知乎 首发于 linux内核技术

x86通用寄存器


小乐叔叔
资深打工人
关注他

13 人赞同了该文章

前言

在linux日常过开发调试中，总避免不了和CPU寄存器、堆栈打交道，尤其是在分析程序core dump文件时，需要对CPU寄存器过程调用约定有深入的理解。本文主要介绍x86和x86_64平台的通用寄存器，以及他们在过程调用中的约定。

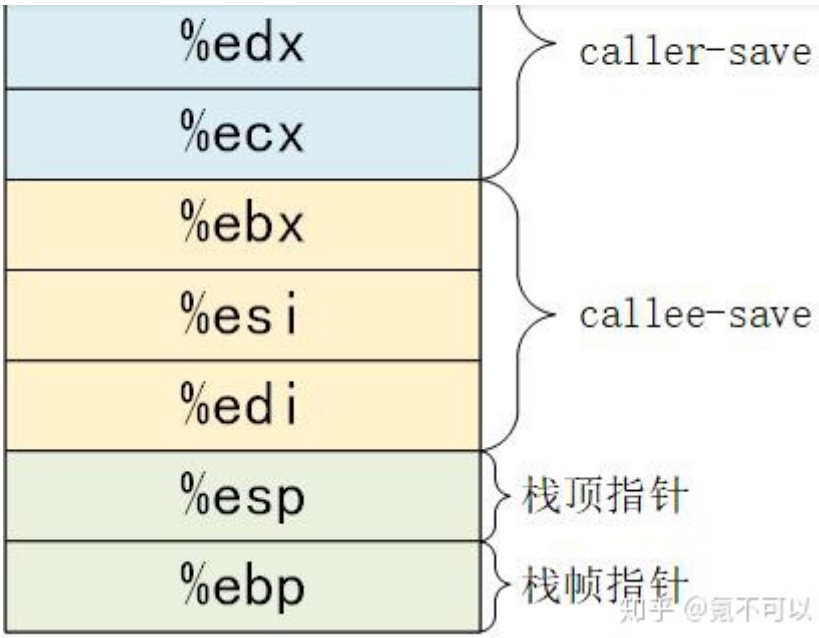
先明确一点，本文将要讨论的是通用寄存器（后简称寄存器）。后面介绍寄存器使用规则或者惯例，只是GCC（G++）遵守的规则，为后面《linux栈回溯》系列文章奠定基础。本文大量参考引用了网络博客文章，文末标注了参考来源，在此感谢博主的无私共享。

x86平台过程调用约定

IA32有8个32位的通用寄存器，这8个通用寄存器都是由8086相应16位通用寄存器扩展成32位而得：

```
EAX：一般用作累加器
EBX：一般用作基址寄存器（Base）
ECX：一般用来计数（Count）
EDX：一般用来存放数据（Data）
ESP：一般用作堆栈指针（Stack Pointer）
EBP：一般用作基址指针（Base Pointer）
ESI：一般用作源变址（Source Index）
EDI：一般用作目标变址（Destinatin Index）
```

作为通用寄存器，过程调用中，**调用者**栈帧需要寄存器暂存数据，**被调用者**栈帧也需要寄存器暂存数据。为防止调用过程中数据不会被破坏丢失，C/C++编译器遵守如下约定的规则：



图一

> 【1】当该函数是处于**调用者**角色时，如果该函数执行过程中产生的临时数据会已存储在**%eax,%edx,%ecx**这些寄存器中，那么在其执行call指令之前会将这些寄存器的数据写入其栈帧内指定的内存区域,这个过程叫做**调用者保存约定(英文原名称:Caller Save)**。
>
> 【2】当该函数是处于**被调用者**角色时，那么在其使用这些寄存器**%ebx,%esp,%edi**之前，那么该函数会保存这些寄存器中的信息到其栈帧指定的内存区域,这个过程叫**被调用者保存约定**。
> 【3】**%eax**总会被用作返回整数值。
> 【4】**%esp,%ebp**总被分别用着指向当前栈帧的**顶部**和**底部**,主要用于在当前函数推出时，将他们还原为原始值。往往会在栈帧开始处保存上一个栈帧的ebp，而esp是全栈的栈顶指针，一直指向栈的顶部。
> 引自：第5篇-戏说程序栈-寄存器和函数状态

