C++ C / C++

关注者
214

如何理解 struct 的内存对齐?

[图片]...显示全部

1 条评论 分享 邀请回答 ...

关注问题

14 个回答 默认排序

張道遠

39 人赞同了该回答

> padding = (align - (offset mod align)) mod align
> new offset = offset + padding = offset + (align - (offset mod align)) mod align
>
> the total size of the structure should be a multiple of the largest alignment of any structure member
>
> ------ wikipedia

3 个因素导致现在的地址对齐约定：

1. 生活很艰难
2. 世界多姿多彩，世上有各种不同的人存在
3. 但我们还是要在一起呀在一起

以下以最简，理想的模型进行讨论。

一个最小存储单位为 8 字节的内存来说。访问地址1, 大小为 4 字节的数据。只需读取地址 0 的 8 字节的数据。然后在出口处移位下就行。因为只需在出口处做一次处理，所以可以不计成本进行移位优化，但这种优化只能在部件内部或者部件组内部进行。不同组件的交互部分对“对齐”还是有不同的看法的。又因为 RISC CPU 的设计，大多精简指令集的指令长就是字长。而指令还需区分取立即数和各种 action, 一字长的指令无法全部用来表示地址空间。综上，大多 RISC CPU 强制地址对齐，地址的低位脑补成 0。顺便也减少了地址线的宽度。

访问内存的速度是非常非常非常……慢的。再加上 CPU 及其指令设计的限制。在这种艰苦条件下，我们必须无所不用其极地减少随机内存访问次数：

- 在一个访问周期里读写最多，但不更多的数据。也就是一字长大小的数据。
- 对于 CPU, 内存的最小存储单元的大小为最大，但不更大的 CPU 字长。

基于此，一般的 RISC CPU 的地址线宽度为 $wordSize - log_2 \frac{wordSize}{byteSize}$. 比如一个 32 位 CPU 的字长是 32bit, 字节大小为 8bit, 那么地址线宽度为 $32 - log_2 \frac{32}{8} = 30$ 可选择 2^{30} 个地址单位， 每个单位的大小是 4 字节 于是总共可管理 $2^{30} * 4 = 2^{30+2}$ 字节的内存。这也是逻辑地址最低 $log_2 \frac{wordSize}{byteSize}$ 位总是为 0 的来历了。对于 32bit 字长的 CPU, 就是最低 2 位为 0 了。
注意，此段所述只是基于 de-facto 习惯(1 byte = 8 bit, mem 空间大小为 $2^{wordSize}$ byte), 为了便于讨论而做的假设。并无特别的意义和强行规定。

以一个字长为 4 字节的 RISC CPU 来进行讨论。单位为字节。
此时，如果我们需要访问地址为 2 的大小为 1 字长也就是 4 字节的数据，也就是 2-5 的数据. 地址 2 mod 大小 4 = 2 不为 0. 这是未对齐的访问。(地址线以二进制表示。最后两位空置，所以始终为逻辑 0)我们需要将地址线设为 0(00) 读出 0-3, 取 2-3, 然后将地址线设为 1(00) 读出 4-7 取 4-5, 然后再合成所需数据。共两次访问。如果要访问地址为 0(00) 或者 1(00) 的大小为 1 字长的数据。则一次访问即可。
这就是地址不对齐导致访问变慢的来历了。

此时若要访问 地址 2 的半个字长大小的数据。也就是 2-3 的数据。我们可以将地址设为 0(00) 读出 0-3 的数据，然后将其在寄存器中右移 2 字节即可。

那么，问题来了。既然如此，我也可以访问地址 1, 大小为 2 字节的数据啊。也就是 1-2 的数据。
将地址线设成 0(00) 读出数据，然后在寄存器内左移 1 字节，再右移 2 字节就行了啊。
这时候 1 mod 2= 1, 不为 0, 但需要访问内存的次数还是一次。
但世界上有许多地方，那儿的 CPU 字长只有 2 字节。当数据到那些地方去旅行时。那儿的 CPU 访问地址为 1, 大小为 2 字节的数据的时候还是需要两次的。

那么，问题又来了。字长为 4 字节的 CPU, 访问 8 字节长度的数据，这个数据反正始终都要读两次，那么它不对齐也是可以的呀。只要他的地址 mod 4 为 0 就可以了。
但世界上还有些地方，那儿的 CPU 字长是 8 字节的，当数据到那些地方去旅行时。那儿的 CPU 访问大小为 8 字节的数据，若其地址 mod 8 为 0 时，只要读一次就够了。此时读两次就是一种浪费了。

