Lab Report Networks Lab 1

9/10/13

Group Id: 4

Devan Buggay - dsb0012@auburn.edu

Nathan Plotts - nwp0002@auburn.edu

Steven Whaley - saw0019@auburn.edu

We implemented our ServerUDP.c in C, ClientUDP.py in python, ServerTCP.py in python, and ClientTCP.py in python. All of our code and testing worked properly when tested on tux machines. To compile and execute the code you must first log on to two tux computers with all the files present, for example I'll use tux190 and tux191. On tux190 type "python ServerTCP.py 10014", next on tux191 type "gcc ClientTCP.c" then "./a.out tux190.eng.auburn.edu 10014 op yourmessage" with op replaced by 85 for vLength or 170 for disemvoweling and yourmessage replaced with whatever string you wish to send and perform operations on. No known bugs or problems.

**example running udp test**

gcc ServerUDP.c -o ServerUDP

./ServerUDP 10014

python ClientUDP.py localhost 10014 107 removemyvowels

**example running tcp test**

py ServerTCP.py 10014

gcc ClientTCP.c -o ClientTCP

./ClientTCP localhost 10014 107 removemyvowels

**You will find all relevant source code here:**

ClientUDP.py

-----------------------------------------------------------------------------------

# group 4

# ClientUDP.py

# run with python CliendUDP.py [hostname] [port] [operation] [string]

import sys, socket, struct, time

BUFFER\_SIZE = 100 # set buffer size

def modified\_unpack(fmt, dat): # custom unpacking method dealing with ending strings

non\_str\_len = struct.calcsize(fmt[:-1]) # get len of everything but the string

str\_len = len(dat) - non\_str\_len # get len of string

str\_fmt = "{0}s".format(str\_len) # get new format

new\_fmt = fmt[:-1] + str\_fmt # add it to the old format

return struct.unpack(new\_fmt, dat) # unpack it with new format

hostname = sys.argv[1] #set hostname

port = int(sys.argv[2]) #set port

operation = int(sys.argv[3]) #set operation

message = sys.argv[4] #set message

requestid = 1; #set the requestid to whatever

print "UDP target hostname:", hostname

print "UDP target port:", port

print "requestid:", requestid

print "operation:", operation

print "message:", message

print "Client: setting socket"

sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM) #set up socket

start = time.clock()

msglength = 5 + len(message); #length is 5 + the length of the string

msgbuf = struct.pack("h h B", msglength, 1, operation) + message #pack the message buffer

print "Client: sending data"

sock.sendto(msgbuf, (hostname, port)) #send the data

#wait

data, addr = sock.recvfrom(BUFFER\_SIZE) #recieve data from server

print "Client: response from ", addr

elapsed = (time.clock() - start) #get elapsed time

print "Time elapsed: ", elapsed

if operation == 85: #if operation 55

print "Operation 85: Number of vowels"

data = struct.unpack("h h h", data)

print "tml:", data[0]

print "requestid:", data[1]

print "vowels:", data[2]

else: #else must be operation 170 (otherwise server would've thrown error)

print "Operation 170: Disemvowel"

data = modified\_unpack("hhs", data)

print "tml:", data[0]

print "requestid:", data[1]

print "string:", data[2]

-----------------------------------------------------------------------------------

ServerUDP.c

-----------------------------------------------------------------------------------

// Group 4

// ServerUDP.c

// compile with gcc ServerUDP.c -o ServerUDP

// run with ./ServerUDP [port]

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <errno.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <netdb.h>

#define MAXBUFLEN 100

// get sockaddr, IPv4 or IPv6:

void \*get\_in\_addr(struct sockaddr \*sa)

{

if (sa->sa\_family == AF\_INET) {

return &(((struct sockaddr\_in\*)sa)->sin\_addr);

}

return &(((struct sockaddr\_in6\*)sa)->sin6\_addr);

}

int is\_vowel(char c) { //check for vowels

switch(c)

{

case 'a':

case 'e':

case 'i':

case 'o':

case 'u':

case 'A':

case 'E':

case 'I':

case 'O':

case 'U':

return 1;

default:

return 0;

}

}

void removeChar(char \*str, char c) { //removeChar function to remove characters from a string

char \*src, \*dst;

for (src = dst = str; \*src != '\0'; src++) {

\*dst = \*src;

if (\*dst != c) dst++;

