Stage 2: HTTP daemon on Renesas M16 board

Cuong Phu Nguyen cpnguyen
Kyung Chul Lee kclee

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

1. Summary

This report covers the time period November 2, 2005 – November 15, 2005 and encompasses Stage 2 - "Design a RAM file system and a set of API" of the project plan. It aims both to describe progress made and discuss the issues encountered.

The achievements of our group so far include:

- Designed and specifying rules and algorithms.
- Implemented a simulated file system on Linux system and currently porting onto M16.

2. Plan

We had the original plan as following for the project:

- Stage 1 (October 18 – 29) - Both Cuong and Kyung
  Get familiar with the Renesas board and environment – Examine and run provided same codes and learn the development environment.

- Stage 2 (October 30 – November 14) - Both Cuong and Kyung
  Design a RAM file system and a set of API – Come up with a feasible file system for a M16 board and create specifications.

- Stage 3 (November 15 – November 22) - Both Cuong and Kyung
  Improvement for the HTTP daemon - Find inefficiency and possible bottlenecks, and improve over those findings.

- Stage 4 (April 23 – April 28) - Both Cuong and Kyung
  Test module - Create possible scenarios and run multiple tests based on the scenarios.

3. Accomplishments & Issues

We are currently working on Stage 2.

Stage 2

We had developed APIs as following:

```
/*
CREATE A FILE

- path: the absolute path name of a file to be created
- size: the final size of a file.  The size is fixed and can not be chaged.

fs_create() returns 0 for success or 1 otherwise
*/

int fs_create(const char* path, int size);



/*
OPEN A FILE

- path: the asolute path name of a file.
- mode: one of the following FS_READ, FS_WRITE, or FS_READ_WRITE

open() return the file descriptor or -1 if an error occured.
*/

int open(const char* path, int mode);


/*
CLOSE A FILE

- fd: file descriptor

close() returns zero on success, or -1 if an error occurred.
*/

int close(int fd);



/*
READ FROM A FILE DESCRIPTOR

- fd: file descriptor
- buf: data to be stored into
- count: number of bytes to read

read() attempts to read up to count bytes from file descriptor fd into the buffer
starting at buf.
If count is zero, read() returns zero and has no other results.
*/

int read(int fd, void *buf, int count);


/*
```

```
WRITE TO A FILE DESCRIPTOR

- fd: file descriptor
- buf: data to be stored from
- count: number of bytes to write

write() writes up to count bytes to the file referenced by the file descriptor fd
from the buffer starting at buf.
*/

int write(int fd, void *buf, int count);


/*
REPOSITION READ/WRITE FILE OFFSET

- fd: file descriptor
- buf: data to be stored from
- count: one of the following
       FS_SEEK_SET: The offset is set to offset bytes.
       FS_SEEK_CUR: The offset is set to its current location plus offset bytes.
       FS_SEEK_END: The offset is set to the size of the file plus offset bytes.

seek() repositions the offset of the file descriptor to the argument offset
according to the directive whence.
*/

int seek(int fd, int offset, int whence);


/*
MAKE A NEW DIRECTORY

- path: the absolute path name of a directory to be created
- max_size: directory size

mkdir() return 1 for sucess or 0 otherwise
*/

int mkdir(const char *path, int size);


/*
READ DIRECTORY LIST

- path: the absolute path name of a directory to be read

readdir() list of strings containing filenames
*/
char** readdir(const char *path);


/*
FORMAT FILE SYSTEM

fs_format() return 0 for sucess or 1 otherwise (general error)
*/


int fs_format();
```

The file system specification as following:

Directory Entries

1 byte	The maximum number of files
1 byte	The number of existing files
Multiple of 20 bytes	File Entries 11 bytes: Filename (maximum 10 characters) 1 bytes: File type - 0 for file - 1 for directory 4 byte: File size 4 byte: File Location

File Descriptor

1 byte	Type - 0 for file not in use - 1 for file currently being read - 2 for file currently being written - 3 for file currently being read and written
4 byte	Operation cursor byte offset from the begging of the file
4 byte	File Location (page index)
4 byte	File Size (# pages)

Page Counter

4 byte	Available Free Space

We are currently porting the simulated system to M16 board. As we expected, we could not many of existing GCC functions and we have to develop our own functions. Another issue is that we could not use any dynamic and pointer arithmetic on M16. Therefore, we have to convert code not to use any pointers and strictly use only use arrays. We came up with that we could store a value to a

predefined place before we call a function in order not to use a pointer for passing a large data.

4. Conclusion and Next Step

Our next step is to complete the file system on M16 and test it out. Once we are able to run the HTTP daemon with our file system, we are going to find inefficiency in the code and try to improve possible bottlenecks.