

- 0 Notes
- 1 Tasks for an HTTP client
- 2 Code explanation
 - Initialization
 - Parsing URLs
 - create a socket and connect to the server
 - send request
 - receive packet
 - postprocessor
 - header extractor
 - second part of the postprocessor
- Test program

0 Notes

1. I tried to modify the code provided by the professor, but I modified too much, and I am very interested in this topic, so I decided to rewrite almost everything and enclose them as a library for later use
2. The code is designed to be able to run on both Windows and UNIX/LINUX systems because when coding, the Winsock and Berkely socket interface are both considered. However, since the programming is done on windows, it hasn't been tried on UNIX/LINUX. So the portability can not be 100% guaranteed, but I will try.
3. One library called "str.h" is from the lecture "Advanced programming" because I have written too much code for processing character strings in C language when I am learning. So this time, I want to do it once and for all. "str.h" is a library for dynamically managing the character string. Since the warmup assignment is not mandatory, the development process is not in a hurry.

1 Tasks for an HTTP client

By analyzing the professor's code and doing some research (also studying the source code of wget), I found that the tasks for an HTTP client to download a file remains the same. 0. Initialization (only for Winsock) 1. Parsing a URL 2. create a socket and connect to the server 3. Fabricate a request and send it to the server (normally, it is GET) 4. Receive response 5. Extract the Body from the response based on the Header, this can be done via postprocessor real-time process (process the data and receive the packets at the same time, specifically for chunked data) 6. Clean up the socket, only for Winsock

```
void ALLinitialization();           // Initialization
void send_request(SOCKET _socket, url _url); // Fabricate and send request
SOCKET connect2server(url _url);    // create socket and connect to server
void ALLcleanup(datapack data, SOCKET _socket); // call this function to do all cleaning work
datapack rcv_response(SOCKET _socket); // Receive response from the server
void headerExtractor(datapack* data); // Extract header info
void postProcessor(datapack* data, FILE* downloadpath); // Extract body info

void perr_exit(const char* msg, int ret_code); // Error Handling
```

2 Code explanation

For each step shown in section 1, I enveloped functions to realize them with the hope that I can maintain them and use them conveniently in the future.

Initialization

This step is done by function ALLinitialization(); but the initialization is specifically for Microsoft Visual C compiler.

```
void ALLinitialization()
{
    // for windows, initialize the socket
#ifdef _WIN32
    /*
     * When using Berkeley sockets, no special initialization is needed, and the socket API is
     * always ready to use.
     * In the main() function, we call WSStartup() on Windows to initialize Winsock. The
     * MAKEWORD macro allows us to request Winsock version 2.2. If our program is unable to
     * initialize Winsock, it prints an error message and aborts.
     */
    WSADATA wsaData; //The WSADATA structure will be filled in by WSStartup() with details about the Windows So
    if (WSStartup(MAKEWORD(2, 2), &wsaData))
        perr_exit("Failed to initialize Winsock.", GETSOCKETERRNO());
#endif
    return;
}
```

Parsing URLs

This step is done by function URL_parse_url(str_p url_origin); since processing URL is another separated library in wget, so I also separate this function into an isolated source file. The parsed URL is stored in a struct. To parse a url, the function will try to find "://" first to separate the protocol and the rest part. If "://" is not found, then the protocol will be set to HTTP as the default value. Then the function will try to find : for the port, and / for the site, and # for the end of parsing.

```
typedef struct
{
    str* url_origin;
    str* protocol;
    str* host;
    str* port;
    str* path;
} url;

url parse_url(str_p url_origin);
```

create a socket and connect to the server

This step is done by the function connect2server. This function request the URL struct as the input, and if the connection is constructed, the socket will be returned. To switch the IPV4 and IPV6 address conveniently in the later, getaddrinfo is used to configure the server info. And to set the time out, the function setsockopt is used. Users can use the Macro _MAX_WAITING_TIME_PER_PACKET to set the timeout of receiving one packet. Since the rcv() function is blocking, it is necessary to set the time out.

```

SOCKET connect2server(url _url) {
    //Configure remote address
    struct addrinfo hints;
    memset(&hints, 0, sizeof(hints));
    hints.ai_socktype = SOCK_STREAM;    //TCP
    struct addrinfo* serverInfo;
    if (getaddrinfo(_url.host->buffer, _url.port->buffer, &hints, &serverInfo))
        perror_exit("Failed configuring info with getaddrinfo().", GETSOCKETERRNO());

    char address_buffer[100];
    char protocol_buffer[100];
    getnameinfo(serverInfo->ai_addr, serverInfo->ai_addrlen,
        address_buffer, sizeof(address_buffer),
        protocol_buffer, sizeof(protocol_buffer),
        NI_NUMERICHOST);

#ifdef _WIN32
    // set the time out
    DWORD timeout = _MAX_WAITING_TIME_PER_PACKET * 1000;
    setsockopt(socket, SOL_SOCKET, SO_RCVTIMEO, (const char*)&timeout, sizeof timeout);
#endif
}

