer/article/1515763

《程序员的自我修养：链接装载与库》