CSC 714 : Project Report 3

Gayathri TK
gtambar@ncsu.edu

Jayush Luniya
jrluniya@ncsu.edu

RTFS: Real Time File System

Project URL

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

Abstract

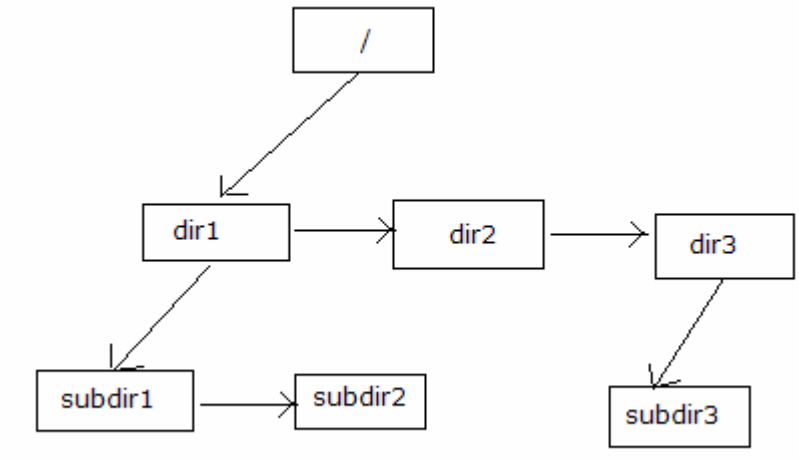
Incorporating a web server to an embedded device provides a powerful mechanism allowing users to monitor and control embedded applications using any standard browser. Web enabling devices provides a new method of interfacing to devices that requires essentially no target side programming and works with 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. RTFS is a RAM-based file system which provides real time guarantees on file access times. The Renesas M16C board typically 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.

Implementation Details

Data Structures Used

1. Directory Control Block (DCB): The directory control block contains metadata about a directory. This includes information about the files in the directory. The number of files in a directory is bounded by MAX_FILE_ENTRIES parameter. The metadata for the subdirectories is maintained in separate DCBs. The directory hierarchy is maintained as a tree as shown below:



```
typedef struct dirEntry_t
{
        char dirName[MAX_DIR_NAME_SIZE];   // Directory Name
        FileEntry files[MAX_FILE_ENTRIES]; // File Control Block Entries
        struct dirEntry_t *leftChild;      // Leftmost Child
        struct dirEntry_t *rightSibling;   // Next subdirectory at the same level.
}DirEntry, *DIRENTRY;
```

2. File Control Block (FCB): The file control block contains the metadata about a file. In order to reduce filesystem overhead, the FCB's are maintained inside the DCBs itself. We maintain an upper bound on the file size (NUM_BLKS_PER_FILE * DATA_BLK_SIZE)

```
typedef struct fileEntry_t
{
        unsigned int fileSize;                  // File Size
        char fileName[MAX_FILE_NAME_SIZE];      // File Name
        int dataBlocks[NUM_BLKS_PER_FILE];      // File data Blocks
        BOOL isValid;                           // File Entry is valid
}FileEntry, *FILEENTRY;
```

3. File Descriptor: File descriptor contains information about the open files in the system. Typically it contains information about the file it refers to and current read position. Since our file system supports writes only in append mode, there is no need for an explicit write pointer.

```
typedef struct fileDesc_t
{
        DIRENTRY dir;                   // Directory in which the file exists
        int fileIndex;                  // Index to the appropriate file entry in the DCB
        unsigned int readPos;           // Read Pointer
        BOOL free;                      // File Descriptor is free
}FileDesc, *FILEDESC;
```

4. Filesystem Superblock: The filesystem superblock contains information about the filesystem itself like the pointer to the DCB of root, array of file descriptors and information about the free data blocks in the system.

```
typedef struct superBlock_t
{
        FileDesc fileDescriptor[MAX_FILE_DESC];         // File Desc Array
        int fdIndex;                                    // Start index to search for free file desc
        char data[MAX_DATA_BLKS][DATA_BLK_SIZE];        // Data Blocks
        char dataBitmap[DATA_BITMAP_SIZE];              // Free/Allocated Data Block Bitmap
        DIRENTRY root;                                  // Pointer to Root DCB
}SuperBlock;
```

Filesystem APIs

RTFS filesystem supports the following APIs:

Directory Functions:

- int f_mkdir(char *pathName)

  Description: Create a new directory. Pathname should be absolute.

