VxWorks 下任务的删除

本文我们聊一聊 VxWorks 系统里如何删除任务。
与任务相关的 API 由系统库 taskLib 提供。

正常情况下，任务执行完自己的最后一条语句后，就主动退出了。不需要做什么额外操作。
不过任务也可以被动态删除。相关的 API 有

```
01: void exit(int code);
02: void taskExit(int code);
03: STATUS taskDelete(TASK_ID tid);
04: STATUS taskSafe (void);
05: STATUS taskUnsafe (void);
```

taskExit()会终止当前任务，并删除任务的 Stack 与 TCB。事实上，执行完最后一条语句时，就会自动调用 taskExit()。也就是说一般情况下，代码中没必要显式的调用 taskExit()。

exit()在 Kernel 模式下，与 taskExit()的作用相同。不过在用户模式下，exit()要强大一些，它会把当前的进程整个删除，并释放进程的内存空间。事实上，RTP 的 main()函数在执行完最后一条语句时，就会自动调用 exit ()。同样，代码中很少显式的调用 exit()。

taskDelete()的参数为 0 时就是删除任务本身，相当于 taskExit(0)。换句话说，**taskDelete()**不同的地方是，它可以删除其它任务。

可以看到，不管任务是主动退出，还是被删除，都只是删除 Stack 和 TCB(因为任务就是由它俩组成的)，这就存在一个很大的风险，即任务在工作时占用的其它系统资源没有被释放。这些系统资源主要有三类：**动态申请的内存，IO 资源，**和**互斥信号量**。所以任务退出或删除时，务必对其资源进行回收(RTP 里好一些，动态内存和私有信号量会随着 RTP 的退出而自动回收)。
另外，任务的退出或被删除，是不会通知其它任务的。而且任务被其它任务删除的话，它自己也是不知道具体的删除位置的。这就无法保证关系资源的回收，甚至可能破坏正在访问的数据。

不过通常情况下，只是在项目的开发调试阶段或软件的异常处理代码中，才会调用这些删除函数。
在正常的代码流程中，为了防止任务执行时被意外删除，可以使用 taskSafe()和 taskUnsafe()对任务进行保护。例如正在访问共享资源 x 的任务 A 突然被删除了，x 的数据一致性就没法保证。而 A 调用 taskSafe()后，其它任务试图删除 A 时，就会被阻塞，直到 A 调用 taskUnsafe()解除保护。这种保护还支持嵌套使用，即调用 taskUnsafe()之前，可以多次调用 taskSafe()，不过后续需要调用相同次数的 taskUnsafe()。

```
01: void ccc()
02:     {
03:     taskSafe();
04:     fd =open("c.txt",O_RDWR,0);
05:
06:     /* critical region code */
07:     ...
08:
09:     close( fd );
10:     taskUnsafe();
11:     }
12:
13: void bbb()
14:     {
15:     taskSafe();
16:     semTake( semMid , WAIT_FOREVER);
17:
18:     /* critical region code */
19:     ...
20:     ccc();
21:
22:     semGive( semMid );
23:     taskUnsafe();
24:     }
25:
26: void aaa()
27:     {
28:     taskSafe();
29:
30:     buf =malloc(10);
31:     bbb();
32:
33:     taskUnsafe();
34:     free( buf );
35:     }
```

可以看到，文件 fd、critical region 和信号量都得到了保护。但动态内存 buf 有未释放的风险。