第4篇:C/C++ 结构体及其数组的内存对齐

铁甲万能狗
自由开发者，专攻C++/Python后端开发(简书平台同号)

数据对象

本节谈及的内存对齐,而在进行这个话题之前，我先引入一个叫**数据对象**的概念。

大部份C语言教程的文章很少会提的一个概念就是数据对象(Data Object),简称**对象**,像基于C衍生出来其他高层语言所理解的"对象"是有些区别的.C中的对象更偏向于内存模型,同样C中的数据对象也适用于汇编,本来C就是"结构化"的汇编语言.数据对象就是本身两个属性.

- 数据值(value)
- 存储地址 (storage location)

也就是C中支持的所有数据类型定义出来的变量或由基本数据类型组合构造的用户定义类型(类类型/也叫结构体)，都统称**数据对象**,**一切类型皆为对象。**而被内存对齐的正是**数据对象。**

内存对齐也叫**字节对齐**(data aligment):就是数据对象的内存大小可以被2的N次方的整数整除,也就是说字节对齐可以用某个2的N次方的整数去对齐.

目前计算机的32位的CPU可以在每个时刻周期从内存读取4个字节并填充数据总线,而64位的CPU每个时刻周期可以读取8个字节,而C语言的的设计者为了遵循CPU的这种特性。就给C编译器，当然后来的C++编译器也继承这一特性，在对C/C++源码编译的时候会对源码中的数据对象自动执行内存对齐操作(对**数据对象**之前**填充**一些没用的字节块)。

备注：上面部分术语摘自相关的维基资料,很官腔套话是吧?!下面来些实质上示例讲解。

为什么要内存对齐?

因为我们要访问物理内存并能够在一次访问中获取整个数据，所以想象以下我们要从内存中读取一个4字节的int，而前两个字节在内存中的一个字中，而后两个字节在另外一个字中。我们不想分两次读取两个字然后将字节读取的字节再次装拼成一个整数，这是一种低效的内存访问。

底效的内存访问演示

未对齐的Data CPU多次加载word示例

这是是位于half-word边界上对齐的未对齐word。 为了对此进行操作,CPU必须做两个word加载。一个用于上half-word，一个用于下half-word。这两项加载操作将进入两个独立的寄存器。然后分别将高层

因此为了有效地一次性执行内存访问:直接读取四个字节或八个字节。就有了内存对齐这个玩意了。

内存对齐规律

1. **一般来说，对于需要X字节的原始数据类型，地址必须是X的倍数.**
2. **struct的起始地址决于其成员变量中的数据类型的sizeof()最大值作为对齐条件。**
3. **编译器在编译阶段对struct中未使用的内存空间执行填充操作,以确保字段对齐。**
4. **char类型不需要对齐,可自由分配任意可用的1字节尺寸的内存空间之中.**

