Upon starting the netfile server, a thread is created to accept new clients. Another thread is created to watch the openfiles and command queue.
In the connection handler thread, we get the ip address of the client, and then start reading from the client.
Workflow for it goes as follows:

```
    1. read the first character of the data. This is the command to be run (o/r/w/c and u/e/t for ext A).
    2. switch on the command:
        1. if o:
            1. read the flag
            2. get the size of the filestring
            3. read the filestring into memory
        2. if r:
            1. read fd
            2. read nbyte
        3. if w:
            1. read fd
            2. read nbyte
            3. read buffer into memory of size nbyte
        4. if c:
            1. read fd
        5. if u/e/t:
            1. set character command
    3. once we have read the indcated amount for each command, break
    4. switch on the command:
        1. if o:
            1. get the access parameter the client is using
            2. get the file from structure to check other permissions
            3. check if can be opened in access parameter in mode
                1. if not, wait
                2. if timed out during wait, send client timeout
            4. check if client has any other open files
                1. if no, add client to clientlist
            5. open file
                1. if error, note error and notify client
            6. if file was not already open, add it to files open for watcher thread
            7. if file was already open, add the access type to filesOpen list
            8. add file to clients list of files open
            9. send response to client with fd
        2. if r:
            1. check if client has anything open
                1. if not, send -1 back to client with EBADF
            2. check if client has that file open
                1. if not, send -1 back to client with EBADF
            3. read from file
                1. if -1, then check error and report to client
            5. send client readsize and buffer
        3. if w:
```

```
        1. check if client has anything open
            1. if not, send -1 back to client with EBADF
        2. check if client has that file open
            1. if not, send -1 back to client with EBADF
        3. write to file
            1. if -1 check error and response to client
        4. report write size to client
    4. if c:
        1. check if client has anything open
            1. if not, send -1 and EBADF
        2. check if client has that fd open
            1. if not, send -1 and EBADF
        3. close fd
            1. if -1 then send -1 and EBADF to client
        4. remove file mode from filesopen
        5. post to filequeue
        6. if modes is null for file, invalidate file
            1. if queue is null remove file
        7. remove file from clients listing of files
            1. if client file list is null, remove client
    5. if u/e/t:
        1. add accessparam to client list
        2. report success to client
```

For the worker thread, we create the socket, set the options, bind, and listen on port 14314. Once we get a connect, we create a new thread for handling a client. Then close the socket.

For the file_queue watcher, we initialize a mutex, and while true, sleep 3 seconds and then get time, and then for each file in queue, check their queued commands if the time they were put in is greater than two seconds from time stored in variable. If yes, then mark invalid and post to that files semaphore. Once all files have been checked unlock and sleep.

Libnetfiles.c:

netserverinit takes the hostname and filemode, checks for the existence of the server, and also sends the mode that the user wishes to access the file in. It returns a server file descriptor to be used to connect, and sets a global hostname string that is used in the connectToServer method. If the server is not found, or an error occurs during communication between the client and the server, it returns -1.

connectToServer is a convenience function that takes a string hostname and makes a connection to the server, returning a file descriptor or -1 on error. This method is called at the start of netopen, netread, netwrite, and netclose, as well as during netserverinit to send the file mode.

netopen takes a file descriptor, buffer, and number of bytes, then creates a message in the format "o/permission/hostnamelength/hostname" and sends it to the server. It receives either a file descriptor or an error message. In the former case, it returns the file descriptor to be used. In the latter, netopen sets errno accordingly.

netread takes a file descriptor, buffer, and number of bytes, then creates a message in the format "r/filedescriptor/numberofbytes", and sends it to the server. It receives the number of bytes read and updates the buffer with the data read, or sets errno accordingly. The function returns the number of bytes read, or -1 on error.

netwrite takes a file descriptor, buffer, and number of bytes, then creates a message in the format "r/filedescriptor/numberofbytes", and sends it to the server. It returns the number of bytes written, or sets errno accordingly and returns.

netclose takes a file descriptor, then closes the socket and returns 0 or -1 based on success or failure.