}

\*dst = '\0';

}

int main(int argc, char \*argv[])

{

if(argc != 2) { //check for correct usage

printf("Usage:./server port\n");

exit(0);

}

char\* port = argv[1]; //set the port

//init a bunch of variables

int sockfd, rv, numbytes;

struct addrinfo hints, \*servinfo, \*p;

struct sockaddr\_storage their\_addr;

char buf[MAXBUFLEN], s[INET6\_ADDRSTRLEN];;

socklen\_t addr\_len;

memset(&hints, 0, sizeof hints);

hints.ai\_family = AF\_UNSPEC; // set to AF\_INET to force IPv4

hints.ai\_socktype = SOCK\_DGRAM;

hints.ai\_flags = AI\_PASSIVE; // use my IP

if ((rv = getaddrinfo(NULL, port, &hints, &servinfo)) != 0) {

fprintf(stderr, "getaddrinfo: %s\n", gai\_strerror(rv));

return 1;

}

for(p = servinfo; p != NULL; p = p->ai\_next) { // loop through all the results and bind to the first we can

if ((sockfd = socket(p->ai\_family, p->ai\_socktype, p->ai\_protocol)) == -1) {

perror("Server: socket");

continue;

}

if (bind(sockfd, p->ai\_addr, p->ai\_addrlen) == -1) {

close(sockfd);

perror("Server: bind");

continue;

}

break;

}

if (p == NULL) { //Check if bound

fprintf(stderr, "Server: failed to bind socket\n");

return 2;

}

int client\_length = (int)sizeof(struct sockaddr\_in); //set client length

//infinite loop

while (1)

{

printf("Server: waiting to recvfrom...\n");

char \*s = NULL;

//struct request\_packet \*message = malloc(MAXBUFLEN - sizeof(struct request\_packet) - 1);

uint8\_t \*message = malloc(MAXBUFLEN - 1); //allocate space for the message

//recieve a request

if ((numbytes = recvfrom(sockfd, message, MAXBUFLEN - 1, 0, (struct sockaddr \*)&their\_addr, &client\_length)) == -1) {

perror("recvfrom");

exit(1);

}

printf("Server: recieved\n");

printf("Server: packet is %d bytes long\n", numbytes);

//unpack packet

uint8\_t \*messagep = message; //new char pointer at message address

uint16\_t tml, requestid; //set up vars

uint8\_t operation;

memcpy(&tml,messagep,sizeof(uint16\_t)); //copy first 2 bytes into tml

messagep += sizeof(uint16\_t); //shift over 2 bytes

memcpy(&requestid,messagep,sizeof(uint16\_t)); //copy 2nd 2 bytes into requestid

messagep += sizeof(uint16\_t); //shift over 2 bytes

memcpy(&operation,messagep,sizeof(uint8\_t)); //copy next byte into operation

messagep += sizeof(uint8\_t); //shift over 1 byte

int stringLength = tml - 5; //calculate length of string (tml - 5(2 + 2 + 1 for header))

char \*buffer = malloc(stringLength + 1); //allocate space including space for '\0'

strcpy(buffer, messagep); // copy the rest of the messagep into buffer string

buffer[stringLength] = '\0'; //set '\0'

// Print some info on the packet

printf("(%d|%d|%d|%s)\n", tml, requestid, operation, buffer);

printf("tml: %d\n", tml);

printf("requestid: %d\n", requestid);

printf("operation: %d\n", operation);

printf("string: %s\n", buffer);

// Number of Vowels

if(operation == 85) {

printf("Operation 85: Number of vowels\n");

printf("String: %s\n", buffer);

uint16\_t total = 0;

char c;

char \*bufferp = buffer;

while(c = \*bufferp++) {

if(is\_vowel(c)) {

total++;

}

}

uint16\_t out[3];

out[0] = 6; //TML will always be 6 because message will always contain 3 shorts

out[1] = requestid; //send back requestid

out[2] = total; //set total

printf("Total vowels: %d\n", total);

printf("Server: Sending response to client\n");

sendto(sockfd, out, 6, 0, (struct sockaddr \*)&their\_addr, client\_length); //sends the out array

