VxWorks 7 SDK for Raspberry Pi 4

Introduction

The VxWorks 7 SDK is a development environment dedicated to VxWorks application developers which includes the following features:

- standard cross-compilation tools based on clang/LLVM which can be used to build both downloadable kernel modules (DKM) and RTP (Real Time Process) applications
- simplified build management: makefile, cmake, roll-your own
- target / architecture specific: includes a generic VxWorks kernel which is bootable on the target platform
- header files and libraries for application development
- Wind River Debugger (wrdbg)
- documentation

This guide helps you get up and running with developing applications for platforms running VxWorks. You can use it for creating new applications, or just exploration of VxWorks capabilities.

Setting up the development environment

You should start by downloading a VxWorks SDK for your platform of choice from https://labs.windriver.com and unpacking it. Refer to the documentation in docs in the unpacked SDK for additional information on creating and debugging applications.

OS requirements

The SDKs are meant to run on Linux hosts. Some of the examples in this document are specific to Debian derivatives.

Prerequisite(s)

Host dependencies

On Debian derivatives, the following packages need to be installed:

```
$ sudo apt install build-essential libc6:i386
```

Having an FTP server installed on your development host will make application deployment easier and allow you to access the host file system from a VxWorks target.

To accommodate for the varying runtime configurations of the VxWorks kernel images included in the SDKs, you may be interested in using an FTP server option based on pyftpdlib.

Install pyftpdlib:

```
$ sudo apt install python-pip
$ sudo pip install pyftpdlib
```

Booting VxWorks on Raspberry Pi 4

1. To create the SD card, download the firmware from:

   https://github.com/raspberrypi/firmware/archive/1.20200212.tar.gz

   E.g. wget https://github.com/raspberrypi/firmware/archive/1.20200212.tar.gz

   Format an SD card as a FAT32 file system and copy the contents of the "boot" directory from the downloaded firmware to the SD card.

2. Compile a u-boot binary for Raspberry Pi 4 and copy it to the SD card

   E.g. on a Ubuntu/Debian host

   $ sudo apt install gcc-aarch64-linux-gnu $ git clone https://gitlab.denx.de/u-boot/u-boot.git
   $ cd u-boot
   $ CROSS_COMPILE=aarch64-linux-gnu- make rpi_4_defconfig
   $ CROSS_COMPILE=aarch64-linux-gnu- make

   Copy u-boot.bin to the SD card as u-boot-64.bin

3. Copy the files from /vxsdk/sdcard/ to the SD card.

4. Copy the VxWorks kernel image /vxsdk/bsps/rpi_4_0_1_1_1/uVxWorks to the SD card.

5. Connect a USB to TTL serial cable to the Raspberry Pi

On the Raspberry Pi 4 Model, GPIOs 14 and 15 are used as UART transmit and receive pins (Mini UART).

Connect the USB to serial cable adapter between the Raspberry Pi and your PC. Then start a serial communication program (e.g. minicom) and configure the serial connnection parameters as follows:

```
Bps/Par/Bits       : 115200 8N1
Hardware Flow Control : No
Software Flow Control : No
```

After plugging in the SD card in your Raspberry Pi and powering it up, the VxWorks kernel previously copied onto the SD card will boot automatically.

```
U-Boot 2020.07-rc2-00133-ged9a3aa645 (May 19 2020 - 12:49:43 +0300)

DRAM:  3.9 GiB
RPI 4 Model B (0xc03112)
MMC:   emmc2@7e340000: 0, mmcnr@7e300000: 1
Loading Environment from FAT... OK
In:    serial
Out:   serial
Err:   serial
Net:
Warning: genet@7d580000 MAC addresses don't match:
Address in DT is                dc:a6:32:86:d8:ef
Address in environment is       dc:a6:32:07:b3:a4
eth0: genet@7d580000
Hit any key to stop autoboot:  0
8944948 bytes read in 773 ms (11 MiB/s)
## Booting kernel from Legacy Image at 00100000 ...
   Image Name:   vxworks
   Image Type:   AArch64 VxWorks Kernel Image (uncompressed)
   Data Size:    8944884 Bytes = 8.5 MiB
   Load Address: 00100000
   Entry Point:  00100000
   Verifying Checksum ... OK
   Loading Kernel Image
   !!! WARNING !!! Using legacy DTB
## Starting vxWorks at 0x00100000, device tree at 0x00000000 ...
```

```
Target Name: vxTarget
Instantiating /tmp as rawFs,  device = 0x1


 _________           _________
 \77777777\          /77777777/
  \77777777\        /77777777/
   \77777777\      /77777777/
    \77777777\    /77777777/
     \77777777\   \7777777/
      \77777777\   \77777/              VxWorks 7 SMP 64-bit
       \77777777\   \777/
        \77777777\   \7/     Release version: SR0660
         \77777777\   -      Build date: Feb  3 2021 14:26:49
          \77777777\
           \7777777/         Copyright Wind River Systems, Inc.
            \77777/   -                 1984-2021
             \777/   /7\
              \7/   /777\
               -   -------

                   Board: Raspberry Pi 4 Model B - ARMv8
                CPU Count: 4
             OS Memory Size: ~3891MB
          ED&R Policy Mode: Deployed
     Debug Agent: Started (always)
         Stop Mode Agent: Not started
       BSP Status: *** UNSUPPORTED ***

 Adding 13658 symbols for standalone.

->
```

Supported features:

- SD card
- USB mass storage
- network, defaulting to DHCP

Application development

Start by opening a Linux terminal window and going to the location of your unpacked VxWorks SDK.

