Design Document: Multi-Threaded HTTP server with logging

Aaron Nguyen
CruzID: anguy200

1 Goals

The goal of this program is to utilize and improve code from the previous assignment to implement multithreading and a logging feature. The code should be able to mimic the functions of HTTP which is mainly being able to receive and send files usually code files and parse these to display information easily read by people.

2 Design

For this program there will be many parts to the code, and as the code is implemented more and more functions will be added or removed based on circumstances. The initial functions to be included into this doc will be a function that runs a server and the functions created in the last assignment HTTP Server which mimics the functionality of a http server. In addition to Multithreading the program must also be able to create a log file which takes the data being received/sent to the server and convert it to hex.

2.1 Handling Arguments and Files

For this program Multi-Threaded HTTP, getopt which uses flags and indexes to parse through arguments. This checks if the function is actually being called and afterwards if it is true all subsequent file names are taken in. From there in the main function steps are taken to initiate a server and prep to receive multiple connections to parsed out to n number of threads.

```
#Input: file names
#Output: data found in each file
For length of args
     Declaration of socket
variables
     Assign socket variables
     Get main socket ip and port
     Listen for connection
     For n
       Create Threads

requests
     while(accept true)
         Semaphores up
         Buff[of sockets]
         Semaphore down
```

2.2 Parsing Headers and Sending Responses

This function is a program that is called for each thread and parses through the header requests when there is a socket in a global buffer. After parsing through the header is should call up other functions which complete respective tasks of type request.

```
#Input: file names
#Output: data found in each file
For length of args
     if filename == -
          take stdin
          Print stdout()
     While recv is not 0
```

```
          Continue to receive
packets
Call second function
```

2.3 Recombing and Printing Data to Socket or local Files

Due to all the data being stored in a heap. All that needs to be done is to pass the heap into a print function. Which will take each string and print it to the stdout

```
Printfile
      While (heap != empty)
      If not valid
            Return -1
      while string is not
empty
            print
      return 1
      free heap
```

2.4 Checking Lines for Valid FileNames

The role of this function is to check whether or not the name of a file being requested or created is a proper file name

```
validity_check(string filename){
    char c;
    if(filename.length() != 27) return false;
    for( u_int i = 0; i < filename.length(); i++){
       c = filename.at(i);
       if((ascii letters, numbers or dash/underscore){
             continue;
       }else{
             return false;
       }
    }
```

```
    return true;
```

2.5 GET Method

Because there are two methods being used in this program this function is used to accomplish the GET task and check for errors while doing it

```
void get_parse(header, socket){
    if(file name is not valid){
       HTTP/1.1 400 Bad Request\r\n
    }
    Open file
    if(Insufficient permission is true){
       HTTP/1.1 403 Forbidden\r\n
       }
    if(if file doesn’t exist){

       HTTP/1.1 404 Not Found\r\n
  }else{
       Print function(file number, socket);
  }
```

2.6 PUT method

All this file does it makes sure the header is passing a valid file name for PUT to use and passes a variable to other functions to begin the PUT process

```
string put_parse(string header, int
socket){
    if(if file name is valid){
       return temp;
    }
    HTTP/1.1 400 Bad Request\r\n
}
```

2.7 PUT method

All this file does it makes sure the header is passing a valid file name for PUT to use and passes a variable to other functions to begin the PUT process

```
int get_put_checker(string line){
    string temp =
line.substr(0,3);
    if(temp == "GET"){
       return 1;
    }
    if(temp == "PUT"){
       return 2;
    }
    return 0;
}
```

2.8 Recombing and Printing Data to Socket or local Files

Due to all the data being stored in a heap. All that needs to be done is to pass the heap into a print function. Which will take each string and print it to the stdout

```
int catch_length(string line){
    if(line.substr(0,7).compare("Content") ==
0){
       int first = (line.find(" ") + 1);
       string temp = line.substr(first);
       int size;
       size = stoi(temp);
       return size;
    }
    return -1;
}
```

2.9 Recombing and Printing Data to Socket or local Files

All this program does is goes through the initial lines sent in by the client and parses through it figure out what method the client is asking for. After discovering method type it calls up the appropriate method functions. This function is called repeatedly every time recv is called

```
void parse_recv(recv is
recieving)
  Checks Method Type{
   if(Get){
      GET FUNCTION;
   }
   If(Put){
     PUT FUNCTION
   }
   Else{
     continue;
   }
}
```

2.10 Logging File Generator

The point of this program is to create and write to a logging file in HEX when the -l file is passed into the argument line whenever a request is made to the server. In addition multiple threads must be able to write to the file at the same time or in sync without overwriting.

```
Void logger (type, body, length)
      Begin to compile char array to be converted into HEX
      For num of lines in HEX = length / 20 + 1
            Added Zero Space
            For every 20 char
```

```
            Convert to hex
            Hex string  + hexchar
      Hex string + new line
Mutex lock
      Write buffer += length of hex string
Mutex unlock
Pwrite hex string
```