```

send request

A request is fabricated and sent by the function send_request. So far, this function only supports sending the easiest request with the command GET.

```

void send_request(SOCKET _socket, url _url) {
    char buffer[_MAX_SEND_BUFFER_SIZE];

    sprintf_s(buffer, _MAX_SEND_BUFFER_SIZE, "GET /%s HTTP/1.1\r\n", _url.path->buffer);
    sprintf_s(buffer + strlen(buffer), _MAX_SEND_BUFFER_SIZE - strlen(buffer), "Host: %s:%s\r\n", _url.host->buffer,
    sprintf_s(buffer + strlen(buffer), _MAX_SEND_BUFFER_SIZE - strlen(buffer), "Connection: close\r\n");    //or keep
    sprintf_s(buffer + strlen(buffer), _MAX_SEND_BUFFER_SIZE - strlen(buffer), "User-Agent: http client beta 1.0\r\n");
    sprintf_s(buffer + strlen(buffer), _MAX_SEND_BUFFER_SIZE - strlen(buffer), "\r\n"); // end of header

    send(_socket, buffer, strlen(buffer), 0);
    printf("Sent:\n%s", buffer);
}

```

receive packet

In this program, I use postprocessing to deal with the body and Header of the HTTP response, which means all the response packets will be received and stored in a temporary file. After the receiving part is finished, the function postProcessor(&data, saveFP); will do the post-process the extract the body. For receiving part, it is done by function recv_response(SOCKET _socket); this function requires the socket as the input and it will return a struct called datapack. In this struct, a file pointer fp is used for operating the temporary file in which all the packet data is stored. This struct also provides information such as headersize, bodysize, headerposition and bodyposition for file handling, and it also has an important bool value "ifchunked" which is important for postprocessing.

postprocessor

Postprocessing, including extract information from the header and extracting file from body, is done by function postProcessor(datapack* data, FILE* downloadpath). This function takes the datapack struct as the input, and it also takes a file pointer for saving the extracted body.

header extractor

The function headerExtractor reads in the temporary file line by line, trying to find the blank line, which is the indicator of the separation of the Header from the body. It also tries to read in the first line, get the HTTP protocol version and the state code. Then the header extractor will also try to find two headers, Transfer-Encoding and Content-Length. If the header Content-Length is found, then the data in the body is not truncated, so the postprocessor will simply save the body without further process. If the transfer encoding: chunked is detected, then the headerextractor will set the ifChunked flag for the postprocessor.

```

we need to determine whether the HTTP server
is using Content-Length or Transfer-Encoding: chunked to indicate body length. If it
doesn't send either, then we assume that the entire HTTP body has been received once the
connection is closed.
*/
// situation: if find content-length - must not be chunked
// then find the encoding - then judge if it is chunked

movingPointer = strstr(tmpArray->buffer, "Transfer-Encoding: ");
movingPointer2 = strstr(tmpArray->buffer, "Content-Length: ");
if (movingPointer2)
{
    //if found content-length, then must not be chunked
    data->contentLength = strtol(movingPointer2 + 16, NULL, 10);
    data->ifChunked = FALSE;
}
else
{
    //if not found content-length
    if (movingPointer)
    {
        //if not found content - length but found encoding
        // make judgement
        if (strstr(movingPointer + 19, "chunked"))
            data->ifChunked = TRUE;
        else //encoding is other type
            data->ifChunked = FALSE;
    }
    else
    {
        //not found anything, assume it is to be not chunked, do nothing since we have already set the default value for chunk
    }
}
}

