Assignment 3

Xiaoxiao He & Zhizhang Deng

Highlights in this project:

1. *We successfully implemented all required functionality stated in the assignment description.*
2. *We utilize and improved the "libtar" library in the iLab machines by reimplementing some of the functions and reduce the potential memory leaks.*
3. *We only store compressed version (tar) of the project at repository and send all files over the network in one compressed file.*
4. *We utilized a common practice called layered programming structure to help us make the program more robust and easier to implement.*
5. *We reused a lot of source code in both client and server in order to reduce workload.*
6. *We implemented Read/Write lock to maximize the multithread efficiency.*
7. *We enforce network protocol in order to make our network more robust to malicious attacks.*
8. *We implemented a path check function so that we can handle all kind of inputs and will normalize the path to our criteria if the path is valid.*
9. *We implemented WTFtest to perform stress test on server with high concurrence input and output. We utilize conditional variable, 2 mutex, one barrier for enforcing simultaneous output to the server.*

1. Introduction:

In Assignment 3 Where's the File, we implement another version of git by using network I/O and multithread programing.
We implemented the following functionality:

1. ./WTF create <project name>: create a folder with our directory structure on both server and client and setup necessary environment.
2. ./WTF destroy <project name>: destroy the corresponding project folder on server.
3. ./WTF add <project name> <file name>: add the file to the project manifest.
4. ./WTF remove <project name> <file name>: remove the file from the project manifest.
5. ./WTF configure <ip addr/hostname> <port number>: write the client configuration file.
6. ./WTF currentversion <project name>: will display server's latest manifest.
7. ./WTF rollback <project name> <project version>: will roll back the project back to given version.
8. ./WTF checkout <project name>: will download the current project from the server.
9. ./WTF update <project name>: will compare the local files with server files and generate .Update file for upgrade.
10. ./WTF upgrade <project name>: will execute the .Update unless there are changes in the folder after update.

11. ./WTF commit <project name>: will compare the local files with server files and generate .Commit file for push.
12. ./WTF push <project name>: will push the local changes to the server unless changes are found between commit and push.

2. Methodology:

2.1 Server Directory Diagram

The following diagram is the folder structure of our server. Since we need to design our server for handling different projects, this is the best way for doing that. We only store tar files of the passed version.

```
--WORKING DIRECTORY
|
| ------ WTFserver
| ------ Projects
         |
         |------Project_NAME1
         |------Project_NAME2
         |------Project_NAME3
         |------Project_NAMEN
                  |
                  |------0
                  |------1
                  |------n (project version)
                         |------ files.tar
                         |------.Commit
                         |------.Manifest
                  |------Currentversion (a file store a number)
                  |------Curr (folder store the files of current version)
                          |
                          |------.Manifest
                          |------……
```

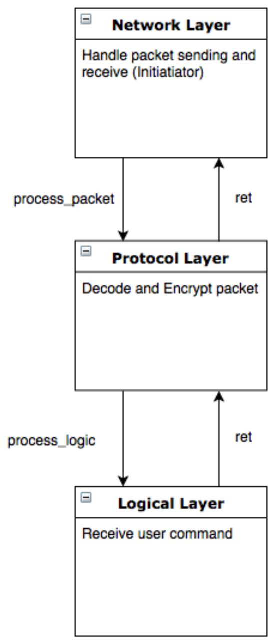
2.2 Features:

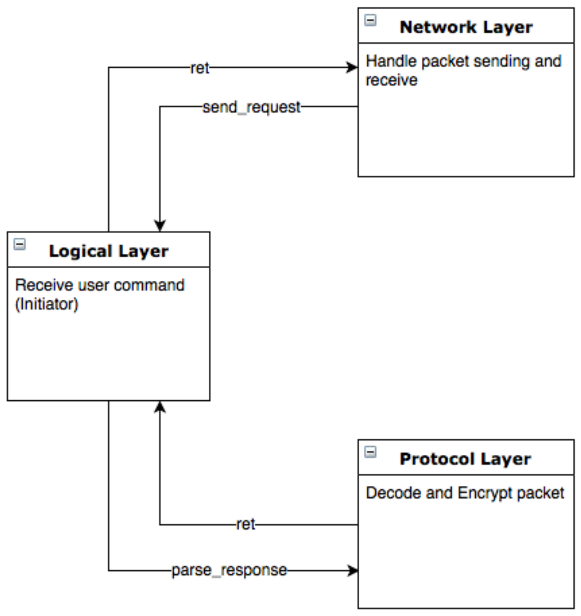
1. Layered program structure

Our program is separated into three different layers: Network Layer, Protocol Layer, Logical Layer.