  Return Value: Returns 0 on success, -1 on error.

- int f_rmdir(char *pathName)

  Description: Remove an empty directory entry. Pathname should be absolute.

  Return Value: Returns 0 on success, -1 on error.

File Control Functions:

- int f_creat (char *pathName)

  Description: Create a new file entry. Pathname should be absolute. f_create also allocates a new file descriptor.

  Return Value: Returns file descriptor number on success, -1 on error.

- int f_remove (char *pathName)

  Description: Remove the file entry. Pathname should be absolute.

  Return Value: Returns 0 on success, -1 on error.

File Access Functions:

- int f_open (char *pathName)

  Description: Open a file. Pathname should be absolute. We do not maintain the filesystem. We don't have any concept of file modes (read-only, write-only, read/write etc).

  Return Value: Returns file descriptor number on success, -1 on error.

- int f_close(int fd)

  Description: Close an open file.

  Return Value: Returns 0 on success, -1 on error.

- int f_read(int fd, void *buf, int count)

  Description: Read up to *count* bytes from file descriptor *fd* into the buffer starting at *buf*

  Return Value: On success, the number of bytes read is returned (zero on EOF), -1 on error.

- int f_write(int fd, void *buf, int count)

  Description: attempts to write *count* of data to the object referenced by the descriptor f*d* from the buffer pointed to by *buf*.

  Return Value: On success, the number of bytes written is returned, -1 on error.

Mini shell

To test the filesystem APIs we implemented a mini shell that supports the following commands:

- mkdir — Command to create a new directory
- rmdir — Command to remove an empty directory
- touch — Create a new file with file size = 0
- remove — Remove a file
- cat — Reads a file sequentially and writes to standard output.
- append — Appends to a file
- cd — Change Present Working Directory (PWD)
- ls — View PWD directory contents
- exit — Exit the shell

Open Issues

- **Porting Issues** – The present filesystem APIs have been tested using a mini-shell as a standalone application. We are currently working on issues and changes required for porting the filesystem to the Renesas board and testing it using the mini-shell on that board.
- **Real Time Guarantees –** Although the filesystem has been designed keeping real time issues in mind, we have to work on specifying the worst case bounds on file operations.

Individual Contributions

Both the team members made equal contribution towards this project milestone.

Task	**Contributor**
Data Structure Design	Gayathri, Jayush
File System APIs : mkdir, rmdir, read, write	Gayathri
File System APIs : creat, remove, open, close	Jayush
Shell Commands: mkdir, append, cd, ls	Gayathri
Shell Commands: rmdir, touch, remove, cat	Jayush

Weekly Milestones

Milestones	Date	Milestone
Milestone 1	Oct 1- Oct 15	Decided on project topic; Came up with project proposal.
Milestone 2	Oct 15- Nov 1	Worked on the Design issues of the project
Milestone 3	Nov 1 - Nov 15	Came up with file system Datastructures; Implemented basic file system APIs; Implemented a mini shell to test the file system APIs; Report
Milestone 4	Nov 15 – Nov 22	Have to work on porting the filesystem to Renesas M16C Board; Provide real time guarantees on file access times.
Milestone 5	Nov 22 – Nov 30	Test and benchmark the filesystem

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] Flash Filesystems for Embedded Linux Systems http://www.linuxjournal.com/node/4678/print
[4] TargetFFS: An Embedded Flash File System http://www.blunkmicro.com/ffs.htm
[5] Renesas M16C Architecture http://www.renesasinteractive.com
[6] Algorithms and Data Structures for Flash Memories http://www.cs.tau.ac.il/~stoledo/Pubs/flash-survey.pdf
[7] A Transactional Flash File System for Microcontrollers http://www.cs.tau.ac.il/~stoledo/Pubs/usenix2005.pdf
[8] ELF: An Efficient Log-Structured Flash File System For Micro Sensor Nodes http://www.cs.colorado.edu/~rhan/Papers/sensys_elf_external.pdf