我们的世界是个艰难但又多姿多彩的世界。为了大家的数据都有一个兼容且一致的模型，方便交换，分析。我们郑重做出约定：

大小为 size 的字段，他的结构内偏移 offset 需符合 offset mod size 为 0.
引用的 wikipedia 的第一段就是对这句话的精确表述。

最后，问题又来了。

struct hi { let: 4 // padding 4 us: 8 // padding 0 play: 1 // padding 1 together: 2 // padding ? }

together 字段的 padding 是要多少？是的 padding 0 就行了。所以大小是 8 + 8 + 2 + 2= 20
那为什么 gcc 告诉我们应该是 24 呢。

因为我们的世界不是孤单的世界。

数据们可以欢乐地组成团队。

hi group[2];

如果我们不能相互体谅，自私地将最后的 padding 设为 0 的话。

假设第一个 hi 位于地址 0, 那么第二个 hi 就得从地址 20 开始了。此时 us 的地址是
$$20 + sizeOf(let) + padding(let) = 20 + 4 + 4 = 28$$
, 而 28 mod 8= 4 不为 0.
如果 hi 的大小为其中最大单元的整数倍也就是 8 * 3= 24 的话。那么 第二个 hi 的 us 字段的地址是 24+8= 32. 而 32 mod 8= 0, 对齐了。所以，最后我们还需要 padding 4 字节。
在这个不孤单的世界里，为了同一类数据能和谐相处。所以我们郑重做出约定：

整个结构的大小必须是其中最大字段大小的整数倍。

于是，不管是在一个数组里没羞没臊地在一起。还是在这个如此多姿多彩各不相同的世界里到处旅行。数据们的美好的生活都可以快速，和谐，一致地进行啦。

最后，若题主有闲，推荐看一下哈佛的 CS101, From NAND to Tetris 课程。从最简单的逻辑门开始，自己动手打造一遍 latch, flip-flop, register，RAM, ALU, CPU, assembler, compiler. 相信到时候你会有更深的体会。

编辑于 2017-03-16

匿名用户

19 人赞同了该回答

ludx/The-Lost-Art-of-C-Structure-Packing · GitHub

发布于 2015-01-30

Courtier
courtier.cc 公众号:CourtierCC

45 人赞同了该回答

5分钟教你秒杀:

前言:(10秒)

前面的各位的方法很好，很传神，不适合速成，想要速成还得看我这种小屁孩写的

公式:(20秒)

公式1:**前面的地址必须是后面的地址正数倍,不是就补齐**

公式2:**整个Struct的地址必须是最大字节的整数倍**

30秒....敌人还有三十秒到达战场...来看看下面的Ex(不是前任...!)

Struct E1

{ int a;char b; char c}e1;

第一地址肯定存放a是4Byte地址，第二地址,b要1Byte的地址，来欢迎**公式一**登场： 4 == 1*N (N等于正整数) 答"**是**"!地址现在为5Byte，下一个c要1Byte的地址同上，所以，就是6Byte。来欢迎**公式二**登场，在这个E1中最大的字节是4，而我们的地址字节是6，4的整数倍不是6，所以，要加2Byte（总地址），So，整个字节为8!

CAUTION:

每个**特定平台**上的编译器都有自己的默认"**对齐系数**"。可以通过预编译命令#pragma pack(n)

喝下茶，缓缓

编辑于 2015-01-31

知乎用户

1 人赞同了该回答

看我背书大法:

许多计算机系统对基本数据类型合法地址做出了一些限制，要求某种类型对象的地址必须是某个值K（通常是2、4或8）的倍数。这种对齐限制简化了形成处理器和存储器系统之间接口的硬件设计。

所以说对齐就是为了优化硬件效率。

另外Linux和Windows的对齐策略似乎是不一样的。

编辑于 2015-01-30

知乎用户

空间换时间不是很常见嘛

发布于 2015-01-30

知乎用户

3 人赞同了该回答

背书式：各成员变量存放的起始地址相对于结构的起始地址的偏移量必须为该变量的类型所占用的字节数的倍数 各成员变量在存放的时候根据在结构中出现的顺序依次申请空间 同时按照上面的对齐方式调整位置 空缺的字节自动填充 同时为了确保结构的大小为结构的字节边界数(即该结构中占用最大的空间的类型的字节数)的倍数，所以在为最后一个成员变量申请空间后 还会根据需要自动填充空缺的字节

多!

简!

单!

