CSC 714 : Project Proposal

Gayathri TK
gtambar@ncsu.edu

Jayush Luniya
jrluniya@ncsu.edu

RTFS : Real Time File System

Project Url

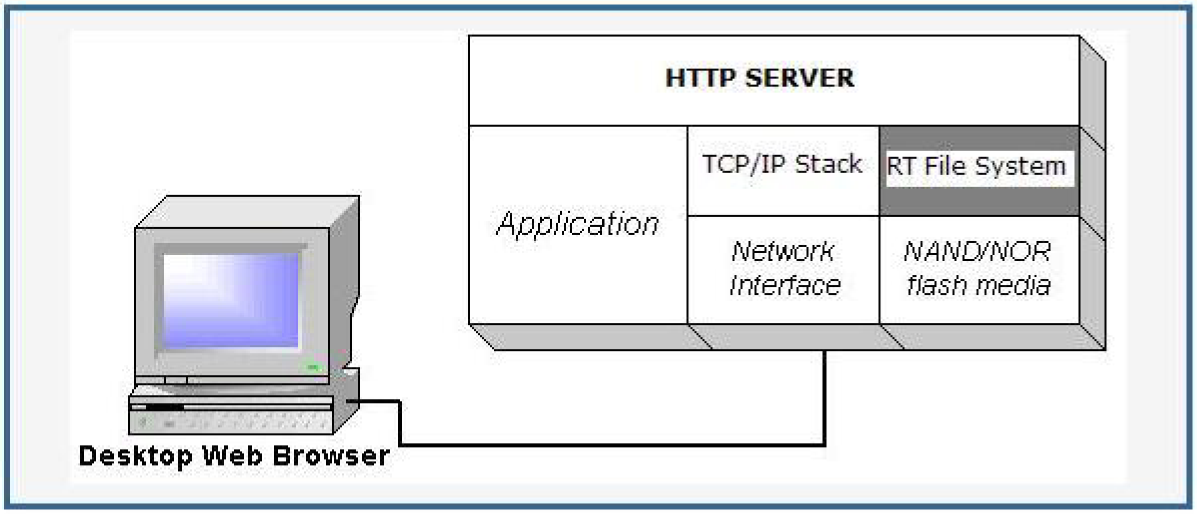
http://www4.ncsu.edu/~jrluniya/rt/

Abstract

Incorporating a web server to an embedded device provides a powerful mechanism allowing users to monitor and control embedded applications using any standard browser. Web enabling devices provides a new method of interfacing to devices that requires essentially no target side programming and works with universally available, standard client software. The web server uses a file system to store embedded web pages in RAM, flash, or on disk. File systems provide capabilities for changing web pages dynamically and maintaining dynamic objects. Pages can be protected with password security to restrict both read and write access. Hence the need for a real-time file system on an embedded device.

The goal of this project is to implement a Real Time File System (RTFS) for Renesas M16C board, which acts as a base station for a cluster of sensor nodes and aggregates data to exchange across clusters. By web-enabling the device, the data can also be available to other base stations and high-end machines in a heterogeneous environment.

Project Description



Need for Web Enabling a Device

1. An embedded web server facilitates the user to interface with the embedded device using any standard browser on his computer running any operating system like Linux, Windows, Solaris eliminating the need to implement new client software.
2. Web servers can be implemented efficiently and in a small footprint. It is the web client, the browser that is the large and sophisticated component of the client/server pair having to parse the HTML and render the graphic output. Fortunately, there is no need to incorporate the browser code when web enabling a device.
3. In addition, a web enabled device can be accessed anywhere in the world if it is connected to the Internet or can be locally connected on a LAN with equal ease.

Design Issues for RTFS

1. RTFS should be a primitive file system with low metadata overhead due to resource constraints on the embedded device. Hence the file system should be similar to FAT[1] or TFAT[2,3] file systems.
2. Typical embedded devices do not have hard disks and hence the filesystem should be a memory based file system.
3. RTFS should support dynamic creation and deletion of files, directories, and links with full read and write capability. Not a static ROM-image file system.
4. A POSIX compliant API would provide well known and easy to use API for faster integration time for application developers. Hence the RTFS API should be POSIX compliant.
5. Traditional file systems, such as FAT, provide satisfactory performance in the transfer of data, but power interruption or system failure can cause corruption of the file system, possibly rendering the device inoperative. This level of reliability, inherent to the FAT file system are not acceptable for many embedded systems, has led to the development of more reliable systems. Hence RTFS should ensure that the FAT table and directory structure are transaction safe across power failures.
6. RTFS should be able to provide real time bounds for file access operations.
7. RTFS should have a thin hardware dependant layer to maximize ease of porting to new architecture. Also RTFS should be minimally dependent on the underlying RTOS.

References

[1] FAT Filesystem http://users.iafrica.com/c/cq/cquirke/fat.htm

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

[3] Reliable File Systems for Windows CE http://ssrc.cse.ucsc.edu/Papers/edel-mascots04.pdf

[4] TargetFFS: An Embedded Flash File System http://www.blunkmicro.com/ffs.htm

[5] Renesas M16C Architecture http://www.renesasinteractive.com