Assignment 3

# Introduction:

# Server Directory Diagram

--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

|------……

# Features:

## Layered program structure

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

Control flow diagram:

|  |  |
| --- | --- |
| Server Side | Client Side |
|  |  |

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.

## 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.

## Utilized Reader/Writer lock to enable high concurrency

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.

## 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. 3-seconds wait time:
   1. The timeout limits for a server thread to receive each data chunk from one single request is 3 seconds. 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.
   2. Upon receiving each data chunk, the timeout is reset
2. Packet size
   1. 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
   1. The maximum packet size that a client can claim to send is 5GB.
   2. 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
   1. Any packet that does not meet our protocol constraints ( See: Communication Protocol ) are marked as invalid and the connection related to it are released.

## 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>

## 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”

# 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) |

# Extra Credits:

1. Compress old versions of the project at the repository:
   1. 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
   1. 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
   1. We have implemented this one also. After we patched libtar version 1.2.11, we utilize our patched version to tar and untar files.