1 Introduction

1.1 Goal

This httpserver.c is to implement a multithread httpserver, creating a pool of threads to handle the connection, the thread is not “busy wait” on locks

1.2 Statement of Scope

The server will respond to simple GET, and PUT commands to read and write files, and the command Head to provide information on existing files. With the following command, the server will respond to the validity of messages, with option argument -l logfile to record the result of each request, Get/health check the the total number of the request processed and the request failed

1.3 Running system

This program is running on the Linux system.

2 Data Design

2.1 Internal data

This program has a httpObject struct data type to store the component of messages from the user.

```
struct httpObject {
   /*
      Create some object 'struct' to keep track of all
      the components related to a HTTP message
      NOTE: There may be more member variables you would want to add
   */
   char method[5];         // PUT, HEAD, GET
   char filename[28];      // what is the file we are worried about
   char httpversion[9];    // HTTP/1.1
   ssize_t content_length; // example: 13
   int status_code;
   uint8_t buffer[BUFFER_SIZE];
 };
```

struct logfile to store the log name from the user

```
   struct logfile{
   char *logdata  //store the name of log file
```

}

The struct QueueObj is to store the client in the order of “first in first out”

```
Struct QueueObj{
 Node front;
 Node back;
Int length;
}
```

2.3 Global data

```
pthread_mutex_t mutex=PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cond=PTHREAD_COND_INITIALIZER;
```

Mutex is for the create a critical section

Cond is to signal the sleeping function

```
int indicateoffset =0;
```

Indicateoffest is to count the number of byte have written in the logfile

```
int error =0;
int process =0;
```

Error and process is to count the number of total process and bad request

3 Architectural and component-level design

3.1 System structure

3.11 creating multi thread in Main function

In the main function, we first initialize the server_socket with address information from the command line. And creating the dispatcher thread by the number of user specified threads using a for-loop and P_thread_create(), with a thread function, thread(), as the arguments. This dispatcher thread will listen to connections and delegate them to threads and listen to the incoming connection. In the while loop, we are listening to the client, accepting the client, store the client_socket in the queue and signal the workerthread to handle the request.

Delegating to thread

The dispatcher thread will create a queue structure and in order to avoid race conditions,we put mutex lock, enter the critical section, put the connection on the queue, increase the length of the queue, leave the critical section, unlock mutex lock and signal the worker thread to handle the connection.

3.12 worker thread keep concurrency without no "busy wait" on locks

In my workerthread function, it will receive the log file as argument, initialize the filename of logfile. Then go to the while loop, first, it enter the critical section, it check the number of client in the queue; if there is no client in the queue, the workerthread will wait the main function to put a client in the queue and receive the signal from the main function, it get the client, delete the that client from the queue, and quit the critical section, and start to process the request from the client.

Elimination busy waiting

To prevent the threads from constantly checking the queue, the dispatch thread will use a condition variable cond to signal worker thread that there is a client on the queue which needs to be processed in the workerthread function, the thread waits on the signal using conditional variable and the lock, after it receive the signal, it will check the existence of client, and jump out of while loop.

3.2 Logging

Goal:

log all the result of request with a hexadecimal representation of the data to a logfile

Design

Since we continue to write the content into the logfile, we need to track the number of bytes we have written to the logfile; when we write the new data to the logfile, we need to offset the number of bytes we have written into the log file. We using pwrite to write the data into log file, **ssize_t pwrite(int** *fd*, **const void ***buf*, **size_t** *count*, **off_t** *offset*);** which take three arguments, the buffer, content length, and offset.

situation1 : write header of request to the logfile

For every request, we need to write the header and the data to the logfile. In order to track of bytes, I set up a "indicateoffset": an global variable to track the number of byte written in the logfile. Every time we write the content into the logfile, we set the mutex lock ,enter the critical section, increase the indicateoffset with size of file data and the size of header,unlock mutex, and leave the critical section, to prevent the race condition

situation2 : write data of source file to the logfile

For get and put, we need convert the content of source file into hexadecimal representation and write it into a log file; creating counter to represent the number of data need to be read and enter a while loop ,we use read() to the get the content of file with 20 bytes every time, for 20bytes, we use sprintf(buf,%02x",writebuf[K], to convert every byte of data into hexadecimal representation, after convert these 20 bytes data, we back to while loop to read another 20 byte from the source file until i have read all the data from the source file indicated by the counter which indicate the number of bytes need to be read.

3.3 Healthcheck

Goal:

Return the number of process and the number of failed process

Design:

I set up two global variables, int process, int error, to count the number of total requests and the number of failed requests. For every request which sends the error message, error will be incremented by1. At the end of the program the process increments by 1. Also we need check the validity of checkhealth request, if there is no log file, report 404 by test the existence of log file;request to head or put the healthcheck will return 403 by check the method of request

4 user interface design

Usage

program will take a port number as a parameter, it will also have two optional parameters defining the number N of threads and the name of the log_file, clients in the different directory send the message.

Example command

/httpserver 8080 -N 4 -l log_file

/httpserver -N 6 1234 • .

/httpserver 8010 -l other_log

/httpserver 3030

Example command

PUT instruction

curl -T client_input.txt http://localhost:8080/server_file

GET instruction

curl http://localhost:8080/server_file

HEAD instruction

curl -I http://localhost:8080/server_file

5 Testing Issues

The program may take a long time to process the request which demands the large size file and write The data of the file into a log file.