# NuttX 的環境架設 QEMU + VSCode – 開發 POSIX 嵌入式系統

Kevin Huang

Jan 24, 2021

NuttX LOGO

## 為什麼用 NuttX?

使用過很多開發版，尤其有現在市面上有很多 Arduino 相容的開發板，要上手也十分容易。而最近幾年，AIoT 這個概念打得火熱，很多廠家都在 MCU 的基礎上把 WiFi 與 Bluetooth 連網的功能加上去，CPU 核心使用了 ARM Cortex-M3 或是更高級的 CPU 核心，SONY 的 Spresense 系列就用了 6 個 Cortex-M4 核心的強大處理器。對於資深的開發者來說，這個組合已經是非常強大，不過跟著原廠出來的開發工具，給 Maker 用的大多就是和 Arduino 相容，或是 FreeRTOS …沒有 POSIX 對於硬核的開發者來說，實在不會很開心。

既然如此，身為 Maker 就需要自己來解決這個問題，我想跑 POSIX 的原因單純只是因為開發 Embedded System 快 25 年經驗累積的不少軟體，從 QNX (現在在車機上大鳴大放的 OS) 到 VxWorks 到 Embedded Linux，大部分都是遵循 POSIX 標準開發的軟體體系。QNX 現在已經是著名的車載系統，現在很多的汽車電子儀表板，裡面跑就是 QNX，堪稱『隱形冠軍』。

FreeRTOS 被 Amazon 買走，我之前慣用的 ThreadX 也被 Microsoft 買走，這兩個 RTOS 在 MCU 的軟體開發上面應該算是蠻主流的 RTOS。而 FreeRTOS 和 ThreadX 上面也都有外掛的 POSIX 支援，但是 NuttX 比較不同的事情是 NuttX 就是原生的 POSIX 支援，在開發或是移植比較大型的軟體上面，這是會讓開發與移植速度更快的優勢。

要快速做到這件事情，我需要建立一個軟體模擬環境，因為我要做的是軟體架構的驗證，真的把系統弄到板子上面去是最後一個階段的工作，一開始不要自己找麻煩，因為光弄到板子，建立開發環境，並且測試就需要花很多時間了，更不用說開發的時候要跟硬體搏鬥，我想節省這個時間，所以我就先建立了 NuttX 在 Ubuntu 20.04 LTS 上面的 QEMU 模擬器，並且使用 gdb 與 VSCode 去開發。

其實 NuttX 的 Build 選項內就有使用 host OS (Linux) 來模擬的選項，但是我不想用這個模式，原因是用這個模式，會導致 symbol table 管理上會混亂。試想，如果今天要使用一個 API 是 pthread_mutex_lock，我到底是使用模擬 NuttX 的 pthread_mutex_lock 還是 Linux 原生的 pthread_mutex_lock? 因此我還是希望能用 QEMU 模擬器，才能更掌握 NuttX 相關的核心調度過程。

## 開始動手

接下來的這個過程，其實流程和我之前寫的一篇文章有點類似 (使用 QEMU + FreeRTOS + Visual Source Code 開發嵌入式系統)，有關 GDB 和 VSCode 設定的部分我就不在這邊贅述了，我在這邊文章先描述如何在 Ubuntu 20.04 LTS 上面修改 QEMU，重新編譯與如何設定 NuttX 的組態，讓他可以在 QEMU 上面跑起來。

## 安裝開發工具

把慣用的開發工具一起安裝上了包含編輯器 (vim)，或是代碼管理工具 (git)，打包工具 (make / cmake / libtool)，網路工具包 (net-tools) … 等等，我們也一併安裝上去。但是和上一篇 FreeRTOS 的過程還是有一些不一樣，這邊我們不能使用 Debian Package Manger 提供的 QEMU，因為上面的 QEMU 裡面並不支援 NuttX 上面

要的開發板型號，爬了一下 QEMU source code，發現其實是有支援的，不過是要悲催地自己下載編譯，而且我們想將 NuttX 的 debug console 導引到 UART1… 這也需要小小修改一下 QEMU 的 source code，稍後再說明。

打入以下指令安裝必要的工具包

```
$ sudo apt install git vim make cmake libtool libtool-bin libfdt-dev gcc gcc-arm-none-eabi gdb gdb-multiarch make net-tools universal-ctags kconfig-frontends -y
```