```

second part of the postprocessor

The second part of the postprocessor is used to extract the body from the raw data. it first checks if the ifChunked flag is set or not. If the flag is set, then the received data is chunked. It will read the size of the fragment and move the file pointer to process the data.

```
if (data->ifChunked)
{
    //if it is chunked, then try to read the 16based data and do modification
    str_p tmpArray = zero_str(_MAX_RECV_BUFFER_SIZE);
    long chunkSize;

    fseek(data->fp, data->bodyPosition, SEEK_SET); //set the cursor to the correct position
    while (fgets(tmpArray->buffer, tmpArray->size, data->fp))
    {
        chunkSize = strtol(tmpArray->buffer, NULL, 16); // read in the size of the chunk
        if (!chunkSize) //when 0, then finish
            break;

        {
            free_str(tmpArray);
            tmpArray = zero_str(chunkSize);
        }

        fread_s(tmpArray->buffer, tmpArray->size, sizeof(char), chunkSize, data->fp);
        fwrite(tmpArray->buffer, sizeof(char), chunkSize, downloadpath);
        fseek(data->fp, 2, SEEK_CUR); //+2 over jump the link break
    }
}
```

Test program

In the Macro, there is a _DEBUG_MODE switch, by opening the switch, there will be more output for printing.

```
#define _DEBUG_MODE //open debug mode for printing more information
#define _MAX_SEND_BUFFER_SIZE 2048
#define _MAX_RECV_BUFFER_SIZE 2000
#define _MAX_WAITING_TIME_TOTAL -1 // set it to be -1 for ultimate waiting time
#define _MAX_WAITING_TIME_PER_PACKET 20
```

For this test, I choose a picture of Prof. Dr. Westermann on his HTTP webpage :) And the download path is also specified

```
//Parsing URL
str_p url_input = new_str("http://www.home.hs-karlsruhe.de/~weth0002/images/TWatVGUsmall.jpg");
url_url = parse_url(url_input);
free_str(url_input); // url_input needs to be freed after use
```

Based on the Parsed URL, the name of the file will be set automatically by moving the pointer to point the last '/', and the remaining part will be set as the name of the file.

```
str_p name = cpy_str(url.path);
char* movingP, *savedPosition;
savedPosition = movingP = name->buffer;
while (*movingP != '\0')
{
    if (*movingP == '/')
        savedPosition = movingP + 1;
    movingP++;
}

str_p filePath = new_str("D:/"); // change the disk here
str_p fullName = append_str(filePath, savedPosition);
free_str(name);

FILE* saveFP;
fopen_s(&saveFP, fullName->buffer, "wb");
if (!saveFP)
    perr_exit("Assignment: Error opening file\n", -1);
```

I am not programming an user interface, because I want it to be a general library.
































Then, by running the program, we can see the result.

URL: http://www.home.hs-karlsruhe.de/~weth0002/images/TWatVGUsmall.jpg
Protocol: http
Host: www.home.hs-karlsruhe.de
Port: 80
Path: ~weth0002/images/TWatVGUsmall.jpg
Connected to 193.196.64.24 with protocol http
Sent:
GET /~weth0002/images/TWatVGUsmall.jpg HTTP/1.1
Host: www.home.hs-karlsruhe.de:80
Connection: close
User-Agent: http client beta 1.0

