# Proposal: HTTP daemon on Renesas M16 board

Cuong Phu Nguyen Kyung Chul Lee

URL: http://www4.ncsu.edu/~kclee/csc714

## 1. Introduction

The Renesas M16 offers a robust platform of 16-bit CISC featuring high ROM code efficiency, extensive EMI/EMS noise immunity, ultra-low power consumption, high-speed processing in actual applications, and numerous and varied integrated peripherals. Extensive device scalability from low- to high-end MCU series, featuring a single architecture as well as compatible pin assignments and peripheral functions, provides support for a vast range of application fields. The Renesas M16 is equipped with two 126kB RAMs, a 10/100 BaseT port, a RS232 port, a LCD, three LEDs, etc. It comes with the Renesas software development tools which include a complete software development tool chain including, HEW (IDE, GUI), NC30WA (C-compiler, assembler, and linker), KD30 (Debugger), and FoUSB (Flash-over-USB Programmer). A real-time, source-level debug environment is implemented using the KD30 debugging software with the RTA-FoUSB-MON Flash Programmer/In-Circuit Debugger (ICD). The Flash-over-USBTM(FoUSB) Programmer software, with the ICD, allows in-system programming. The ICD and firmware provide a convenient USB (Universal Serial Bus) interface between the board and the host PC. This interface reduces resource requirements on the processor.

Our project is aiming to implement a filesystem on Renesas to be used by a HTTP daemon. The existing network stack is implemented as a single dispatcher, which bases on the input packet types to invoke the appropriate protocol entities. HTTP daemon is on of such protocols running on top of TCP layer. For each new connection to HTTP daemon, data can be read or written. In order to operate, the HTTP daemon uses a set of filesystem APIs to store, retrieve and modify user data. As a goal of the project, the HTTP daemon needs to handle sensors data, collected by other group's projects. The sensor data will be sent to the HTTP daemon via 10/100 BaseT port or via 801.11b/g interface. As data need to go in and out fast enough, we need to implement a RAM filesystem and provide the HTTP daemon with a set of APIs. As the RAM capacity of the board is limited, the filesystem needs to be compact enough. Hence the lengths of file name and the number of directory levels should be limited.

## 2. Approach.

We will adopt a step-by-step approach to achieve our goal and break down our project into four different stages. However, as we approaching the problem, there may be more sub stages to be added.

**Stage 1 - Get familiar with the Renesas board and environment.**

There board comes with an integrated development environment for Windows. This IDE allow us to create and maintain the project, compile the code, and download the code from PC through the USB port. The debugging capability is also provided. There are open source software modules, such as BSP, drivers, and sample applications. These open source programs need to be explored in order to get familiar with the board hardware. Ideas about specific implementation may come from these source codes too.

**Stage 2 - Design a RAM file system and a set of API.**

We may take a look at existing filesystem like Extended File System, FAT, etc. to design our own RAM file system. Based on the requirement and the limitation of the RAM space, we may not implement the fancy features like long file names, extended number of file entries, too many directory levels, etc. The file system may have the following functional blocks:

*File/Directory entry blocks.*
This block will be searched first in order to seek for a file. The total number of entries will be fixed. Each entry contains the file name, file size and a pointer to file location.

*File data block.*
This block contains the actual data for a file. The advantage of the RAM file system is that it is more flexible than other sect based file system. On the other hand, we need to take into account memory allocation/deallocation in our code.

The set of APIs to maintain and manipulate the filesystem, as well as to provide services for the HTTP daemon. Since the task structure is simple, the API may not need to deal with the re-entrence issue: only one function is called at a time. One of the top priority requirement to the APIs is the execution time need to be short. There won't be any blocking waits.

**Stage 3 - Improvement for the HTTP daemon**

As we are implementing the project, we need to review the code of the HTTP daemon again and again. If you see any inefficiency, we need to improve the code. We may also need to change the HTTP daemon a little bit to so that it can work seamlessly with the file system. To some extent, we can event improve the protocol stack if needed.

**Stage 4 - Test module.**

In order to test our project, we need to write some test modules. The basic idea is to post and get data and verify if the data are correct. There are two types of test programs. One can communicate with the HTTP daemon over ethernet, 801.11b/g interfaces or the test. Other can run on the Renesas board and communicate with the HTTP daemon internally.

## 3. References


- The File system, 1996-1999 David A Rusling
  *http://www.tldp.org/LDP/tlk/fs/filesystem.html*

- HEW (Highperformance Embedded Workshop)
  *http://www2.eu.renesas.com/products/mpumcu/tool/hew/documents.html*

- HTTP - Hypertext Transfer Protocol *http://www.w3.org/Protocols/*

- M16C Family, Renesas
  *http://america.renesas.com/fmwk.jsp?cnt=m16c_family_landing.jsp&fp=/products/mpumcu/m16c_family/*

- SFS: Simple Filesystem *http://www.eros-os.org/devel/ObRef/standard/SFS.html*

- Transaction-Safe FAT File System
  *http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wcemain4/html/cmcontransaction-safefatfilesystem.asp*