VxWorks 的应用程序自启动的方法

我们知道，要想在 VxWorks 里启动应用程序(APP/Application/Task+RTP)，常见的方法是使用 taskSpawn()/rtpSpawn()来创建并激活相应的 Task/RTP。那我们的第一条 taskSpawn()/rtpSpawn()应该放在哪呢？也就是如何让 VxWorks 系统**自动**启动 APP 呢？
只要找到 VxWorks 系统启动时执行的最后一个函数，让这个函数来调用我们自己的 APP 就可以了。
打开工程目录里的 prjConfig.c 文件，可以看到 VxWorks 的第一个 Task**tRootTask** 的入口函数 usrRoot()。而 usrRoot()调用的最后一个函数是 **usrAppInit**()

*prjConfig.c

```
683 */
684
685 void usrRoot (char *pMemPoolStart, unsigned memPoolSize)
686     {
687     usrKernelCoreInit ();               /* core kernel facilities */
688     poolLibInit();                      /* memory pools of fixed size items
689     memInit (pMemPoolStart, memPoolSize, MEM_PART_DEFAULT_OPTIONS); /* full
690     memPartLibInit (pMemPoolStart, memPoolSize); /* core memory partition ma
691     pgPoolLibInit();                    /* Generic Page Pool Allocator */
692     pgPoolVirtLibInit();                /* Page Pool Allocator for Virtual
693     pgPoolPhysLibInit();                /* Page Pool Allocator for Physical
694     /*
695      * .
696      * .
697      * .
698      */
699     cpuPwrLightMgrInit (); cpuPwrMgrEnable (TRUE); /* Idle-halt CPU power ma
700     cplusCtorsLink ();                  /* run compiler generated initializa
701     cplusDemanglerInit ();              /* Support library for kernel shell
702     usrScInit ();                       /* The system call initialization se
703     usrToolsInit ();                    /* software development tools */
704     usrAppInit ();                      /* call usrAp
705     }
```

然后在工程里可以找到文件 usrAppInit.c

usrAppInit.c

```
30 /************************************************************************
31 *
32 * usrAppInit - initialize the users application
33 */
34
35 void usrAppInit (void)
36     {
37 #ifdef  USER_APPL_INIT
38     USER_APPL_INIT;     /* for backwards compatibility */
39 #endif
40
41     /* add application specific code here
42     }
```

从注释里可以看出来了，我们的 APP 可以从这里开始。
写个小程序试试，例如 printf()

```
#include <stdIo.h>
#include <taskLib.h>
/*******************************************************************************
*
* usrAppInit - initialize the users application
*/

void usrAppInit (void)
    {
#ifdef  USER_APPL_INIT
    USER_APPL_INIT;     /* for backwards compatibility */
#endif

    /* add application specific code here */
    int pri;
    TASK_DESC desc;
    taskPriorityGet(0,&pri);
    taskInfoGet(0,&desc);
    printf ("\n I'm task %s, priority is %d, stack size is 0x%x\n\n",
            taskName(0),pri,desc.td_stackSize);
    }
```

启动 VxWorks，看看效果

```
 UUUUUUUU      UUUUUUUU          tttt            iiii   lllllll
 U::::::U      U::::::U       ttt:::t           i::::i  l:::::l
 U::::::U      U::::::U       t::::::t           iiii   l:::::l
 UU:::::U      U:::::UU       t::::::t                  l:::::l
  U:::::U      U:::::U ttttttt:::::ttttttt     iiiiiii   l::::l
  U:::::D      D:::::U t:::::::::::::::::t     i:::::i   l::::l
  U:::::D      D:::::U t:::::::::::::::::t      i::::i   l::::l
  U:::::D      D:::::U tttttt:::::::tttttt      i::::i   l::::l
  U:::::D      D:::::U       t::::::t           i::::i   l::::l
  U:::::D      D:::::U       t::::::t           i::::i   l::::l
  U:::::D      D:::::U       t::::::t           i::::i   l::::l
  U::::::U   U::::::U       t::::::t    tttttt i::::i   l::::l
  U:::::::UUU:::::::U       t:::::::tttt::::::ti::::::il::::::l
   UU:::::::::::::UU        tt:::::::::::::::ti::::::il::::::l
     UU:::::::::UU            tt:::::::::::tti::::::il::::::l
       UUUUUUUUU                ttttttttttt  iiiiiiiillllllll
OS: VxWorks 6.9.  BSP: VmWare - Yang.
CPU: VMWARE.  Memory Size: 0xfefa000 (~254Mb).
Created: Mar  5 2019 11:51:15
ED&R Policy Mode: Deployed
WDB_COMM_END Ready.

I'm task tRootTask, priority is 0, stack size is 0x2400

-> _
```

