**Computer Networks**

**Homework#4**

**2015318622 Noh Mirae**

1. (100pts) Consider figure below. Assuming TCP Reno is the protocol experiencing the behavior shown above, answer the following questions.



* 1. Identify the intervals of time when TCP slow start is operating.
     1. 1-6, 23-26
  2. Identify the intervals of time when TCP congestion avoidance is operating.
     1. 6-16, 17-22
  3. After the 16th transmission round, is segment loss detected by a triple duplicate ACK or by a timeout?
     1. 3 dup ACKs
  4. After the 22nd transmission round, is segment loss detected by a triple duplicate ACK or by a timeout?
     1. Time-out
  5. What is the initial value of ssthresh at the first transmission round?
     1. 32
  6. What is the value of ssthresh at the 18th transmission round?
     1. 41/2 = 21
  7. During what transmission round is the 70th segment sent?
     1. 1 + 2 + 4 + 8 + 16 + 32 = 63.

63 + 33 = 96.

64 < 70 < 96, so the 70th segment is sent in **7th transmission round**.

* 1. Assuming a packet loss is detected after the 26th round by the receipt of a triple duplicate ACK, what will be the values of the congestion window size and of ssthresh?
     1. ssthresh: 8/2 = 4

congestion window size: 4 + 3 = 7

* 1. Suppose TCP Tahoe is used (instead of TCP Reno), and assume that triple duplicate ACKs are received at the 16th round. What are the ssthresh and the congestion window size?
     1. ssthresh: 42 / 2 = 21

congestion window size = 1

* 1. Again, suppose TCP Tahoe is used, and there is a timeout event at 22nd round. How many packets have been sent out from 17th round till 22nd round, inclusive?
     1. 24 + 25 + 26 + 27 + 28 + 29 = 159 packets

1. (40pts) Socket Programming Experience: Controlling Drones

* Development Environment

platform: Linux Ubuntu

compiler: gcc compiler

* Folder structure and files



drone\_client.c: c file for client program

drone\_server.c: c file for server program

headerfiles.h: headerfile for client and server program

client: compiled client program

server: compiled server program

moves.txt: text file containing commands

position.txt: text file saving final position of the drone

* How to compile and run

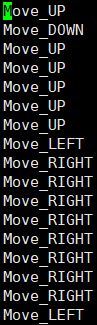
<server>



<client>



test input (moves.txt)

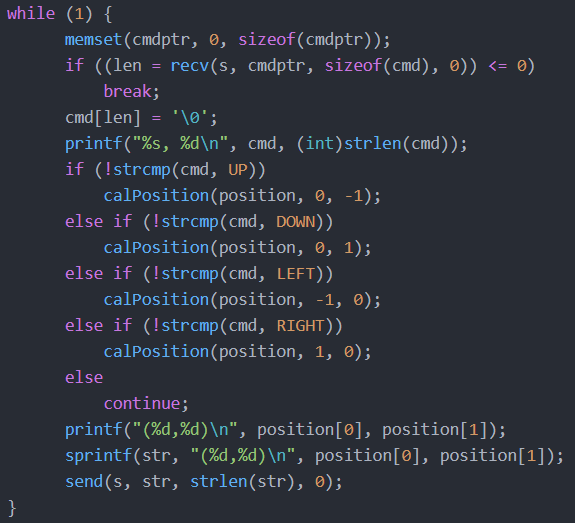


test output (position.txt)

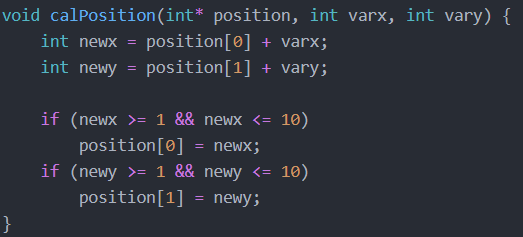


* Important implementation functionalities

<server>

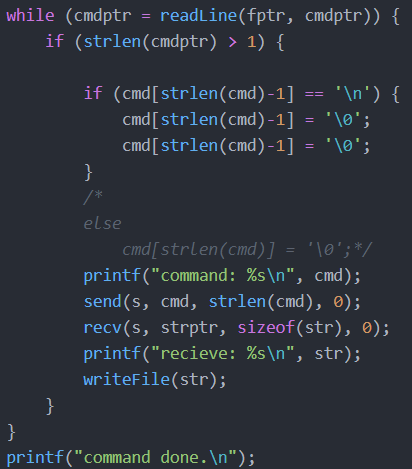


This part of the server code receives each command string, compares it to given commands, calculates position by calling calPosition(), and send back the position.

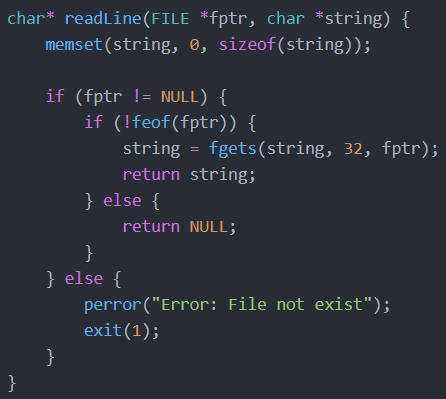


Function calPosition() of the server gets current position and values to move, calculates new position, and save it on the array position[].

