

Vxworks 6.6使用说明及网络配置



2016-11-21

火星人俱乐部-杨攀

版权所有-请勿抄袭

目录

[笔者序 1](#_Toc467522428)

[1. Vxworks简介 2](#_Toc467522429)

[1.1 Vxworks操作系统的组成部分： 2](#_Toc467522430)

[2. WorkBench简介及使用说明 3](#_Toc467522431)

[2.1 Workbench 3.0简介 3](#_Toc467522432)

[2.2 Workbench 3.0使用说明 3](#_Toc467522433)

[2.3 VxWorks模拟器 7](#_Toc467522434)

[3 Vxworks模拟器网络配置 8](#_Toc467522435)

[3.1 Vxworks模拟器网络通信原理 8](#_Toc467522436)

[3.2 编译一个具有网络模块的Vxworks image 9](#_Toc467522437)

[3.3 配置simulator前的预设条件 11](#_Toc467522438)

[4. Workbench中创建一个网络连接项目 15](#_Toc467522439)

[5. 如何运行一个RTP项目 15](#_Toc467522440)

[附录 18](#_Toc467522441)

[（一） Vxworks上TCP Client代码 18](#_Toc467522442)

[（二） Vxworks上TCP Servert代码 19](#_Toc467522443)

[（三） 主机上作为中转站的Tcp代码 21](#_Toc467522444)

## 笔者序

本文是基于Vxworks 6.6,、WorkBench 3.0编写，所用的主机环境为Windows XP Pro，经笔者测试，WorkBench 3.0的开发环境也可以在Windows 7 Pro（32bit）中安装使用，而对于64的Windows系统，由于提供的网络适配器的不兼容，不建议使用WorkBench 3.0的开发环境。

如果在开发过程中遇见问题，建议查阅官方的help文档。WorkBench界面，选择Help >

Getting Started Resources，虽然都是英文文档，确实是笔者在学习过程找到的最有效的开发资料。

## Vxworks简介

VxWorks 操作系统是美国WindRiver公司于1983年设计开发的一种嵌入式实时操作系统（RTOS），是嵌入式开发环境的关键组成部分。良好的持续发展能力、高性能的内核以及友好的用户开发环境，在嵌入式实时操作系统领域占据一席之地。它以其良好的可靠性和卓越的实时性被广泛地应用在通信、军事、航空、航天等高精尖技术及实时性要求极高的领域中。

### 1.1 Vxworks操作系统的组成部分：

#### 1.1.1内核

特点：

多任务调度（采用基于优先级抢占方式，同时支持同优先级任务间的分时间片调度）、任务间的同步、进程间通信机制、中断处理、定时器和内存管理机制

#### 1.1.2 I/O 系统

VxWorks 提供了一个快速灵活的与 ANSI C 兼容的 I/O 系统，包括 UNIX 标准的Basic I/O，Buffer I/O 以及POSIX 标准的异步 I/O。VxWorks 包括以下驱动程序：网络驱动、管道驱动、RAM盘驱动、SCSI驱动、键盘驱动、显示驱动、磁盘驱动、并口驱动等 。

#### 1.1.3 文件系统

支持四种文件系统：dosFs，rt11Fs，rawFs 和 tapeFs、支持在一个单独的 VxWorks 系统上同时并存几个不同的文件系统。

板级支持包 BSP（Board Support Package）

板级支持包向VxWorks操作系统提供了对各种板子的硬件功能操作的统一的软件接口，它是保证VxWorks操作系统可移植性的关键，它包括硬件初始化、中断的产生和处理、硬件时钟和计时器管理、局域和总线内存地址映射、内存分配等等。每个板级支持包包括一个 ROM 启动（Boot ROM）或其它启动机制。

#### 1.1.4 网络支持

它提供了对其它VxWorks系统和TCP/IP网络系统的"透明"访问，包括与BSD套接字兼容的编程接口，远程过程调用（RPC），SNMP（可选项），远程文件访问（包括客户端和服务端的NFS机制以及使用RSH，FTP 或 TFTP的非NFS机制）以及BOOTP 和代理ARP、DHCP、DNS、OSPF、RIP。无论是松耦合的串行线路、标准的以太网连接还是紧耦合的利用共享内存的背板总线，所有的 VxWorks 网络机制都遵循标准的 Internet 协议。

