Lab Session 1: Computer Networks

Note: This assignment has both a theoretical and a practical part! The exercises for the theoretical part can be found under the practical assignment.

Practical Assignment: File Transfer (7 points)

Introduction

Throughout this assignment you will be implementing a small client which is able to upload and download files from the server. All submissions for this assignment will on Themis and will be implemented in C.

Input

The input for the client will be read from the standard input. On the first line, you will be given an IP and a port to connect to. After that, on each subsequent line you will be given an operation to do. There are three possible operations, download (D), upload (U), and close (C). The last operation in the file will always be a close operation. Download and upload operations will be followed by a file name on the same line telling you what file the operation is happening on.

Example Input

```
127.0.0.1 25565
D download_example.txt
C
```

Output

After each request sent by the client (except for the close request) the server will send a small response letting the client know what it needs to do next. This response can either be marked as successful or unsuccessful depending on the situation. In the case of a download request a successful response will be followed by the byte content of the file, and an unsuccessful response represents the fact that the requested file does not exist on the server. In the case of an upload, the server will again, send the same response, with it being successful meaning the server will now expect the bytes for the file, and unsuccessful indicating that there already exists a file with this name on the server. Both the server and the client will write the contents of the file to the standard out once the transfer is completed. Remember to add a newline when you print the contents of the file!

Note: Make sure that after you are done downloading a file you also write the contents to a file on disk!

Successful upload output example

```
File upload_example.txt uploaded.
```

Unsuccessful upload output example

```
File upload_example.txt already exists on server.
```

Successful download output example

```
File download_example.txt downloaded.
Hello this is the data in the file!
```

Unsuccessful download output example

```
File inexistent_download_example.txt does not exist on server.
```

Sockets

In order to solve this assignment we will be using the TCP/IP protocol. Here is an example of connecting to a server.

```c
char ip[16];
short port;

int sockFileDescriptor = socket(AF_INET, SOCK_STREAM, 0);
if (sockFileDescriptor == -1) {
    printf("Error creating socket\n");
    return 1;
}

sockaddr_in serverAddress;
serverAddress.sin_family = AF_INET;
serverAddress.sin_port = port;
serverAddress.sin_addr.s_addr = inet_addr(ip);

int connectResult = connect(sockFileDescriptor, (struct sockaddr*)&
    serverAddress, sizeof(serverAddress));
if (connectResult == -1) {
    printf("Error connecting to server\n");
    return 1;
}
```

Working with Files

Unlike previous assignments from other courses, in this assignment you will need to be able to work with files. Here are a couple useful things you might need while working with files

Calculating File Size

```c
int findSize(char* file_name) {
    FILE* fp = fopen(file_name, "r");
    if (fp == NULL) {
        printf("File Not Found!\n");
        return -1;
    }
    fseek(fp, 0L, SEEK_END);
    long int res = ftell(fp);
    fclose(fp);
    return res;
}
```

Reading and Writing to a File

```c
FILE *in_file  = fopen("name_of_file", "r"); // read only
FILE *out_file = fopen("name_of_file", "w"); // write only

char buffer[1024];
fread(buffer, sizeof(char), byte_count, out_file); // Reads up to
    byte_count characters
fwrite(buffer, sizeof(char), byte_count, out_file) // Prints
    byte_count chars
```

Sending and reading structs in C

In order to allow the client and the server to talk to each other, we need some sort of structured data to be passed around. Luckily this is done quite easily in C, as we can quite simply send the memory behind a struct, and C will interpret it correctly on the other end.

Here is an example we can use for sending and receiving a struct in C

```c
struct MyDataStructure {
    int age;
    char name[64];
};

MyDataStructure sending = {20, "John"};
// Sending the struct over the network
send(socketDescriptor, &sending, sizeof(MyDataStructure), 0);

// Reading the struct over the network
MyDataStructure response = {0};
recv(sockFileDescriptor, &response, sizeof(MyDataStructure), 0);
```

Protocol

The client you will implement needs to follow a specific protocol in order to be able to communicate with the server. You will be given a template on Themis which contains the relevant structs, so there's no need to copy and paste them over.

The client and the server are able to continuously talk to each other, each transfer will always be followed up by a transfer response, and, depending on that, either the client or the server will be expecting some bytes to be sent.

The server response will contain "Y" (yes) or "N" (no) depending on whether it is willing to accept the transfer. It is also guaranteed that all files will be under 1KB (1024 bytes), so reading the entire buffer in one call is fine.

It should also be noted that in the case of an upload, the fileSize contained in the response will be 0 (as it's unused). Similarly, when sending a download request, you should set the fileSize to 0 (as there is no way the client would know what filesize the file has).

```
struct Transfer {
    char request;
    char filename[64];
    int fileSize;
};

struct TransferResponse {
    char response;
    int fileSize;
};
```

Here's a full example of communication for every possible scenario:

Successful upload protocol example

```
(Client -> Server) Transfer {'U', Byte count}
(Server -> Client) TransferResponse {'Y'}
(Client -> Server) File content, byte by byte
```

Unsuccessful upload protocol example

```
(Client -> Server) Transfer {'U', Byte count}
(Server -> Client) TransferResponse {'N'}
```

Successful download protocol example

```
(Client -> Server) Transfer {'D'}
(Server -> Client) TransferResponse {'Y', Byte count}
(Server -> Client) File content, byte by byte
```

Unsuccessful download protocol example

```
(Client -> Server) Transfer {'D'}
(Server -> Client) TransferResponse {'N'}
```

Testing locally

You will be provided on Themis with the binary for the server. We recommend placing the server and the client in separate directories. Both the server and the client will need to operate with files within the directory they are placed in. Running the server is quite easy, and it has a built in port of 25565.

To run it, open a separate terminal in the directory of the server and run.

```
./ref_server
```

And you should get the following output

```
Server listening on 127.0.0.1:25565
```

Once that's done you will be able to run your client and connect to the server.

Theoretical Assignment (3 points)

Exercise 1 (1 point)

If an ICMP Time Exceeded message is sent back in response to an ICMP Echo Request packet, calculate the time taken for the response to reach the original sender, assuming a network delay of 5 milliseconds per router hop. Assume the ICMP Echo Request packet sent from a host with a time-to-live (TTL) value of 64.

Exercise 2 (1 point)

In a local area network (LAN) segment, the ARP broadcast domain has 50 devices. Each device sends out an ARP request packet every 30 seconds to refresh its ARP cache. Calculate the total number of ARP packets transmitted in the LAN segment per minute.

Exercise 3 (1 point)

Explain the concept of IPv6 Anycast addressing and discuss its applications in load balancing and high availability scenarios.