}

// Disemvowel

else if(operation == 170) {

printf("Operation 107: Disemvowel\n");

printf("String in: %s\n", buffer);

removeChar(buffer, 'a'); //remove all vowels

removeChar(buffer, 'e');

removeChar(buffer, 'i');

removeChar(buffer, 'o');

removeChar(buffer, 'u');

removeChar(buffer, 'A');

removeChar(buffer, 'E');

removeChar(buffer, 'I');

removeChar(buffer, 'O');

removeChar(buffer, 'U');

printf("String out: %s\n", buffer);

//set up returning packet

char \*buf = malloc(4 + strlen(buffer)); //allocate space (4 + length of returning string)

char \*pos = buf; //set up pointer

\*(u\_short\*)pos = 4 + strlen(buffer); //fill in tml

pos += sizeof(u\_short); //shift

\*(u\_short\*)pos = requestid; //fill in requestid

pos += sizeof(u\_short); //shift

strcpy( pos, buffer ); //fill in string

//packet now looks like (h,h,s)

printf("Server: Sending response to client\n");

int bytes = sendto(sockfd, buf, 4 + strlen(buffer), 0, (struct sockaddr \*)&their\_addr, client\_length); //send the packet to the client

printf("Sent %d bytes\n", bytes);

}

// No op

else {

printf("Server: Unknown operation\n");

}

//free everything

free(message);

free(buffer);

//null all the pointers

message = NULL;

messagep = NULL;

buffer = NULL;

}

// Cleanup

freeaddrinfo(servinfo);

close(sockfd);

return 0;

}

-----------------------------------------------------------------------------------

ClientTCP.c

-----------------------------------------------------------------------------------

// Group 4

// ClientTCP.c

// compile with gcc ClientTCP.c -o ClientTCP

// run with ./ClientTCP [hostname] [port] [operation] [string]

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <errno.h>

#include <string.h>

#include <netdb.h>

#include <sys/types.h>

#include <netinet/in.h>

#include <sys/socket.h>

#include <stdint.h>

#include <inttypes.h>

#include <arpa/inet.h>

#include <time.h>

#define MAXDATASIZE 100

// get sockaddr, IPv4 or IPv6:

void \*get\_in\_addr(struct sockaddr \*sa) {

if (sa->sa\_family == AF\_INET) {

return &(((struct sockaddr\_in\*)sa)->sin\_addr);

}

return &(((struct sockaddr\_in6\*)sa)->sin6\_addr);

}

int main(int argc, char \*argv[])

{

//set up some variables

int sockfd, numbytes;

struct addrinfo hints, \*servinfo, \*p;

int rv;

char s[INET6\_ADDRSTRLEN];

if (argc != 5) { // check for correct args

fprintf(stderr,"usage: client [hostname] [port] [operation] [message]\n");

exit(1);

}

memset(&hints, 0, sizeof hints);

hints.ai\_family = AF\_UNSPEC;

hints.ai\_socktype = SOCK\_STREAM;

if ((rv = getaddrinfo(argv[1], argv[2], &hints, &servinfo)) != 0) {

fprintf(stderr, "getaddrinfo: %s\n", gai\_strerror(rv));

return 1;

}

// loop through all the results and connect to the first we can

for(p = servinfo; p != NULL; p = p->ai\_next) {

if ((sockfd = socket(p->ai\_family, p->ai\_socktype,

p->ai\_protocol)) == -1) {

perror("client: socket");

continue;

}

if (connect(sockfd, p->ai\_addr, p->ai\_addrlen) == -1) {

close(sockfd);

perror("client: connect");

continue;

}

break;

}

if (p == NULL) {

fprintf(stderr, "client: failed to connect\n");

return 2;

}

printf("Sending message: %s\n", argv[4]);

// setup variables

uint16\_t tml;

uint16\_t requestid = 1; // requestid can be whatever

uint8\_t operation = atoi(argv[3]); // set operation

inet\_ntop(p->ai\_family, get\_in\_addr((struct sockaddr \*)p->ai\_addr), s, sizeof s);

printf("client: connecting to %s\n", s);