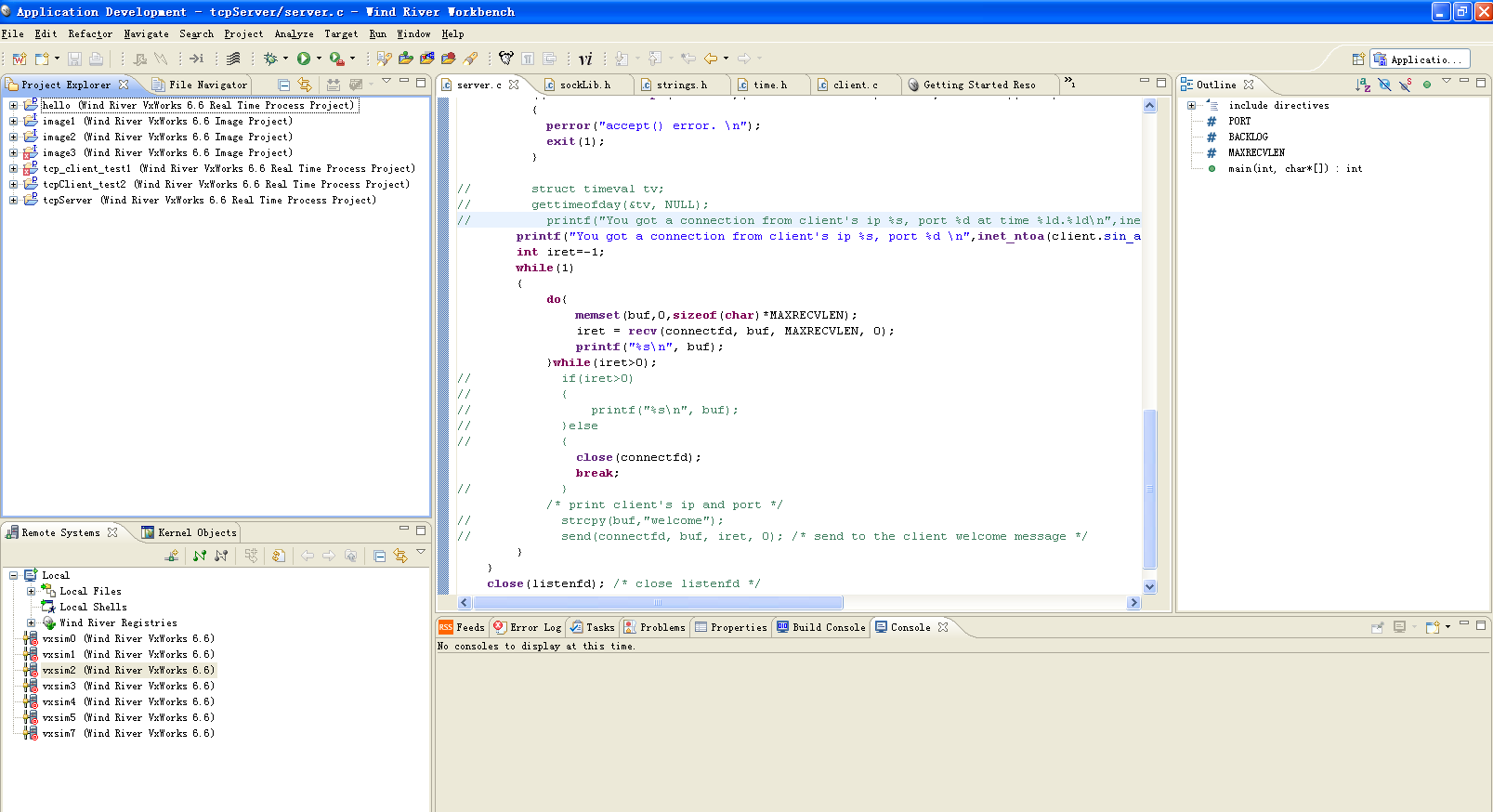
## WorkBench简介及使用说明

### Workbench 3.0简介

风河Workbench 3.0是基于eclipse平台发展出的开发工具（所以其风格非常类似eclipse），提供非常有效的方式去开发实时嵌入式的应用程序，而不必过多的依赖于目标系统。

风河WorkBench课运行在window、liunx、和Solaris主机上，笔者则是介绍在windows主机上使用WorkBench 3.0.

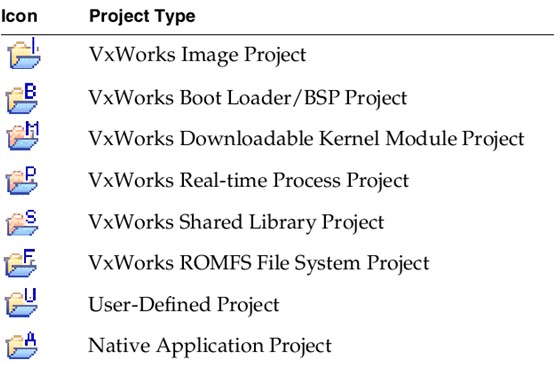
一个Workbench 3.0 界面



### Workbench 3.0使用说明

#### 2.2.1 工程简介

不同类型的工程有着不同的用处，Workbench支持多种类型的工程，在Project Explorer窗口可以看见不同类型的工程有着不同的符号标志，见下图：



##### 2.2.1.1 VxWorks Image Project（VIP)

用VxWorks Image Project 工程配置（自定义或裁剪）并编译一个内核映像用以启动目标板（ boot target）。通过添加VxWorks ROMFS 文件系统、内核模块（kernel modules）、应用程序（applications）、 库（libraries）、 数据文件（data files），可以把整个系统联接进如一个映像（single image）。

一个VxWorks Image project既可基于一个同类型的Image project也可基于一个BSP( Board Support Package)。

##### 2.2.1.2 VxWorks Boot Loader/BSP Project（BSP）

VxWorks Boot Loader/BSP工程用于创建一个VxWorks boot loader也称为VxWorks boot ROM 在启动时下载一个带vxworks内核的目标映像。也可以把一个已经存在的同类型的工程的源文件复制到使用者创建的工程中，然后在不改变 VxWorks 所在目录的情况下自定义工程。

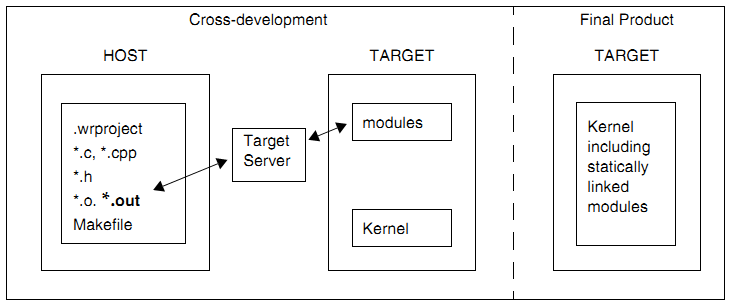
在开发环境中boot loader被用来从主机下载一个VxWorks映像；在最终的产品中boot loader和操作系统映像存储在同一个磁盘上。

boot loader和VxWorks并不需要在ROM中单独存储。

##### 2.2.1.3 VxWorks Downloadable Kernel Module Project(DKM)

利用VxWorks Downloadable Kernel Module 工程管理和建立一个存在于内核空间（kernel space）模块（modules）。可以单独建立模块运行和调试，可以自由的在目标（target）上运行VxWorks，下载，卸载，重载。一旦模块完成就可以静态的链接到内核（kernel），或者模块也可以使用一个存在的文件系统。

DKM工程示意图（不包含文件系统）



内核模式的开发是VxWorks 传统的开发方法，所有的任务（tasks）都在启动（spawn）之后运行在一个没有保护的环境。

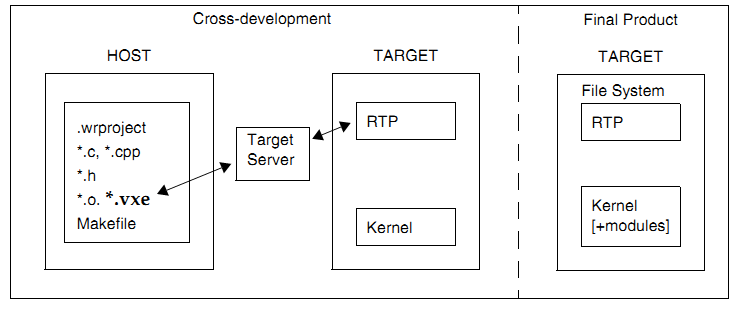
Downloadable Kernel Module在目标已经启动之后，可以启动应用程序。

##### 2.2.1.4 VxWorks Real-time Process Project（RTP)

VxWorks Real-time Process可以管理和建立可执行文件在内核空间之外。可以单独运行和调试可执行文件。

对于Real-time任务就像一个完全独立的程序下载到一个单独的程序运行地址空间（ process address space）。一个Real-time的程序的二进制文件可以像ROMFS文件系统一样存在于目标的（target-side）文件系统。

