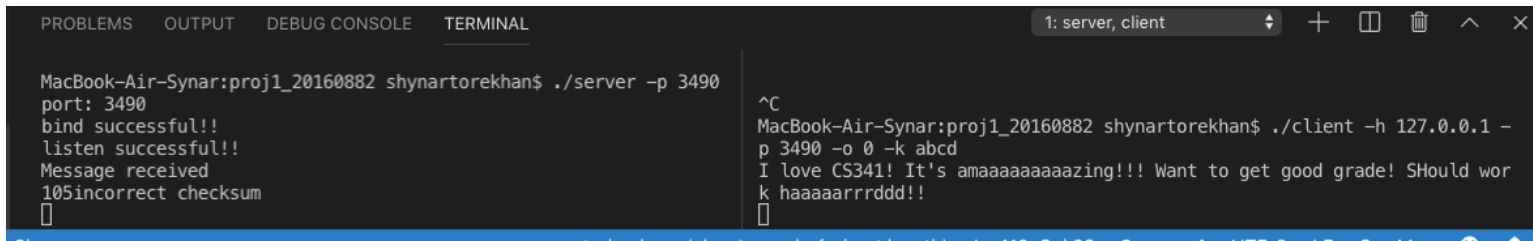


CS341: Project 1 report Torekhan Shynar 20160882

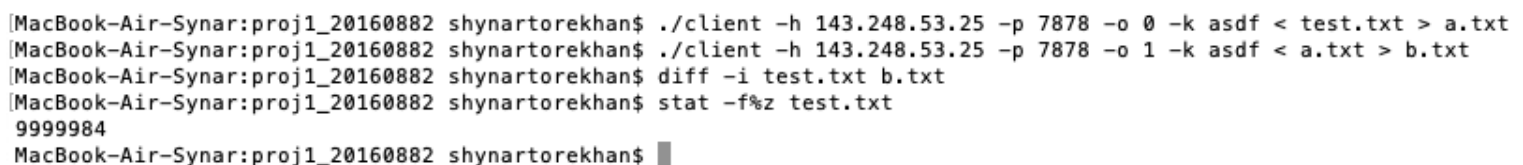
Instructions to compile programmes and self-test results.

Programs (client and server) can be compiled by calling “make all”. The programmes were tested by the test vectors and test servers provided. Some corner cases were tested manually.



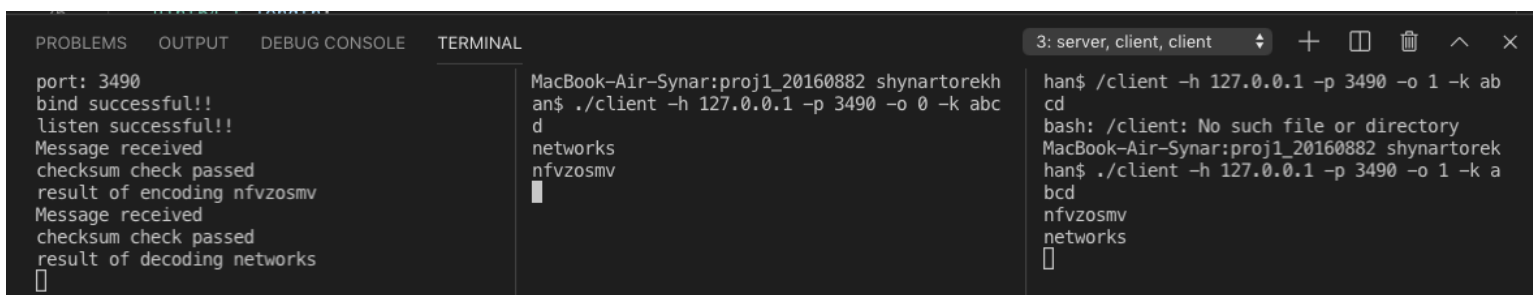
```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: server, client
MacBook-Air-Synar:proj1_20160882 shynartorekhan$ ./server -p 3490
port: 3490
bind successful!!
listen successful!!
Message received
105incorrect checksum
^C
MacBook-Air-Synar:proj1_20160882 shynartorekhan$ ./client -h 127.0.0.1 -p 3490 -o 0 -k abcd
I love CS341! It's amaaaaaaaazing!!! Want to get good grade! SHould work haaaaarrddd!!
```