//create packet

uint8\_t \*buf = malloc(5 + strlen(argv[4])); // allocate space for packet

uint8\_t \*pos = buf; //point pointer

\*(uint16\_t\*)pos = 5 + strlen(argv[4]); //set first 2 bytes to total message length (5 + strlen(message))

pos += sizeof(uint16\_t); // move pointer 2 bytes over

\*(uint16\_t\*)pos = requestid; // set next 2 bytes to rid

pos += sizeof(uint16\_t); // move over 2 bytes

\*(uint8\_t\*)pos = operation; // set next bte to operation

pos += sizeof(u\_int8\_t); // move over 1 byte

strcpy( pos, argv[4] ); // copy string into rest of packet

// send packet

write(sockfd, buf, 5 + strlen(argv[4]));

uint8\_t \*in = malloc(MAXDATASIZE);

if ((numbytes = read(sockfd, in, MAXDATASIZE-1)) == -1) {

perror("recv");

exit(1);

}

// disemvowel operation

if (operation == 170) {

uint8\_t \*inp = in; //pointer to packet

uint16\_t tml, rid; // set up variables

//unpack (h,h,s) packet

memcpy(&tml, inp, sizeof(uint16\_t)); //copy first two bytes into tml

inp += sizeof(uint16\_t); //move pointer 2 bytes over

memcpy(&rid, inp, sizeof(uint16\_t)); //copy next two bytes into rid

inp += sizeof(uint16\_t); //move pointer 2 bytes over

char \*inbuf = malloc(tml - 4); //allocate space for string

strcpy(inbuf, inp); //copy rest of packet into string

inbuf[tml - 4] = '\0'; // set null terminator

//print out some info

printf("client: received tml is '%hu'\n",tml);

printf("client: received rid is '%hu'\n",rid);

printf("client: received disemvoweled message '%s'\n",inbuf);

}

// vowel count operation

else {

uint8\_t \*inp = in; // pointer to packet

uint16\_t tml, rid, vowels; //set up variables

// unpack (h,h,h) uint16\_t

memcpy(&tml, inp, sizeof(u\_short)); // copy first 2 bytes into tml

inp += sizeof(uint16\_t); // move pointer 2 bytes over

memcpy(&rid, inp, sizeof(uint16\_t)); // copy next 2 bytes into rid

inp += sizeof(uint16\_t); // move pointer 2 bytes over

memcpy(&vowels, inp, sizeof(uint16\_t)); // copy last two bytes into vowels

printf("client: received tml is '%hu'\n",tml);

printf("client: received rid is '%hu'\n",rid);

printf("client: received vowel length '%hu'\n",vowels);

}

freeaddrinfo(servinfo); // all done with this structure

close(sockfd);

return 0;

}

-----------------------------------------------------------------------------------

ServerTCP.py

-----------------------------------------------------------------------------------

# group 4

# ServerTCP.py

# run with python ServerTCP.py [port]

import socket, sys, struct

port = int(sys.argv[1]) # set port

BUFFER\_SIZE = 100 # set buffer size

def disemvoweling(str): # disemvowel

for char in str:

if char in "aeiouAEIOU":

str = str.replace(char,'')

return str

def vlength(str): # vowel count

count = 0

for char in str:

if char in "aeiouAEIOU":

count = count + 1

return count

def modified\_unpack(fmt, dat): # custom unpacking method dealing with ending strings

non\_str\_len = struct.calcsize(fmt[:-1]) # get len of everything but the string

str\_len = len(dat) - non\_str\_len # get len of string

str\_fmt = "{0}s".format(str\_len) # get new format

new\_fmt = fmt[:-1] + str\_fmt # add it to the old format

return struct.unpack(new\_fmt, dat) # unpack it with new format

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) # set socket

s.bind(('', port)) # bind on port

s.listen(1) # listen

# infinite loop

while 1:

conn, addr = s.accept() # accept connection

print 'Connection address:', addr

data = conn.recv(BUFFER\_SIZE) # recieve data

if not data: break

tup = modified\_unpack("h h B s", data) # unpack data

tml = tup[0] # extract tml

rid = tup[1] # extract rid

operation = tup[2] # extract operation

message = tup[3] # extract message

# display some info about the packet

print "Tml:", tml

print "RID:", rid

print "Operation:", operation

print "Message:", message

# vowel length

if operation == 85:

message = vlength(message) # get vowel count

tml = 6 #set tml (will always be 6)

out = struct.pack("h h h", tml, rid, message) # pack packet

conn.send(out) # send packet

# disemvowel

if operation == 170:

message = disemvoweling(message) # get disemvoweled message

tml = 4 + len(message) # recalculate tml

out = struct.pack("h h", tml, rid) + message # pack packet

conn.send(out) # send packet

conn.close() # close connection

-----------------------------------------------------------------------------------