Sample for Assignment 4

CS370 Assignment 4

Sample for Assignment 5

Maximum points 200

Implement a Simple Stateless Network File Server (SSNFS) that supports *remote file service model* (just to make things easy). Your file server will use a UNIX file as a virtual disk to store the files created by the clients. Your server and client should be implemented as Sun RPC server and client. For simplicity, you can think of the virtual disk as a sequence of blocks, each block containing 512 bytes. You can also assume that the virtual disk capacity is 10MB. Each client should be assigned by the server a home directory. Your client program should be written in such a way that it facilitates the testing of the correctness of your server implementation. You will be provided with an interface definition file. You **must** use that interface definition file so that the file servers implemented by each of you will export the same interface and hence will work correctly with the clients implemented by the others. **You should not modify the interface definition file. If you modify, you will not get any credit**. If you think you have to modify for the program to work correctly, talk to me first. The server exports the following operations for the client:

Create: creates a file with the give name in the user's directory. Each user is assigned his/her own directory with his/her login name as the name of the directory. The login name of a user can obtained from the password file (using *getpwuid(getuid())->pw_name*). Initially, a newly created file is allocated 8 blocks. Your implementation should allow the files to grow dynamically. i.e., if more than 8 blocks are needed for an already created file, additional blocks in multiples of 8 blocks should be allocated. You can assume a maximum file size of 64 blocks.

Read: reads a specified number of bytes from the specified position of a specified file and returns it to the client. Use variable length buffer. **Returns appropriate error message if trying to read past the end of file**.

Write: writes the specified number of bytes from a buffer to the specified file from a specified location. Use variable length buffer. **Returns appropriate error message if trying to write in a location past the end of file**.

List: lists the names of all files in the user's directory.

Copy: takes the names of two files as parameter and copies the contents of the first file to the second file.

Delete: deletes the specified file.

Append: appends the specified number of bytes from a buffer to the specified file (i.e., writes at the end of the specified file).

Grading:

- Program compiles and produces some meaningful output (50 points)
- **Create** works correctly (space is allocated as specified). (20 points)
- **Read** works correctly. (15 points)
- **Write** works correctly. Make sure you can write to random location in a file even if the initial segment of the file is empty. (15 points)
- **List** works correctly. (10 points)
- **Copy** works correctly. (10 points)
- **Append** works correctly. (10 points)
- **Delete** works correctly. (10 points)
- **Append** works correctly. (10 points)
- Your server works correctly with the clients of others. (15 points)
- When the server crashes and is restarted, files and directories on the disk(virtual) are not lost. When the server restarts, it uses the already created fielsystem and does not initialize the disk. (25 points)
- Documentation, meaningful error messages and appropriate user interface. (10 points)

Additional features you may try to implement for fun if you have time:

- Provide support for creation hierarchical UNIX like directories and files structure
- provide support for the creation of symbolic links.
- Provide support for security and authentication

You need to submit a tar file containing all the source code files, makefile and a README file.

A sample run of the client and the corresponding output (shown in **bold**).

>`Create file1`
>**file1 created for user manivann**
>`Write file1 0 10 abababababab`
>**10 characters written to file1**
>`Create file2`
>**file2 created for user manivann**
>`List`
>**The files are: file1 file2**
>`Create file1`
>**Error: file already exists**
>`Copy file1 file2`
>**file copied successfully**
>`Copy file1 file3`
>**Error: file3 does not exist**
>`Read file2 2 5`
>**Content read: ababa**
>`Delete file1`
>**file1 deleted**
>`List`
>**The files are: file2**
>`Append file2 aaaaaggggggghhh 10`
>**10 charaters "aaaaaggggg" appended to file2**

Expected learning outcome: Develop your own implementation of a component of a distributed system. Useful for virtualization.