

2021/05/27_x64汇编与逆向_第4课_函数参数的传递、变量、数组、结构体

笔记本: x64汇编与逆向

创建时间: 2021/5/27 星期四 10:03

作者: ileemi

- [课前会议](#)
- [三目运算](#)
- [函数的工作原理](#)
- [调用约定默认值](#)
- [函数参数的传递](#)
 - [整型传参](#)
 - [浮点型传参](#)
 - [混合传参](#)
 - [返回值](#)
- [变量在内存中的位置](#)
 - [全局变量的识别](#)
 - [局部变量的识别](#)
 - [静态局部变量](#)
 - [堆变量初始化](#)
- [数组](#)
 - [数组作为参数](#)
- [结构体](#)
- [构造析构](#)
- [异常](#)

课前会议

C++多态:

编译期多态: 函数重载(显示原理就是通过名称粉碎)、RTTI

运行期多态: 虚函数

函数指针: 主要应用在软件设计上(比如虚表的结构)。

泛型编程: 实现方式 --> 模板(泛型编程的应用, 一种具体的实现方式, 泛型编程是一种设计的理念), 主要解决数据类型导致函数不通用, 使用 void* 来定义参数的类型。泛型编程不止C++可以实现, C语言也可以实现。

宏: 注意大括号, 运算符的优先级以及嵌套问题。应用: 增加可读性以及性能。当函数中的局部变量较多时就不适合使用宏, 为此C++发明了内联函数。

注入、hook的差别:

注入: 将自己的代码注入到目标进程并执行。远程线程注入、apc注入、dll劫持、调试器注入

hook：监控进程中API的操作。IAT HOOK（需要远程线程注入）、SetWindowsHookEx、inline hook、SSDT hook（内核，不需要注入）

MFC消息机制：实现原理描述一下

调试器的实现原理：就是异常处理（指令执行出现异常，调试器接收异常）。

三目运算

- $x == 0 ? 8 : 9$ 这种两值相差一的，利用 setz/setnz reg 条件设置指令
- 其他情况利用 cmovz 条件传送指令

函数的工作原理

函数调用后，寄存器和内存的情况：



调用约定默认值

默认情况下，x64 应用程序二进制接口 (ABI) 使用四寄存器 fast-call 调用约定。系统在调用堆栈上分配空间作为影子存储，供被调用方保存这些寄存器。

函数参数的传递

默认情况下，x64 调用约定将前 4 个参数传递给寄存器中的函数。用于这些参数的寄存器取决于参数的位置和类型。剩余的参数按从右到左的顺序推送到堆栈上。

下表总结了如何从左侧按类型和位置传递参数：

参数类型	第 5 个和更高位置	第 4 个	第 3 个	第 2 个	最左侧
浮点	堆栈	XMM3	XMM2	XMM1	XMM0
整数	堆栈	R9	R8	RDX	RCX
聚合 (8、16、32 或 64 位) 和 <code>__m64</code>	堆栈	R9	R8	RDX	RCX
其他聚合，作为指针	堆栈	R9	R8	RDX	RCX
<code>__m128</code> ，作为指针	堆栈	R9	R8	RDX	RCX

整型传参

最左边 4 个位置的整数值参数从左到右分别在 RCX、RDX、R8 和 R9 中传递。如前所述，第 5 个和更高位置的参数在堆栈上传递。寄存器中的所有整型参数都是向右对齐的，因此被调用方可忽略寄存器的高位，只访问所需的寄存器部分。

```
func1(int a, int b, int c, int d, int e, int f);  
// a in RCX, b in RDX, c in R8, d in R9, f then e pushed on stack
```

浮点型传参

YMM0 -- 32字节

XMM0 -- 16字节

前四个参数中的所有浮点和双精度参数都在 XMM0 - XMM3（具体视位置而定）中传递。存在 varargs 参数时，浮点值只放在整数寄存器 RCX、RDX、R8 和 R9 中。有关详细信息，请参阅 Vararg。同样，当相应的参数为整数或指针类型时，将忽略 XMM0 - XMM3 寄存器。

```
func2(float a, double b, float c, double d, float e, float f);  
// a in XMM0, b in XMM1, c in XMM2, d in XMM3, f then e pushed on stack
```

混合传参

```
func3(int a, double b, int c, float d, int e, float f);  
// a in RCX, b in XMM1, c in R8, d in XMM3, f then e pushed on stack
```

返回值

可以适应 64 位的标量返回值（包括 `__m64` 类型）是通过 **RAX** 返回的。非标量类型（包括浮点数、双精度数和矢量类型，例如 `__m128`、`__m128i`、`__m128d`）以 **XMM0** 的形式返回。返回到 RAX 或 XMM0 中的值的未使用位数的状态未定义。

变量在内存中的位置

变量的识别：

全局变量的识别

通过IDA可进行观察，直接调用，在data区。

局部变量的识别

64位程序的堆栈空间不能直接判定是参数还是局部变量。一般使用预留空间存储局部变量（需要根据函数调用上下文进行判断，预留空间是否做为参数进行传递，没有做为参数就是存放局部变量）。

静态局部变量

会使用标志（全局标志）判断是否进行初始化（在程序中只能初始化一次）。

堆变量初始化

通过 `_initterm` 进行初始化赋值操作。

数组

寻址公式和32位一样。使用 XMM0 等寄存器对数组成员进行批量初始化（一次16字节）。

数组作为参数

和32位基本一致。

结构体

64位结构体默认对齐值是8字节（受编译器版本的影响）。

成员访问：结构体首地址+成员偏移

做为函数参数传参：栈拷贝，传递结构体指针

返回结构体：返回结构体指针

构造析构

区别：64位程序中函数的调用约定为fastcall，虚表由四字节改为8字节。

异常

64位程序不在使用SEH处理异常，采用新的异常框架（FH）