VxWorks 系统下 Page Fault 浅析

相信很多人在 X86 的设备里执行 VxWorks 应用时，遇到过 Page Fault 错误。

```
Page Fault

Page Dir Base   : 0x015df000
Esp0 0x01723bf0 : 0xeeeeeeee, 0xeeeeeeee, 0xeeeeeeee, 0xeeeeeeee
Esp0 0x01723c00 : 0xeeeeeeee, 0xeeeeeeee, 0xeeeeeeee, 0xeeeeeeee
Program Counter : 0x004eab8b
Code Selector   : 0x00000008
Eflags Register : 0x00010206
Error Code      : 0x00000002
Page Fault Addr : 0x00000020

Task: 0x1626b98 "tShell0"
0x1626b98 (tShell0): task 0x1626b98 has had a failure and has been stopped.
0x1626b98 (tShell0): The task been terminated because it
that raised the signal 11.
```

这是 X86 CPU 的 14 号异常，指的是访问存储器的指令发生了页异常。

我以往的经验，这样情况多是地址错误引起的，而主要的地址错误就是栈溢出(或者叫栈越界)。我们写个例子模拟一下

```
1  #include <vxWorks.h>
2  #include <taskLib.h>
3  #include <stdIo.h>
4  void aaa()
5  {
6      TASK_DESC taskDesc;
7      char sss[0x10001]="hello";
8
9      taskInfoGet(0,&taskDesc);
10     printf("td_stackSize=0x%x=%d\n",taskDesc.td_stackSize,taskDesc.td_stackSize)
11     printf("td_pStackBase=0x%x\n",taskDesc.td_pStackBase);
12     printf("td_pStackEnd=0x%x\n",taskDesc.td_pStackEnd);
13     printf("sss=0x%x\n",sss);
14     printf("sss[0x10000]=0x%x\n",sss+0x10000);
15 }
```

在 X86 的 VxWorks 里启个任务执行它

```
-> sp aaa
Task spawned: id = 0x162e250, name = t1
value = 23257680 = 0x162e250 = 'P'
-> td_stackSize=0x4e20=20000
td_pStackBase=0x172aaa0
td_pStackEnd=0x1725c80
sss=0x171a9db
sss[0x10000]=0x172a9db
```

可以看到任务 t1 的栈用的是默认值 20000，但是代码中有个 0x10001 的数组 sss，很明显栈不够用的。sss 的结束地址 0x172a9db 还在任务栈内，但 sss 的起始地址 0x171a9db 已经超出了栈的范围[0x172aaa0,0x1725c80]。这时候再按一下键盘，就会出现刚刚的 Page Fault 了。

在实际工作中，给我们带来困扰的一般是这个任务(例如 t1)已经退出了，因此出现 Page Fault 时，用 i 或 checkStack 命令已经找不到罪魁祸首了。

在《Task 之系统任务》里提到过，可以在任务的最后位置添加一个 taskSuspend(0)，把它挂起来。然后就可以用 checkStack 了

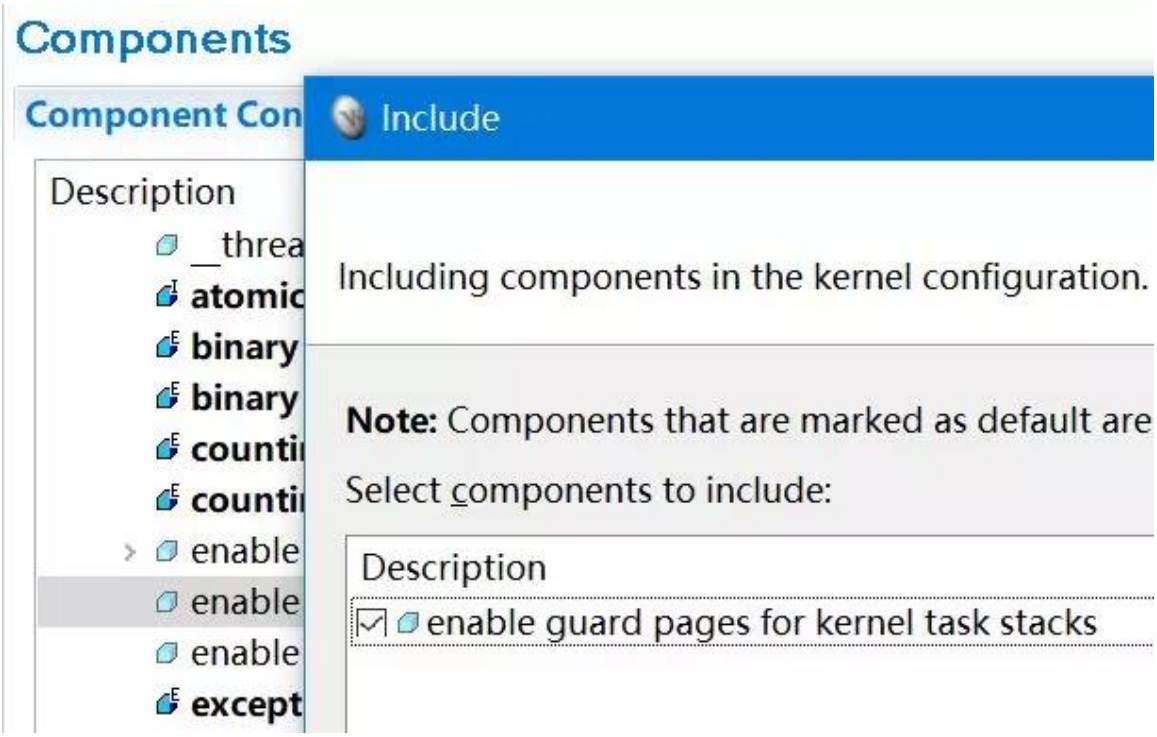
```
#include <vxWorks.h>
#include <taskLib.h>
#include <stdIo.h>
void aaa()
{
    TASK_DESC taskDesc;
    char sss[0x10001]="hello";

    taskInfoGet(0,&taskDesc);
    printf("td_stackSize=0x%x=%d\n",taskDesc.td_stackSize,taskDesc.td_stackSize);
    printf("td_pStackBase=0x%x\n",taskDesc.td_pStackBase);
    printf("td_pStackEnd=0x%x\n",taskDesc.td_pStackEnd);
    printf("sss=0x%x\n",sss);
    printf("sss[0x10000]=0x%x\n",sss+0x10000);
    taskSuspend(0);
}
```

```
-> checkStack t1
  NAME        ENTRY        TID       SIZE   CUR  HIGH  MARGIN
---------- ------------ ---------- ------ ------ ------ -------
t1          aaa          0x162e250  20000      0 20000       0 OVERFLOW
(Exception Stack)                     6880      0   100    6780
value = 1 = 0x1
->
```

