前言
代码：版本1
分析原因
执行char *pData = 0;
执行do_work(pData, 128);
执行p = (char *)malloc(size + 1);
代码：版本2
执行char *pData = 0;
执行do_malloc(&pData, 128);
执行*p = (char *)malloc(size + 1);

前言

今天同事问了一个问题：在函数参数中传递指针的指针，很常用的一个场景，重新梳理一下记录于此，以后如果有类似的问题直接发这篇小总结就可以了。

代码：版本1

```
void do_malloc(char *p, int size)
{
    p = (char *)malloc(size + 1);
    memset(p, 0, size + 1);
}


int main(int argc, char *argv[])
{
    char *pData = 0;
    do_malloc(pData, 128);
    sprintf(pData, "%s", "abc");
    printf(pData);
    return 0;
}
```

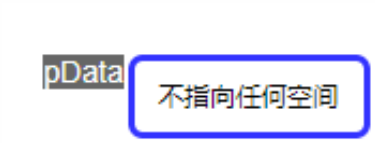
代码本意是：do_work()函数向系统堆空间申请size个字节的空间，然后返回给main函数中的pData指针。
但是，执行的时候报错：Segmentation fault (core dumped)。

分析原因

我们可以把char*类型的指针看成一个遥控器，如果给这个指针赋值，就相当于把这个遥控器与一个设备进行绑定，可以通过遥控器来控制这个设备。

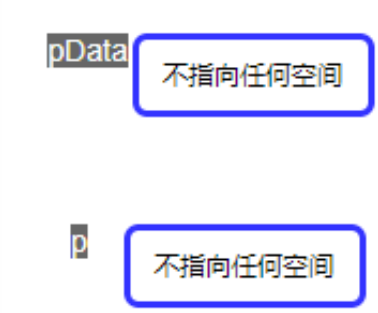
执行char *pData = 0;

pData内容为空，相当于这个遥控器没有与任何设备绑定，如下图：



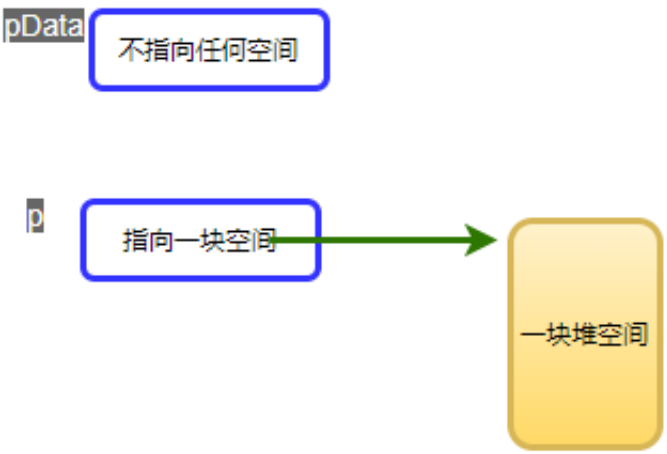
执行do_work(pData, 128);

这里传递的参数是pData本身，所以进入void do_work(char *p, int size)函数之后，实参pData的内容就赋值给形参p，所以指针p的内容也为空，也就是说：p这个遥控器也没有与任何设备绑定，如下图：



执行p = (char *)malloc(size + 1);

这句话的作用是把申请到的堆空间的首地址，赋值给p。就是说：现在p指向了内存中的一块空间，就相当于一个p这个遥控器与一个设备进行绑定了，可以控制这个设备了，如下图：



到这里就已经看到程序崩溃的原因了：虽然给指针p赋值了，但是实参pData中的内容一直为空，因此从do_malloc函数返回之后，pData仍然是一个空指针，所以就崩溃了。当然，p指向的堆空间也就泄露了。

代码：版本2

代码的本意是在do_malloc函数中申请堆空间，然后把这块空间的首地址赋值给pData。在do_malloc函数中，调用系统函数malloc成功之后返回所分配空间的首地址，关键是要把这个首地址送给pData指针，也就是说要让pData指针变量中的值等于这个堆空间的首地址。

那应该如何通过中间的一个函数来完成这个功能呢，如下代码：

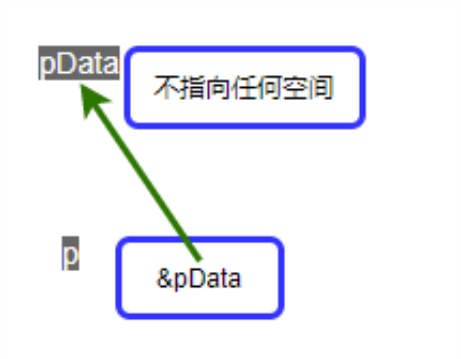
```
void do_malloc(char **p, int size)
{
    *p = (char *)malloc(size + 1);
    memset(*p, 0, size + 1);
}


int main(int argc, char *argv[])
{
    char *pData = 0;
    do_malloc(&pData, 128);
    sprintf(pData, "%s", "abc");
    printf(pData);
    return 0;
}
```

执行char *pData = 0;

这一句没有变化。

执行do_malloc(&pData, 128);

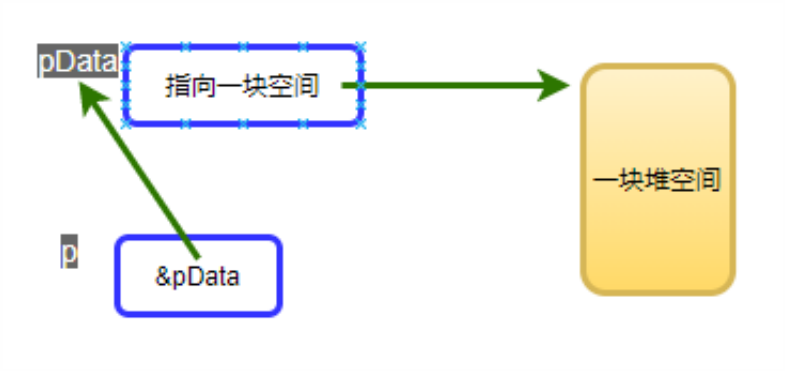
把pData指针的地址作为实参进行传递，因为pData本身就是一个指针，加上取地址符&，就是指针的指针(二级指针)，因此do_malloc函数的第一个参数就要定义成char**类型，此时示意如图：



p此时是一个二级指针，参数赋值之后，p里面的内容就变成了pData这个指针变量的地址，也就是说p指向了pData这个变量。

执行*p = (char *)malloc(size + 1);

这句话首先搞明白*p是啥意思，刚才说了，p是一个指针，它指向了pData这个变量。那么在p前面加上取值操作符*，就相当于是取出指针p中的值，它里面的值就是pData！
因此，malloc函数返回的堆空间首地址，就相当于是赋值给了pData，如下图：



此时，pData这个遥控器就与分配的这块堆空间绑定在一起，随后再操作pData就没有问题了。

【原创声明】

> 作者：道哥(公众号: IOT物联网小镇)
>
> 知乎：道哥
>
> B站：道哥分享
>
> 掘金：道哥分享
>
> CSDN：道哥分享

如果觉得文章不错，请转发、分享给您的朋友。

我会把十多年嵌入式开发中的项目实战经验进行总结、分享，相信不会让你失望的！

长按下图二维码关注，每篇文章都有干货。

推荐阅读

[1] 原来gdb的底层调试原理这么简单
[2] 生产者和消费者模式中的双缓冲技术
[3] 深入LUA脚本语言，让你彻底明白调试原理
[4] 一步步分析-如何用C实现面向对象编程
[5] 关于加密、证书的那些事