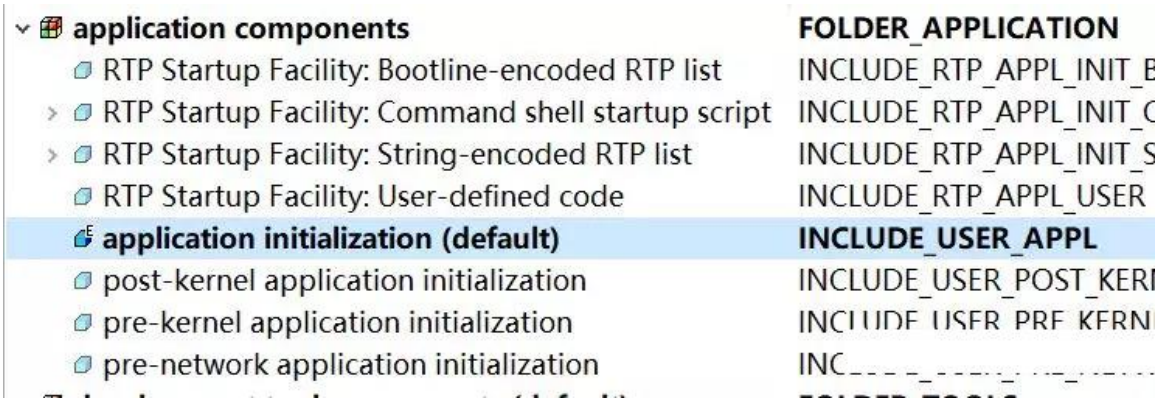
可以看到 APP 的 printf()执行了，不过它是运行在 tRootTask 的上下文里的。这样不好，一来它的优先级太高；二来出了问题后，影响太大。所以我们的 APP 应该由 usrAppInit()中的 taskSpawn()来启动，例如

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

void test()
    {
    int pri;
    TASK_DESC desc;
    taskPriorityGet(0,&pri);
    taskInfoGet(0,&desc);
    printf("\n I'm task %s, priority is %d, stack size is 0x%x\n\n",
            (int)taskName(0),pri,desc.td_stackSize,0,0,0,0,0);
    }
/************************************************************************
*
* usrAppInit - initialize the users application
*/

void usrAppInit (void)
    {
#ifdef  USER_APPL_INIT
    USER_APPL_INIT;     /* for backwards compatibility */
#endif