Source the SDK development environment before using the SDK.

```
$ source sdkenv.sh
```

Developing applications from the command line

The VxWorks compiler driver (wr-cc/wr-c++) utility lets you invoke VxWorks toolchains to compile your real time process (RTP) application from command line. The SDK development environment includes a set of environment variables (CC, CXX, CPP, etc) which match the various compiler driver utilities.

Procedure:

1. Create a C source file foo.c:

```
#include <stdio.h>

int main(void)
    {
    printf("hello, world!\n");
    return 0;
    }
```

1. Compile the source file foo.c to create a Real Time Process application.

```
$ wr-cc -rtp foo.c -static -o foo.vxe
```

or

```
$ $CC -rtp foo.c -static -o foo.vxe
```

Alternatively, you can also create a Makefile and then call make to compile the source file.

```
all:
    $CC -rtp foo.c -static -o foo.vxe
```

```
$ make
```

Developing applications using CMake

By default, a generated VxWorks SDK includes CMake. When the application developer sources the SDK environment, they will automatically have access to CMake. There is no need to download, install and configure CMake separately. With VxWorks CMake, you can build existing CMake examples that are included in the generated SDK, or you can create your own projects.

CMake examples can be found in the directories /examples/rtp/hello_cmake_rtp and /examples/dkm/hello_cmake_dkm

Procedure: 1. Create a project directory, called my_project.

```
$ mkdir my_project
```

1. Create the CMakeLists.txt file in the project folder.
2. To build your project, run the following command:

```
$ cmake -D CMAKE_TOOLCHAIN_FILE=${WIND_SDK_HOME}/vxsdk/sysroot/mk/rtp.toolchain.cmake .
```

> **Note**: RTP is used in the example. If you would like to build a DKM project instead, change the toolchain name to dkm.toolchain.cmake.

Running applications

Start an FTP server on port 21 with user "target" and password "vxtarget" and FTP root in the current user's home.

```
$ sudo python -m pyftpdlib -p 21 -u target -P vxTarget -d $HOME &
```

Make a note of the IP address of your development host.

If your Raspberry Pi has successfully completed DHCP negotiation, it will be able to access the host file system via a VxWorks remote file device.

In the vxWorks shell run the following command to create a device named "wrs".

```
-> netDevCreate ("/wrs", "192.168.10.191", 1)
```

> **Note:** Replace 192.168.10.191 with the IP address of your development host (i.e. the one on which you're running the FTP server).

Switch to the cmd shell and navigate to the location of your application, now available on the target via the remote file device created previously.

E.g.

```
-> cmd
[vxWorks *]# cd /wrs
[vxWorks *]# pwd
/wrs/
[vxWorks *]# ls foo.c
foo.c
[vxWorks *]# more foo.c
#include <stdio.h>

int main(int argc, char **argv) {
    printf("Hello World\n");
    return 0;
}
[vxWorks *]#
[vxWorks *]# foo.vxe
Launching process 'foo.vxe' ...
Process 'foo.vxe' (process Id = 0xffff8000005e85d0) launched.
Hello world
```

Creating and deploying Python applications

Prerequisites

The SD card you have created earlier includes all the prerequisites for running python applications on VxWorks.

Launch Python from the cmd shell to print the Hello World! string:

```
[vxWorks *] cd /usr
[vxWorks *]# python3
Launching process 'python3' ...
Process 'python3' (process Id = 0xffff800000641380) launched.
Python 3.8.0 (default, Feb  2 2021, 19:28:52)
[Clang 10.0.1.1 (http://gitlab.devstar.cloud/compilers/llvm/clang.git 5449acbae on vxworks
Type "help", "copyright", "credits" or "license" for more information.

>>> import sys
>>> print("Hello World!")
Hello World!
>>> print (sys.platform)
vxworks
```

Python scripts can be executed as

```
[vxWorks *] python3 helloworld.py

Hello World!
```

> **Note**: If you launch Python from the shell, you may need to set the value of rtpSpStackSize before starting Python (for example, set the rtpSpStackSize to 0x1000000). For more information, see rtpSp( ) in the API reference.

https://labs.windriver.com/downloads/wrsdk-vxworks7-docs/2103/README_raspberrypi4b.html