发布于 2015-05-02

KDF5000
一点都不文艺的程序猿

2 人赞同了该回答

字节对齐主要是为了提高内存的访问效率，比如intel 32为cpu，每个总线周期都是从偶地址开始读取32位的内存数据，如果数据存放地址不是从偶数开始，则可能出现需要两个总线周期才能读取到想要的数据，因此需要在内存中存放数据时进行对齐。

通常我们说字节对齐很多时候都是说struct结构体的内存对齐，比如下面的结构体:

```
struct A{
    char a;
    int b;
    short c;
}
```

在32位机器上char 占1个字节，int 占4个字节，short占2个字节，一共占用7个字节.但是实际真的是这样吗?

我们先看下面程序的输出:

```
#include <stdio.h>

struct A{
    char a;
    int b;
    short c;
};
int main(){
    struct A a;
    printf("A: %ld\n", sizeof(a));
    return 0;
}
```

测试输出的结果是A: 12, 比计算的7多了5个字节。这个就是因为编译器在编译的时候进行了内存对齐导致的。

内存对齐主要遵循下面三个原则:

1. 结构体变量的**起始地址**能够被其最宽的成员大小整除
2. 结构体每个成员相对于**起始地址的偏移**能够被其**自身大小整除**，如果不能则在**前一个成员后面**补充字节
3. 结构体总体大小能够**被最宽的成员的大小**整除，如不能则在**后面**补充字节

其实这里有点不严谨，编译器在编译的时候是可以指定对齐大小的，实际使用的有效对齐其实是取指定大小和自身大小的最小值，一般默认的对齐大小是4。

再回到上面的例子，如果默认的对齐大小是4，结构体a的其实地址为0x0000，能够被最宽的数据成员大小(这里是int， 大小为4，有效对齐大小也是4)整除，姑char a的从0x0000开始存放占用一个字节即0x0000~0x0001，然后是int b，其大小为4，故要满足2，需要从0x0004开始，所以在char a后填充三个字节，因此a对齐后占用的空间是0x0000~0x0003，b占用的空间是0x0004~0x0007, 然后是short c其大小是2，故从0x0008开始占用两个字节，即0x0008~0x000A。 此时整个结构体占用的空间是0x0000~0x000A， 占用11个字节，11%4 ！ = 0, 不满足第三个原则，所以需要在后面补充一个字节，即最后内存对齐后占用的空间是0x0000~0x000B，一共12个字节。

编辑于 2017-08-04

▲ 2 ▼ 添加评论 分享 收藏 感谢 收起

羊垒
油女一族，优秀的Bug制造者。\

2 人赞同了该回答

一个简单的算法就是：
如果n位对齐，那么n位以下的所有基本类型，地址必须为本类型长度的整数倍；
n位以上的基本类型，保持n位对齐。

编辑于 2015-01-30

▲ 2 ▼ 添加评论 分享 收藏 感谢

AloneMonkey
猿，是改变世界的动物！

1 人赞同了该回答

blogfshare.com/memory-a...

发布于 2015-01-30

▲ 1 ▼ 1 条评论 分享 收藏 感谢

Zbtrik
跟帖局副局长

这篇博客给的例子挺详细内存对齐规则之我见

发布于 2017-04-02

▲ 0 ▼ 添加评论 分享 收藏 感谢

知乎用户

CPU是按字读取内存。所以内存对齐的话，不会出现某个类型的数据读一半的情况，需要再二次读取内存。可以提升访问效率。

Reference: Purpose of memory alignment

发布于 2017-03-24

▲ 0 ▼ 添加评论 分享 收藏 感谢

xjsxjtu

我认为归根结底是编译器想通过空间换时间，通过适当增加padding，使每个成员的访问都在一个指令里完成，而不需要两次访问再拼接。
理解有误的话请无视...

发布于 2015-01-31

▲ 0 ▼ 1 条评论 分享 收藏 感谢

lesten
荣耀的背后刻着一道孤独。

就题主贴出的图
如果 都是 自带的数据类型，则结构体内部按照占用内存最大的数据类型对齐。
E1~E6 都是。 拿E3举例，E3中long long占用8字节（因为题主context int占用了四字节，所以都是按int 4字节计算）。所以E3中 longlong 前面的 a(1字节)， b(2字节)， c(1字节) 总共是4字节，因为结构体内部按照最大的8字节对齐 所以E3 是16字节。
如果 有自定义数据类型，并且自定义数据类型 比 自带数据类型占用内存大时，将自定义数据类型占用内存扩充至最大自带数据类型的整数倍进行对齐。

发布于 2015-01-30

▲ 0 ▼ 添加评论 分享 收藏 感谢

私家课 · Live 推荐

刘看山 · 知乎指南 · 知
侵权举报 · 网上有害信
违法和不良信息举报：
儿童色情信息举报专区
联系我们 © 2017 知乎

下载知乎
与世界分享

相关问题

知乎用户

用空间换时间
也可能是编译器优化不到位

编辑于 2015-01-30