    /* add application specific code here */
    taskSpawn("tTest",100,0,0x1000,(FUNCPTR)test,0,
    }
```

这样就可以了

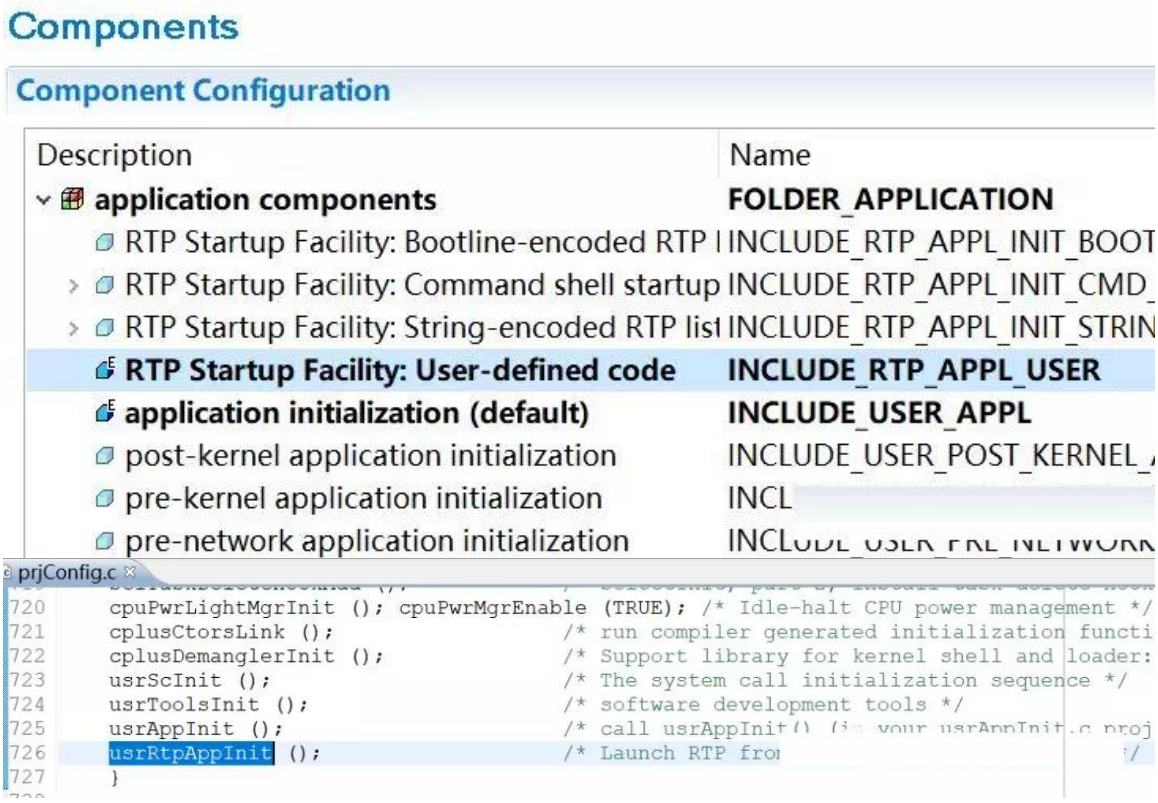
```
 UUUUUUUU     UUUUUUUU          tttt            iiii  lllllll
 U::::::U     U::::::U       ttt:::t           i::::i l:::::l
 U::::::U     U::::::U       t:::::t            iiii  l:::::l
 UU:::::U     U:::::UU       t:::::t                  l:::::l
  U:::::U     U:::::U ttttttt:::::ttttttt     iiiiiii  l::::l
  U:::::D     D:::::U t:::::::::::::::::t     i:::::i  l::::l
  U:::::D     D:::::U t:::::::::::::::::t      i::::i  l::::l
  U:::::D     D:::::U tttttt:::::::tttttt      i::::i  l::::l
  U:::::D     D:::::U       t:::::t            i::::i  l::::l
  U:::::D     D:::::U       t:::::t            i::::i  l::::l
  U:::::D     D:::::U       t:::::t            i::::i  l::::l
  U::::::U   U::::::U       t:::::t    ttttttt i::::i  l::::l
  U:::::::UUU:::::::U       t::::::tttt:::::ti::::::il::::::l
   UU:::::::::::::UU        tt::::::::::::::ti::::::il::::::l
     UU:::::::::UU            tt:::::::::::tti::::::il::::::l
       UUUUUUUUU                ttttttttttt  iiiiiiiillllllll
OS: VxWorks 6.9.  BSP: VmWare - Yang.
CPU: VMWARE.  Memory Size: 0xfefa000 (~254Mb).
Created: Mar  5 2019 12:04:19
ED&R Policy Mode: Deployed
WDB_COMM_END Ready.
->
I'm task tTest, priority is 100, stack size
```

usrRoot()之所以会调用 usrAppInit()，是因为包含了组件 INCLUDE_USER_APPL



可以看到还有几个组件与 startup 相关，有兴趣的话，我们试试其它组件

INCLUDE_RTP_APPL_USER

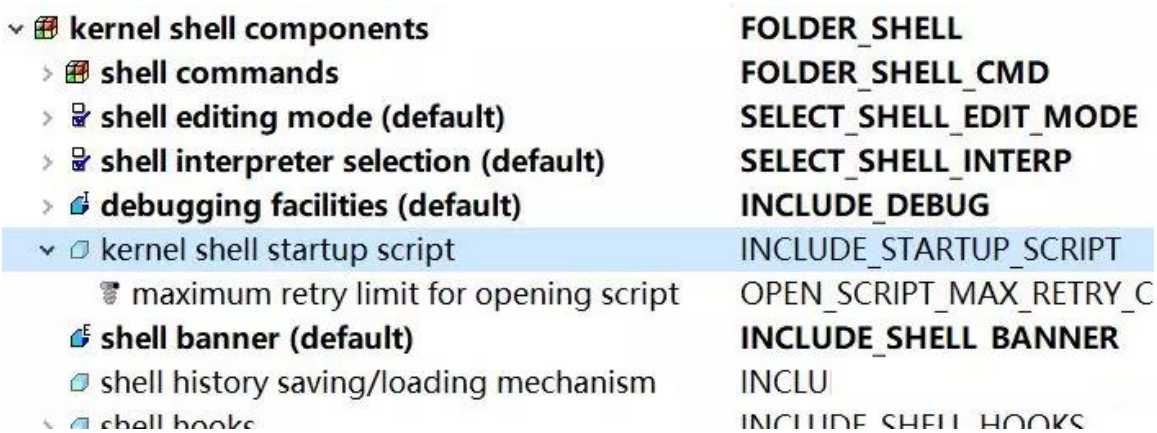


usrRtpAppInit.c

```
/*******************************************************************
*
* usrRtpAppInit - initialize the users RTP applications.
*/

void usrRtpAppInit (void)
    {
    /* TODO - add your own application lau
    }
```

可以看到，包含了这个组件后，工程里多出来一个与 usrAppInit()类似的 **usrRtpAppInit**()。从注释来看，它是用来启动 RTP 的。不过它俩默认都是空函数，没什么本质区别的。也就是说，在 usrRtpAppInit()里来启动 Kernel Task 也是没问题的。

INCLUDE_STARTUP_SCRIPT



这个组件在另一个目录里，包含之后，usrRoot()会在调用 usrShell()之前解析并执行 bootrom 的 startup script 参数

prjConfig.c

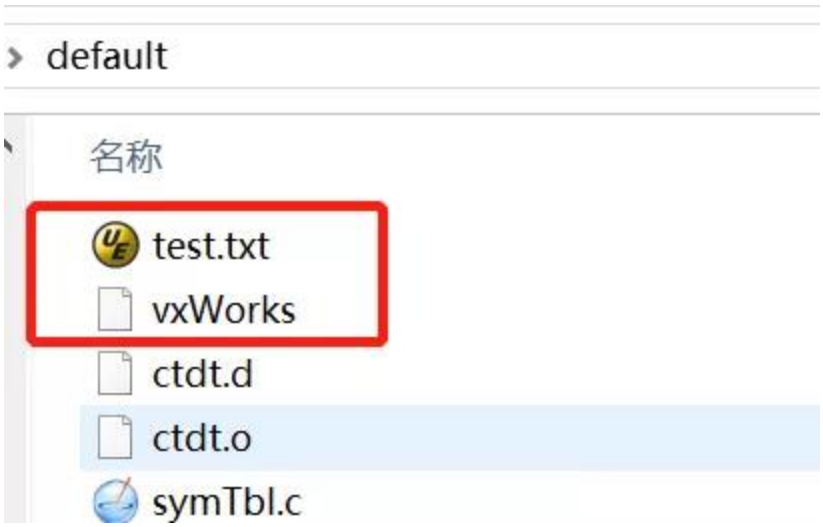
```
void usrShellInit (void)
    {
    shellLibInit ();                    /* Handles the shell core files */
    { extern size_t dbgLibTrcBufSz; dbgLibTrcBufSz = DEBUG_STACK_TRACE_BUF_SIZ
    vxdbgRtpLibInit ();                 /* Initialize process debugging librar
    usrBanner ();                       /* Display the WRS banner on startup *
    ledModeRegister (viLedLibInit);     /* Editing mode similar to the Vi edit
    shellInterpRegister (shellInterpCInit); /* The C interpreter for the kerne
    shellInterpRegister (shellInterpCmdInit); /* The command interpreter for t
    usrShellCmdInit ();                 /* The kernel shell commands initializ
    usrStartupScript (startupScriptFieldSplit (sysBootParams.startupScript));
    usrShell ();                        /* Interpreter
    }
```

