CS 570 - Assignment 4

Maximum points 200

Implement a Simple Stateful 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 user 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 are 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 clients:

open_file: Opens the file with given name in the user's directory and returns a file descriptor (a positive integer) to the client. The file descriptor is used in subsequent reads and writes. If there is no file with the given file name in the user's directory, it creates a file with that name and returns the file descriptor. If it cannot open a file for some reason (for example, no space on the disk to create a new file, or file table has grown large) it returns -1 as file descriptor. 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*). A newly created file is allocated 64 blocks. File size is fixed.

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

write_file: writes the specified number of bytes from the buffer to the file represented by the file descriptor from the current position. Uses variable length buffer. **Returns appropriate error message if write fails**.

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

delete_file: deletes the specified file.

close_file: closes the file with given file descriptor. After a file is closed, you should not be able to read from the file.

Note that the file server is stateful. i.e., it maintains a file table in memory which contains information about currently open files (i.e, user name current position of file pointer, etc). Files are read only sequentially from the beginning to the end and are also written sequentially. Random access is not supported.
Your client program should implement the following wrapper functions corresponding to the above-mentioned operations supported by the server: **Open, Read, Write, List, Delete, Close**. This will help me test your program.

Grading:

- Program compiles and produces some meaningful output (50 points)
- **Open** works correctly . (20 points)
- **Read** works correctly. (20 points)
- **Write** works correctly. (20 points)
- **List** works correctly. (10 points)
- **Delete** works correctly. (10 points)
- **Close** works correctly. (10 points)
- Your server works correctly with the clients of other students. (15 points)
- When the server is restarted after it crashes, files and directories on the disk (virtual) are not lost. When the server restarts, it uses the already created filesystem and does not reinitialize the disk. Of course, information about open files are lost. (25 points)
- Documentation, meaningful error messages. (20 points)

Extra Credit: (100 points) If you did not use the sample code given and implemented a solution on your own and also implemented the following additional feature.
• Provide support for creation hierarchical UNIX like directories and files structure
You need to submit a tar file or zip file (no rar file) containing all the source code files, makefile and a README file.

An example sample code that I will use in the main() of your client program for testing is provided below.

```
int i,j,k;
int fd1,fd2,fd3;
char buffer[100];
fd1=Open("File1"); // opens the file
for (i=0; i< 20;i++){
  Write(fd1,  "funny contents in the file 1", 15);
}
Close(fd1);
fd2=Open("File1");
for (j=0; j< 20;j++){
    Read(fd2, buffer, 10);
    printf("%s\n",buffer);
}
Close(fd2);
Delete("File1");
List();
```

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