Asgn 2 Design Document

Samuel Shin
Cruzid: sayshin

CSE 130, Fall 2019

1 Goal

The goal of this program is to implement multithreading and logging on top of Assignment 1. The program should be able to detect the options -N and -l using getopt(). -N is for the number of threads (default 4) and -l [log file] to specify logging. The program should be able to create a pool of worker threads with N amount, and when a connection is accepted, the dispatcher will have one of the threads in the pool do the work and handle the GET/PUT request like in Asgn1. If logging is specified, it should also log GET, PUT, and FAILs accordingly. If all threads are in use, and another request comes in, it should wait until a thread opens up.

2 Assumptions

I'm assuming that the testing will be done with "curl http://localhost:8080 --request-target filename_27_character" for GET requests and "curl -T localfile http://localhost:8080 --request-target filename_27_character" for PUT requests. I'm assuming that my GET works fine because on Piazza it says that it's okay if we're getting this warning "Warning: Binary output can mess up your terminal. Use "--output -" to tell Warning: curl to output it to your terminal anyway, or consider "--output Warning: " to save to a file." I'm assuming that the user will input something like 127.0.0.1 for the address name when running ./httpserver [address name] [optional port num] because my program can't handle something like http://localhost. I'm assuming that if using the default port number 80, use sudo ./httpserver [address name] to start the server. Also for -N, use something like -N8 (Note the lack of space). I'm assuming that the grader will try the test commands over and over again until it works because sometimes it freezes up (maybe an ubuntu error or something, maybe like 5 times should be okay).

3 Design

The way I am approaching this problem is being able to parse the command line command using getopt(), catch options for -N and -l and be able to save this information in the program. Then I'm going to get logging done first because it's easier than multithreading. To accomplish logging, I have to detect first if -l was given and if a filename was given with it. Have a bool var that lets the rest of the program know if logging has been requested. If logging is active, then log GET requests accordingly, PUT requests appropriately along with FAILs. Next, try to get the pool of worker threads working, figure out how to multithread with a simple function first that takes in no args, then multithread on the assignment 1 code and break it up into a function you can call for pthread_create.

4 Pseudocode

Procedure httpserver

If num of args < 3 || > 6
Error out
Int Option = getopt(N::l:) //N has an optional arg and l needs an arg after
If (option == 'N')
Set number of threads to arg if given
If (option == 'l')
Set bool var to true telling that log has been requested
If (option == ':' || option == '?')
Something went wrong, handle error

Create thread pool based off of number specified from command line

Have a vector of bools initialized to false that represents each of the worker threads called available_threads //this is to tell other requests if a thread is available or not
While (true)
Call socket(), setsockopt(), bind(), listen(), accept() in this order to establish connection

Somewhere in above steps, set port number if port number was given, otherwise default is 80
On accept() that's when the dispatcher knows to hand the request to a thread. Once a request is given to a thread, set the thread associated with the available_threads to false so that another request can't be processed on that thread.

recv() request(s) from client

Parse http header

If header has GET

Parse and check if filename is legal

Basically do the dog algorithm we did, but instead of writing to stdout Write to the client by doing a send(sockfd, buffer, bytes, 0)

Else If header has PUT

Parse and check if filename is legal

Get the content length to see how many bytes to read

Call recv again to get the data to put into the file

Then place file contents into given filename

Else

Send a error 500 code

Free up the available_worker that the thread is associated with