```
[VxWorks Boot]: c

'.' = clear field;   '-' = go to previous field;

boot device          : lnPci0
processor number     : 0
host name            : host
file name            : vxworks
inet on ethernet (e) : 192.168.11.111:ffffff00
inet on backplane (b):
host inet (h)        : 192.168.11.1
gateway inet (g)     :
user (u)             : vm
ftp password (pw) (blank = use rsh): vm
flags (f)            : 0x8
target name (tn)     : vm
startup script (s)   :
```

我们写一个 script 文件来试试这个功能：

新建一个文件，为了方便编辑，可以把文件后缀写作 txt，在文件中添加几行**可以在 VxWorks 的 Kernel Shell 里执行的语句**。我们之前定义了一个函数 test()，这次也调用它

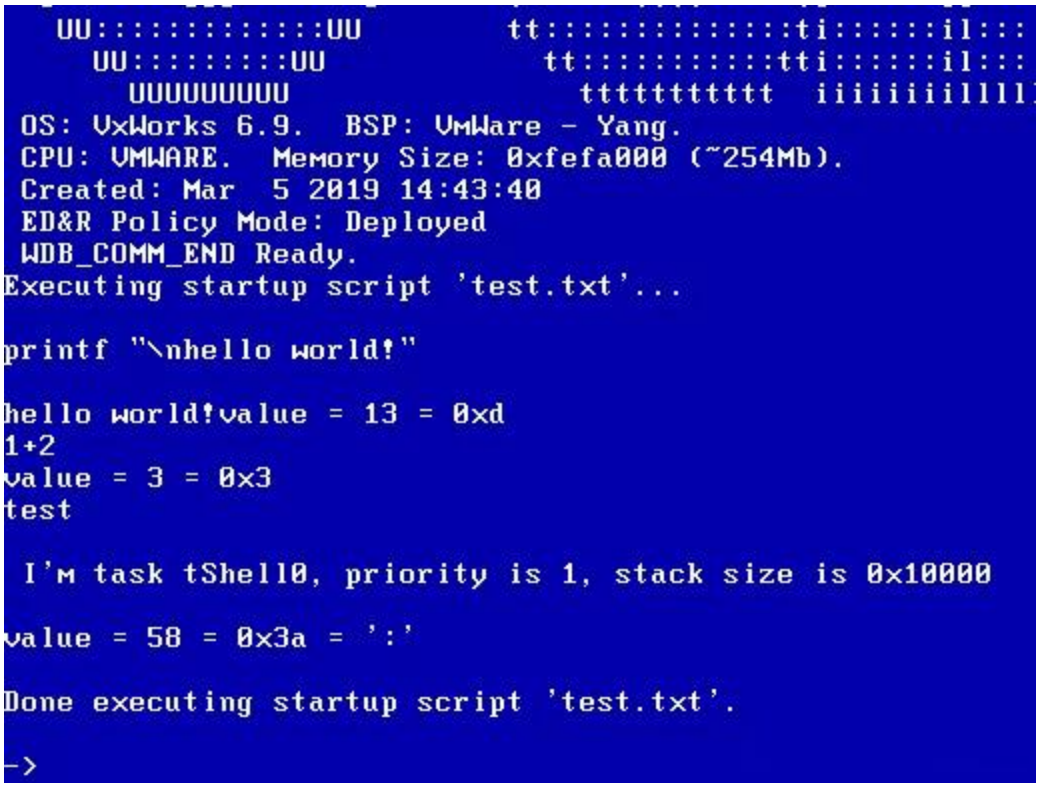
把这个文件存放到 VxWorks 文件的目录里



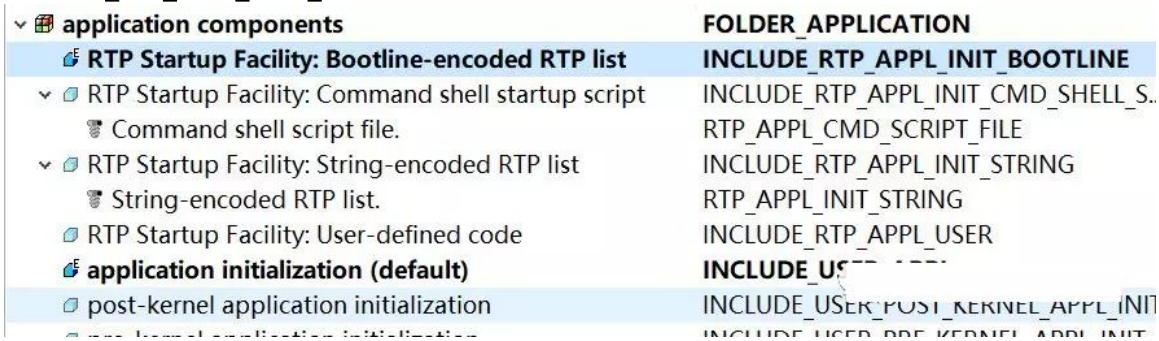
修改 bootrom 的 startup script 参数