## 抓取 NuttX 代碼

接著去 GITHUB 把 NuttX 代碼抓下來，新版本的 NuttX 把 Kernel 和 APP 的目錄整個切開了，所以需要分別去把 NuttX Kernel 與 APP 抓下來。另外要注意的是，在 Build Kernel 的時候，他認得的目錄叫做 “nuttx” 與 “apps”，要如同下面的命令去把 NuttX 和 APP 放在正確的目錄內。我自己的習慣是在 $HOME 目錄下面建立一個 “work” 目錄，把 Source Code 放在裡面。

```
$ mkdir -p $HOME/work
$ cd $HOME/work
$ git clone https://github.com/apache/incubator-nuttx.git nuttx
$ git clone https://github.com/apache/incubator-nuttx-apps.git apps
```

## Build Nuttx Kernel

```
$ cd $HOME/work/nuttx
```

這時候要告訴 tools/configure.sh

1. 我們是在 Linux 的環境上編譯
2. 我們的 app 目錄 (注意：app 目錄一定要是 nuttx 目錄的相對位置，在這裡就是 “../apps” )
3. 最後是開發版與 app 名字，我們選擇 ” stm32f103-minimum:nsh“

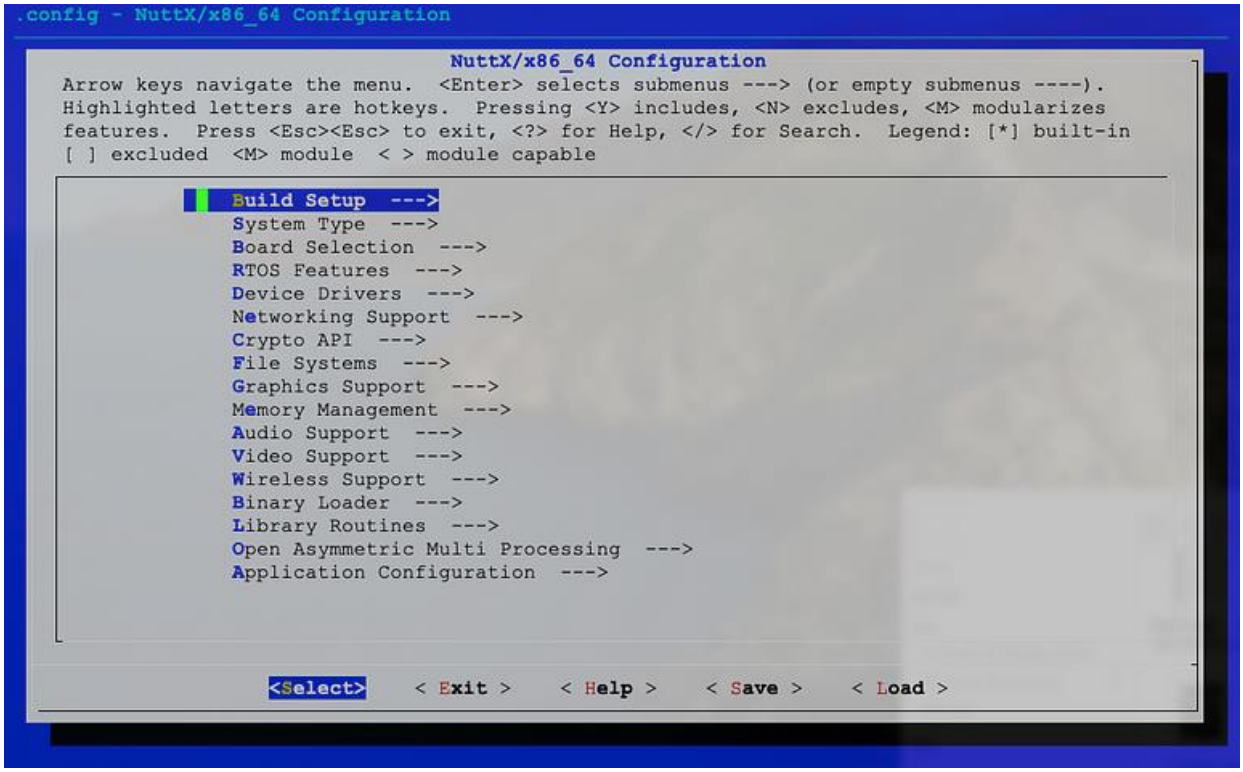
stm32f103 這個開發板就是 Ubuntu 20.04 LTS package manager 裡面的 QEMU 並沒有支援到的板子，所以等一下我們要自己抓 QEMU 下來編譯。選擇 minimum kernel 的組態，執行 NuttShell。

```
$ ./tools/configure.sh -E -l -a ../apps stm32f103-minimum:nsh
```

NuttX 會開始做組態，組態完成之後先不急著 Build，打入下面的指令進入 NuttX 的 Config 選項。

```
$ make menuconfig
```

