CISC 3595 Operating Systems

Course Project

Overview

You work in the new product development group of a large company called *Acme Computer Systems*. You and your team have been assigned the task of constructing the file system module of the new (Linux based) operating system ATOS (Acme Table Operating System) for your company's soon to be released latest model tablet computer. Your team is responsible for all parts of this project including

- Development and documentation of the file system specifications, emphasizing what is important for a tablet based file system
- Development and documentation of the design of the file system, levering what you knew about file systems and tablet requirements to pick the best Linux inspired design approach
- Development and documentation of all the software
- Development and documentation of the test cases and test results used to validate your software.

What software your team needs to build

You need to write and test file system simulation software ATOS-FS that does the following:

1. Implements a file storage mechanism that respects the constraints of the tablet application. All your files are to be stored in memory only using a DISK API (described below) as your storage mechanism. You cannot just 'pass through' your directory and file systems commands to the underlying file system on the computer you are using for development.
2. Implement an API for the following file system operations: *Create, Delete, Open, Close, Read, Write, Stats*.
3. Implement *at least* sequential access files.
4. Allow files to be opened in *read mode* or in *write mode* and Read and Write operations should behave accordingly for that opened file.
5. Implement *at least* a one-level, flat directory structure and an API for the directory command *List* to list the directory.
6. Provide a front-end user interface ATOS-UI that implements the following commands:
    a. CREATE filename -- creates the file and enters it into the directory with zero size
    b. DELETE filename – deletes the file and reclaims its space
    c. DIR – lists all the filenames, their sizes and the amount of free space left.
    d. EDIT filename – whatever you type after this goes into the file until you press the EOF character (^D).
    e. TYPE filename – prints the contents of the file to the screen.
    f. EXIT – ends your program

Schedule

3/7,10	*Class project starts. Divide into teams.*
3/13-19	**No class, Spring break**
3/21,24	
3/28,31	*Project specification document due (1 page)*
4/4,7	
4/11	*Project design document due (2 pages)*
4/13-17	*No Class, Easter Recess*
4/18,21	
4/25,28	*Project testing document due (2 pages)*
5/2	
5/4	*Last day of classes*
	Project Due

Grading

- Everybody on the team needs to have visibly contributed: each document will have a lead author, and each API routine will have an author
- You will deliver documents by email to me per team
- Each team will have all its source code in one folder called *teamname* with a makefile that will compile and link your code with the name: CISC3595_teamname_atos-fs
- Project counts for 10% of your grade
    - Documents count for 4%
    - Working code counts for 6%

ATOS-FS API Specifications – Overview

`bool Create(string filename);` // creates 0 sized file called filename, returns true if successful else false.

`bool Delete(string filename);` // deletes file called filename, returns space back to the free space pool.

`int Open(string filename, string mode);` // opens file called filename in the mode request and returns an integer file handle that can be used for read or write depending on mode

`bool Close(int handle);` // closes the file with this handle, returns true if successful or false if not.

`int Read(int handle, int numchars, char *buffer);` // reads numchars from file with this handle and returns these in buffer. Returns the number of characters actually read (might be less than numchars), or -1 if there is an error (e.g., no such handle).

`int Write(in handle, int numchars, char *buffer);` // writes numchars chars that are in buffer to the file with this handle. Returns the number actually written or -1 if there is an error.

`int Stats(filename);` // returns the current size of filename

`vector<string> List();` // returns a list of all currently existing file names as stings

The DISK API

bool CreateDisk(int blocksize, int numblocks); // creates disk with this size

bool WriteDisk(int offset, char blockbuffer[blocksize]);// writes a block to the disk at location offset

bool ReadDisk(int offset, char blockbuffer[blocksize]);// reads the disk block at location offset into the block buffer

struct { vector<char> data; } blockType;

struct { vector<blockType> disk; int blocksize; int numblocks; } diskType;

ATOS-UI Specifications – Overview

```
$ ./CISC3595_myteamname_atos-fs
$$ CREATE my1stfile.new
$$
$$ EDIT my1stfile.new
Xyz abc defg h456 8
1 2 3 4 that's all
^D
$$ DIR
        ATOS-FS Directory Listing
        FILENAME                SIZE (blks)
        my1stfile.new           1

        FREE SPACE 999 blks
$$ TYPE my1stfile.new
Xyz abc defg h456 8
1 2 3 4 that's all
$$ DELETE my1stfile.new
$$ DIR
        ATOS-FS Directory Listing
        FILENAME                SIZE (blks)

        FREE SPACE 1000 blks
$$ EXIT
$
```

Notes:

1. The prompt is a double dollar sign followed by a space.
2. In DIR there is one tab at the start of each row and 3 tabs between names and file sizes.