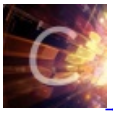
对堆栈8字节对齐问题的讨论

原创

MacLodge 于 2018-05-07 11:34:10 发布 2800 收藏 18
分类专栏： C/C++ 文章标签： 字节对齐 C语言

本文链接：https://blog.csdn.net/hsl416604093/article/details/80223261



目录

一、字节对齐原则

1、结构(**struct**)(或联合(**union**)) 中的第一个数据成员放在 **offset** 为 **0** 的地方，以后每个数据成员存储的起始位置要从该成员大小或者成员的子成员大小（只要该成员有子成员，比如说是数组，结构体等）的整数倍开始(比如 **int** 型变量在 32 位编译环境下为 4 字节,则要从 4 的整数倍地址开始存储；

2、如果一个结构里有某些结构体成员,则结构体成员要从其内部最大元素大小的整数倍地址开始存储.(如：**struct a** 里存有 **struct b, b** 里有 **char, int , double** 等元素,那 **b** 应该从 **8** 的整数倍开始存储.)；

3、结构体的总大小，也就是 **sizeof** 的结果，必须是其内部最大成员的整数倍，不足的要补齐；

```
typedef struct Demo_A
{
    double length;  // 0 - 7;
    int id;         // 8 - 11;
    char op;        // 12 - 13;
    float weight;   // 16 - 19;
}AA;


typedef struct Demo_B
{
    char name[5];   // 0 - 4 ;
    int id;         // 8 - 11 ;
    double score;   // 16 - 23 ;
    float ui;       // 24 - 27 ;
    short grade;    // 28 - 31 ;
    char weight;    // 32 ;
}BB;
```

//总大小为最大成员变量大小的整数倍，sizeof(AA) = 24； sizeof(BB) = 32；

#pragma pack()

在代码前加一句 #pragma pack(1)，会发现 sizeof(AA) = 17； sizeof(BB) = 24；

AA 是 8+4+1+4=17;

BB 是 5+4+8+4+2+1=24;

这就是理想中的没有内存对齐的情况，所以#pragma pack(1)是告诉编译器,所有的对齐都按照1的整数倍对齐,换句话说就是没有对齐规则。

即#pragma pack(n)就是所有的对齐都按照n的整数倍对齐。

ps:

Vc,Vs等编译器默认是 **#pragma pack(8)**，所以测试我们的规则会正常；

gcc 默认是 #pragma pack(4)，并且 gcc 只支持 1, 2, 4 对齐。套用三原则里计算的对齐值是不能大于 **#pragma pack** 指定的n值。

另参考：C语言结构体字节对齐原则

二、为什么要保证堆栈8字节对齐

AAPCS 规则要求堆栈保持 8 字节对齐。如果不对齐，调用一般的函数也是没问题的。但是当调用需要严格遵守 AAPCS 规则的函数时可能会出错。

例如调用 sprintf 输出一个浮点数时，栈必须是 8 字节对齐的，否则结果可能会出错。

实验验证：

```
#include "stdio.h"
#include "string.h"
float fff=1.234;
char buf[128];
int main(void)
{
    sprintf(buf,"%.3f\n\r",fff);//A
    while(1);
}
```

1.在 A 处设置断点，让程序全速运行至 A
2.在 MDK 中修改 MSP 的值使 MSP 满足 8 字节对齐
3.全速运行程序，观察 buf 中的字符为 1.234 结果正确
4.回到第 2 步，修改 MSP 使之只满足 4 字节对齐而不满足 8 字节对齐
5.全速运行程序，观察 buf 中的字符为 -2.000 结果错误

该实验证明了调用 sprintf 输出一个浮点数必须要保证栈 8 字节对齐。

三、编译器为我们做了什么

先看一个实验

```
#include "stdio.h"
#include "string.h"
float fff=1.234;
char buf[128];

void fun(int a,int b,int c,int d)
{
    int v;
    v=v;
}
void test(void)
{}
int main(void)
{
    fun(1,2,3,4);
    test();       //A
//  sprintf(buf,"%.3f\n\r",fff);
    while(1);
}
```

保证初始的时候堆栈是 8 字节对齐的；
1.在 A 处设置断点；
2.全速运行至 A，观察 MSP=0x2000025c，没有 8 字节对齐；
3.略微修改一下 main 函数代码如下，其他部分代码不变；

```
int main(void)
{
    fun(1,2,3,4);
//  test();
    sprintf(buf,"%.3f\n\r",fff);//A
    while(1);
}
```

4.同样在 A 处设置断点；
5.全速运行至 A，观察 MSP=0x200002d8，这次 8 字节对齐了；

这个实验说明了如果编译器发现了某个函数需要调用浮点库时会自动调整编译生成的汇编代码，从而保证调用这些浮点库函数时堆栈是8字节对齐的。换句话说如果我们保证了栈初始的时候是8字节对齐的，那么编译器可以保证以后调用浮点库时堆栈仍是8字节对齐的。

