Brandon Smith and Benjamin De Brasi
Asst3: Remote Files

Readme

Int netopen(const char *pathname, int flags)

Takes a string representing a file path, and file mode listed below. Connects to the server and decides whether the file can be opened. There are multiple reasons the open may fail. First Are the errors listed below. Second may be since I implemented extension A the file could be denied access if not in the correct mode. If a file is blocked because of extension A it will return EWOULDBLOCK. Also errno will be set on these errors. If the errno EWOULDBLOCK is set this means the queue time ran out. It is set to 2 seconds.

ERRORS:
EACCES
EINTR
EISDIR
ENOENT
EROFS
ENFILE
EWOULDBLOCK

MODES:
O_RDONLY,
O_WRONLY, or O_RDWR

RETURN VALUE:
netopen() returns the new file descriptor, or -1 if an error occurred (in which case, errno is set appropriately).

ssize_t netread(int files, void *buf, size_t nbyte)

This attempts to read n bytes from the file given by the file descriptor in netopen. These read bytes are placed in the file descriptor given by buf. If an error occurs while reading the data will not be sent and an appropriate errno will be set.

ERRORS:
ETIMEDOUT
EBADF
ECONNRESET

RETURN VALUE:
Upon successful completion, netwrite() should return the number of bytes actually written to

the file associated with fildes. This number should never be greater than nbyte. Otherwise, -1 should be returned and errno set to indicate the error.

ssize_t netwrite(int fildes, const void *buf, size_t nbyte)

This attempts to write n bytes from the file given by the file descriptor in netopen. These write bytes are given by buf. If an error occurs while writing the data will not be sent and an appropriate errno will be set.

ERRORS:
ETIMEDOUT
EBADF
ECONNRESET

RETURN VALUE
Upon successful completion, netwrite() should return the number of bytes actually written to the file associated with fildes. This number should never be greater than nbyte. Otherwise, -1 should be returned and errno set to indicate the error.

int netclose(int fd)

ERRORS:
EBADF

Closes current file access. This will close(fd) on the remote server. It will also attempt to notify the other threads in the waiting queue that the file is available.

RETURN VALUE
netclose() returns zero on success. On error, -1 is returned, and errno is set appropriately.

netserverinit(char * hostname, int filemode)
Determines if the host exists. Also sets the global transfer mode. If the transfer mode is invalid or the hostname does not exist an error is thrown and h_errno is set appropriately.

RETURN
0 on success, -1 on error and h_errnor set correctly
ERRORS:
HOST_NOT_FOUND
INVALID_FILE_MODE

EXTENSIONS DONE(A,B,C,D)

Extension A: This extension associates a transfer mode with every open request. I implemented it the way it is written in the spec sheet. Each open operation is a client, so if you try to open 2 of the same files in exclusive mode from the same program one of them will fail.

Extension B: I decided to have each file over 2k bytes be split into 4 streams. Since the spec sheet said no more than 10 sockets should be open at one time if there are no more sockets left the client will report a ECONNRESET error and the data will not be transferred. The ports are 25566-25576.

Extension C+D: I implemented this queue differently, but it still works. I used a combination of read() and pipelines to achieve this. If there is a conflict the read() will sit there until the close method writes to this pipe, or the monitor thread wakes up and scans for queues that have elapsed more than 2 seconds.

Testing Methodology for Extensions

Extension A: Opening the same file with different permissions in different modes would be an adequate way to test this.

Extension B: Transferring a large file and if you do not trust the ports written to output can monitor all the outgoing/incoming connections. Also doing a md5hash on the data transferred to verify that all the parts were successfully put back together.

Extension C+D: Since I am not sure if my program works correctly for multithreading the best way to test this would be to open multiple terminal windows and attempt to open multiple conflicting files and watch as the queue is processed, or they are terminated.

Definitions

```
#define PORT 25565
#define INVALID_FILE_MODE 6969
#define UNRESTRICTED 0
#define EXCLUSIVE 1
#define TRANSACTION 2
```

Compiling

The Makefile I included compiles the library with a test c file. This test C file has code which will copy the contents from filename to filename2. These files do not exist and must be created. This also creates the netfileserver binary.