My Code Platform Home Language

结构体成员在内存中的对齐方式

≥ myCode ○ 2016年2月15日 □ C/C++

这个话题还是很早以前讨论过,当时并没有好好的理解,最近在复习知识的时候又重新看了一遍资料,自己做一下总结,也希望后面有人需要学习时可以对他

 \Box 2

有所帮助。以下我会举两个结构体的例子,分别画图的方式表达对齐的原则。

结构体对齐的公式

记住以下这些规则,把结构体往里面套就可以了。结构体对齐的原则就是牺牲空间的方式来减少时间的消耗,空间用完还可以复用,而时间过去了就再也不会, 回来了。

- 以 #pragma pack(x) 中 x 的大小和结构中占用空间最大的成员做比较,取小值为 n (外对齐依据)
- 以 n 值和结构体每个成员比较,得出结果列表为 m[x]
- 根据每个成员的大小依次向内存中填充数据,要求填充 成员的起始地址 减去 构体起始地址 的差都可以整除 m[x] ,如不能整除则向后移动,直到可以整除 再填充成员到内存(内对齐依据)
- 当全部成员填充完毕后所占用的字节若不能整除 n,则扩充内存到可以整除 n 为止。

案例—

我们来看一个简单的案例,#pragma pack(4) 为 4,结构体中有 char、short、int 3个成员,其对齐的方式如下图表示:

```
n = #pragma pack(4) > sizeof(int) ? sizeof(int) : #pragma pack(4)
#pragma pack(4)
                          m[1] = sizeof(char) > n ? n : sizeof(char)
typedef struct data
                          m[2] = sizeof(short) > n ? n : sizeof(short)
                                                                           2
                          m[3] = sizeof(int) > n ? n : sizeof(int)
   char a;
   short b;
   int c;
                            0 可以整除 m[1] ,存放 char
} DATA, *PDATA;
                              1 无法整除 m[2],向后移动直到可以整除的位置填充数据
                                  2 可以整除 [m2] ,存放 short
                                        4 可以整除 [m3],存放 int
short
                            结构体实际占用内存大小8字节
```

```
#include <stdio.h>
#pragma pack(4)
typedef struct data
} DATA, *PDATA;
int main(int argc, char* argv[])
   printf("sizeof(DATA) = %lu\n", sizeof(DATA));
   return 0;
```

以上代码运行后,得到如下结果

```
mycode@vmware:~/Desktop$ gcc struct_size.c -o struct_size
mycode@vmware:~/Desktop$ ./struct_size
sizeof (DATA) = 8
```

案例二

这个案例中,我们把 #pragma pack(8) 设定为 8,结构体中有三个成员 char、double、int,其对齐方式如下图:

```
n = \#pragma pack(8) > sizeof(double) ? sizeof(double) : \#pragma pack(8)
#pragma pack(8)
                                 m[1] = sizeof(char) > n ? n : sizeof(char)
typedef struct data
                                 m[2] = sizeof(double) > n ? n : sizeof(double)
                                                                                   8
                                 m[3] = sizeof(int) > n ? n : sizeof(int)
   char a;
   double b;
                                 0 可以整除 m[1],存放 char
   int c;
} DATA, *PDATA;
                                    1 无法整除 m[2],向后移动直到可以整除 m[2] 的位置填充数据
                                                           8 可以整除 m[2] ,存放 short
                                                                                   16 可以整除 m[3] ,存放 int
                                                                                               20 无法整除 n ,走外对齐原则
                                                          结构体实际占用内存大小 24字节
```

```
#include <stdio.h>
#pragma pack(8)
typedef struct data
} DATA, *PDATA;
int main(int argc, char* argv[])
   printf("sizeof(DATA) = %lu\n", sizeof(DATA));
```

是不是有点毁三观啊?以上代码运行后的结果,的确就是 24。

```
mycode@vmware: \(^\)/Desktop\(\sigma\) gcc struct_size.c -o struct_size
mycode@vmware: \[ \textsup \] ./struct_size
sizeof (DATA) = 24
```

要注意的是,如果你把这个案例中 int 和 double 成员颠倒个位置,再编译代码你会发现其占用空间变成了 16,按上面的规则推断一下,很明显,首先是第二个 成员 int 在内存的第 4 号位置就可以驻留了,第三个成员 double,在第 8 号位置也同样可以驻留。最终大小会变成 16 个字节。

● 进程空间

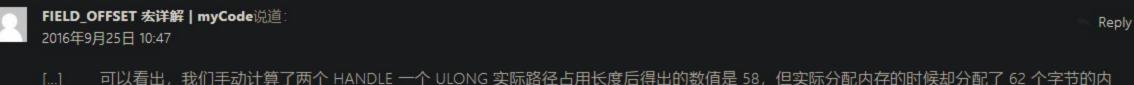
- 重载 new、delete 检测内存泄露
- AWE 内存管理

• Windows 堆内存管理

完成端口实现高性能服务端通信层的关键问题 数据全排列组合

2 Replies to "结构体成员在内存中的对齐方式"

Pingback: FIELD_OFFSET 宏详解 | myCode



可以看出,我们手动计算了两个 HANDLE 一个 ULONG 实际路径占用长度后得出的数值是 58,但实际分配内存的时候却分配了 62 个字节的内 存。后面我们拷贝数据到分配的内存中以后,有 4 个字节的空间没有使用而被浪费。那为什么 sizeof(PATH_INFO) 会比我们自己计算的长度多出来 4 个字 节呢? 这涉及到对结构体内存对齐的知识的了解了,请看本站曾经写过的一篇关于结构体内存对齐的非常详细的文章(图文,非常易懂) http://www.mycode.net.cn/language/cpp/1489.html [...]

发表评论

您的电子邮箱地址不会被公开。 必填项已用*标注

```
发表评论
```