Assignment #3 : Design document

1. Client / Server communication protocol

Message format is a comma-separated text --- “netFunc, parm1, parm2, parm3”. The first field is an integer representing the net function. The remaining 3 parameters depend on the net function chosen.

The net function parameter is an enumeration of the following:

//

// Net server function types

//

typedef enum {

NET\_SERVERINIT = 1,

NET\_OPEN = 2,

NET\_READ = 3,

NET\_WRITE = 4,

NET\_CLOSE = 5,

INVALID = 99

} NET\_FUNCTION\_TYPE;

//

// Supported file connection mode

//

typedef enum {

UNRESTRICTED\_MODE = 1,

EXCLUSIVE\_MODE = 2,

TRANSACTION\_MODE = 3,

INVALID\_FILE\_MODE = 99

} FILE\_CONNECTION\_MODE;

//

// Supported file open flags

//

typedef enum {

O\_RDONLY = 0,

O\_WEONLY = 1,

O\_RDWR = 2,

} FILE\_OPEN\_FLAGS\_TYPE;

typedef struct {

int sockfd; // file transfer socket

int port; // port number

int inUse; // TRUE= socket in use. Otherwise, FALSE.

NET\_FD\_TYPE netFd;

} FILE\_TRANSFER\_SOCKET\_TYPE;

typedef struct {

int fd; // File descriptor (must be negative)

FILE\_CONNECTION\_MODE fcMode; // File connection mode

int fileOpenFlags; // Open file flags

char pathname[256]; // file path name

} NET\_FD\_TYPE;

typedef struct {

char hostname[64];

FILE\_CONNECTION\_MODE fcMode;

} NET\_SERVER;

typedef struct {

int port;

int netfd;

int seqNum;

char \*buf;

int iStartPos;

int iLength;

} FILE\_PART\_TYPE;

Here is the communication protocol between the client and the server:

| Net function | Client | Server |
| --- | --- | --- |
| netserverinit | Send  “1,0,0,0” | Respond with    “resultCode,0,0,0”  where  resultCode = (0 for success, 1 for failure) |
| netopen | Send  “2,fcMode,flag,pathname” | Respond with  “resultCode, errno, h\_errno, netFd”  where  resultCode = (0 for success, 1 for failure)  errno = system error number  h\_errno = system error number  netFd = network file descriptor |
| netclose | Send  “5,netFd,0,0” | Respond with  “resultCode, errno, h\_errno, netFd”  where  resultCode = (0 for success, 1 for failure)  errno = system error number  h\_errno = system error number  netFd = network file descriptor closed |
| Netwrite | Send  “4,netFd,nBytes,0”  Spawn a “sendData” thread for each port:   |  | | --- | | Send to each port listener | | Send “4,netFd, SeqNum,nBytes”  Send “nBytes” of data  Receive response msg for the number of bytes written to server. | | Respond with configuration message:  “resultCode, errno, h\_errno, netFd,  PortCount,  PortNum, PortNum, PortNum, ….”  where  resultCode = (0 for success, 1 for failure)  errno = system error number  h\_errno = system error number  netFd = network file descriptor closed  PortCount = Total number of available ports  PortNum= Socket port number to use for writing  Spawn a “netwriteListener” thread for each port:   |  | | --- | | “netwriteListener” thread on each port | | Respond with:  “resultCode, errno, h\_errno, SeqNum, nBytes”  Receive “nBytes” of data from the client  Respond with:  “resultCode, errno, h\_errno, nBytes” | |
| Wait for all “sendData” threads to finish. | After all “netwriteListener” threads finished, reconstruct the received file from the sequence of all file piece parts. |
| Read the “nTotalBytes” returned from the server. | Final respond with:  “resultCode, errno, h\_errno, nTotalBytes”  where  resultCode = (0 for success, 1 for failure)  errno = system error number  h\_errno = system error number  nTotalBytes = Total bytes written. This is the sum of all bytes written by each part in the entire sequence. |
| Netread | Send  “3,netFd,nBytesWant,0”  Where  nBytesWant = bytes want to read from server  Spawn a “getData” thread for each port:   |  | | --- | | “getData” thread on each port | | Send “3,netFd, SeqNum,iStartPos, nBytes”  Received “nBytes” of data from the server  Respond with:  “resultCode, errno, h\_errno, nBytes” | | Respond with configuration message:  “resultCode, errno, h\_errno, netFd,  nFileSize, PortCount,  PortNum, PortNum, PortNum, ….”  where  resultCode = (0 for success, 1 for failure)  errno = system error number  h\_errno = system error number  netFd = network file descriptor closed  nFileSize = File size in bytes  PortCount = Total number of available ports  PortNum= Socket port number to use for writing  Spawn a “netreadListener” thread for each port:   |  | | --- | | “netreadListener” thread on each port | | Respond with sending “nBytes” of data  Receive response msg for the number of bytes received by client. | |
|  | Wait for all “getData” threads to finish | Wait for all “netreadListener” threads to finish |
|  | Receive “nTotalBytes” read from the server. | Final respond with:  “resultCode, errno, h\_errno, nTotalBytes”  where  resultCode = (0 for success, 1 for failure)  errno = system error number  h\_errno = system error number  nTotalBytes = Total bytes read. This is the sum of all bytes sent by “netreadListener” threads. |

