

CS 423 MP4 – Multi-threaded Secure File Server

Jin Heo – TA

Tarek Abdezlalher - Instructor

Goal

- Implement a multi-threaded, secure network file server
- Help you understand:
 - File systems
 - Socket programming
 - Client-server systems

Accounts and Machine

- Dedicated machine is prepared by TSG
- csil-vmserve1.cs.uiuc.edu
- Remote SSH access
- Your work will be graded on this machine

Communication Protocol

- You will write the code for the server side of the protocol
- Five types of requests from client to server:
 - FS_SESSION
 - FS_READ
 - FS_APPEND
 - FS_CREATE
- Each request is identified
 - Session number + sequence number
- Upon error, the server should close the connection (without sending a response message).
- The client side of the protocol is carried out by the functions in `libfs_client.a`
 - `fs_session()`, `fs_read()`, `fs_append()`, `fs_delete()`, `fs_delete()`

FS_SESSION

- Request a new session
- REQ: "FS_SESSION <username> <session> <sequence><NULL>"
 - <username> is the name of the user making the request
 - <session> and <sequence> are each 0
 - <NULL> is the ascii character '\0'
- RES: "<session> <sequence><NULL>"
 - should assign the *lowest-numbered* unused session
 - the first returned session number should be 0
 - <session> is the new session number
 - <sequence> should be 0

FS_READ

- Read an existing file
- REQ: "FS_READ <username> <session> <sequence> <filename> <offset> <size><NULL>"
 - <filename> is the name of the file being read
 - <offset> specifies the starting byte of the file portion being read
 - <size> specifies the number of bytes to read from the file
- RES: "<session> <sequence><NULL>"<data>"
 - <data> is the data that was read from the file.
 - The size of <data> should be the <size> from the request message.
 - the file server should check the validity of its parameters.
 - Check that the file exists, is owned by <username>, and is large enough to satisfy the request.

FS_APPEND

- Appends to an existing file
- REQ: "FS_APPEND <username> <session>
<sequence> <filename> <size><NULL>"<data>
 - <size> specifies the number of bytes to append to the file
 - <data> is that data to append to the file. The size of <data> is given in <size>.
- RES: "<session> <sequence><NULL>"
 - No data should be appended to the file for unsuccessful requests.

FS_CREATE

- Creates a new file
- REQ: "FS_CREATE <username> <session>
<sequence> <filename><NULL>"
 - <filename> is the name of the file being created
- RES: "<session> <sequence><NULL>"
 - check that the file does not yet exist and that there is sufficient disk space to create a new file.

FS_DELETE

- Delete an existing file
- REQ: "FS_DELETE <username> <session>
<sequence> <filename><NULL>"
 - <filename> is the name of the file being deleted
- RES: "<session> <sequence><NULL>"
 - Check the validity of its parameters. The server should also check that the file exists and is owned by <username>.

Encryption

- All messages between the client and file server will use encryption
 - secret-key encryption based on the user's password.
 - The file server will be given a list of user and passwords when it is started
- We will provide encryption and decryption functions.
 - Explained later

Encryption Cont.

- All request messages will be encrypted using the password parameter that was passed to the client function.
 - A cleartext request header followed by the request message.
 - "<username> <size><NULL>"<body>
 - The file server uses the user name to decrypt the ensuing request message.
- All response messages will be encrypted using the user's password.
 - A cleartext response header before sending the response message.
 - "<size><NULL>"<body>

File System Structure

- The file system consists of a single directory of files.
 - Stored in disk block 0
 - Contains fs_direntry entries (one entry per file).
 - File name and inode block #
 - Unused directory entries have inode_block=0.
- Each file in the file system is described in an inode.
 - Each inode is stored in a single disk block.
 - Owner
 - file size
 - Blocks array: for data blocks

File Server Internals – Argument and Input

- Your file server should be able to be called with 0 or 1 command-line arguments.
 - Specifies the listen port number
- Your file server will be passed a list of usernames and passwords via stdin
 - user1 password1
user2 password2
user3 password3
user4 password4
- Example:
 - "fs 8000 < passwords" or "fs < passwords"

File Server Internals - Initialization

- Read the list of usernames and passwords from stdin.
- Set up the listen socket
- Read the directory block from disk into memory, then read the inodes for all valid directory entries
 - Your file server should be able to start with any valid file system (empty or with files).