As shown on the picture above, the server was tested manually. In this case the client sent message with the wrong checksum, and server detected the wrong checksum (as you can see “incorrect checksum” is printed), so client did not receive any response from the server. Provided test servers by teaching assistants also don’t respond when client sends message with the wrong checksum. So, we can conclude that server handles wrong checksums from the client as expected.



```
MacBook-Air-Synar:proj1_20160882 shynartorekhan$ ./client -h 143.248.53.25 -p 7878 -o 0 -k asdf < test.txt > a.txt
MacBook-Air-Synar:proj1_20160882 shynartorekhan$ ./client -h 143.248.53.25 -p 7878 -o 1 -k asdf < a.txt > b.txt
MacBook-Air-Synar:proj1_20160882 shynartorekhan$ diff -i test.txt b.txt
MacBook-Air-Synar:proj1_20160882 shynartorekhan$ stat -f%z test.txt
9999984
MacBook-Air-Synar:proj1_20160882 shynartorekhan$
```

In the screenshot above one can observe that client successfully handles big messages with the size 10MB. Both test.txt and b.txt are identical. We can see that the size of test.txt is 9999984, which is 10MB - 16 (the size of op + checksum + keyword + length fields of message), thus the message which is handled by client is 10MB.



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 3: server, client, client
port: 3490
bind successful!!
listen successful!!
Message received
checksum check passed
result of encoding nfvzosmv
Message received
checksum check passed
result of decoding networks
MacBook-Air-Synar:proj1_20160882 shynartorekhan$ ./client -h 127.0.0.1 -p 3490 -o 0 -k abcd
networks
nfvzosmv
han$ ./client -h 127.0.0.1 -p 3490 -o 1 -k abcd
bash: ./client: No such file or directory
MacBook-Air-Synar:proj1_20160882 shynartorekhan$ ./client -h 127.0.0.1 -p 3490 -o 1 -k abcd
nfvzosmv
networks
```

In the screenshot above one can observe that server successfully handles connections from multiple clients (by using fork()). The leftmost part of the terminal is the server, and it can be observed how server handles operations of both clients. The middle client sends encoding requests to the server, whereas the rightmost server sends decoding requests to the server. Both of the clients are served by the server simultaneously.

In the screenshot below, you can observe that client detects the messages from the server with the incorrect checksum. It was tested by connecting to the misbehaving server provided by the teaching assistants. As it was unspecified in the slides how should specifically the client handle the case when server sends message with incorrect checksum, my implementation of client let’s user know

```

[root@CS341-14:~/CN_proj1# ./client -h 143.248.53.25 -p 7879 -o 0 -k cake
[networks
incorrect checksum
pedaqrw
[I love cs341
incorrect checksum
k lyzg cc341
^C
[root@CS341-14:~/CN_proj1# ./client -h 143.248.53.25 -p 7878 -o 0 -k cake
[networks
pedaqrw
[I love cs341
k lyzg cc341
^C
root@CS341-14:~/CN_proj1# █

```

that incorrect checksum was detected by printing out “incorrect checksum” and the payload. Then client connects to the “good” server on the port 7878, and it can be observed that when message with correct checksum is sent to the client, client doesn’t complain as in the case with the

