Includes Extension A and C.

run in the form.

./client “design.cs.rutgers.edu” “mode”

Please enter a valid pathname

libnetfiles.c
void seterrno() : reads error message from server, decodes it and sets errno.

int initalizeSocket() : Helper Function which Initialzes the socket connection to the hostname

int netserverinit(char* hostName, int mode); Checks if hostname is correct or not.

int netopen(Char* pathname, flags): Calls initalizeSocket() to make a connection with server. Sends in the parameters in a char* array form. This char* buffer includes; 1 : pathname : flags : mode. 1 is used on the server side to decode which function is called (1 : netopen). Client sends this message to buffer and waits for respone from server. Response is checked for errors and is reported to Errno. Response is a file descriptor which is returned.

Way of Calling netopen is : int FileDescriptor = netopen(pathname, O_RDWR);

netread(), write(), close() : All these functions send a char* buffer message with their parameters which is decoded on server side. If an error is rensponded, errno is set and program is terminated. They all return int.

RUN IN THIS FORM :

```
char* pathname = "/.autofs/ilab/ilab_users/sss269/A3/bstesting.txt\0";

char* b = malloc(10000);

int x = netopen(pathname, O_RDWR);

read_status = netread(x,b,10000);

write_status = netwrite(x, "some string her", 30);

close_status = netclose(x);
```

netfileserver.c

```
typedef struct fileInfo{
        int flag;
        int mode;
        int fileDesc
char* filename;} fileinfo;
```

```
#define unrestricted 0
#define exclusive    1
#define transaction  2
```

```
#define INVALID_FILE_MODE 9998
```

Main : Create one thread which will initialize the socket connection and keep the socket connection running forever.

InitalizeSocket() : Keep connecting with clients, whenever a client is connected, also create a thread for that client. This thread will decode the messages sent by the client and execute them.

DecodeRequest() : Decode the message read from the client, and also executes the request.
Returns the output of the request back to the client. If the error is returned by executing the request, an error message is formed and given back to the client. If the error message is INVALID_FILE_MODE which is caused to due policy issues, the request is placed on a queue. (EXTENSION C) and a confirmation is sent to the client, informing about the queue. All the error handling is done in this function.

Int simulatenetopen, simulatenetread, simulatenetwrite, simulatenetclose :

These are macro functions called by decode request to execute the request. They are simply local open() write()….

Int checkCanOpen() : Whenever a file is opened, its information is stored in an array of fileInfo. This structure is used to check which file is opened in which mode. This also checks if a given file can be opened or not. Uses the following funcitons to decide :

checkFileOpen(pathname) : Checks whether a file is open.
CheckTransaction() : checks if any file is open in transaction.
CheckExclusive() : checks if any file is open in eclusive write mode
checkUnrestrictedWrtite() checks if any file is open in unrestrictied write mode

char ** getTokens() : helper function, returns tokens according to delim.

int getFileIndexFromFileDescp : helper function

StartQueue() : whenever Invalid_file_mode is encountered that file is put on a queue.

CheckQueue() : after any file is closed, a thread is spawned to check whether the first element in the queue can be executed now. If so, thread executes it and terminates.

Queueu.c :

Standard queue functions in the from of a linked list. Managed by netfileserver.c