NuttX 的整個組態的設定方式和 Linux kernel 一樣，用選的，我覺得是滿好的方式，如下圖




NuttX 的組態畫面

1. 在 “System Type” 下面選擇 “ARM” → STMicro STM32 F1/F2/F3/F4/G4/L1 這個選項。
2. 在 “Build Setup” 下面選擇 “Build Host Platform” 為 “Linux”
3. 在 “Build Setup” 下面選擇 “Debug Options” → “Enable Debug Feature”
4. 在 “Build Setup” 下面選擇 “Generate Debug Symbols”

5. 離開之前記得 <Save> 把你的 .config 儲存起來

然後你就可以 build 了…

```
$ make -j16
```

很快的 (NuttX 其實很小)，Kernel 和 Symbol Table 就 Build 出來了，在 $HOME/work/nuttx 目錄下面會產生兩個檔案，一個是 nuttx.bin，這是給 QEMU 執行的，另外一個是 nuttx，這是給 GDB 看的。

## 小修改一下 QEMU 並且編譯

GITHUB 上面有一個針對 STM32 ARM 出來的 QEMU https://github.com/beckus/qemu_stm32.git 雖然有點舊，但是用這個版本測試是沒有問題的，而且簡單並且上手速度快，所以我們下載這個

```
$ git clone https://github.com/beckus/qemu_stm32.git
```

下載完成之後，要小修改一下 UART 的順序，主要是我想將 NuttShell 的 console 從 UART1 打出來，我用 “-serial stdio” 就可以用 Linux 的 tty 操作，而原來的 QEMU 的順序剛好相反，所以小修一下：

```
$ cd $HOME/work/qemu_stm32
$ vi hw/arm/stm32_p103.c
```

還沒有修改前長這樣

```
/* Connect RS232 to UART */
    stm32_uart_connect(
            (Stm32Uart *)uart2,
            serial_hds[0],
            STM32_USART2_NO_REMAP);

    /* These additional UARTs have not been tested yet... */
    stm32_uart_connect(
            (Stm32Uart *)uart1,
            serial_hds[1],
            STM32_USART1_NO_REMAP);
```

```
    stm32_uart_connect(
            (Stm32Uart *)uart3,
            serial_hds[2],
            STM32_USART3_NO_REMAP);
```

修改前

修改之後長這樣

```
/* Connect RS232 to UART */
    stm32_uart_connect(
            (Stm32Uart *)uart1,
            serial_hds[0],
            STM32_USART2_NO_REMAP);

    /* These additional UARTs have not been tested yet... */
    stm32_uart_connect(
            (Stm32Uart *)uart2,
            serial_hds[1],
            STM32_USART1_NO_REMAP);

    stm32_uart_connect(
            (Stm32Uart *)uart3,
            serial_hds[2],
            STM32_USART3_NO_REMAP);
```

修改後

其實我只是把 uart2 和 uart1 顛倒過來而已 – 疑? 好像改完之後才是正確的啊…之後你就可以打下面的指令，去把 ARM 版本的 QEMU 建造出來。

```
$ ./configure --disable-werror
$ make -j16
```

順利的話，QEMU fo ARM 就已經被 build 起來成功了，現在切換到 nuttx 的目錄內：

```
$ cd $HOME/work/nuttx
```

打入下面的指令，讓 QEMU 執行 nuttx kernel，選擇 hardware 為 stm32-p103，選擇 NuttX kernel (nuttx.bin)，把第一個 serial port (UART) 轉到 stdio，並且停在第一行指令，聽 port 1234 等待 GDB 來連接。

```
$ ../qemu_stm32/arm-softmmu/qemu-system-arm -machine stm32-p103 -
kernel ./nuttx.bin -nographic -s -S -serial stdio
```

執行完成之後，開啟另外一個 terminal 打入下面的命令啟動 gdb-multiarch，載入 nuttx (symbol)。

```
$ gdb-multiarch nuttx
```

在 gdb 下打入下面的指令。

```
(gdb) target remote localhost:1234
```

就可以連接到 QEMU 下的 NuttX，可以瀏覽 source code，到這裡，就完成了開發環境的設定，可以開心地研究 NuttX 了。

```
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from nuttx...
(gdb) target remote localhost:1234
Remote debugging using localhost:1234
__start () at chip/stm32_start.c:284
284       stm32_clockconfig();
(gdb) list
279                       "r"(CONFIG_IDLETHREAD_STACKSIZE - 64) :);
280     #endif
281
282       /* Configure the UART so that we can get debug output as soon as possible */
283
284       stm32_clockconfig();
285       stm32_fpuconfig();
286       stm32_lowsetup();
287       stm32_gpioinit();
288       showprogress('A');
(gdb)
```

