Design Document
Billy Kwong
Bqkwong
CSE 130 Fall19

ASGN2 Design

Goal

The goals for Assignment 2 are to modify the HTTP server that you already implemented to have two additional features: multi-threading and logging.

Assumptions

I'm assuming that the server will run on Unix environment and user will generate header as required by server else it will generate the error. In addition, it will only support the GET and PUT request from user and present working directory of user will serve as the storage of resources.

Design

The general approach I'm taking is to make the server listen to a queue on clients one by one (as it is a multi-threaded server) it will read the request header of client and perform some operation (PUT or GET) on the basis of that request and provide the response header while making log if given in parameters. It will generate an appropriate error if the request gets failed or server crashes.

*** PIAZZA Comments***
1) Create threads with pointers to a shared buffer
2) Make all threads wait on a lock
3) When requests come in:
   a) Update the shared buffer
   b) Signal to lock, waking up a thread
4) If all threads are busy, queue the requests

At the beginning of the program, I create N number of threads based on user input and i pass my dispatcher function into p_thread_create. From the dispatcher function, I wait for and accept any incoming requests. Then from there, I hand off the connection to the next available thread

Pseudocode

define BUFFER_SIZE 4096
define DEFAULT_PORT 80
initialize pthread_mutex_t mutex;

Procedure httpserver

```
char *logfile=NULL;
```

```
char hostbuffer[256];
char *IPbuffer;
struct hostent *host_entry;
int hostname;
hostname = gethostname(hostbuffer, sizeof(hostbuffer));
host_entry = gethostbyname(hostbuffer);
IPbuffer = inet_ntoa(*((struct in_addr*)host_entry->h_addr_list[0]));
int port = DEFAULT_PORT;
if(argc == 3 && strcmp(argv[1],"-N"))
{
        N = atoi(argv[2]);
}
else if(argc == 3 && strcmp(argv[1],"-l"))
{
        logfile=argv[2];
}
else if(argc == 3)
{
        IPbuffer=argv[1];
        port = atoi(argv[2]);
}
else if(argc == 2)
{
        IPbuffer=argv[1];
}
else (argc < 2 || argc > 3)
{
        exit_with_error("Usage: ./httpserver <ip> <port> or ./httpserver <ip>.");
}
pthread_t thread[N];

if(logfile == NULL)
        int lfd=open("/dev/null",O_WRONLY);
Else
        lfd=open(logfile,O_WRONLY & O_CREAT);

server_sockfd <- 0
client_sockfd <- 0
Define structure sockaddr_in server_addr;
Define structure sockaddr_in client_addr;

Create passive socket for the server
Create an address structure containing server IP addr and port, then
```

```
Bind the server_sockfd with this address

Create connection queue and wait for clients
int count <- 0
While true
        if(count > N)
                create dispatcher thread for count
                pthread join for count
                count  <- count +1
        else diplay error
close(server_sockfd)
close(client_sockfd)

End

procedure DISPATCHER
        lock mutex
        Accept a connection, blocks until connection from client is established
        will return a brand new descriptor for comm with this single connection

        if client_sockfd == -1
                display_error "HTTP/1.1 500 INTERNEL SERVER ERROR\r\n"
                write(lfd,er,sizeof(er));
                Continue
        else
                count <- 0;
                Read from sockfd
                define buf[BUFFER_SIZE];
                define rv;
                while rv <- recive data from client
                        tokenize the string and push them in vector

                if GET request
                        if file doesn't exist
                                display_error "HTTP/1.1 404 NOT FOUND\r\n"
                                write(lfd,er,sizeof(er));
                        else if exists but is not accessible
                                display_error "HTTP/1.1 403 FORBIDDEN\r\n"
                                write(lfd,er,sizeof(er));
                        Else
                                fd <- open file descriptor
                                while data in file
                                        send data to client_sockfd
```

```
                    send data to lfd
close fd;
            display_error "HTTP/1.1 200 OK\r\n"
            write(lfd,er,sizeof(er));
else if PUT request
        fd <- open file's descriptor
        if fd == -1
            display_error "HTTP/1.1 403 FORBIDDEN\r\n";
            write(lfd,er,sizeof(er));
            int fd = open(tokens[1].c_str(),O_RDWR & O_CREAT);
        Else
            vector<string> content;
            stringstream cl(header[1]);
            while(getline(cl, intermediate, ' '))
                content.push_back(intermediate);
                if(content[0] == "Content-Length:")
                    int len = atoi(content[1].c_str());
                    int i=2;
                    while(len>0 && i<count)
                      {
                      write(fd,header[i].c_str(),header[i].length()+1);
                      write(lfd,header[i].c_str(),header[i].length()+1);
                      len -= header[i].length();
                      I++;
                     }
                    Else
                        for(int i=1;i<count;i++)
                            if(header[i] != "\r")
                            write(fd,header[i].c_str(),header[i].len
                        gth()+1);
                            write(lfd,header[i].c_str(),header[i].le
                        ngth()+1);
                    close(fd);
                    char er[] = "HTTP/1.1 201 CREATED\r\n";
            send(client_sockfd, er, sizeof(er),0);
        write(lfd,er,sizeof(er));
        Else
            if count<-content length
                close fd;
                    display "HTTP/1.1 201 CREATED\r\n"
                    write(lfd,er,sizeof(er));
        Else
            display_error = "HTTP/1.1 400 BAD REQUEST\r\n";
```

```
        write(lfd,er,sizeof(er));
unlock mutex
return
```