```
[VxWorks Boot]: c

'.' = clear field;  '-' = go to previous field;  ^D = quit

boot device          : lnPci0
processor number     : 0
host name            : host
file name            : vxworks
inet on ethernet (e) : 192.168.11.111:ffffff00
inet on backplane (b):
host inet (h)        : 192.168.11.1
gateway inet (g)     :
user (u)             : vm
ftp password (pw) (blank = use rsh): vm
flags (f)            : 0x8
target name (tn)     : vm
startup script (s)   : test.txt
```

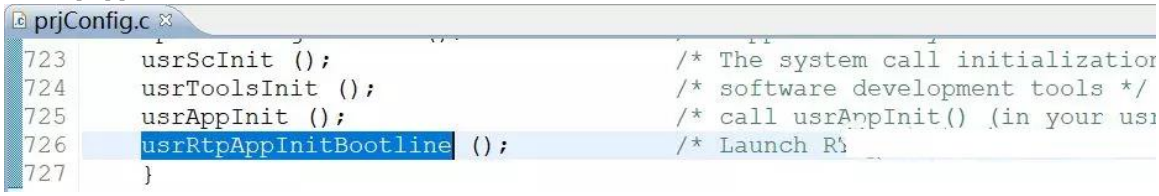
启动 VxWorks



INCLUDE_RTP_APPL_INIT_BOOTLINE

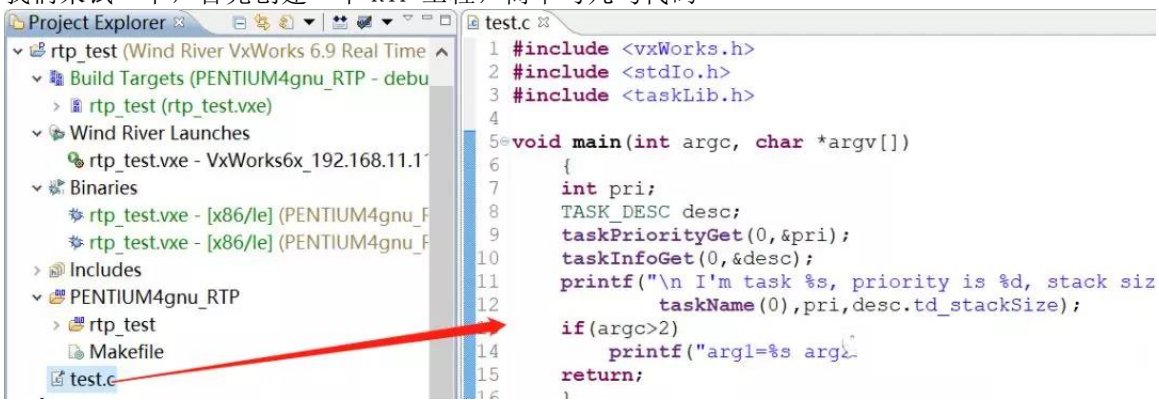


包含这个组件后，usrRoot()会在 usrAppInit()之后再调用一个函数 usrRtpAppInitBootline()。

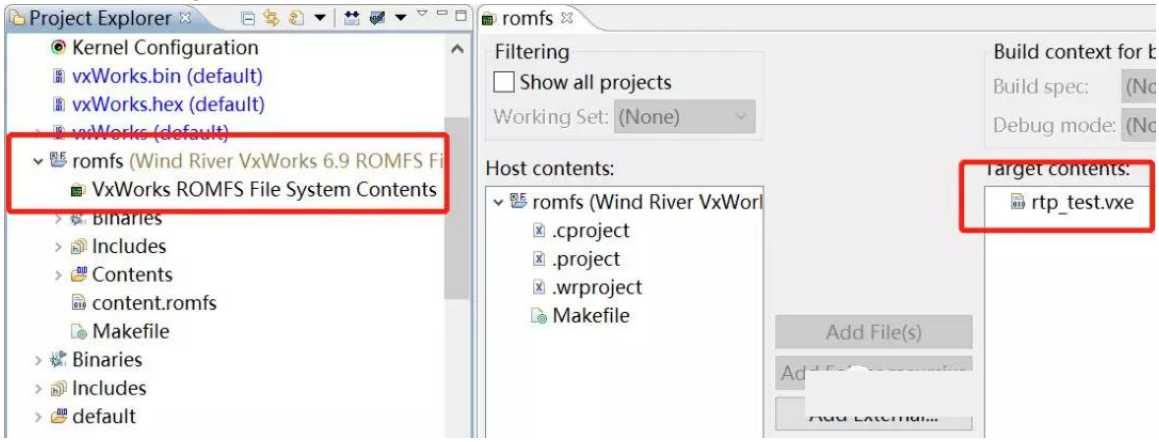


这个函数的作用也是解析 startup script，不过它解析的不是 script 里的文件，而是直接用使用 startup script 指定的 RTP 文件。格式是： #RTP 文件 1^参数 1^参数 2…#RTP 文件 2…即符号#之后是 RTP 的文件名，符号^之后是 RTP 的入参。有几个 RTP 就用几个#，有几个参数就用几个^

我们来试一下，首先创建一个 RTP 工程，简单写几句代码



新建一个 ROMFS 工程，用于存放 RTP 工程编译后的 rtp_test.vxe，然后把 ROMFS 工程放到 VxWorks 的 Image 工程里



修改 bootrom 的 startup script 参数，可以保留之前的 script 文件，不过要放到第一个#之前。并给 RTP 传递两个参数：x 和 y