基本数据类型的对齐要求

在不同的硬件架构和OS上执行的内存对齐是不一样,我们下面有个表,

数据类型	IA32	x86_64
char	1	1
short	2	2
int / float	4	4
double	4	8
任意pointer	4	8

需要特别指出的是

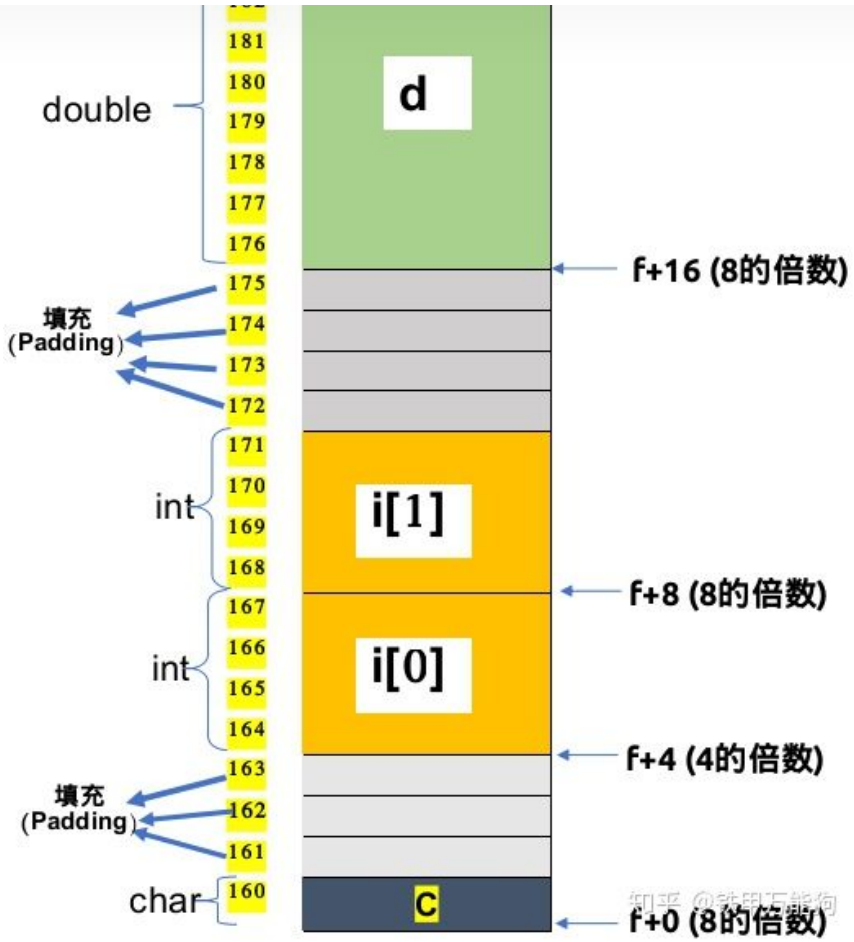
- 对于char类型没有任何对齐要求。
- 对于double类型即便是IA32的硬件架构,事实上可以通过gcc编译的时候指定命令行选项-malign-double,double类型也会以8字节对齐而不是4字节对齐。
  从这个表可知,不同类型的CPU,对齐操作是不一样的.

x86_64环境下的对齐示例

以下示例的左侧的数字表示内存地址,为了一目了然,使用十进制表示,并且在IA32 Windows 或 x86_64环境下，这个示例主要用来解析上面的对齐规则。

```
struct foo{
      char c;
      int [2];
      double d;
}
```

- 首先,在结构体内部,必须满足每个成员的对齐条件
- 纵观整个struct的成员变量
  - 每个结构体都由一个对齐条件整数K，K即是结构体中所有成员变量中的类型尺寸的最大值。
  - 起始地址和结构体的长度必须是整数K的倍数。



那么该结构体的对齐条件整数K是多少?

- 根据规则1和2:由于这个结构体的对齐条件是8字节,因为它是struct成员变量中对齐条件最大是8个字节,所以这个结构体的起始地址必须是8的倍数,从低位r地址算起当然是地址160.
- 根据规则4:由于char类型是没有任何对齐条件的限制,并且在结构体中第一个声明的变量,所以它就落在结构体的起始地址,也就是地址160的位置.
- 根据规则1,由于int数组类型的每个元素占用4个字节,那么int类型的起始地址以4的倍数开始的那么就自然落到了164这个地址,紧挨着的i[2]元素自然落在168这个地址.
- 成员变量c和int[0]之间有还有未使用的内存空间,根据规则3,会填充未使用的字节.
- 根据规则1,Double类型的成员以8的倍数对齐,自然落在自struct起始地址起第16个字节的位置,即地址176的地方算起占用8个字节表示成员变量d的数据.
- 根据规则3,另外编译器会填充成员变量d和数组元素int[1]之间未使用的内存位置.

IA32 Linux环境下，上面示例对齐条件整数K是多少呢？你可以自行思考一下。

节省内存空间

从上面的例子我们得知,虽然内存对齐操作有助于优化CPU对内存的访问,但会带来一下副作用，就是会浪费一些内存空间.但这个问题不能全赖在编译器身上，作为程序员不良的写码习惯也是很大关系滴!!

