Computer Networks

HTTP-protocol

Muhammed Kamal Abd Elrhman 60

Introduction:

use sockets to implement a simple web client that communicates with a web server using a restricted subset of HTTP.

Multi threaded based server:

Using thread based server since it is lightweight processes.
Run server using command line "*simple_server PORTNUMBER (optionally number of maximum clients)*"

Pseudocode:

```
while true: do
    Listen for connections
    Accept new connection from incoming client and delegate it to worker thread/process
    Parse HTTP/1.1 request and determine the command (GET or POST)
    Determine if target file exists (in case of GET) and return error otherwise
    Transmit contents of file (reads from the file and writes on the socket) (in case of GET)
    Wait for new requests (persistent connection)
    Close if connection timed out
end while
```

Code organisation:

- Server.h & server.cpp:

  Simple class represent the server using system calls

```
class server {
public:
    server();
    bool bindOnSocket(char* portNumber);
    void start_listening(int queueSize);
    int acceptConnection();
    void startServer();
    vector<string> requestParser(string req,int *body_length,char** body);
    void handleRequest(vector<string> request,int sock,char* buffer,int recived);

private:
    int socket_fd;
    int new_socket_fd;
    int queueSize;
    vector<client_attr> connected_clients;
    int client_num;
    map<string,string> file_extension_map;
    string get_header_of_file(string file_name);
    bool file_exist(string file_name);
    vector<char> readfile(string file_name);
    void handleGET(string file,int sock);
    void handlePOST(string file_name,int sock,char* buffer,int recived);
};
```

- Struct client_attr:

  Struct carry necessary information about the client that connect to the server

```
struct client_attr{
    int socket_fd;//socket to respond with to the client.
    clock_t time;//time of the client's last request.
    void *s;// thread function that handle client requests.
};
```

- Every client has thread in the server that receive requests from that client

- Interrupter thread that monitoring the client threads and terminate idle ones

```
//intrupt thread that monitering the clients and close connection with
//idle clients.
void* interupt(void* arg){
    wrapper *w = (wrapper*)arg;
    vector<client_attr>* vec = (vector<client_attr>*)w->p;
    int max = w->max;
    int defaultTimeOut = 10;
    while(1){
        for(int i = 0;i < vec->size();i++){
            if((clock() - vec->at(i).time)/CLOCKS_PER_SEC > (defaultTimeOut - (vec->size()/max)*3)){
                close(vec->at(i).socket_fd);
                cout<<"timeout for : "<< vec->at(i).socket_fd<<endl;
                vec->erase(vec->begin() + i);
            }
        }
    }

}
```

Simple Client:

Pseudocode:

Create a TCP connection with the server
while more operations exist **do**
    Send next requests to the server
    Receives data from the server (in case of GET) or sends data (in case of POST)
end while
Close the connection

Code organization:

- Class client.h/cpp:

```
class client {
public:
    bool connectToServer(string hostname,int port);
    vector<request> parseCommandFile(string file_name);
    void closeConnection();
    bool processCommand(request req,int index);
    void sendRequests(string file_name);
private:
    request commandParser(string command);
    string handleGET(string request);
    string handlePOST(string request);
    int server_socket;

};
```