Real-time 程序执行时对于内核就像单独的实体，如下图所示：



关于如何连接到共享库（Shared Libraries），参看平台所带的帮助信息。

##### 2.2.1.5 VxWorks Shared Library Project（Libs）

VxWorks Shared Library的库文件会动态连接到运行的VxWorks实时程序，要连接到RTP工程，必须将共享库（ shared library）储存在目标侧的文件系统。也可以通过VxWorks Shared Library工程建立一个子工程在编译时静态链进入其他的工程。

更多的信息在平台帮助信息中查看，其路径为： Help > Cheat Sheets > Wind River Workbench > Setup a VxWorks RTP with a shared library 。

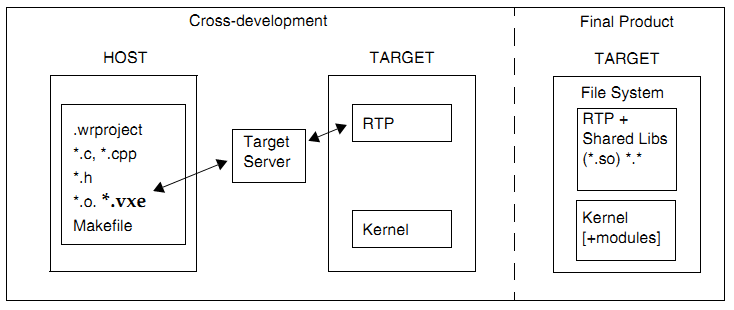
##### 2.2.1.6 VxWorks ROMFS File System Project（ROMFS)

VxWorks ROMFS File System 工程可以作为任何一个需要目标侧文件系统的工程的子工程。

这种工程类型设计用来建立应用程序或者其他任何类型的文件，带一个VxWorks系统映像的ROMFS文件系统，除了是vxworks的引导映像并不需要任何的存储介质**。**所以并不需要其他的文件系统存储文件，系统也具有完全的功能而不需要求助local文件系统、NFS 驱动、RSH 、或者FTP 协议等等。

需要注意的是ROMFS与ROM 媒介没有任何关系，名字代表的意思是Read Only Memory File System。

ROMFS工程示意图：



##### 2.2.1.7 User-Defined Projects（UDP)

User-Defined projects 为用户设置和保存自己的编译系统、文件系统布局等等。用户接口提供以下支持：

可以用编译命令（build command ）开展用户的编译功能，这样就允许在Workbench GUI界面进行编译。

在Project Explorer窗口中建立的工程，它的编译规则可以反映在makefiles文件中。这样就允许在Project Explorer窗口选择任何用户自己的编译规则。

可以在Build Console窗口看到编译工程时的输出信息。

##### 2.2.1.8 Native Application Project（NAP)

Native Application工程用于主机环境下的C/C++应用程序的开发。Wind River Workbench为当地GNU 2.9x, GNU 3.x, 和Microsoft 开发功能（ssembler, compiler, linker, archiver)提供编译和源文件分析支持，但没有调试器，所以得选择合适的当地工具用于调试（debugging）。

### VxWorks模拟器

#### 2.3.1 Vxworks Simulator简介

风河VxWorks模拟器可以模拟硬件目标作为VxWorks的样机和试验台环境。通过VxWorks模拟器，可以在自己的主机系统上开发，运行和测试应用程序，同时VxWorks模拟器还可以让你建立一个模拟目标网络的开发和测试复杂的网络应用。

note：Vxworks simulator不能用来开发驱动程序

#### 2.3.2 如何自己建立一个模拟器

##### 2.3.2.1 编译一个属于自己的内核镜像

编译一个默认的Vxworks镜像：创建一个VxWorks image project：选择**File > New> VxWorks Image Project**，即可创建一个image项目。

这里比较的重要的是选择Vxworks simulator BSPs，具体如下表：

|  |  |
| --- | --- |
| 主机 | BSP Name |
| Windows | simpc |
| Solaris | solaris |
| Linux | linux |

其余选择默认配置即可

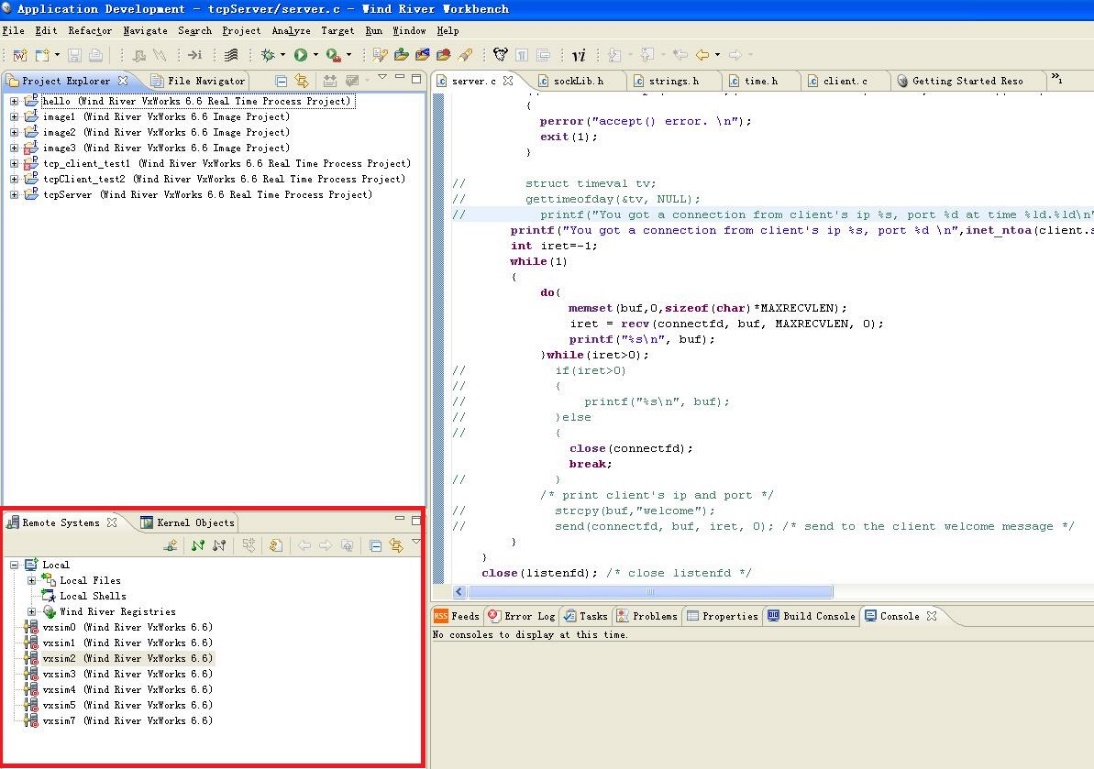
note：在创建镜像项目的过程中可以修改镜像的配置，项目创建完成后，在项目中双击Kernel Configuration，则可以添加自己需要的内核组件，添加完成后重新build即可。