One time received:30000 bytes, spent 0 seconds, have received 30000 bytes in total, consumed 0 seconds so far
One time received:3120 bytes, spent 0 seconds, have received 33120 bytes in total, consumed 0 seconds so far
One time received:30000 bytes, spent 0 seconds, have received 63120 bytes in total, consumed 0 seconds so far
One time received:4500 bytes, spent 0 seconds, have received 67620 bytes in total, consumed 0 seconds so far
One time received:30000 bytes, spent 0 seconds, have received 97620 bytes in total, consumed 0 seconds so far
One time received:360 bytes, spent 0 seconds, have received 97980 bytes in total, consumed 0 seconds so far
One time received:30000 bytes, spent 0 seconds, have received 127980 bytes in total, consumed 0 seconds so far
One time received:5880 bytes, spent 0 seconds, have received 133860 bytes in total, consumed 0 seconds so far
One time received:30000 bytes, spent 0 seconds, have received 163860 bytes in total, consumed 0 seconds so far
One time received:1740 bytes, spent 0 seconds, have received 165600 bytes in total, consumed 0 seconds so far
One time received:30000 bytes, spent 0 seconds, have received 195600 bytes in total, consumed 0 seconds so far
One time received:3120 bytes, spent 0 seconds, have received 198720 bytes in total, consumed 0 seconds so far
One time received:30000 bytes, spent 0 seconds, have received 228720 bytes in total, consumed 0 seconds so far
One time received:4500 bytes, spent 0 seconds, have received 233220 bytes in total, consumed 0 seconds so far
One time received:30000 bytes, spent 0 seconds, have received 263220 bytes in total, consumed 0 seconds so far
One time received:7260 bytes, spent 0 seconds, have received 270480 bytes in total, consumed 0 seconds so far
One time received:30000 bytes, spent 0 seconds, have received 300480 bytes in total, consumed 0 seconds so far
One time received:360 bytes, spent 0 seconds, have received 300840 bytes in total, consumed 0 seconds so far
One time received:30000 bytes, spent 0 seconds, have received 330840 bytes in total, consumed 0 seconds so far
One time received:14160 bytes, spent 0 seconds, have received 345000 bytes in total, consumed 0 seconds so far
One time received:30000 bytes, spent 0 seconds, have received 375000 bytes in total, consumed 0 seconds so far
One time received:8640 bytes, spent 0 seconds, have received 383640 bytes in total, consumed 0 seconds so far
One time received:30000 bytes, spent 0 seconds, have received 413640 bytes in total, consumed 0 seconds so far
One time received:12780 bytes, spent 0 seconds, have received 426420 bytes in total, consumed 0 seconds so far
One time received:30000 bytes, spent 0 seconds, have received 456420 bytes in total, consumed 0 seconds so far
One time received:19680 bytes, spent 0 seconds, have received 476100 bytes in total, consumed 0 seconds so far
One time received:30000 bytes, spent 0 seconds, have received 506100 bytes in total, consumed 0 seconds so far
One time received:360 bytes, spent 0 seconds, have received 506460 bytes in total, consumed 0 seconds so far
One time received:30000 bytes, spent 0 seconds, have received 536460 bytes in total, consumed 0 seconds so far
One time received:7260 bytes, spent 0 seconds, have received 543720 bytes in total, consumed 0 seconds so far
One time received:30000 bytes, spent 0 seconds, have received 573720 bytes in total, consumed 0 seconds so far
One time received:7260 bytes, spent 0 seconds, have received 580980 bytes in total, consumed 0 seconds so far
One time received:30000 bytes, spent 0 seconds, have received 610980 bytes in total, consumed 0 seconds so far
One time received:14160 bytes, spent 0 seconds, have received 625140 bytes in total, consumed 0 seconds so far
One time received:2760 bytes, spent 0 seconds, have received 627900 bytes in total, consumed 0 seconds so far
One time received:30000 bytes, spent 0 seconds, have received 657900 bytes in total, consumed 0 seconds so far
One time received:1740 bytes, spent 0 seconds, have received 659640 bytes in total, consumed 0 seconds so far
One time received:30000 bytes, spent 0 seconds, have received 689640 bytes in total, consumed 0 seconds so far
One time received:27960 bytes, spent 0 seconds, have received 717600 bytes in total, consumed 0 seconds so far
One time received:22235 bytes, spent 0 seconds, have received 739835 bytes in total, consumed 0 seconds so far
Connection closed by remote.
Have finished receiving, total received size:739835, consumed 0 seconds in total

And a file is also saved

➤ This PC > DATA (D:)

Name	Date modified	Type	Size
	2022.1.22 1:21		
	2022.1.24 7:51	File folder	
	2022.4.5 7:00	File folder	
	2022.1.21 13:00	File folder	
	2022.1.16 13:35	File folder	
	2022.1.18 18:40	File folder	
	2022.4.5 10:00	File folder	
	2022.4.5 10:00	File folder	
	2022.4.5 10:00	File folder	
	2022.4.5 10:00	File folder	
	2022.4.5 10:00	File folder	
	2022.4.5 10:00	File folder	
	2022.4.5 10:00	File folder	
	2022.4.5 10:00	File folder	
	2022.4.5 10:00	File folder	
	2022.4.5 10:00	File folder	
	2022.4.5 10:00	File folder	
	2022.4.5 10:00	File folder	
	2022.4.5 10:00	File folder	
	2022.4.5 10:00	File folder	
	2022.4.5 10:00	File folder	
	2022.4.5 10:00	File folder	
	2022.4.5 10:00	File folder	
	2022.4.5 10:00	File folder	
	2022.4.5 10:00	File folder	
	2022.4.5 10:00	File folder	
	2022.4.5 10:00	File folder	
	2022.4.5 10:00	File folder	
	2022.4.5 10:00	File folder	
	2022.4.5 10:00	File folder	
	2022.4.6 0:49	JPG File	723 KB