x86_64平台寄存器使用约定

x86_64架构有16个通用寄存器，相比IA32多了8个（r8 至 r15时x86_64新增的）。

图二

寄存器集成在CPU上，存取速度比存储器快好几个数量级，寄存器多了，GCC就可以更多的使用寄存器，替换之前的存储器堆栈使用，从而大大提升性能。

和IA32主要区别是：

> 1）6个寄存器用来保存参数，多出的参数类似x86入栈；
> 2）若存在闲置的寄存器的话，局部变量可以直接缓存到闲置寄存器中，过多局部变量类似x86入栈；
> 3）因为是64位，rsp栈指针每次移动8个字节：类似 movq -8(%rsp),%rsp
> 4）函数可以访问%rsp之后最多128个字节的内存：“红色区域”，意味着可以在通过%rsp的来在“红色区域”内存储一些临时数据。而不必使用使用多条指令。参考栈帧章节。
> 5）在编译优化时，栈帧指针rbp被弃用，成为通用一般寄存器。所有对当前栈帧中的内存字段的访问引用,由%rsp进行相对寻址来实现。参考栈帧章节。

在32bit时代，参数传递是通过入栈实现的，相对CPU来说，存储器访问太慢；这样函数调用的效率就不高。在**x86-64**时代，寄存器数量多了，CPU就可以利用多达6个寄存器来存储参数（图二），多于6个的参数，依然还是通过**入栈**实现传递。

因此在x86_64位机器上编程时，需要注意：

1）为了效率尽量使用少于6个参数的函数；

2）传递比较大的参数，尽量使用指针，因为寄存器只有64位；

C源码

```
#include <stdio.h>
#include <stdlib.h>

int foo(int arg1, int arg2, int arg3, int arg4, int arg5, int arg6, int arg7)
{
    int array[] = {100,200,300,400,500,600,700};
    int sum = array[arg1]+ array[arg7];

    return sum;

}      /* -----  end of function foo  ----- */

int main(int argc, char *argv[])
{
    int i = 1;
    int j = foo(0,1,2, 3, 4, 5,6);

        printf("i=%d,j=%d\n", i, j);

    return 0;
}
```

编译：gcc -O1 -S -o args_test.s args_test.c 。查看汇编程序args_test.s（节选）：

```
 5 foo:
 6 .LFB20:
 7         movl    $100, -40(%rsp)
 8         movl    $200, -36(%rsp)
 9         movl    $300, -32(%rsp)
10         movl    $400, -28(%rsp)
11         movl    $500, -24(%rsp)
12         movl    $600, -20(%rsp)
13         movl    $700, -16(%rsp)
14         movslq  8(%rsp),%rax
15         movslq  %edi,%rdi
16         movl    -40(%rsp,%rax,4), %eax
17         addl    -40(%rsp,%rdi,4), %eax
18         ret
```

```
29          subq    $8, %rsp
30  .LCFI0:
31          movl    $6, (%rsp)
32          movl    $5, %r9d
33          movl    $4, %r8d
34          movl    $3, %ecx
35          movl    $2, %edx
36          movl    $1, %esi
37          movl    $0, %edi
38          call    foo
39          movl    %eax, %edx
40          movl    $1, %esi
41          movl    $.LC0, %edi
42          movl    $0, %eax
43          call    printf
44          movl    $0, %eax
45          addq    $8, %rsp
46          ret
```

1）前六个参数依次存在六个通用寄存器中（倒序，类似入栈），第七个参数入栈保存（注意$6是立即数6）；

2）在foo函数中，7 - 13行是把数组常数入栈（方便计算，且是访问了栈顶之外，ABI规范）；

3）14和15行分别去参数1和参数7，rax暂存参数7，最后需要保存函数**返回值**；

4）汇编语句“ -40(%rsp,%rax,4), %eax”中，%rsp作为数组基地址，%rax是数组下标，4是数组元素长度即5）sizeof(int)，最后-40偏移找到正确的栈中的位置。

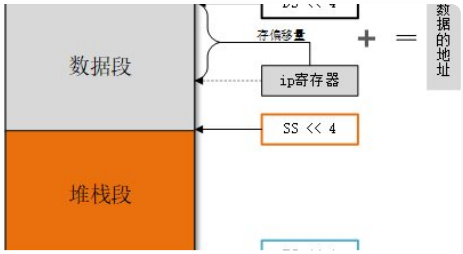
发布于 2020-11-13

知乎 首发于 **linux内核技术**

文章被以下专栏收录

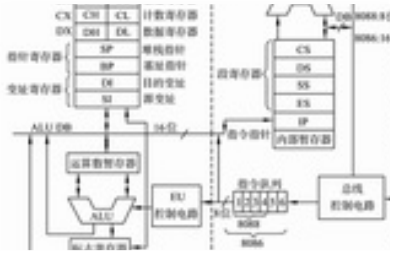
linux内核技术
分享、探讨linux kernel技术

推荐阅读



x86段寄存器和分段机制

小乐叔叔 发表于linux...



x86汇编之——8086寄存器解

Hero

还没有评论

写下你的评论...