编译好的Vxworks镜像在项目路径 > default > vxWorks，在default目录下有两个名vxWorks的文件，我们需要选择的是文件类型为“文件”的，而非“SYM文件”

##### 2.3.2.2 创建一个属于自己的Vxworks simulator

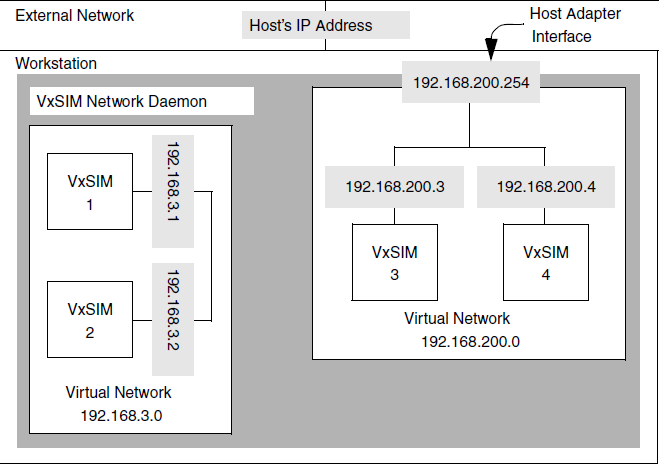
**Target >New Connection > Wind River VxWorks 6.x Simulator Connection**，一直选择默认设置则可以创建一个最简单的模拟器

在Workbench默认工作界面的坐下角的Remote System选项卡可以打开刚刚创建的模拟器。



## Vxworks模拟器网络配置

### Vxworks模拟器网络通信原理



如图所示，简单的说，不同的VxSIM间可以通过 Network daemon通信，而Vxworks与主机则是通过一个NetWork Adapter进行通信。

如果要使Vxworks simulator能实现网络通信功能，WorkBench 默认的simulator使用的Vxworks镜像编译时并没有加入网络功能的模块，所以这里需要编译属于自己的Vxworks镜像，利用该image配置一个simulator，并利用simulated subnets为该模拟器分配一个IP地址。

### 3.2 编译一个具有网络模块的Vxworks image

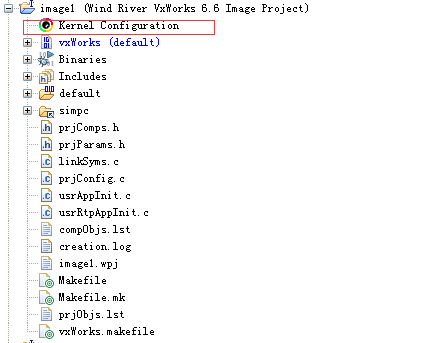
note： 在笔者找到的风河官方的模拟器使用教程里有提到可以通过adapter实现模拟器与external network的通信，这里的external network不知道是外网还是额外的网络，在多方求证无果的情况下，笔者用了两周的时间，并没有使得simulator能与外网通信。最后是通过window Xp主机做交换，完成模拟器与其他同一局域网内其他主机的网络通信

1. 创建一个VxWorks image project：
2. 选择**File > New> VxWorks Image Project**
3. 输入项目名称，并选择项目的location（可以自行选择，不影响后续结果），click **Next**
4. 选择时候导入一个现有的项目配置（不导入）

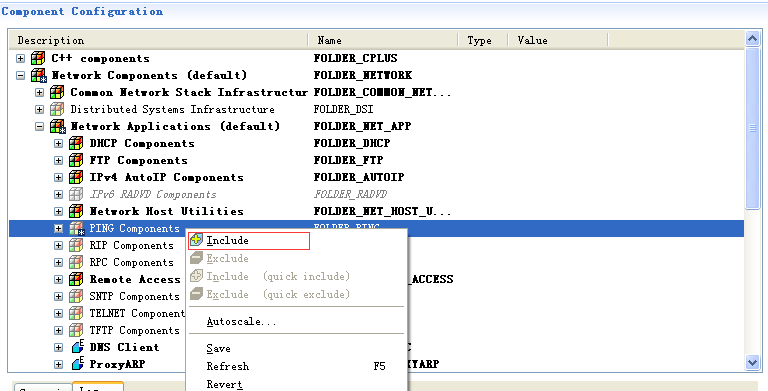
选择A board support package ,具体的选择simpc（详见2.3.2.1小节），Tool chain选择diab（这是风河自己开发的编译器）

click **Next**

1. 勾选IPv6 enabled kernel libraries（不勾选的话需要在项目创建完成后再去配置内核选项），click **Next**
2. 其余选择默认配置即可，直接click **Finish**
3. 项目创建完成后，双击项目列表里的**Kernel Configuration，**进入内核配置界面

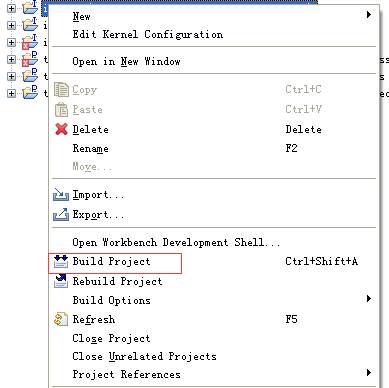


如图所示，在内核组件中加入Ping Component，ping command和ping client都需要勾选，然后**Finish**



1. 编译Vxworks image

在项目名称右击，然后选择**Build Project**



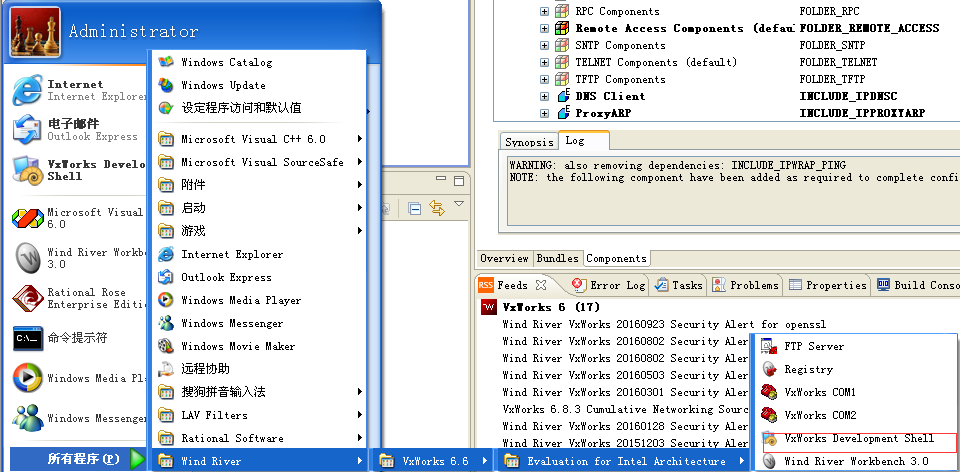
1. 编译完成

编译好的Vxworks镜像在项目路径 > default > vxWorks，在default目录下有两个名vxWorks的文件，我们需要选择的是文件类型为“文件”的，而非“SYM文件”

