30 | ABI 与 API 究竟有什么区别？

2022-03-07 于航

《深入C语言和程序运行原理》 课程介绍 >

讲述：于航

时长 08:40 大小 7.95M

你好，我是于航。

今天我们来聊另外一个老生常谈的话题：“ ABI 与 API 这两个概念究竟有什么区别？”

也许你之前也思考过这个问题。ABI 与 API 这两个英文缩写只差一个字符，因此它们对应的概念在很多线下讨论和博客文章中会被经常混用，甚至是乱用。当然，时不时地，这个问题也会成为人们在技术社交圈内的丰富谈资。这一讲，就以你熟悉的 C 语言体系为例，我们来一起看看 ABI 与 API 二者分别指代什么内容，有什么区别。

API

API 的全称为“应用程序编程接口（Application Programming Interface）”。从它的名字我们就能看出来，这一类接口的侧重点在于“编程”。因此，通过遵循 API 规范，我们可以在相应的编

程语言代码中使用这些接口，以操作计算机系统来完成某项特定任务。而对 C 语言来说，那些由 C 标准库提供的，被定义在不同头文件中的函数原型，便是一种 API 的具体表现形式。

重要特征

API 具有的一个最重要特征，便是隐藏了其背后具体功能的内部实现细节，只公开对编码有意义的部分（如接口名称、可接收参数的个数与类型等）。通过保持这部分特征的一致性，API 提供者与调用者便可在相对隔离的环境下被独立维护。在这种情况下，这部分相对统一和稳定的特征也可被单独抽离出来，成为相应的 API 规范。

如下面的代码所示，C 标准库函数 fopen 在 API 层面有着稳定的特征，但实际上，使用这些 API 构建的应用程序却可以在不修改代码的情况下，灵活选用不同的接口实现方案（如 musl 和 glibc）。这其中的一个重要原因，便是 **C 标准库的 API 规范是独立于具体实现的**。

```
FILE *fopen(const char* filename, const char* mode)
```

发展简史

API 对应的概念最早出现于上世纪 40 年代，而直到 1968 年，API 这个术语才在发表于 AFIPS（American Federation of Information Processing Societies）会议的一篇论文中首次出现，随后逐渐被业界广泛采用。而随着计算机互联网的普及，API 一词开始被更多地用于表示与 Web 领域相关的标准接口。其中，使用最为广泛的当属那些基于 REST、SOAP 等风格构架的 Web API。下面是一个简单的例子：

```
POST https://docs.googleapis.com/v1/documents/{documentId}:batchUpdate
```


截至目前，在没有具体上下文的情况下，我们通常默认用 API 一词来表示那些与 Web 领域相关的标准接口。

总的来看，API 的最重要特征在于**提供相应功能的同时隐藏实现细节，让其使用者可以按照较为统一和稳定的方式来使用系统能力**。按照这个结论，API 的概念可以被应用在更加广泛的领域中。比如，对于操作系统调用来说，我们也可以称其为由操作系统内核提供的一种 API。通

过这些 API，系统库乃至上层应用都可以使用内核的能力，但使用者又无需了解它的内部具体实现细节。

可以看到，API 的概念还是十分简单和清晰的。接下来我们看看与它仅有一字之差的 ABI，到这里，事情变得复杂了起来。

ABI

ABI 的全称为“应用程序二进制接口（Application Binary Interface）”。与 API 不同的是，ABI 的侧重点并不在于 “Programming”，而在于 “Binary”，即机器指令层面的具体格式。因此，与 API 相反，**ABI 将程序与操作系统、硬件平台之间紧密协作需要遵守的特定规则暴露了出来**。这些规则指定了基于这个体系运行的二进制应用程序，应该如何在机器代码层面进行数据访问或函数调用等一系列操作。

规则的重要性不言而喻。我曾在 05 讲 中介绍过，运行在 x86-64 平台上的类 Unix 系统会遵循名为 “System V AMD64 ABI” 的调用约定，来进行函数的实际调用过程。而在这个约定中，函数调用需要使用固定的寄存器传递参数，并按照一定顺序，在满足栈对齐等一系列要求的情况下进行。当然，你可能会问：如果不满足这些要求，程序就无法正常运行吗？下面，让我们来做一个简单的实验。

不遵循 ABI 的程序能否运行？

这里，为了验证不满足 ABI 要求的程序能否正常运行，我们将直接使用 Intel 格式的汇编代码来编写程序入口函数 _start 的具体实现。在该函数内部，我们会按照违背 SysV 调用约定的方式，调用直接由 C 代码编译而来的函数 sub，并将所得计算结果作为程序的最终返回值。

首先，我们来看 sub 函数的实现代码：


```c
// sub.c
int sub(int x, int y) {
  return x - y;
}
```


该函数的逻辑十分简单：它接收两个整型参数 x 与 y，并返回表达式 x - y 的值。紧接着，我们来编写 _start 部分的逻辑，代码如下所示：

复制代码

```
1  # main.asm
2  extern sub
3  global _start
4  section .text
5  _start:
6    and   rsp, 0xfffffffffffffff0
7    sub   rsp, 1
8    mov   esi, 2  # the 1st param.
9    mov   edi, 1  # the 2nd param.
10   call  sub
11   mov   edi, eax
12   mov   eax, 60
13   syscall
```