<client>



This part of the client reads command lines int moves.txt one by one, sends the line to the server, receives new position, and write it on position.txt.



Function readLine() reads one line of given file pointer and return it. If the file doesn’t exist, it terminates the program.

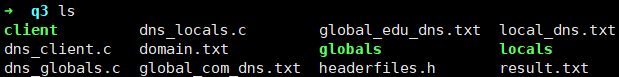
1. (60pts) Socket Programming Experience: Simple DNS simulation

* Development Environment

platform: Linux Ubuntu

compiler: gcc compiler

* Folder structure and files



dns\_client.c: c file for dns client program

dns\_locals.c: c file for dns local server program

dns\_globals.c: c file for dns global server program

client: compiled dns client program

locals: compiled dns local server program

globals: compiled dns global server program

domain.txt: text file which contains the domain name

local\_dns.txt: text file which contains dns addresses and their ip addresses of local

global\_edu\_dns.txt: text file which contains dns addresses which end with “edu” and their ip addresses of global server

global\_com\_dsn.txt: text file which contains dns addresses which end with “com” and their ip addresses of global server

* How to compile and run

<global server>



<local server>



<client>



<test input(domain.txt)>

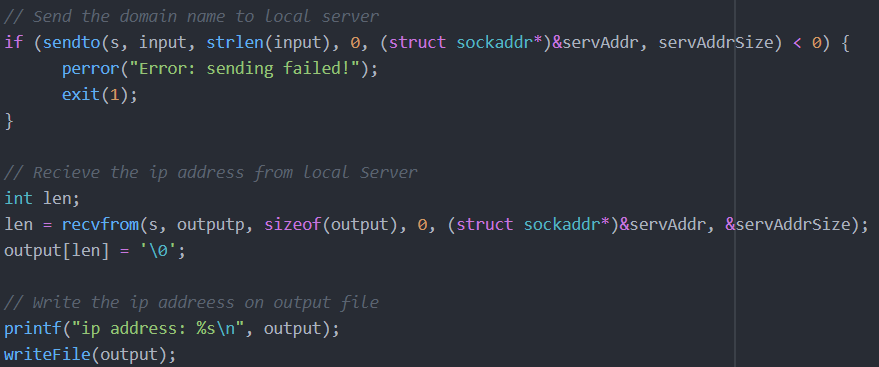


<test output(result.txt)>



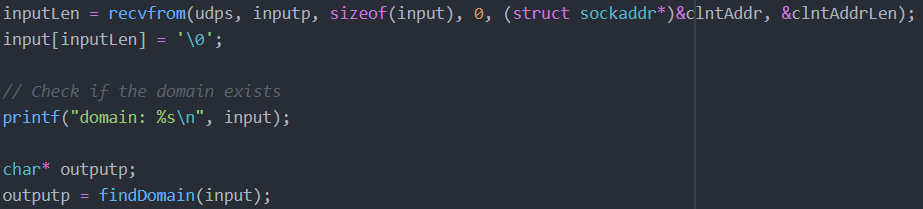
* Important implementation functionalities

<client>

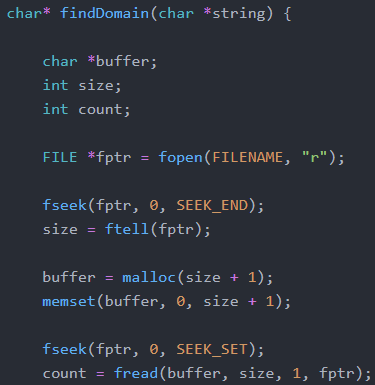
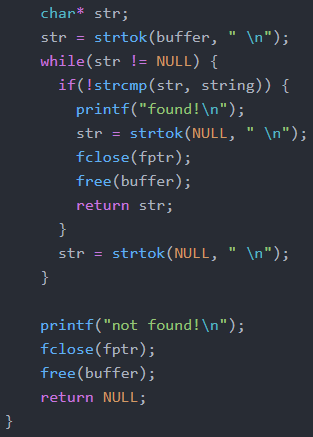


This is main part of the client program. It sends the domain(input) to the local server, receive the ip address of the domain from the local server, and write it on output(result.txt) file.

<local server>



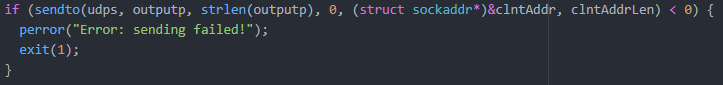
Local server first finds the received domain name from its file(local\_dns.txt). It finds it and save it to the variable outputp by calling the function findDomain(). Below is whole codes of the function, findDomain().

It scans local\_dns.txt and check if it contains the given domain name by using the functions strtok() and strcmp(). If it founds the domain, it returns the ip address which is existing right after the domain.



In the main function, if it cannot find the ip address of received domain name in local\_dns.txt, it connects to the global server by TCP/IP protocol, sends the given domain name to it, and receives the ip address(or “Not found” message) from it.



Lastly, the local server program sends found ip address(from local\_dns.txt or from the global server program) to the client.

<global server program>



Global server program receives the request(domain name string from the client) from the local server. It first founds the string in “global\_edu\_dns”, found in “global\_com\_dns” if formal call returns NULL, and sends the found data to the local server. If the string not exist neither “global\_edu\_dns” nor “global\_com\_dns”, it sends “Not found” to the local server.