```

MacBook-Air-Synar:proj1_20160882 shynartorekhan$ ./client -h 143.248.53.25 -p 7878 -o 0 -k asdf < burrito.txt > a.txt
MacBook-Air-Synar:proj1_20160882 shynartorekhan$ ./client -h 143.248.53.25 -p 7878 -o 1 -k asdf < a.txt > b.txt
MacBook-Air-Synar:proj1_20160882 shynartorekhan$ diff -i burrito.txt b.txt
MacBook-Air-Synar:proj1_20160882 shynartorekhan$ ./client -h 143.248.53.25 -p 7878 -o 0 -k asdf < crossbow.txt > a.txt
MacBook-Air-Synar:proj1_20160882 shynartorekhan$ ./client -h 143.248.53.25 -p 7878 -o 1 -k asdf < a.txt > b.txt
MacBook-Air-Synar:proj1_20160882 shynartorekhan$ diff -i crossbow.txt b.txt
MacBook-Air-Synar:proj1_20160882 shynartorekhan$ ./client -h 143.248.53.25 -p 7878 -o 0 -k asdf < sample.txt > a.txt
MacBook-Air-Synar:proj1_20160882 shynartorekhan$ ./client -h 143.248.53.25 -p 7878 -o 1 -k asdf < a.txt > b.txt
MacBook-Air-Synar:proj1_20160882 shynartorekhan$ diff -i sample.txt b.txt
MacBook-Air-Synar:proj1_20160882 shynartorekhan$ ./client -h 127.0.0.1 -p 3490 -o 0 -k asdf < burrito.txt > a.txt
MacBook-Air-Synar:proj1_20160882 shynartorekhan$ ./client -h 127.0.0.1 -p 3490 -o 1 -k asdf < a.txt > b.txt
MacBook-Air-Synar:proj1_20160882 shynartorekhan$ diff -i burrito.txt b.txt
MacBook-Air-Synar:proj1_20160882 shynartorekhan$ ./client -h 127.0.0.1 -p 3490 -o 0 -k asdf < crossbow.txt > a.txt
MacBook-Air-Synar:proj1_20160882 shynartorekhan$ ./client -h 127.0.0.1 -p 3490 -o 1 -k asdf < a.txt > b.txt
MacBook-Air-Synar:proj1_20160882 shynartorekhan$ diff -i crossbow.txt b.txt
MacBook-Air-Synar:proj1_20160882 shynartorekhan$ ./client -h 127.0.0.1 -p 3490 -o 0 -k asdf < sample.txt > a.txt
MacBook-Air-Synar:proj1_20160882 shynartorekhan$ ./client -h 127.0.0.1 -p 3490 -o 1 -k asdf < a.txt > b.txt
MacBook-Air-Synar:proj1_20160882 shynartorekhan$ diff -i sample.txt b.txt
MacBook-Air-Synar:proj1_20160882 shynartorekhan$ ./client -h 127.0.0.1 -p 3490 -o 0 -k asdf < crossbow.txt > a.txt
MacBook-Air-Synar:proj1_20160882 shynartorekhan$ ./client -h 143.248.53.25 -p 7878 -o 0 -k asdf < crossbow.txt > b.txt
MacBook-Air-Synar:proj1_20160882 shynartorekhan$ diff a.txt b.txt
MacBook-Air-Synar:proj1_20160882 shynartorekhan$ ./client -h 127.0.0.1 -p 3490 -o 0 -k asdf < burrito.txt > a.txt
MacBook-Air-Synar:proj1_20160882 shynartorekhan$ ./client -h 127.0.0.1 -p 3490 -o 0 -k asdf < burrito.txt > a.txt
MacBook-Air-Synar:proj1_20160882 shynartorekhan$ ./client -h 143.248.53.25 -p 7878 -o 0 -k asdf < burrito.txt > b.txt
MacBook-Air-Synar:proj1_20160882 shynartorekhan$ diff a.txt b.txt
MacBook-Air-Synar:proj1_20160882 shynartorekhan$ ./client -h 127.0.0.1 -p 3490 -o 0 -k asdf < sample.txt > a.txt
MacBook-Air-Synar:proj1_20160882 shynartorekhan$ ./client -h 143.248.53.25 -p 7878 -o 0 -k asdf < sample.txt > b.txt
MacBook-Air-Synar:proj1_20160882 shynartorekhan$ diff a.txt b.txt

```

misbehaving server described above.

The image shown above demonstrates how both client and server were tested on the vectors (sample text files) provided by teaching assistants. This testing method was suggested in the slides and KLMS announcements. So, the first 9 lines check the functionality of client with the “good” server. The following 9 lines check the functionality of decryption and encryption. The last 9 lines check the functionality of the server by comparing it with the behavior of “good” server. As there was no output from “diff” operation, it can be said that implementation of client and server work correctly on the test vectors provided by the teaching faculty.

Explanation of the server and client structures.

The code written in the submitted binaries is thoroughly commented, and the names of methods and variables are designed to make code as self-explanatory and readable as possible (although I am not very experienced, but I tried hard to achieve readability). In this section I am going to provide overall description of the server and client functionalities. For more specific specifications it is recommended to refer to the code provided. Both client and server share lots in common. Both their send and recv functions are wrapped in the way that whole message will be send or recv(received). For example, send_all function will try to send message at once, but if it is not possible, the function will send message chunk by chunk until it is not fully sent. The logic of client goes in the following way: first of all, the arguments are parsed, if arguments are passed in the wrong way message will be printed. Then socket is set_up, the method returns socket file descriptor. getaddrinfo(), socket() and connect() are wrapped inside of setup_socket() function. Then client keeps getting input from StdIn in the while loop. When input is received, the data is wrapped according to the protocol provided(the checksum is calculated, and etc). The send_all() function described before is called. Then we keep recv (receiving) message from the server until the whole message is received. We check the integrity of the checksum. Then the payload is outputted.

For the server, the flow is “similar”. The differences are that in the setup_socket() we have call to the bind() and listen(). Also the code for encryption and decryption is in the server part. Then in the while loop we keep calling accept(). After some client is connected, we receive message, validate wether it obeys the protocol, then we encrypt/decrypt the data, pack the message, and send to the client. The most interesting part in the server was handling multiple connections. It was important to deal with the possible zombie processes after child process quits. For this sigchld handler was written.

In conclusion, I am satisfied to code this assignment, as I was able to learn a lot about socket programming. The bonus part was not implemented.