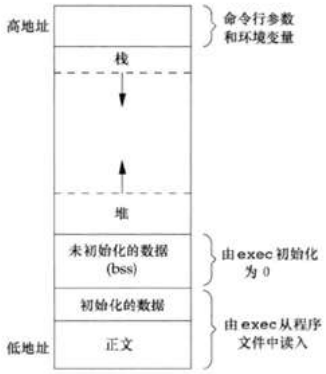
# 函数调用过程中函数栈详解

原创 展开

芹泽 最后发布于2018-08-14 16:19:51 阅读数 3302 收藏




当进程被加载到内存时，会被分成很多段

1. 代码段：保存程序文本，指令指针EIP就是指向代码段，可读可执行不可写，如果发生写操作则会提示segmentation fault
2. 数据段：保存初始化的全局变量和静态变量，可读可写不可执行
3. BSS：未初始化的全局变量和静态变量
4. 堆(Heap)：动态分配内存，向地址增大的方向增长，可读可写可执行
5. 栈(Stack)：存放局部变量，函数参数，当前状态，函数调用信息等，向地址减小的方向增长，可读可写可执行
6. 环境/参数段（environment/argumentssection）：用来存储系统环境变量的一份复制文件，进程在运行时可能需要。例如，运行中的进程，可以通过环境变量来访问路径、shell 名称、主机名等信息。该节是可写的，因此在缓冲区溢出（buffer overflow）攻击中都可以使用该段

## 寄存器

EAX：累加(Accumulator)寄存器，常用于函数返回值

EBX：基址(Base)寄存器，以它为基址访问内存

ECX：计数器(Counter)寄存器，常用作字符串和循环操作中的计数器

EDX：数据(Data)寄存器，常用于乘除法和I/O指针

ESI：源变址寄存器

DSI：目的变址寄存器

ESP：堆栈(Stack)指针寄存器，指向堆栈顶部

EBP：基址指针寄存器，指向当前堆栈底部

EIP：指令寄存器，指向下一条指令的地址

## 入栈push和出栈pop

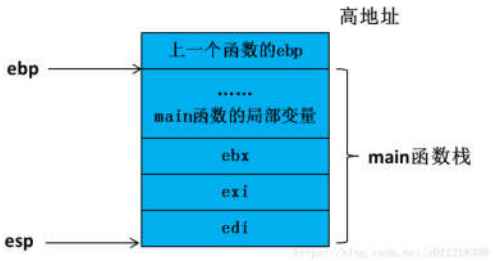
push ebp就等于将ebp的值保存到栈中，并且将当前esp下移

pop ebp就等于将ebp的值从栈中取出来，将ebp指向这个值

**下面用一个例子来讲函数调用过程中栈的变化**

```
int sum(int _a,int _b)
{
    int c=0;
    c=_a+_b;

    return c;
}


int main()
{
    int a=10;
    int b=20;
    int ret=0;

    ret=sum(a,b);

    return 0;
}
```

