README:

My codes are based on the template codes generated by

```
rpcgen -a -C ssnfs.x
```

and Sun RPC tutorial which is available online: `http://www.cs.rutgers.edu/~pxk/rutgers/notes/rpc`.

Before testing on your own computer, run

```
make
```

on the command line.

Brief Description: This program implements a Simple Stateless Network File Server that supports remote file service model. The file server use a UNIX file called "Virtual_Disk" as a virtual disk to store files created by the clients.

The virtual disk a s sequence of blocks, each block containing 512 bytes. The capacity of the disk is 10 MB which is 20480 blocks. These blocks are assigned to files 8 by 8. We call 8 blocks as a bunch. The disk contains 2560 bunches and 8 of them are reserved for storing SSNFS information.

SSNFS system information consists of:

- Number of users.
- An array indicates whether the 2560 bunches are free.
- The available number of bunches.
- A list of users which contain their names and names and locations of their files.

The SSNFS consists of six operations:

- **Create:** creates a file with the give name in the user's directory.

  When the server receives a `create` request from a client, it will first check whether the virtual disk is created. If not, the server will create and initialize it. Then, it will check whether the file exists. If not, it will create it for the user. When the file is created, a bunch(8 blocks) is allocated for this file. When the virtual disk is full, it returns appropriate message.

- **Read:** reads a specified number of bytes from the specified position of a specified file and returns it to the client.

  When the server receive a `read` request from a client, it will check whether the virtual disk is created and whether the file exists and return appropriate message. It will return 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.

  When the server receive a `write` request from a client, it will check whether the virtual disk is created and whether the file exists and return appropriate message. It will return appropriate error message if trying to write in a location past the end of file. Dynamically allocating memory is employed(Available bunches will be assigned if needed). Error message will be sent if the virtual is out of memory.

  When `write` is invoken on client, the program requires the user to type in at most `numbytes` bytes and can be terminated using `Ctrl + D` in UNIX environment. If more than `numbytes` bytes are typed in, the program will only take the first `numbytes` bytes.

- **Lists:** lists the names of all files in the users directory.

  When the server receive a `list` request from a client, it will check whether the virtual disk is created and return appropriate message.

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

  When the server receive a `copy` request from a client, it will check whether the virtual disk is created and whether the files exist and return appropriate message. The `copy` operation will succeed only if when the two files are already created and there is enough memory on the virtual disk.

- **Delete:** deletes the specified file.

  When the server receive a `delete` request from a client, it will check whether the virtual disk is created and whether the file exists and return appropriate message. Memory is released after the file is deleted.

USER INTERFACE:

```
******************SSNFS environment*****************
CREATE A FILE:    create [file_name]
LIST ALL FILES:   list
DELETE A FILE:    delete [file_name]
WRITE TO A FILE:  write [file_name] [offset] [numbytes]
READ A FILE:      read [file_name] [offset] [numbytes]
COPY A FILE:      copy [from_filename] [to_filename]
EXIT:             exit
```

Examples:

Here is an example run on my computer. To check the correctness of dynamically allocation of memory, we can change the BLOCKSIZE to be a smaller number, e.g. 16. Make sure you also enlarger `RES`(number of bunches reserved for system information) to fit `diskinfo`(19208 bytes).

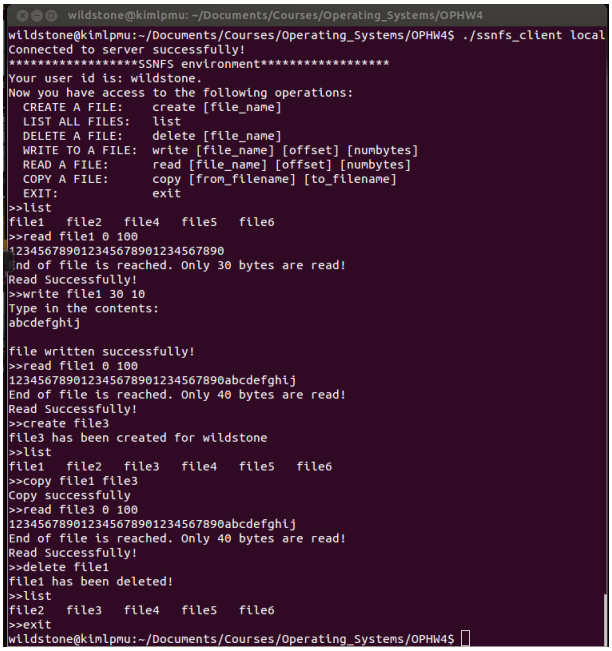

Figure 1: Client Interface

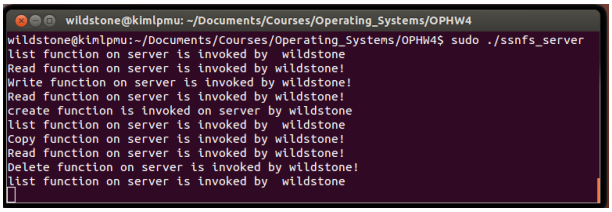

Figure 2: Server