# VxWorks 下的 C++调用

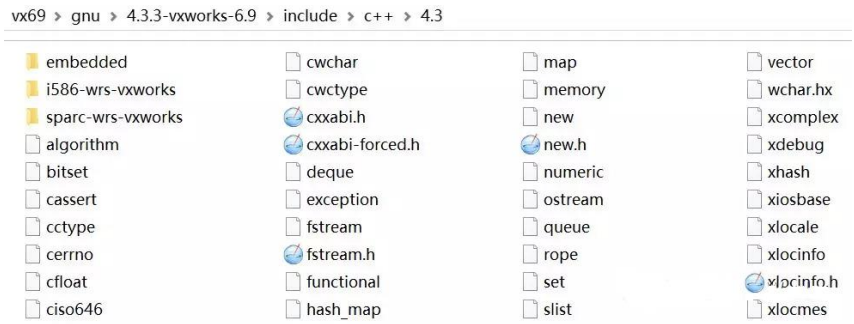
VxWorks 支持 C++编程，毕竟面向对象语言功能强大。不过因为性能的原因，在实时系统里，更多的还是使用 C 语言来编程。尤其在信号和中断处理函数里调用 C++的话，一些动态行为不能使用，例如非静态成员函数、实例化对象、删除对象、exception、run-time type identification(RTTI)等

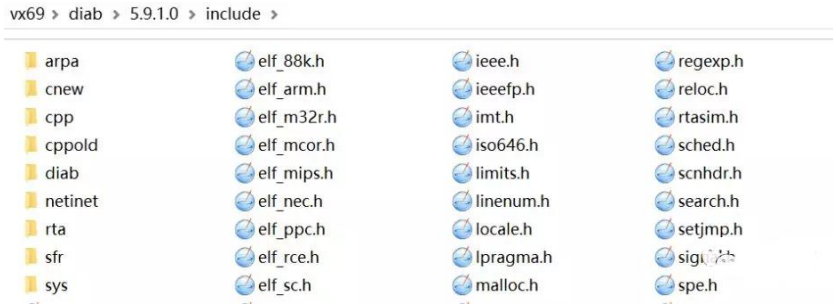
VxWorks 中的 C++组件有

- INCLUDE_CTORS_DTORS - 默认包含，确保内核启动时调用编译器生成的初始化函数(包含 C++静态对象的初始化)
- INCLUDE_CPLUS - C++应用的基本支持，通常与 INCLUDE_CPLUS_LANG 一起使用
- INCLUDE_CPLUS_LANG - C++语言特性的支持，例如 new、delete、异常处理等
- INCLUDE_CPLUS_IOSTREAMS - 包含所有类库功能
- INCLUDE_CPLUS_DEMANGLER - C++符号的还原(demangler)，使用 Kernel Shell 加载器时，会用到它。使用 INCLUDE_CPLUS 和 INCLUDE_SYM_TBL 时，会包含它


风河集成的两个编译器(diab 和 gnu)都支持 C++，不过它俩基于 C++源码编译得到二进制文件并不兼容！当然也包括它俩的 C++系统库 libcplus.a


另外，它俩的头文件也不相同，而且各自使用自己的路径，而不是系统的 target/h



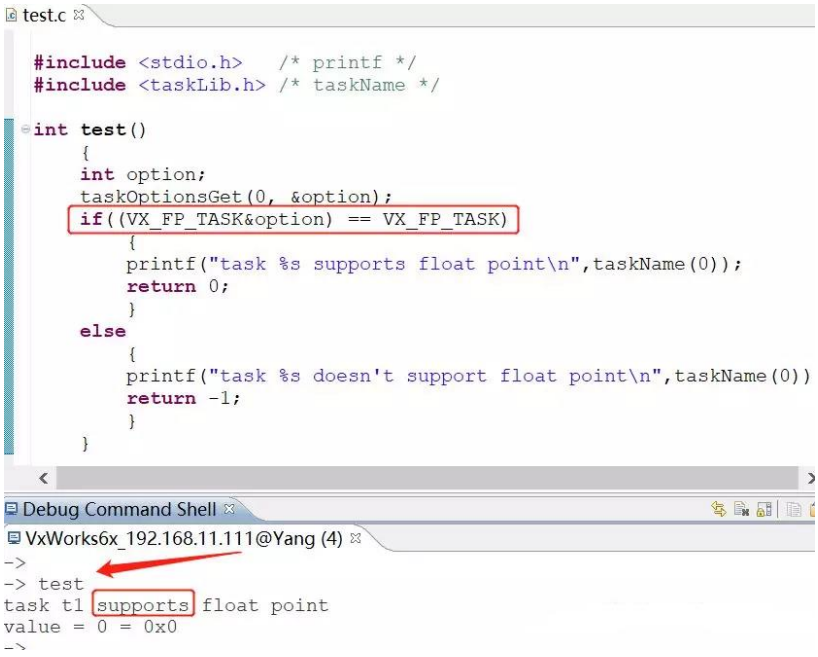

不过开发者不用关心这些信息，编译器会自动识别、自动查找它们的 ^_^

开发者需要注意的是，在启动包含了 C++的任务时，需要包含 VX_FP_TASK 选项，因为 C++的异常处理机制使用了浮点寄存器。否则上下文切换时，系统不会保存/恢复浮点寄存器的值，就有可能出现意想不到的错误



```
taskSpawn(NULL, 100, VX_FP_TASK, 2000)
```

幸运的是，Host 端的工具(例如 Host Shell)默认包含了这个选项



因为 C++支持函数重载，这样经过编译器编译后，符号表中的函数名肯定不能与源码中的函数名一致，因此 C 文件中无法直接访问 C++的符号。为了解决这个问题，C++的符号需要使用 **extern "C"**来声明

看个例子

```
#ifdef __cplusplus
extern "C" void funCpp0();
extern "C" void funCpp1(int i);
extern "C" void funC0();
extern "C" void funC1();
#else
extern void funCpp0();
extern void funCpp1(int i);
extern void funC0();
extern void funC1();
#endif
```

在头文件中，使用宏__cplusplus 来声明函数，这样 C++和 C 文件都可以引用它

C++函数如下

[testcpp.cpp (F:\test)]

O选项 V视图 W窗口 H帮助

```
00001:
00002: #include <stdIo.h>
00003: #include "test.h"
00004:
00005: void funCpp0()
00006:     {
00007:     printf("this is %s\n",__FUNCTION__);
00008:     }
00009: void funCpp1(int i)
00010:     {
00011:     printf("this is %s: %d\n",__FUNCTION__,i);
00012:     funC0();
00013:     }
```

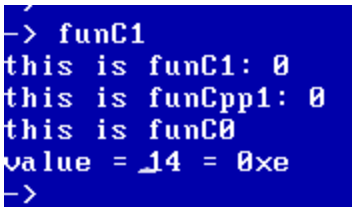
C 函数如下

[test.c (F:\test)]

O选项 V视图 W窗口 H帮助

```
00001:
00002: #include <stdIo.h>
00003: #include "test.h"
00004:
00005: void funC0()
00006:     {
00007:     printf("this is %s\n",__FUNCTION__);
00008:     }
00009: void funC1(int i)
00010:     {
00011:     printf("this is %s: %d\n",__FUNCTION__,i);
00012:     funCpp1(i);
00013:     }
```

跑一下试试




可以看到 C 与 C++相互调用成功了：C 里的 funC1()调用了 C++的 funCpp1()；funCpp1()又调用了 C 的 funC0()