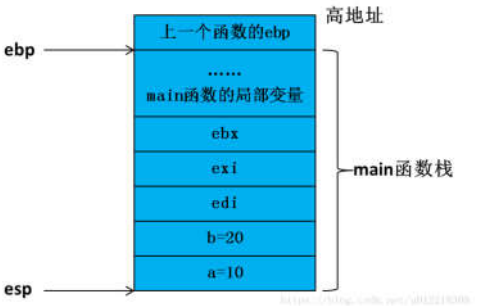
main函数的栈在调用之前如图：




Ok现在讲一讲ret=sum(a,b);的执行过程

**Step 1：**

函数参数从右至左入栈



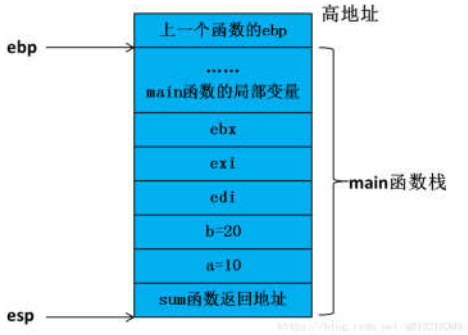

**Step 2：**

**ret=sum(a,b);**

call @ILT+0(sum) (00401005) call指令实际上分两步

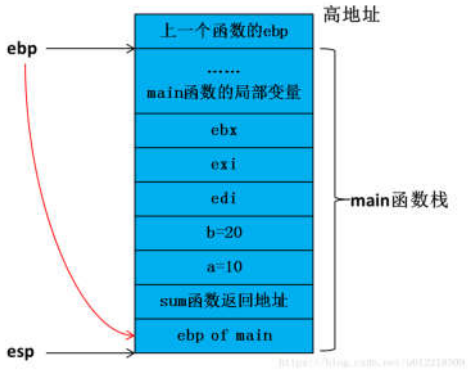
push EIP 将下一条指令入栈保存起来

esp-4 esp指针下移

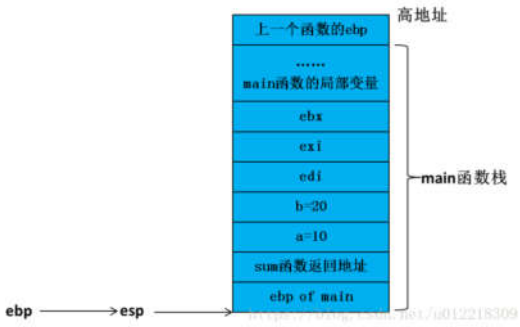



Step 3:

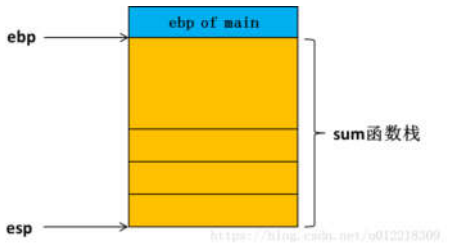
push ebp 将main函数基指针入栈保存




mov ebp esp 将esp的值存入ebp也就等于将ebp指向esp




sub esp 44H将esp下移动一段空间创建sum函数的栈栈帧




**Step 4:**

push ebx

push esi

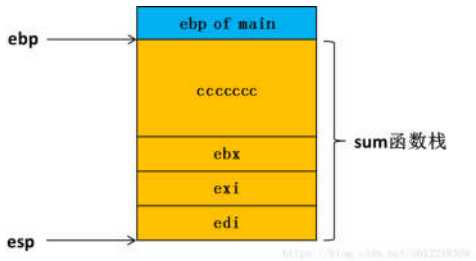
push edi

lea edi,[ebp-44h] 从ebp-44h的地方开始拷贝

mov ecx,11h 拷贝11次

mov eax,0CCCCCCCCh 拷贝内容为0CCCCCCCCh

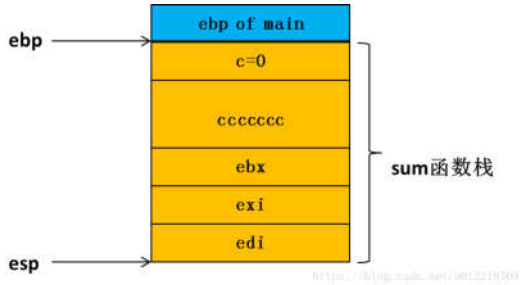
rep stos dword ptr [edi] 每次拷贝双字




**Step 5:**

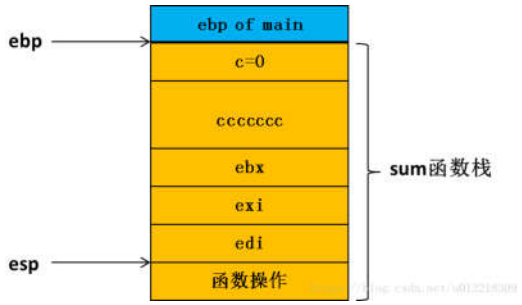
**int c = 0;**

mov dword ptr [ebp-4],0 将sum的局部变量c放入[ebp-4]的空间内




**Step 6:**

执行函数操作




**Step 7:**

**return c = 0;**

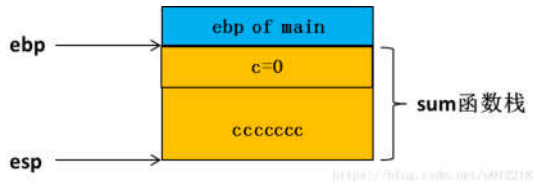
mov eax,dword ptr [ebp-4] 将返回值(变量c所在的地址的内容)放入eax寄存器保存住