```
boot device          : lnPci
unit number          : 0
processor number     : 0
host name            : host
file name            : vxworks
inet on ethernet (e) : 192.168.11.111:ffffff00
host inet (h)        : 192.168.11.1
user (u)             : vm
ftp password (pw)    : vm
flags (f)            : 0x8
target name (tn)     : vm
startup script (s)   : test.txt#/romfs/rtp_test.vxe x y
```

这次 script 文件 test.txt 的内容如下

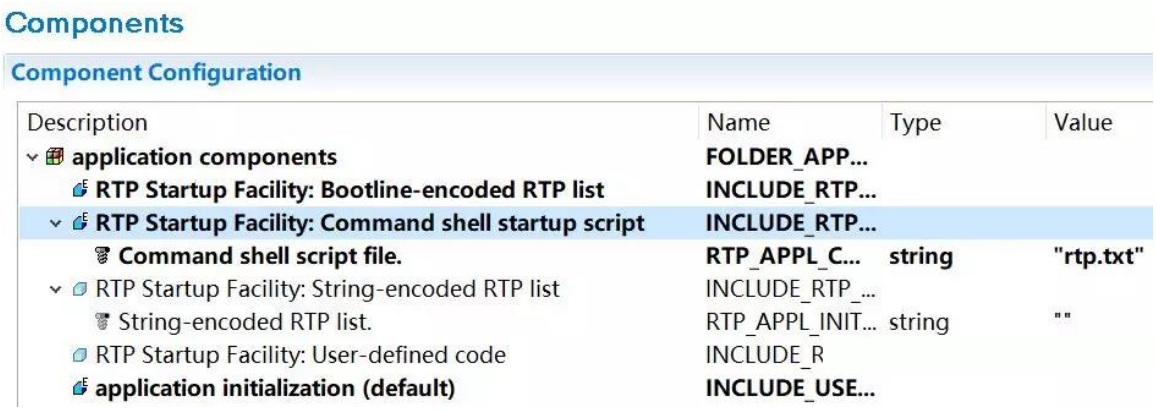
启动 VxWorks，看看得到了什么

```
     UU:::::::::UU              tt:::::::::::tti::::::il::::::l
       UUUUUUUUU                tttttttttttt  iiiiiiiillllllll
 OS: VxWorks 6.9.  BSP: VmWare - Yang.
 CPU: VMWARE.  Memory Size: 0xfefa000 (~254Mb).
 Created: Mar  5 2019 16:05:49
 ED&R Policy Mode: Deployed
 WDB_COMM_END Ready.
Executing startup script 'test.txt'...

printf "\n hello world!"

 hello world!value = 14 = 0xe
2+3
value = 5 = 0x5

Done executing startup script 'test.txt'.

Spawning RTP: /romfs/rtp_test.vxe
->
 I'm task iRtp_test, priority is 220, stack size is 0xff40e458

