The Trivial File System

Posted on **May 1, 2012**

This posting is the third in a series that began with a look at Non-volatile Options and then moved on to an examination of the M25P80 serial memory device. With this posting we look at the Trivial File System or TFS created to exploit the M25P80 as a storage medium in an embedded development system (The mikromedia for PIC32).

The TFS is a very rudimentary file system for storing parameters, settings and infrequently changed data for an embedded system. The TFS is not designed to compete or be compatible with any “real” big computer file systems. It is not designed to hold “user” data that may need to be updated frequently. Rather, the TFS is best thought of as a combination of EEPROM for parameters and an extension to flash program memory for data.

In examining the TFS, three questions would seem to be of the greatest relevance:

1. What does the TFS do?
2. How is the TFS used?
3. How does the TFS work?

The TFS lives up to its name by doing as little as possible and providing only basic services with ten functions. These functions tell the story of what the TFS is all about:

```
emRESULT emTFSOpenFS(void)
void emTFSCloseFS(void)
```

These simple functions open and close the file system. The open function can return some interesting values:

emOK – the file system is opened and available.
emWRN_WRITE_PROTECT – a warning that the device is write protected.
emERR_NO_RESPONSE – can’t initialize the device.

emERR_UNFORMATTED – unformatted media.
emERR_UNKNOWN_VERSION – the file system version is unknown.

```
emRESULT emTFSFormatMedia(U32 dirSize)
```

The format function erases *all* data on the M25P80 and prepares the media to receive data. There are two reasons to call the format function. 1) The open function has returned emERR_UNFORMATTED or emERR_UNKNOWN_VERSION or 2) there is a need to reclaim file space in the media. The format function is the only function that calls the erase routines of the flash memory, and because of this, it can be extremely slow, up to 20 seconds!

Note that the format function takes one argument, the directory size in bytes. Memory space for the directory must be pre-allocated and cannot be expanded. Each file directory entry consumes 4 bytes plus one byte for each character in the name. Every time a file is replaced with a new version, a new directory entry is required. It is important that the directory area be large enough for all the files it needs to hold and yet not so large that it eats into the space needed to store the file data.

```
emRESULT emTFSOpenFile(PC8 name)
```

This function is used to open an existing file with the given name. The file name is a null terminated string of one to 15 characters and is case sensitive. Except for null, there are no reserved characters. This function returns:

A handle value if successful.
emERR_FILE_NOT_FOUND – the file could not be found.
emERR_OUT_OF_HANDLES – the supply of handles is exhausted.

```
emRESULT emTFSCreateFile(PC8 name, U32 size)
```

This function is used to create new files and replace existing ones. The name is a null terminated string, the size is the amount of file space reserved for the file. The total space used must be known when the file is created. If a file of the same name exists, it is marked as pending erase. The new file is created as write in progress. Possible return values include:

A handle value if successful.
emERR_OUT_OF_HANDLES – the supply of handles is exhausted.
emERR_INVALID_FILE_NAME – the name is not valid
emERR_OUT_OF_HANDLES – the supply of handles is exhausted.
emERR_OUT_OF_SPACE – there is not enough space for the file.
emERR_WRITE_PROTECT – the media is write protected.

```
emRESULT emTFSCloseFile(emTFSHANDLE hnd)
```

This function closes a file, freeing up the handle used to access it. If the file was newly created, its status is changed from write in progress to valid. If an older file was being replaced, its status is changed from pending erase to erased. Possible return values include:

emOK – no errors, all OK
emERR_INVALID_HANDLE – the handle was not of an opened file.

```
emRESULT emTFSDeleteFile(PC8 name)
```

This simple function deletes the file of the given name. Note however that this function does NOT free up space on the media. Only a format command can do that. The delete command merely makes the file inaccessible. Possible return values are:

emOK – all OK!
emERR_WRITE_PROTECT – the media is write protected.
emERR_FILE_NOT_FOUND – no file of that name found to delete.

```
emRESULT emTFSReadFile(emTFSHANDLE hnd, U32 adr, U16 len, PU8 buf)
```

This function is used to read data from a file where hnd is the handle of the file, adr is the offset in bytes from the start of the file (0 is the start of the file), len is the number of bytes to be read, and buf is a pointer to a buffer to hold the data. Possible return values include:

emOK – all OK
emERR_INVALID_HANDLE – the handle was not of an opened file.
emERR_INVALID_POSN – the read was not within the scope of the file.

```
emRESULT emTFSWriteFile(emTFSHANDLE hnd, U32 adr, U16 len, PU8 buf)
```

This function is used to write data to a file, sort of. The actual operation performed is to bitwise AND the new data with the data already on the media. Fresh media has the value 0xFF so this emulates a write operation the first time it is performed. Subsequent writes to the same section of media only AND the old and new values. To get a clean slate, a new file must be created (see above).

The parameters are hnd, the handle of the file, adr is the offset in bytes from the start of the file (0 is the start of the file), len is the number of bytes to be written, and buf is a pointer to a buffer to that holds the data to be written. Possible return values are:

emOK – all OK
emERR_INVALID_HANDLE – the handle was not of an opened file.
emERR_WRITE_PROTECT – the media is write protected.
emERR_INVALID_POSN – the write was not within the scope of the file.

```
U32 emTFSFileSize(emTFSHANDLE hnd)
```

The function returns the size of the file attached to the handle in bytes or zero if there is an error.

There. A whole file system in only ten modest functions. In future postings, the usage and internals of the TFS shall be examined. Until then, please take the time to comment or make suggestions.

Peter Camilleri (aka Squidly Jones)

This entry was posted in **Embedded Development** by **Peter Camilleri**. Bookmark the **permalink [http://teuthida-technologies.com/?p=947]** .