C语言对齐

一、为什么要对齐

不同的处理器访问内存的方法不同，一般来讲都支持单字节访问。为了提高效率16位机可能还支持按2字节访问，32位机可能还支持按4字节访问。

按多字节访问一般需要地址对齐。比如按2字节访问时，要求地址的最低位为0，即按2字节对齐。按4字节访问时，要求地址的最低2位为0，即按4字节对齐。

如果地址是符合对齐要求的，就可以实现多字节一次访问，提高访问效率。否则的话则须拆成单个字节逐个访问。

二、C语言的对齐

C语言是跨平台的编程语言，他默认的对齐方式是按照变量的长度进行对齐。比如char为一个字节对齐，short为2个字节对齐，long为4个字节对齐。

为了提高内存的利用率，对于全局变量，编译器会把所有同长度的变量组合在一起分配空间，空间的首地址符合对齐关系。比如给所有非零初值的单字节变量分配一块空间。

例1：char a = 1; short b = 0; char c = 0; char d = 3; short e;

那么“a”和“d”会被分配在同一块空间，空间首地址为1字节对齐，“b”和“e”会被分配在同一块空间，空间首地址为2字节对齐。（无初值一般等同于初值为零）

三、结构体的对齐

结构体里面的变量默认是单独符合对齐规律的（因为结构体的变量必须连续分配，不能够拆分开统一分配）。

通过#pragma pack(x)可以改变默认的对齐规律，把大于“x”字节对齐的变量压缩到“x”字节对齐，小于“x”字节对齐的变量不受影响。

例2：

```
typedef struct
{
    u8 a;
    u16 b;
    u32 c;
}test1;
```

test1的内存分配如下表：

a	填充	b	b	c	c	c	c

```
typedef struct
{
    u32 c;
    u16 b;
    u8 a;
}test2;
```

test2的内存分配如下表：

c	c	c	c	b	b	a	填充

```
#pragma pack(1)   //对于16位和32位，使用1字节压缩对齐会严重影响效率，慎用！
typedef struct
{
    u8 a;
    u16 b;
    u32 c;
}test3;
#pragma pack()   //恢复默认压缩字节数
```

test3的内存分配如下表：

a	b	b	c	c	c	c

四、结构体的尾部填充

一些编译器（如KEIL）会在结构体的尾部填充，使结构体的大小为其内部最大对齐数的整数倍。比如结构体内部有4字节对齐的变量，那么结构体就会被填充成4的倍数。

（作者猜想：可能结构体是按其最大对齐数统一分配空间的。比如结构体的最大对齐数为2，那么他就和short型的变量统一分配空间。尾部填充成2的倍数，使后面的变量符合对齐规律。）

可以通过#pragma pack(x)，把最大对齐数压缩到“x”，从而改变结构体尾部的填充。

五、常用相关函数

取结构体长度的函数：sizeof()

取结构体成员的相对位置的函数：offsetof(结构名，成员名)。

注：使用offsetof函数需包含要<stddef.h>标准头文件。

例3：

sizeof(test1)等于8；sizeof(test2)等于8；sizeof(test3)等于7；

offsetof(test1,c)等于4；offsetof(test2,a)等于6；offsetof(test3,c)等于3；

六、应用举例

在实际应用中，可能需要对结构体按字节校验，然后把校验的结果保存到结构体的末尾。

例4：

```
typedef struct
{
    u8 a;
    u16 b;
    u32 c;
    u16 crc;
}test4;
```

在KEIL下面编译，得到的分配如下：

a	填充	b	b	c	c	c	c	crc	crc	填充	填充

在计算校验的时候，我们一般需要得到除了校验字符以外其他数据的长度。对于test4来说其他数据的长度就是8。

用sizeof表示就是sizeof (test4)-2。这里的2为尾部填充字节数，不宜理解，且需根据实际尾部填充情况变化。

用offsetof表示就是offsetof(test4,crc)。这是一种完美的表示方式，强烈推荐。

zhugean 2013-8-19