arg1=x arg2=y
```

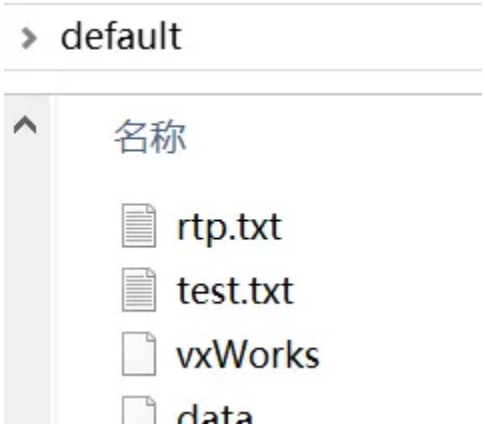
VxWorks 在打印 Kernel Shell 提示符“->”之前执行了 test.txt 里的语句；在打印“->”之后，又启动了 startup script 里指定的 RTP。

这就是 INCLUDE_RTP_APPL_INIT_BOOTLINE 的作用：执行 bootrom 参数 startup script 中符号#之后的 RTP

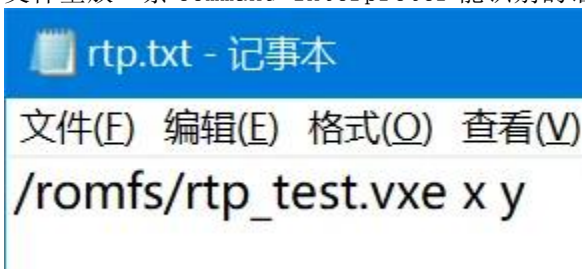
INCLUDE_RTP_APPL_INIT_CMD_SHELL_SCRIPT



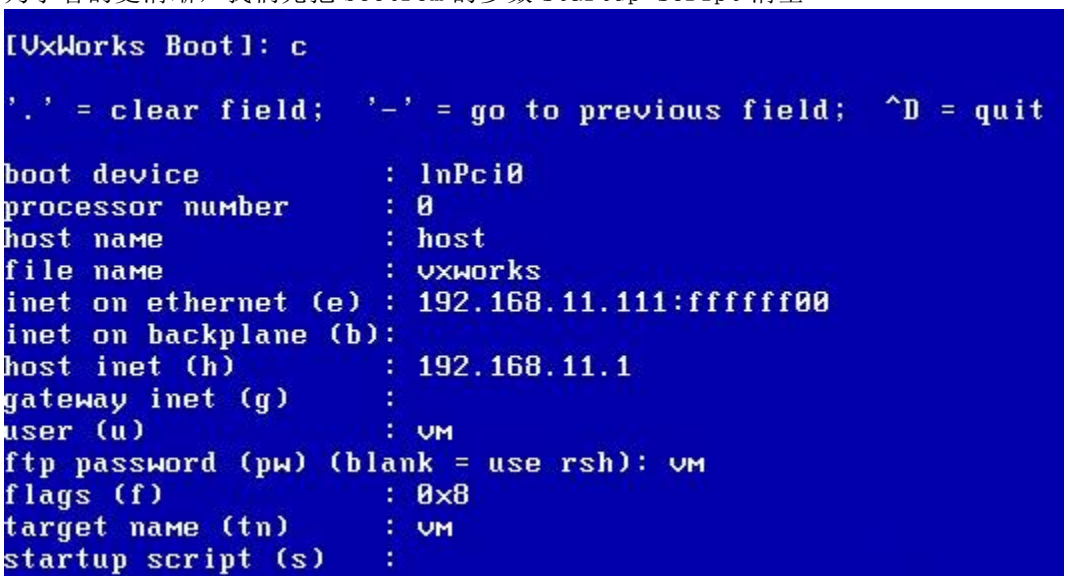
这个组件也是在 VxWorks 启动 Kernel Shell 之后来启动 RTP 的。不过它的输入是由参数 RTP_APPL_CMD_SCRIPT_FILE 来指定的文件，而且它解析文件时，用的是 VxWorks6 特有的 Command Interpreter。
例如我们在 VxWorks 镜像文件的目录里新建一个 rtp.txt 文件，并把这个文件名赋值给参数 RTP_APPL_CMD_SCRIPT_FILE



文件里放一条 Command Interpreter 能识别的语句



为了看的更清晰，我们先把 bootrom 的参数 startup script 清空



然后启动 VxWorks

```
  U:::::D      D:::::U        t:::::t             i::::i   l::::l
  U:::::D      D:::::U        t:::::t             i::::i   l::::l
  U::::::U    U::::::U        t:::::t    ttttttt  i::::i   l::::l
  U:::::::UUU:::::::U        t::::::tttt:::::ti::::::il::::::l
   UU:::::::::::::UU         tt::::::::::::::ti::::::il::::::l
     UU:::::::::UU             tt:::::::::::tti::::::il::::::l
       UUUUUUUUU                 ttttttttttt  iiiiiiiillllllll
OS: VxWorks 6.9.  BSP: VMWare - Yang.
CPU: VMWARE.  Memory Size: 0xfefa000 (~254Mb).
Created: Mar  5 2019 16:34:25
ED&R Policy Mode: Deployed
WDB_COMM_END Ready.

-> Executing startup script 'rtp.txt'...

/romfs/rtp_test.vxe x y
Launching process '/romfs/rtp_test.vxe' ...
Process '/romfs/rtp_test.vxe' (process Id = 0x1643010) launched.