### 3.3 配置simulator前的预设条件

#### 安装Network daemon

1. 打开Vxworks Development Shell（必须以管理员权限打开）



1. 卸载原有服务

在打开的命令行中输入以下指令

installDir**/vxworks-6.x/host/x86-win32/bin/vxsimnetds\_inst.exe /u**

note：其中的installDir是Vxworks的安装目录

1. 安装一个新的network daemon

*installDir***/vxworks-6.***x***/host/x86-win32/bin/vxsimnetds\_inst.exe**

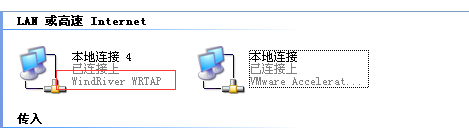
#### 开启Wind river的服务

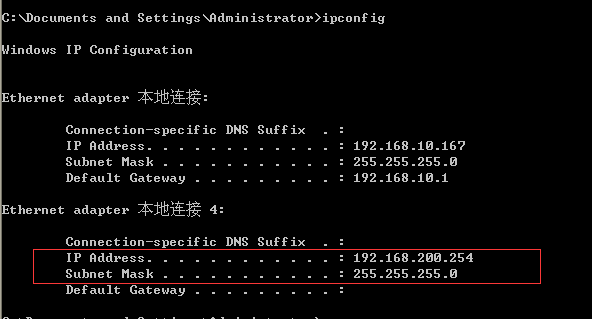
1. 打开控制面板 > 管理工具 > 服务
2. 选择**Wind River Network Daemon for VxWorks Simulator**，右键鼠标选择**Start，**开启服务

note：这种默认的安装方式，得到的网段是在192.168.200.0子网，即创建的simulator的IP设置应该为192.168.200.X（否则会出错）

#### 安装Wind River的网络适配器

1. 打开控制面板，双击添加硬件，在打开的向导中click **Next**，选择“是，我已经连接了此硬件”。click **Next**
2. 选择“添加新的硬件设备”,c click **Next**，选择“安装我手动从列表选择的硬件” click **Next**
3. 选择“网络适配器”，click **Next**
4. 选择“从磁盘安装”，click **Next**
5. 浏览installDir\vxworks-6.x\host\x86-win32\bin
6. 选择“wrtap.inf”，click **Open**
7. 选择**WindRiver WRTAP**, click **Next**.
8. 其余选择**next**，即可
9. 验证，打开网络连接，应当如图所示，如果WindRiver WRTAP的网络连接显示未连接，请重启电脑

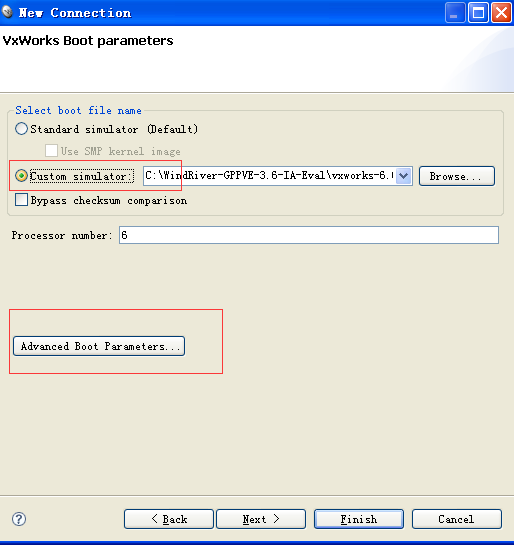




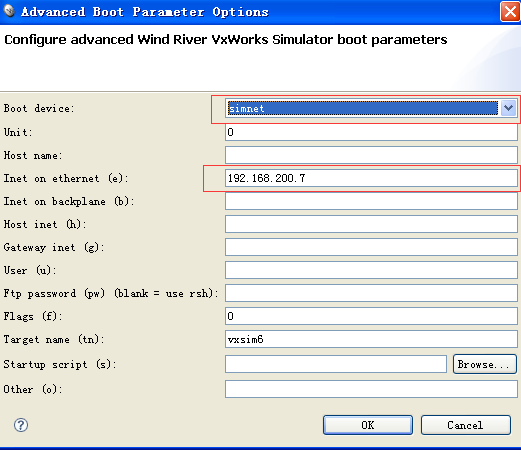
#### 3.1.4 配置具有IP的simulator

1. **Target >New Connection > Wind River VxWorks 6.x Simulator Connection**
2. 在打开的界面勾选**Custom simulator**，选择刚刚编译好的Vxworks image

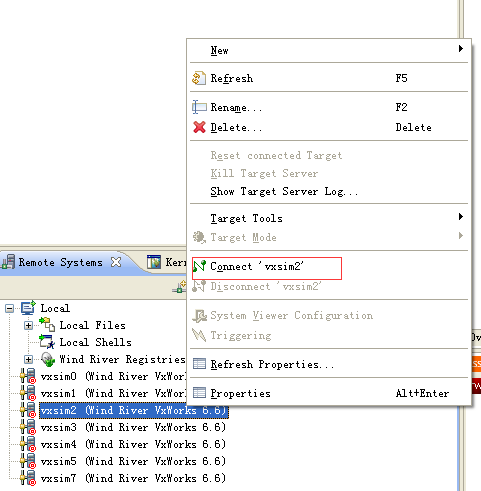
click **Advanced Boot Parameters**



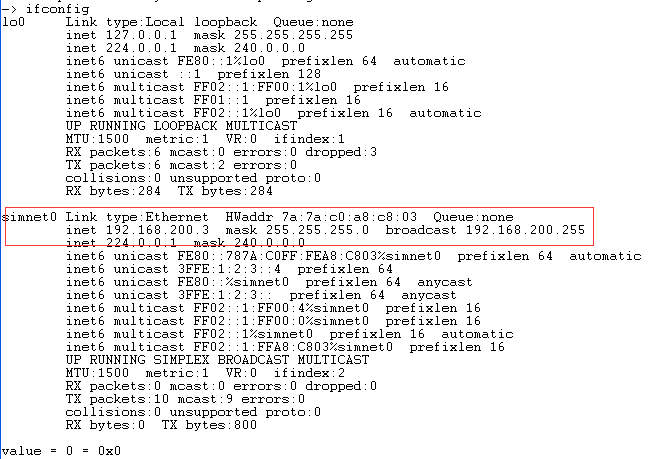
1. 在打开的配置界面，**Boot device**选择simnet，IP地址会默认分配好（不建议更改，容易出现问题），然后click **OK**