Sequence of TCP Socket function calls used by client and server

|  |  |
| --- | --- |
| Client | Server |
| Client and server each creates a socket. Server also set up a “listener”. | |
| sockfd = socket( AF\_INET, SOCK\_STREAM, 0);  set :  serv\_addr.sin\_addr.s\_addr = host addr  serv\_addr.sin\_port = htons(PORT\_NUMBER)  connect( sockfd, &serv\_addr,…) | sockfd = socket( AF\_INET, SOCK\_STREAM, 0);  set :  serv\_addr.sin\_addr.s\_addr = INADDR\_ANY;  serv\_addr.sin\_port =tons(PORT\_NUMBER);  bind( sockfd, &serv\_addr, ……)  listen( sockfd, 50)  newsockfd = accept( sockfd, ……) |
| Client sends “net” function command | |
| write( sockfd, msg, strlen(msg)) | Spawn worker thread using newsockfd  read( newsockfd, msg, MSG\_SIZE -1) |
| Server sends result of executing the net function | |
| read( sockfd, msg, MSG\_SIZE -1) | write( newsockfd, msg, strlen(msg) ) |
| Finish communication | |
| Close( sockfd ) | Close( newsockfd ) |

1. Netopen access policy

This table describes access policy implemented with the “netopen” function. It determines if the combination of the open flags and open connection mode is allowed to access the given file. The decision to allow access is made based on how the specified file may have already been opened by another client before.

|  |  |  |  |
| --- | --- | --- | --- |
| Open connection mode | Open flags | | |
| O\_RDONLY | O\_WRONLY | O\_RDWR |
| Unrestricted | 1) Allowed if the specified file is not found in FD table.  2) Disallowed if file is already opened in transaction mode.    3) Otherwise, allowed. | 1) Allowed if the specified file is not found in FD table.  2) Disallowed if file is already opened in transaction mode, exclusive O\_WRONLY or exclusive O\_RDWR.  3) Otherwise, allowed. | 1) Allowed if the specified file is not found in FD table.  2) Disallowed if file is already opened in transaction mode, exclusive O\_WRONLY or exclusive O\_RDWR.  3) Otherwise, allowed. |
| Exclusive | 1) Allowed if the specified file is not found in FD table.  2) Disallowed only if the file is already opened in transaction mode.  3) Otherwise, allowed. | 1) Allowed if the specified file is not found in FD table.  2) Disallowed if file is either opened in transaction mode, exclusive O\_WRONLY or exclusive O\_RDWR.  3) Otherwise, allowed. | 1) Allowed if the specified file is not found in FD table.  2) Disallowed if file is either opened in transaction mode, exclusive O\_WRONLY or exclusive O\_RDWR.  3) Otherwise, allowed. |
| Transaction | 1) Disallowed if the specified file is found in FD table.  2) Otherwise, allowed. | 1) Disallowed if the specified file is found in FD table.  2) Otherwise, allowed. | 1) Disallowed if the specified file is found in FD table.  2) Otherwise, allowed. |