 I'm task iRtp_test, priority is 220, stack size is 12445312

arg1=x arg2=y
```

可以看到，组件 INCLUDE_RTP_APPL_INIT_CMD_SHELL_SCRIPT 的作用就是使用 Command Interpreter 来解析 RTP_APPL_CMD_SCRIPT_FILE 指定的文件

INCLUDE_RTP_APPL_INIT_STRING

Components

Component Configuration

Description	Name	Type	Value
application components	FOLDER_APPLICATION		
RTP Startup Facility: Bootline-encoded RTP list	INCLUDE_RTP_APPL_INIT_BOOTLINE		
RTP Startup Facility: Command shell startup script	INCLUDE_RTP_APPL_INIT_CMD_SH...		
Command shell script file.	RTP_APPL_CMD_SCRIPT_FILE	string	"rtp.txt"
RTP Startup Facility: String-encoded RTP list	INCLUDE_RTP_APPL_INIT_STRING		
String-encoded RTP list.	RTP_APPL_INIT_STRING	string	""
RTP Startup Facility: User-defined code	INCLUDE_RTP_APPL_USER		
application initialization (default)	INCLUDE_USER_APPL		
post-kernel application initialization	INCLUDE_USER_POST_KERNEL_APPL ...		

这个组件的作用类似于前文中的 INCLUDE_RTP_APPL_INIT_BOOTLINE，解析的格式也是一样的。不过它的输入不是 bootrom 的 startup script 参数，而是自己的参数 RTP_APPL_INIT_STRING。

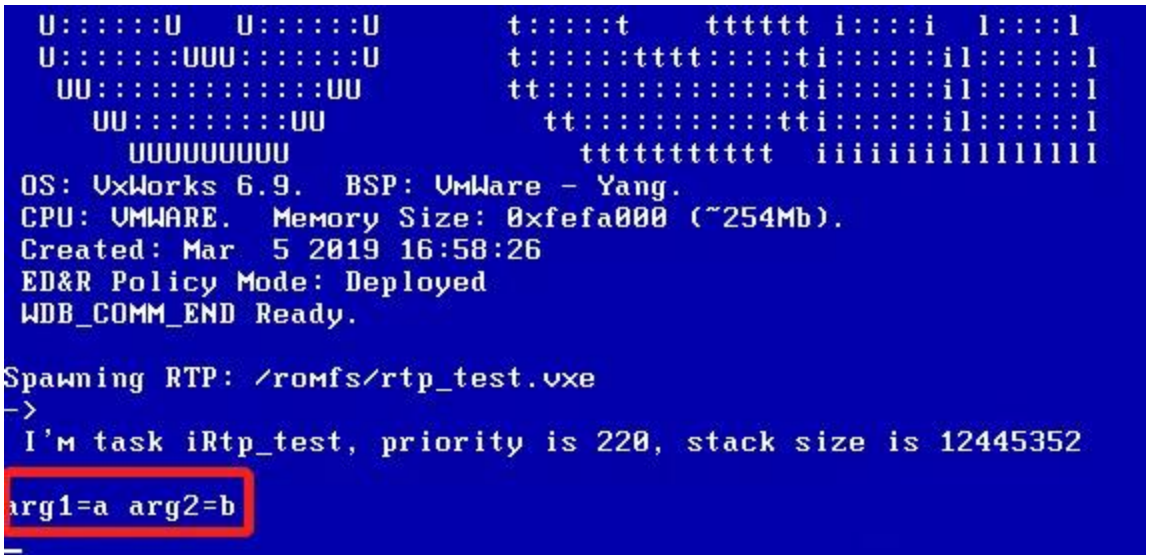
我们修改一下这个参数试试，同样为了看的清晰，我们把上个例子中的 RTP_APPL_CMD_SCRIPT_FILE 置为空，并把 RTP 的参数改为 a 和 b

Components

Component Configuration

Description	Name	Type	Value
application components	FOLDER_APPLICATION		
RTP Startup Facility: Bootline-encoded RTP lis	INCLUDE_RTP_APPL_INIT_BOOTLINE		
RTP Startup Facility: Command shell startup s	INCLUDE_RTP_APPL_INIT_CMD_SH...		
Command shell script file.	RTP_APPL_CMD_SCRIPT_FILE	string	""
RTP Startup Facility: String-encoded RTP list	INCLUDE_RTP_APPL_INIT_STRING		
String-encoded RTP list.	RTP_APPL_INIT_STRING	string	"/romfs/rtp test.vxe^a^b"
RTP Startup Facility: User-defined code	INCLUDE_RTP_APPL_USER		
application initialization (default)	INCLUDE_USER_APPL		

启动 VxWorks



总结一下

	Component	Application Position	Interpreter
1	INCLUDE_USER_APPL	usrAppInit.c/usrAppInit()	C
2	INCLUDE_RTP_APPL_USER	usrRtpAppInit.c/usrRtpAppInit()	C
3	INCLUDE_STARTUP_SCRIPT	bootrom/startup script	Kernel Shell
4	INCLUDE_RTP_APPL_INIT_BOOTLINE	bootrom/startup script	#RTP1^arg1^arg2#RTP2^arg1
5	INCLUDE_RTP_APPL_INIT_CMD_SHELL_SCRIPT	RTP_APPL_CMD_SCRIPT_FILE	
6	INCLUDE_RTP_APPL_INIT_STRING	RTP_APPL_INIT_STRING	#RTP1^arg1^arg2#RTP2^arg1

1、3 用于启动 Task，2、4-6 用于启动 RTP。

1、2 在代码中；3、4 在引导参数中；5、6 在组件参数中。

1、2 使用 C 语法；3 使用 Shell 命令，类似于 C 语法；4、6 使用符号#和^；5 使用专用于 RTP 的 Command 命令。