这样就能抓到现形了。

在 VxWorks 里有个组件，INCLUDE_PROTECT_TASK_STACK，用于保护栈的溢出。我们加上它来试一试



这时候再执行同样的程序后，VxWorks 立刻重启了，添加了 taskSuspend(0)也没用。在 bootrom 里，用 e 命令可以看到重启的原因

```
excOsm:FATAL: Null taskIdCurrent, or bad stack pointer, errno = 0x00000000


Press any key to stop auto-boot...
 1


[VxWorks Boot]: e

xcOsm:FATAL: Null taskIdCurrent, or bad stack pointer, errno = 0x00000000

[VxWorks Boot]: _
```

有了这个保护，再有越界就会立刻重启，不会把危险推后。因为有的时候越界，并不会立刻暴露问题。

既然说到了越界，还有一些比较常见的情况，例如数组越界、指针越界。看个例子

```
 1 #include <vxWorks.h>
 2 #include <stdIo.h>
 3
 4 int a1=1;
 5 int a2[1]={2};
 6 int a3=3;
 7 void aaa()
 8 {
 9     printf("a1=%d,a2[0]=%d,a2[1]=%d,a3=%d\n",a1,a2[0],a2[1],a3);
10
11     a2[1]=100;
12     printf("set a2[1] to 100\n");
13
14     printf("a1=%d,a2[0]=%d,a2[1]=%d,a3=%d\n",a1,a2[0],a2[1],a3);
15
16 }
```

数组 a2 只有一个成员 a2[0]，但赋值时，写入了两个成员 a2[0]和 a2[1]。

```
-> sp aaa
Task spawned: id = 0x162e128, name = t1
value = 23257384 = 0x162e128 = '('
-> a1=1,a2[0]=2,a2[1]=3,a3=3
set a2[1] to 100
a1=1,a2[0]=2,a2[1]=100,a3=100
```

可以看到，写 a2[1]时，实际操作的 a3 的地址

```
-> d &a1,1,4
NOTE: memory values are displayed in hexadecimal.
0x005e6bf0:  00000001
value = 0 = 0x0
-> d &a2,2,4
NOTE: memory values are displayed in hexadecimal.
0x005e6bf0:           00000002 00000064
value = 0 = 0x0
-> d &a3,1,4
NOTE: memory values are displayed in hexadecimal.
0x005e6bf0:                    00000064
value = 0 = 0x0
->
```

如果当前文件中，a2[]后面没有声明其它变量，那被操作的地址就很隐蔽了。我们再试一下

```
#include <vxWorks.h>
#include <stdIo.h>

int a1=1;
int a3=3;
int a2[1]={2};
void aaa()
{
    printf("a1=%d,a2[0]=%d,a2[1]=%d,a3=%d\n",a1,a2[0],a2[1],a3);

    a2[1]=100;
    printf("set a2[1] to 100\n");

    printf("a1=%d,a2[0]=%d,a2[1]=%d,a3=%d\n",a1,a2[0],a2[1],a3);

}
```

重启 VxWorks，我们先看看 a2 后面是谁

```
-> lkAddr &a2
0x005e6bf4 a3                          data
0x005e6bf8 a2                          data
0x005e6bfc runtimeName                 data
0x005e6c00 vxWorksVersion              data
0x005e6c04 runtimeVersion              data
0x005e6c08 runtimeVersionFull          data
0x005e6c0c creationDate                data
0x005e6c10 __cplusXtors_o              data
0x005e6c14 cplusXtorStrategy           data
0x005e6c18 linkedCtorsInitialized      data
0x005e6c1c __cplusDem_o                data
0x005e6c20 cplusDemanglerMode          data
value = 0 = 0x0
-> runtimeName
runtimeName = 0x5e6bfc: value = 5853753 = 0x595239 = '9'
->
```

哦，有个变量叫 runtimeName，人家的值是 0x595239

然后执行程序，再看看它的值

```
-> sp aaa
Task spawned: id = 0x15fc350, name = t1
value = 23053136 = 0x15fc350 = 'P'
-> a1=1,a2[0]=2,a2[1]=5853753,a3=3
set a2[1] to 100
a1=1,a2[0]=2,a2[1]=100,a3=3

-> runtimeName
runtimeName = 0x5e6bfc: value = 100
-> _
```

后面那个变量 runtimeName 被修改了…

只要这个 VxWorks 系统没有关机，在若干年之后，就可能有个任务访问变量 runtimeName，那时就会出问题，但那个时候，它只能背锅了… 执行 aaa 的任务早跑了