四、os下应该怎样设置任务堆栈

由上面的讨论可知给任务分配栈时需要保证栈是 8 字节对齐的，不然在该任务中凡是调用 sprintf 的函数均会出错，因为栈一开始就是不对齐的。

是否保证了栈初始是8字节对齐了就万事大吉了呢。no！大家请看一种特殊的情况：

```
#include "stdio.h"
#include "string.h"
float fff=1.234;
char buf[128];
void fun(int a,int b,int c,int d)
{
    int v;
    v=v;
}
int main(void)
{
    fun(1,2,3,4);
    while(1);
}
void SVC_Handler(void)
{
    sprintf(buf,"%.3f\n\r",fff);//B
}
```

mian函数的反汇编如下：

```
0x080001DC B500 PUSH {lr}
0x080001DE 2304 MOVS r3,#0x04 ;A
0x080001E0 2203 MOVS r2,#0x03
0x080001E2 2102 MOVS r1,#0x02
0x080001E4 2001 MOVS r0,#0x01
0x080001E6 F7FFFFF5 BL.W fun (0x080001D4)
0x080001EA BF00 NOP
0x080001EC E7FE B 0x080001EC
```

保证初始的时候堆栈是 8 字节对齐的；
1.在 A 处设置断点；
2.全速运行至 A，观察此时 MSP=0x200002e4 未对齐；
3.在 MDK 中将 SVC 的挂起位置 1；
4.在 B 处设置断点；
5.全速运行至 B，观察此时 MSP=0x200002b4 未对齐；
6.继续全速执行，观察 buf 中的字符为: -2.000 出错了；

这个实验说明了即使保证栈初始是 8 字节对齐的，编译器也只能保证在调用 sprintf 那个时刻栈是 8 字节对齐的，但不能保证任意时刻栈都是 8 字节对齐的，如果恰巧在 MSP 没有 8 字节对齐的时刻发生了中断，而中断中又调用了 sprintf，这种情况下仍会出错。

五、Cortex-M3 内核为我们做了什么

Cortex-M3 内核提供了一种硬件机制来解决上述这种中断中栈不对齐问题。

CM3 中可以把 NVIC 配置控制寄存器的 STKALIGN 置位，来保证中断中的栈 8 字节对齐。

具体实现过程如下：当发生中断时由硬件自动检测 MSP 是否 8 字节对齐，如果对齐了，则不进行任何操作，如果没有对齐，则自动将 MSP 减 4 这样便对齐了，同时将 xPSR 的第 9 位置位来记录这个 MSP 的非正常的变化，在中断返回若发现 xPSR 的第 9 位是置位的则自动将 MSP 加 4 调整回原来的值。

实验验证：

```
#include "stdio.h"
#include "string.h"
float fff=1.234;
char buf[128];
void fun(int a,int b,int c,int d)
{
    int v;
    v=v;
}
int main(void)
{
    fun(1,2,3,4);
    while(1);
}
void SVC_Handler(void)
{
    sprintf(buf,"%.3f\n\r",fff);//B
}
```

mian函数的反汇编如下：

```
0x080001DC B500 PUSH {lr}
0x080001DE 2304 MOVS r3,#0x04 ;A
0x080001E0 2203 MOVS r2,#0x03
0x080001E2 2102 MOVS r1,#0x02
0x080001E4 2001 MOVS r0,#0x01
0x080001E6 F7FFFFF5 BL.W fun (0x080001D4)
0x080001EA BF00 NOP
0x080001EC E7FE B 0x080001EC
```

1.在 A 处设置断点；

2.全速运行至 A，观察此时 MSP=0x200002e4 未对齐；

3.在 MDK 中将 SVC 的挂起位置 1，同时将 0xE000ED14 处的值由 0x00000000 改为 0x00000200 (即将 NVIC 配置控制寄存器的 STKALIGN 置位)
4.在 B 处设置断点；
5.全速运行至 B，观察此时 MSP=0x200002b0 对齐了；
6.观察中断返回时的 MSP=0x200002e4 调整回来了；
7.继续全速执行，观察 buf 中的字符为:1.234 正确；

这个实验说明了将NVIC配置控制寄存器的STKALIGN置位可以保护中断时栈仍是8字节对齐

六、总结

综上所述，为了能够安全的使用严格遵守AAPCS规则的函数(比如sprintf)需要做到以下几点：
1.保证MSP在初始的时候是8字节对齐的
2.如果用到OS的话需要保证给每个任务分配的栈是保持8字节对齐的
3.如果用的是基于CM3内核的处理器需将NVIC配置控制寄存器的STKALIGN置位

七、使用系统时的操作(uCOS-Ⅲ)

OS_TCB FloatTaskTCB; //任务控制块；

__align(8) CPU_STK FLOAT_TASK_STK[FLOAT_STK_SIZE]; //任务堆栈；

其中 __align(8) 是保证开辟堆栈保持8字节对齐；