这部分逻辑的实现过程主要分为下面几个步骤：

1. 代码第 6 行，通过 and 指令，栈顶首先被对齐到 ABI 要求的 16 字节，接着通过下一行的 sub 指令，我们让栈失去“对齐”这一特性；
2. 代码第 8~9 行，我们使用寄存器 esi 存放传入 sub 函数的第一个参数，使用 edi 存放其第二个参数（与 SysV 中的规定相反）；
3. 代码第 10 行，调用 sub 函数；
4. 代码第 11~13 行，调用 exit 系统调用，并将存放在寄存器 eax 中的计算结果作为程序结束的返回值。

紧接着，分别执行下面这几行命令，以完成上述代码的编译过程（注：这里我们使用的 nasm 是一个 x86 汇编器，详情可以参考这个链接）：

复制代码

```
1  gcc -c sub.c -o sub.o
2  nasm -f elf64 ./main.asm -o ./main.o
3  ld ./main.o ./sub.o -o ./main
```


最后，运行程序，通过命令 `echo $?`，我们可以查看到程序退出时的实际返回值。

在这一系列操作后，你会发现程序可以被正常编译和执行，只是得到的返回结果并不符合我们的预期。实际上，出现这个问题的原因正是由于我们编写的 _start 入口代码没有遵循统一的

SysV 调用规范。

在代码的第 8~9 行，我们以相反的顺序（即先 esi 后 edi）传入了 sub 函数需要的两个实参。而在 sub 函数对应的机器代码实现中，GCC 会按照 SysV 规范来分配该函数需要使用的各个参数。因此，实际上由 esi 传入的第一个参数，便会被 sub 函数当作第二个参数。同样地，由 edi 传入的第二个参数则会被当作第一个参数。

除此之外，虽然未对齐的栈顶并未导致 sub 函数的调用失败，但这只是由于我们的实验用例比较幸运。在真实的 x86-64 体系中，有很多指令（如 `movapd` 、`movaps`）在被实际调用前，都是需要栈顶位于特定边界（如 16、32、64 字节）对齐的，否则会抛出相应异常（如 #GP）。这些指令可能被广泛地使用在各类标准库实现中，因此，**为了能够正确使用这些函数，满足栈对齐这一要求也是不可或缺的**。

ABI 的主要内容

总的来看，ABI 规范通常会涵盖以下这些内容：

- 函数调用规范；
- 处理器可以访问的数据类型其大小与对齐方式；
- 进程初始化细节（如栈和寄存器的状态变化）；
- 对象文件（如 .o）的基本结构；
- 程序载入和动态链接的细节；
- ……

对于 C 程序来说，稳定的 ABI 对**保障同一个程序在多个不同平台（如不同的类 Unix 系统）上的兼容性**有着重要作用。C 标准中并未规定 C 代码应该按照怎样的方式执行，内存应该如何布局和分配。而与 C 程序运行时相关的一切细节，实际上都是由相应平台的 ABI 来决定的。比如，在 SysV ABI 规范手册的第一章内容中，我们可以看到下面这句话：

> No attempt has been made to specify an ABI for languages other than C. However, it is assumed that many programming languages will wish to link with code written in C, so that the ABI specifications documented here apply there too.

翻译过来就是：“我们没有尝试为 C 以外的语言指定 ABI。然而，假定许多其他编程语言都希望与使用 C 语言编写的代码进行链接，那么这里记录的 ABI 规范也同样适用于那些情况。”可以看到，C 语言的特殊地位使得操作系统厂商会选择使用它来作为编写相应 ABI 规范的“基准语言”。

比如在 SysV 中，规范将 C 语言里的各种基本类型与处理器类型进行了一一映射，对结构和联合的对齐要求进行了说明，对可变参数列表的具体实现方式进行了说明，等等。所有这些细则都将共同约束编译器，使其按照某一特定方式编译和构建 C 程序。

总结

今天，我主要为你讲解了 API 与 ABI 这两个概念之间的区别。

API 的全称为“应用程序编程接口”，它是程序员可以通过编程语言调用的一种特殊资源。API 通过隐藏功能的内部实现细节隔离了其使用方与提供方，使得 API 实现与应用程序实现可以通过遵循统一、稳定的 API 规范，来达到各自独立维护的目的。API 有着多种具体表现形式，比如源代码形式的函数，或是基于互联网的 Web 接口。

ABI 的全称为“应用程序二进制接口”，它是一套描述了应用程序应该如何在机器指令层面与特定操作系统和硬件平台正常协作的一系列规范。这些规范决定了应用程序应该如何进行数据访问、函数调用，乃至使用操作系统的能力。不遵循 ABI 规范的应用程序也许可以运行，但它却可能会失去在同一个体系下的兼容性以及运行正确性。

思考题

尝试使用 x86-64 汇编语言编写一个程序，使它能够产生由于栈未对齐而导致的异常。希望你可以动手实践下，然后在评论区分享你的经验，或者提出你遇到的问题。

今天的课程到这里就结束了，希望可以帮助到你，也希望你在下方的留言区和我一起讨论。同时，欢迎你把这节课分享给你的朋友或同事，我们一起交流。

上一篇 29 | C 程序的入口真的是 main 函数吗？

下一篇 31 | 程序如何与操作系统交互？

精选留言 (1)

Y

2022-03-07

API：规定了螺丝和螺丝母的规格

ABI：规定了制作螺丝的材料和制作细节

😂

作者回复: 可以说是十分形象了！