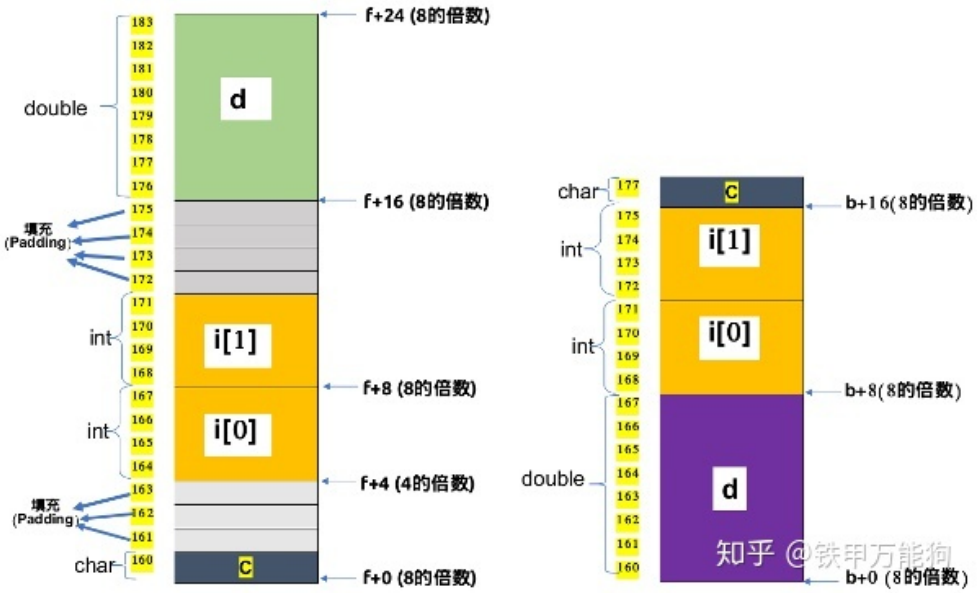
我们在看看下面的示例从下图可以得知**结构体内部成员变量声明的先后顺序和编译器对内存对齐后,结构体所占的内存空间有很大的关系**.

- 下图左手边的内存布局是x86_64架构下对应的C代码

```
struct{
```

- 下图右手边的内存布局是x86_64架构下对应的C代码

```
struct{
    double   d;
    int      i[2];
    char     c;
} *b;
```



由上面的的内存布局对比,我们得到得到一个基本结论:
在struct内部,将成员变量按照其类型的sizeof()值由大到小,依存声明的话能够不仅可以最大限度地减少编译器填充未使用内存块的操作,而且填充的内存块出现的次数越少,那么CPU每次从对应内存块中加载数据到寄存器中执行shift运算的次数也会相应地减少。同时能够兼顾CPU对对齐内存的访问.

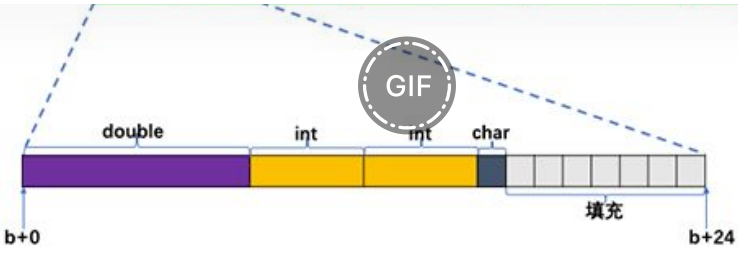
有读者对我上面的结果提出疑问，无论在32位还是64位计算机上，sizeof(double)始终为8个字节。 不同之处在于在32位Linux系统上，double对齐倍数是4字节数据类型一样。 要将double对齐的倍数为8个字节，请使用-malign-double（编译时选项)。

结构体数组的内存分布

灰常不幸的是!!，结构体的数组是无法满足上面的所讲的节省内存的特性的。

```
typedef struct{
      double   d;
      int      i[2];
      char     c;
  }  b;

b[4];
```



我们已经知道结构体b的对齐条件是8的倍数,由于结构体b的占据17个字节的内存空间,因此和它相关的数组中的每个元素占用内存空间必须要达到24个字节,才能达成每个元素的对齐条件是8的倍数, 因此每个结构体元素,编译器还需要为每个元素填充7个字节囧rz...

发布于 2020-08-15 02:02

汇编语言 寄存器 C++

文章被以下专栏收录

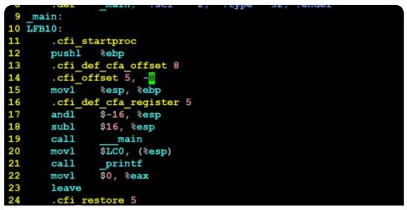
C/C++内存管理
侧重C/C++内存模型，反编译分析

推荐阅读

第7篇：C/C++程序栈-帧

本篇详细讲解有关IA32约定中的程序栈帧，我栈顶到栈底的方向逐一回顾一下。首先分析栈底层的内存细节，需要你具备非常基础的汇编语法知识即可。如果你毫无概念，可以自行查看我以...

铁甲万能狗 发表于C/C++...



从C语言到汇编（七）结构体赋值与复制

pk201... 发表于从C语言到...

【C/C++】C语言之程分析

Boots 发表于

2 条评论 ⇌ 切换为时间排序

写下你的评论...

黄辛未 2021-08-31
建议讲一下类的内存布局