Control flow diagram:

Server Side   Client Side




Network Layer
Handle packet sending and receive
ret
send_request
Logical Layer
Receive user command (Initiator)
Protocol Layer
Decode and Encrypt packet
ret
parse_response

Each layer can be changed independently as long as their interfaces stay intact. This design enables us to reuse source codes for both clients and servers.

2. Source code reuse on both Client and Server

By utilizing different compile-time macro definition, we managed to compile the same source code files from Protocol Layer and Network Layer for both client and server.

3. Utilized Reader/Writer lock to enable high concurrency (different project need to write)

We used a reader starving pthread_rwlock for every single project. Reader/Writer lock in this case is very efficient as it allows us to enable multiple people downloading the same project but only one people can write to a project.

4. Network Defensive programming

By deploying multiple defensive mechanism in our network layer and protocol layer, we can prevent server from serving requests that are generated by attackers. Some of the defensive mechanisms we deployed:

1. 5 minutes wait time:
    a. The timeout limits for a server thread to receive each data chunk from one single request is 5 minutes. If such timeout is reached

    before the receiving a full packet, the request is marked as invalid, and the resources related to it are released.
    b. Upon receiving each data chunk, the timeout is reset
2. Packet size
    a. After a client claimed to send X bytes of data, but we received more, the connection is marked as invalid and the resources related to it are released.
3. MAX_PACKET_SIZE
    a. The maximum packet size that a client can claim to send is 5GB.
    b. When a client claimed to send more than 5GB of data (as claimed in the first 8 bytes of the packet), the request is marked as invalid, and the resources related to it are released
4. Protocol validation
    a. Any packet that does not meet our protocol constraints (See: Communication Protocol) are marked as invalid and the connection related to it are released.

5. Patched the iLab libtar library

When we are doing some research on what tar libraries we should use, we find out that libtar.so.1.2.11 is a working library. However, this version of the library has some memory leaks in it. We have to make some patches to this library in order to make it work correctly. The patch of code changes that we made are available at the file "libtar.patch". We also made the our modified libtar open source in order to comply GPL license: https://github.com/dzz007/libtar

6. Handle relative path

We utilize various Linux API such as realname() to make sure the WTF client works perfectly even when it is receiving relative path as input such as: "../../a/b/cdf/gg.c"

7. Extra Credits:

1. Compress old versions of the project at the repository:
    a. We have implemented it, please refer to our Server's folder structure diagram at page 1
2. Compress all files to be sent over the network
    a. We have implemented it as well. The project compressed tar will be sent using our Payload2 field in our protocol, and we selectively decompress files from it that are interested
3. Compress using libtar or gzip but not the system call
    a. We have implemented this one also. After we patched libtar version 1.2.11, we utilize our patched version to tar and untar files.

Test Plan:

We will test the following:
1. Ordinary test:
configure the client, open the server on 5555.
create a project, add a file, commit and push.

use another computer, or in another folder, configure the client and checkout the same project.

make any modification in the project and commit and push.

go to the first client and execute update and upgrade.

Check currentversion and history.

Destroy the project.

(Additional test is recorded in the testcases.txt file)

2. Server Stress Test
We will use WTFtest to perform push, checkout, create (executed in sequence) with 50 concurrence connection.

3. Abnormal operation
Please read the section 2 in testcases.txt

Appendix

Communication Protocol

Request:

Packet Size	OP Code	Project Name Size	Project Name	IsTwoPayload	DATA
8 bytes	1 byte	8 bytes	(pns)	1 byte	(?)

Response:

Packet Size	Status Code	IsTwoPayload	DATA
8 bytes	2 bytes	1 byte	(?)

Data Section specification is defined by IsTwoPayload

When IsTwoPayload=1
Data Section defined as:

Payload1 Size	Payload1	Payload2
8 bytes	(p1s)	(?)

(Note: Payload1 is normally used as transferring .Commit / .Manifest file, while Payload2 is normally used to transfer TAR binary)

When IsTwoPayload=0
Data Section defined as:

Payload1
(p1s)