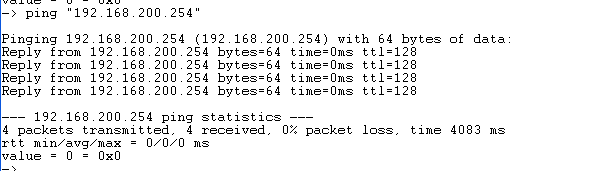
1. 其余选择默认配置即可
2. 验证是否配置成功。按照2.3.2.2小节中所述打开模拟器



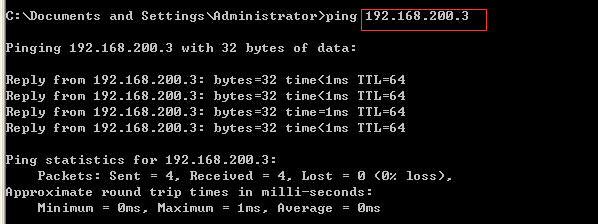
1. 在打开的界面，输入命令ifconfig，出现刚刚配置好的IP地址（这里由于笔者打开的是之前已经创建好的模拟器，所以IP不是刚刚创建的192.168.200.7）



1. 验证网络连通性,在模拟器中输入命令：ping “192.168.200.254”



在windows 主机中



## Workbench中创建一个网络连接项目

1. 创建一个RTP项目（详见2.2.1.4）
2. 具体项目代码见附录（所设置的IP地址均是笔者电脑的IP，请自行更改）
3. 运行方法见5小节，注意选择目标模拟器时必须选择刚刚编译的具有IP地址的模拟器

由于Vxworks模拟器只能与主机的windRiver的网络适配器连接（IP为192.168.200.254），所以如果模拟器作为Tcp client连接主机的Tcp server，client设置的目标服务器IP必须为192.168.200.254

实现模拟器与其他主机的Tcp通信；

假设安装Vxworks的主机为A，Vxworks需要通信的主机为B

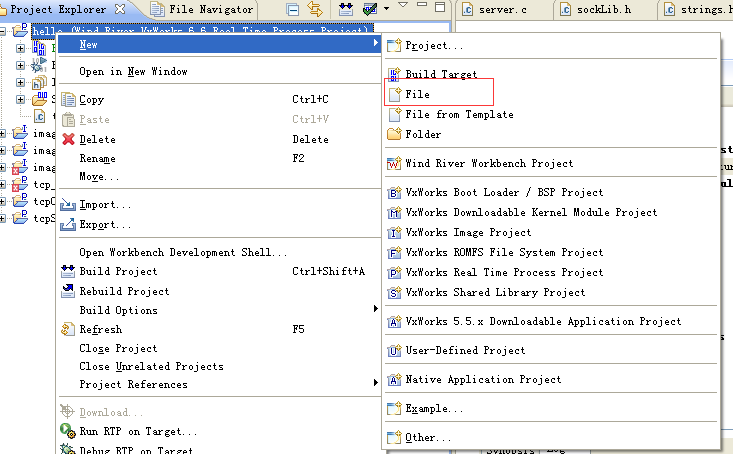
则有如下方案：

1. 在Vxworks模拟器上运行一个Tcp server，主机B上运行一个Tcp server，有数据data；
2. 主机A上写一个中转程序（示例见附录），该程序连接主机B，得到主机B发送的data，主机A的中转程序同时连接A，将data转发给Vxworks模拟器

## 如何运行一个RTP项目

笔者以一个简单的hello world程序为例

1. 建立一个RTP项目，项目名为hello（详见2.2.1.4）
2. 在项目名上右击，**New > File** ，文件名为test.c



1. 在test.c文件里写下如下代码

#include <stdio.h>

int main()

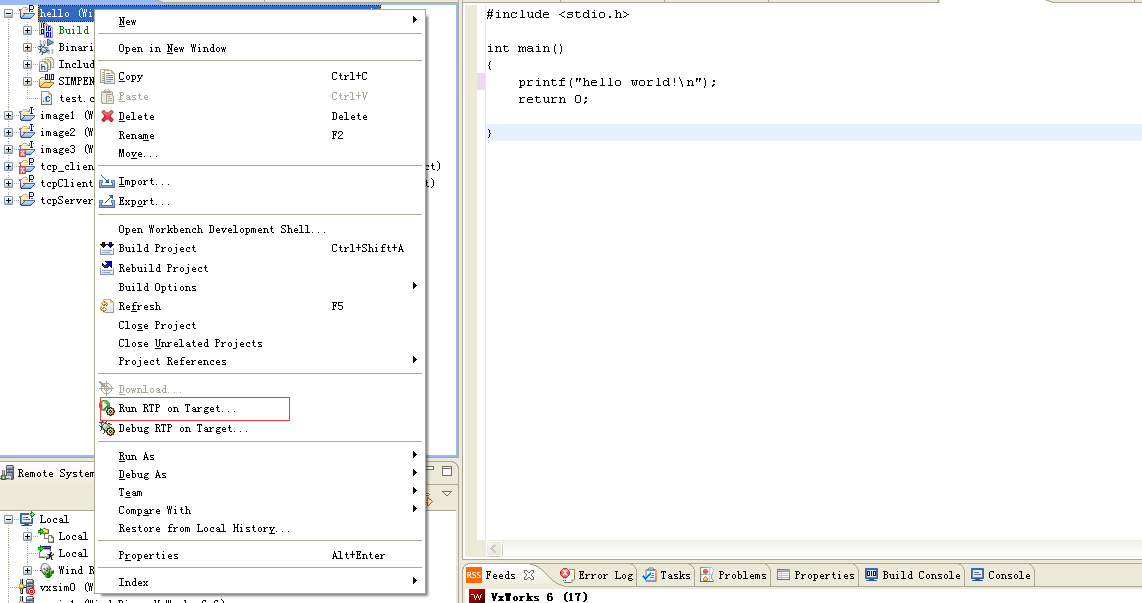
{

printf("hello world!\n");