File Server Internals - Concurrency

- Multi-threaded server
 - Serve *any* number of clients at the same time.
- The following operations are allowed in parallel:
 - FS_SESSION with all other requests
 - Multiple FS_READ's of the same file
 - FS_READ or FS_APPEND of one file, with FS_READ or FS_APPEND of a different file.
- Should NOT proceed in parallel:
 - FS_APPEND of a file with FS_READ or FS_APPEND of the same file.
 - FS_CREATE or FS_DELETE of any file with FS_READ, FS_APPEND, FS_CREATE, or FS_DELETE of any file
 - FS_CREATE and FS_DELETE write the directory.
- **Hint:** use pthread reader-writer locks

Encryption/Decryption Utility

- fs_encrypt() and fs_decrypt() to encrypt/decrypt an arbitrary buffer of data.
 - Return a pointer to the transformed data and its size
 - Free the memory allocated by fs_encrypt() and fs_decrypt()
- Two types of encryption: CLEAR and AES.
 - CLEAR: a trivial encryption scheme that leaves the data visible
 - AES: use the AES (Rijndael) algorithm.
 - Set the FS_CRYPT environment variable to CLEAR or AES.
 - "setenv FS_CRYPT CLEAR" or "setenv FS_CRYPT AES".
 - "export FS_CRYPT=CLEAR" or "export FS_CRYPT=AES".

Other Useful Functions and Programs

- "createfs"
 - Create a newly initialized file system in the Linux file "/tmp/fs_tmp.<loginid>.disk"
- "showfs"
 - Show the current file system contents
- disk_readblock, disk_writeblock()
 - Read and write a disk block

Sockets and TCP

- Berkeley sockets
 - A common programming interface used in network programming
 - A brief introduction to socket programming
 - <http://www.beej.us/guide/bgnet/output/html/singlepage/bgnet.html>

Pthreads

- Your server is multi-threaded: use pthread library
- Useful pthread APIs:
 - pthread_create
 - pthread_detach
 - pthread_mutex_init
 - pthread_mutex_lock
 - pthread_mutex_unlock
 - pthread_cond_init
 - pthread_cond_wait
 - pthread_cond_signal
 - pthread_cond_broadcast
 - pthread_rwlock_init
 - pthread_rwlock_rdlock
 - pthread_rwlock_wrlock
 - pthread_rwlock_unlock
 - pthread_attr_init
 - pthread_attr_setstacksize

Test Cases

- Write your own test cases
- You should test your file server with both serial and concurrent client requests.
- Possible categories of test cases that you can consider
 - basic functionality
 - error handling
 - large, serial (i.e. non-concurrent) operations
 - start with pre-existing file systems
 - concurrent operations

Others

- Files are in csil-vmsserv1:/home/class/sp10/cs423/mp4.
- Some important rules
 - MUST use disk_readblock/disk_writeblock to write the disk, fs_send to send messages, and fs_close to close a network socket.
 - Send each response via two calls to fs_send: one to send the cleartext response header, the second to send the response itself.
 - When allocating a disk block, always choose the lowest-numbered free disk block.
 - When FS_CREATE allocates a directory entry, it should choose the lowest-numbered free directory entry.

Submission

- What should be submitted
 - Tar and gzipped file containing
 - C++ program for your file server (name should be "fs.cc")
 - Readme file: what you have done
- Email the TA by 10:00am (i.e., before class) on April 26th.
- Send an email with the subject line including "CS423 SP10 MP4"
- You can resubmit your work anytime before the deadline.

Questions