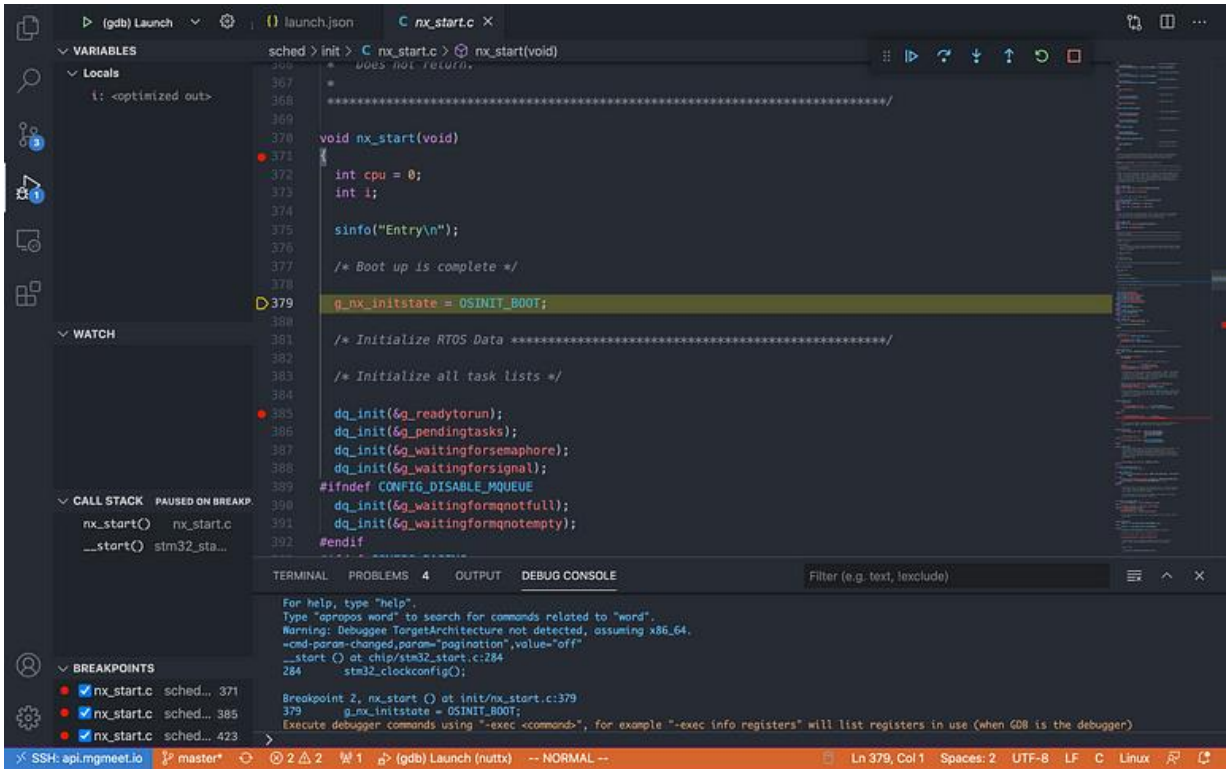
GDB + QEMU + NuttX

至於使用 VSCode，和我上一篇文章，使用 FreeRTOS 的時候完全一樣，請參閱 (使用 QEMU + FreeRTOS + Visual Source Code 開發嵌入式系統)。

打開 VSCode，打開目錄到 $HOME/work/nuttx 之下，選擇 VSCode 的 Run → Start Debuggin，如果是第一次執行，VSCode 會跳出 launch.json 請您修改，更改 launch.json 如下，大同小異，只是把 program 改成 $(workspaceFolder)/nuttx

```
// Use IntelliSense to learn about possible attributes.
// Hover to view descriptions of existing attributes.
// For more information, visit:
https://go.microsoft.com/fwlink/?linkid=830387
{
    "version": "0.2.0",
    "configurations": [
        {
            "name": "(gdb) Launch",
            "type": "cppdbg",
            "request": "launch",
            "program": "${workspaceFolder}/nuttx",
            "miDebuggerServerAddress": "localhost:1234",
            "args": [],
            "stopAtEntry": false,
            "cwd": "${workspaceFolder}",
            "environment": [],
            "externalConsole": false,
            "MIMode": "gdb",
            "setupCommands": [
                {
                    "description": "Enable pretty-printing for gdb",
                    "text": "-enable-pretty-printing",
                    "ignoreFailures": true
                    }
            ],
            "miDebuggerPath": "/usr/bin/gdb-multiarch",
            "miDebuggerServerAddress": "localhost:1234"
        }
    ]
}
```

之後就可以用 VSCode 在 QEMU 與 GDB 的環境下研究或是開發 NuttX 了。




NuttX + QEMU + VSCode Source Level Debug