**Step 8:**

pop edi //将之前入栈的值重新返回给edi寄存器

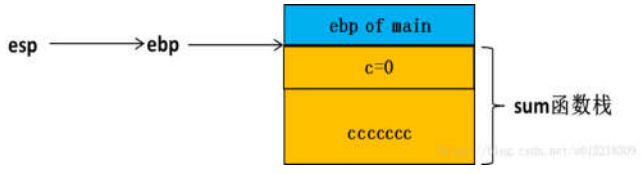
0040104C pop exi ////将之前入栈的值重新返回给exi寄存器

0040104D pop ebx ////将之前入栈的值重新返回给ebx寄存器



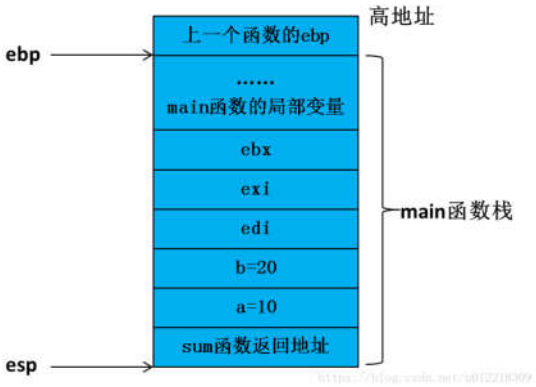

**Step 9:**

mov esp ebp //将ebp的值赋给esp，也就等于将esp指向ebp，销毁sum函数栈帧



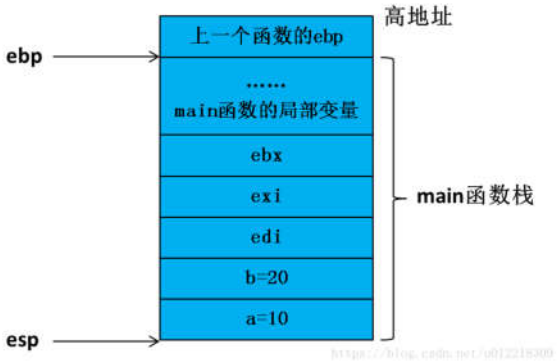
**Step 10:**

pop ebp //ebp出栈，将栈中保存的main函数的基址赋值给ebp



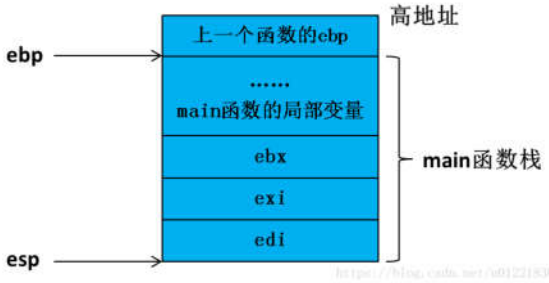

**Step 11:**

ret //ret相当于pop eip 就是把之前保存的函数返回地址(也就是main函数中下一条该执行的指令的地址)出栈

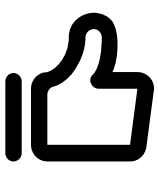
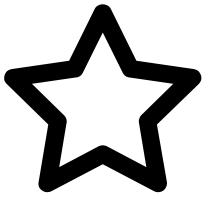
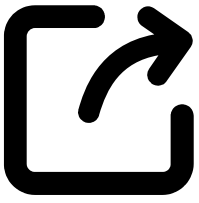



**Step 12:**

add esp,8 //此时若传入sum函数的参数已经不需要了，我们将esp指针上移




此时函数整个调用过程就结束了，main函数栈恢复到了调用之前的状态

点赞 4 收藏 分享