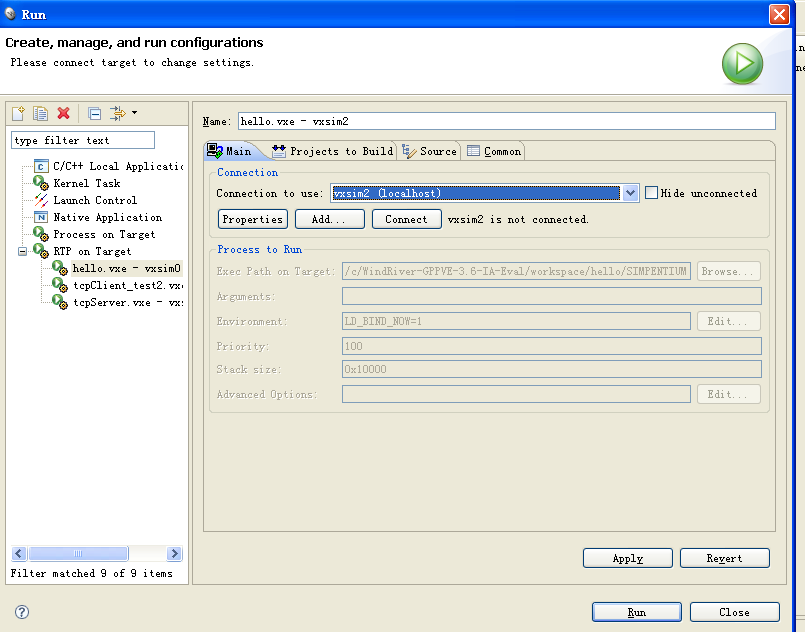
return 0;

}

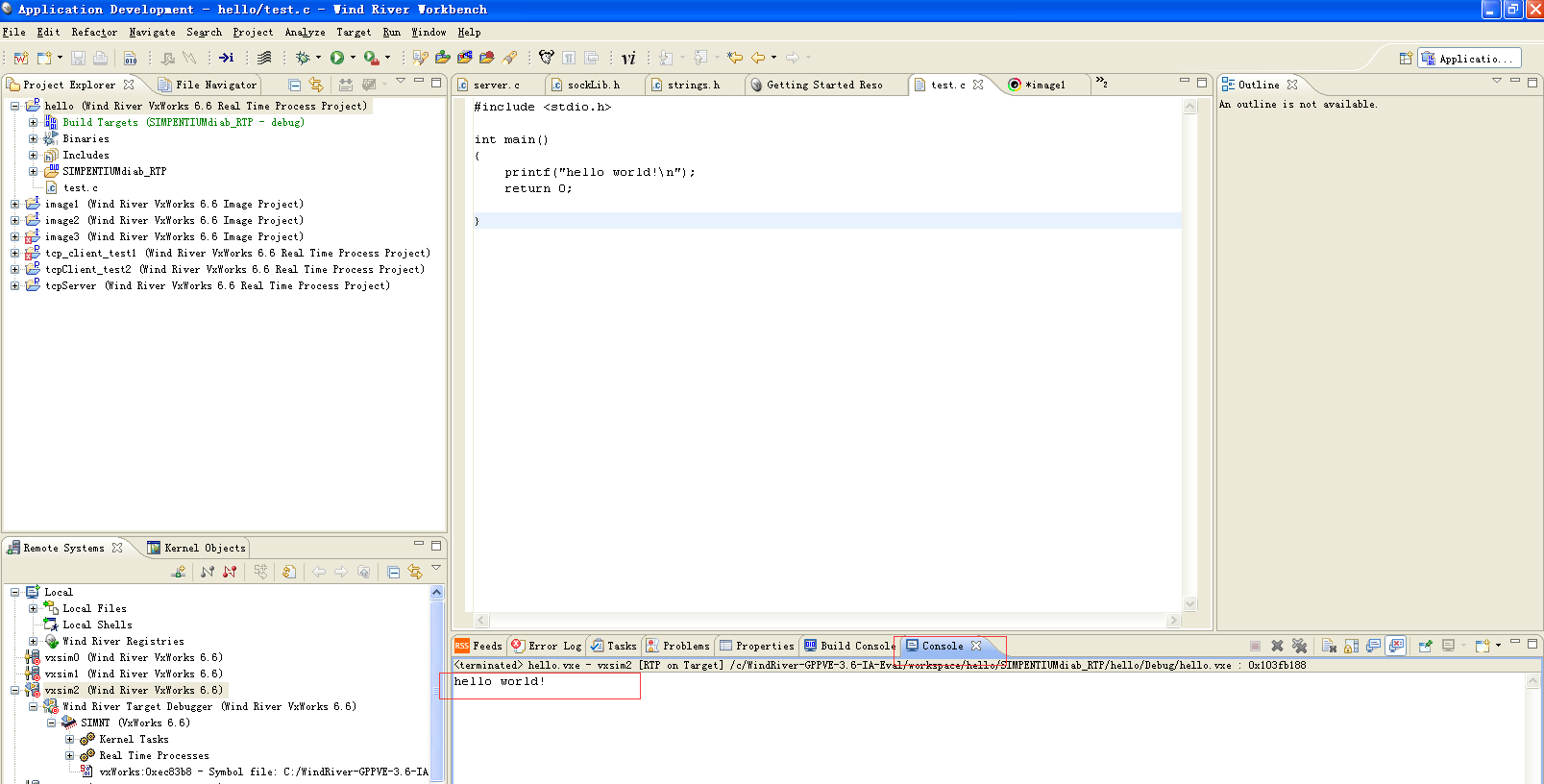
1. 运行项目：在项目名上右击，选择**Run RTP on Target**



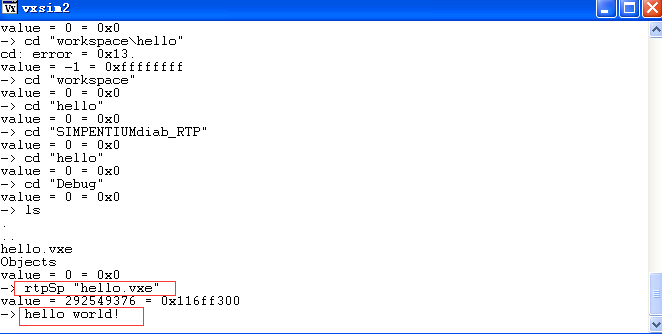
1. 选择好目标模拟器，然后click **Run**



1. 在下方的Console界面会出现模拟器运行的仿真结果



1. 如何在模拟器界面运行一个程序：
2. 进入程序界面：cd "host:c:"（进入主机C盘，建议将可执行程序hello.vxe复制到根目录下运行）
3. 运行：rtpSp "hello.vxe"



## 附录

### Vxworks上TCP Client代码

**#include** <stdlib.h>

**#include** <stdio.h>

**#include** <unistd.h>

**#include** <string.h>

**#include** <sys/types.h>

**#include** <sys/socket.h>

**#include** <netinet/in.h>

**#include** <netdb.h> /\* netdb is necessary for struct hostent \*/

**#include** <wrapper/wrapperHostLib.h>

**#include** <sockLib.h>

**#include** <strings.h>

**#define** PORT 1024 /\* server port \*/

**#define** MAXDATASIZE 100

**int** **main**(**int** argc, **char** \*argv[])

{

**int** sockfd, num; /\* files descriptors \*/

**char** buf[MAXDATASIZE]; /\* buf will store received text \*/

**struct** hostent \*he; /\* structure that will get information about remote host \*/

**struct** sockaddr\_in server;

**char** \*IP\_address ="192.168.200.254";

**if** ((he = **gethostbyname**(IP\_address)) == NULL)

{

**printf**("gethostbyname() error\n");

**exit**(1);

}

**if** ((sockfd = **socket**(AF\_INET, SOCK\_STREAM, 0)) == -1)

{

**printf**("socket() error\n");

**exit**(1);

}

server.sin\_family = AF\_INET;

server.sin\_port = htons(PORT);

server.sin\_addr = \*((**struct** in\_addr \*)he->h\_addr);

**if** (**connect**(sockfd, (**struct** sockaddr \*)&server, **sizeof**(server)) == -1)

{

**printf**("connect() error\n");

**exit**(1);

}

**char** str[] = "horst\n";

**if** ((num = **send**(sockfd, str, **sizeof**(str), 0)) == -1){

**printf**("send() error\n");

**exit**(1);

}

**if** ((num = **recv**(sockfd, buf, MAXDATASIZE, 0)) == -1)

{

**printf**("recv() error\n");

**exit**(1);

}

buf[num - 1] = '\0';

**printf**("server message: %s\n", buf);

**close**(sockfd);

**return** 0;

}

### （二） Vxworks上TCP Servert代码

**#include** <sys/time.h>

**#include** <stdlib.h>

**#include** <stdio.h>

**#include** <string.h>

**#include** <unistd.h>

**#include** <sys/types.h>

**#include** <sys/socket.h>

**#include** <netinet/in.h>

**#include** <arpa/inet.h>

**#include** <sockLib.h>

**#include** <strings.h>

**#include** <sys/time.h>

**#define** PORT 1025

**#define** BACKLOG 1

**#define** MAXRECVLEN 1024

**int** **main**(**int** argc, **char** \*argv[])

{

**char** buf[MAXRECVLEN];

**int** listenfd, connectfd; /\* socket descriptors \*/

**struct** sockaddr\_in server; /\* server's address information \*/

**struct** sockaddr\_in client; /\* client's address information \*/

socklen\_t addrlen;

/\* Create TCP socket \*/

**if** ((listenfd = **socket**(AF\_INET, SOCK\_STREAM, 0)) == -1)

{

/\* handle exception \*/

**perror**("socket() error. Failed to initiate a socket");

**exit**(1);

}

/\* set socket option \*/

**int** opt = SO\_REUSEADDR;

**setsockopt**(listenfd, SOL\_SOCKET, SO\_REUSEADDR, &opt, **sizeof**(opt));

**bzero**(&server, **sizeof**(server));

server.sin\_family = AF\_INET;

server.sin\_port = htons(PORT);

server.sin\_addr.s\_addr = htonl(INADDR\_ANY);

**if**(**bind**(listenfd, (**struct** sockaddr \*)&server, **sizeof**(server)) == -1)

{

/\* handle exception \*/

**perror**("Bind() error.");

**exit**(1);

}

**if**(**listen**(listenfd, BACKLOG) == -1)

{

**perror**("listen() error. \n");

**exit**(1);

}

addrlen = **sizeof**(client);

**while**(1){

**if**((connectfd=**accept**(listenfd,(**struct** sockaddr \*)&client, &addrlen))==-1)

{

**perror**("accept() error. \n");

**exit**(1);

}

**printf**("You got a connection from client's ip %s, port %d \n",**inet\_ntoa**(client.sin\_addr),htons(client.sin\_port));

**int** iret=-1;

**while**(1)

{

**do**{

**memset**(buf,0,**sizeof**(**char**)\*MAXRECVLEN);

iret = **recv**(connectfd, buf, MAXRECVLEN, 0);

**printf**("%s\n", buf);

}**while**(iret>0);

**close**(connectfd);

**break**;

}

}

**close**(listenfd); /\* close listenfd \*/

**return** 0;

}

### （三） 主机上作为中转站的Tcp代码

#include "stdafx.h"

#include <WINSOCK2.H>

#include <STDIO.H>

#pragma comment(lib,"ws2\_32.lib")

void test\_client(char \*meassage);

int main(int argc, char\* argv[])

{

WORD sockVersion = MAKEWORD(2, 2);

WSADATA data;

if (WSAStartup(sockVersion, &data) != 0)

{

return 0;

}

SOCKET sclient = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

if (sclient == INVALID\_SOCKET)

{

printf("invalid socket !");

return 0;

}

sockaddr\_in serAddr;

serAddr.sin\_family = AF\_INET;

serAddr.sin\_port = htons(1024);

serAddr.sin\_addr.S\_un.S\_addr = inet\_addr("192.168.10.174");

if (connect(sclient, (sockaddr \*)&serAddr, sizeof(serAddr)) == SOCKET\_ERROR)

{

printf("connect error !");

closesocket(sclient);

return 0;

}

char \* sendData = "你好，TCP\_Vxworks，我是客户端!\n";

send(sclient, sendData, strlen(sendData), 0);

char\* recData = new char[255];

int ret;

do{

memset(recData, 0, sizeof(char)\* 255);

ret = recv(sclient, recData, 255, 0);

recData[ret] = 0x00;

printf("%s\n",recData);

test\_client(recData);

} while (ret > 0);

closesocket(sclient);

WSACleanup();

return 0;

}

void test\_client(char \*meassage)

{

WORD sockVersion = MAKEWORD(2, 2);

WSADATA data;

if (WSAStartup(sockVersion, &data) != 0)

{

return;

}

SOCKET sclient = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

if (sclient == INVALID\_SOCKET)

{

//printf("invalid socket !");

return;

}

sockaddr\_in serAddr;

serAddr.sin\_family = AF\_INET;

serAddr.sin\_port = htons(1025);

serAddr.sin\_addr.S\_un.S\_addr = inet\_addr("192.168.200.3");

if (connect(sclient, (sockaddr \*)&serAddr, sizeof(serAddr)) == SOCKET\_ERROR)

{

//printf("connect error !");

closesocket(sclient);

}

char recData[255];

send(sclient, meassage, strlen(meassage), 0);

closesocket(sclient);

WSACleanup();

}