מטלה 3 -רשתות תקשורת

מגישים: 318845500 , 324460682

חלק א:

בחלק הזה כתבנו 2 קבצי c – TCP_Sender, TCP_Receiver

ה sender שולח קובץ בגודל FILE_LENGTH

הגדרנו את כתובות ה IP להיות IPv4

אפשר להחליף את ה congestion control algorithms ל reno או ל cubic

נעבור על הקוד:

נתחיל עם הקוד של ה sender

רשימה של משתנים:

```
#define SERVER_IP "127.0.0.1" // A default IP address for the server
#define SERVER_PORT 5060 // A default port for the server
#define BUFFER_SIZE 1024 // The buffer size to store the received message
#define INVALID_SOCKET (-1) // define a constant for an invalid socket
#define FILE_LENGTH 2000000 // A variable to store the length of the file to send
```

נגדיר פונקציה ליצירת קובץ עם data רנדומלי (לקחנו מה appendix c):

```
char *util_generate_random_data(unsigned int size) {
    char *buffer = NULL;
// Argument check.
    if (size == 0) {
        return NULL;
    }
    buffer = (char *) calloc( nmemb: size,  size: sizeof(char));
    if (buffer == NULL){
        printf( format: "calloc(3) failed\n");
        return NULL;
    }
// Randomize the seed of the random number generator.
    srand( seed: time( timer: NULL));
    for (unsigned int i = 0; i < size; i++) {
        *(buffer + i) = ((unsigned int) rand() % 256);
    }
    return buffer;
}
```

עכשיו נגדיר את ה main:

קודם נקבל את הארגומנטים של PORT הפורט שאליו נשלח את החבילה ו ALGO האלגוריתם של ה congestion control

אם אין מספיק ארגומנטים, נכניס את הפורט והאלגוריתם הדיפולטיבי שהגדרנו מראש.

```
int main(int args, char** argv) {
    printf( format: "TCP Sender starting\n");
    char *receiver_ip = NULL;
    int receiver_port = 0;
    char *CC_algo = 0;

    if (args != 7) {
        printf( format: "Invalid number of arguments, inserting default values\n");
        receiver_ip = SERVER_IP;
        receiver_port = SERVER_PORT;
        CC_algo = "reno";
    } else if (args == 7) {
        receiver_ip = argv[2];
        receiver_port = atoi( nptr: argv[4]);
        CC_algo = argv[6];
    }
```

עכשיו אנחנו יוצרים משתנה socket וגם struct לשמור את הכתובת של receiver (ה ip והסוג של ה ip במקרה שלנו IPv4. וגם את הפורט) נאתחל את ה struct באפסים

בנוסף ניצור את התוכן (content) של ההודעה שנשלח (בגודל שקבענו)

```
// The variable to store the socket file descriptor, and the server's address.
int sock = INVALID_SOCKET;
struct sockaddr_in server_addr;
// Create a message to send to the server.
char *content = util_generate_random_data( size: FILE_LENGTH); // generate 2 MB of data
int content_len = FILE_LENGTH;
// Reset the server structure to zeros.
memset( s: &server_addr, c: 0, n: sizeof(server_addr));
```

נכניס את כל הנתונים ל struct של ההכתובת

```
// Convert the server's address from text to binary form and store it in the server structure.
// This should not fail if the address is valid (e.g. "127.0.0.1").
if (inet_pton( af: AF_INET, cp: receiver_ip, buf: &server_addr.sin_addr) <= 0) {
    perror( s: "inet_pton(3)");
    close( fd: sock);
    free( ptr: content);
    return -1;
}
// Set the server's address family, port, and address.
server_addr.sin_family = AF_INET; //IPv4
server_addr.sin_port = htons( hostshort: receiver_port); // set the server's port
```

נשנה את האלגוריתם של הסוקט לפי מה שקיבלנו בארגומנטים

```
//try to change the Congestion Control algorithm
if (setsockopt( fd: sock,  level: IPPROTO_TCP,  optname: TCP_CONGESTION,  optval: CC_algo,  optlen: strlen( s: CC_algo)) < 0) {
    perror( s: "setsockopt(2)");
    close( fd: sock);
    free( ptr: content);
    return -1;
}
printf( format: "CC_algo set to: %s\n", CC_algo);
```

ננסה ליצור קשר עם ה receiver

```
// Try to connect to the server.
printf(  format: "Connecting to %s : %d\n", receiver_ip, receiver_port);
// Try to connect to the server.
if (connect( fd: sock,  addr: (struct sockaddr *) &server_addr,  len: sizeof(server_addr)) < 0) {
    perror( s: "connect(2)");
    close( fd: sock);
    free( ptr: content);
    return -1;
}
fprintf( stream: stdout,  format: "Successfully connected to the receiver!\n"
                "Sending message to the receiver.\n");
```

ננסה לשלוח הודעה ל receiver

```
// Send the message to the server.
int bytes_sent = send( fd: sock,  buf: content,  n: content_len,  flags: 0);
// If the message sending failed, print an error message and return 1.
if (bytes_sent <= 0) {
    perror( s: "send(2)");
    close( fd: sock);
    free( ptr: content);
    return -1;
}
fprintf( stream: stdout,  format: "Sent %d bytes to the receiver successfully\n",bytes_sent);
```

עכשיו ניכנס ללולאה כך שה sender יוכל לשלוח שוב את החבילה כמה פעמים שהוא רוצה. אם הוא מכניס 1 החבילה נשלחת שוב, אם הוא מכניס 0 אז הוא שולח חבילה ריקה לסמן ל receiver שהוא מתנתק.

הקוד לקבל את התשובה מהמשתמש:

```
while(1){

    printf( format: "To resend the message press 1, to exit press 0\n");
    int choice;
    if(scanf( format: "%d", &choice)<0){
        perror( s: "scanf(2)");
        close( fd: sock);
        break;
    }
```

הקוד למצב שהמשתמש מכניס 0 (שולחים חבילה בלי דאטה כדי לסמן למקבל על התנתקות):

```c
if(choice == 0){
    int exit_bytes_sent = send( fd: sock,  buf: NULL,  n: 0,  flags: 0);
    // If the message sending failed, print an error message and return 1.
    if (exit_bytes_sent < 0) {
        perror( s: "send(2)");
        close( fd: sock);
        free( ptr: content);
        return -1;
    }
    printf(  format: "Exiting Program now\n");
    break;
}
```

הקוד למצב שבו המשתמש מכניס 1 (כלומר הוא רוצה לשלוח שוב את החבילה):

```c
    if (choice == 1) {
        // Send the message to the server.
        bytes_sent = 0;
        bytes_sent = send( fd: sock,  buf: content,  n: content_len,  flags: 0);
        // If the message sending failed, print an error message and return 1.
        if (bytes_sent < 0) {
            perror( s: "send(2)");
            close( fd: sock);
            free( ptr: content);
            return -1;
        }
        fprintf( stream: stdout,  format: "Sent %d bytes to the server!\n", bytes_sent);
    }
}
```

סיום התוכנית (סוגרים את הסוקט, ומשחררים זיכרון):

```c
    // Close the socket with the server.
    close( fd: sock);
    free( ptr: content);

    fprintf( stream: stdout,  format: "Connection closed!\n");

    // Return 0 to indicate that the client ran successfully.
    return 0;
}
```

עכשיו נעבור על הקוד של ה receiver:

ההגדרות שלנו בתחילת הקוד:

```c
#define SERVER_PORT 5060 // A default port for the server
#define MAX_CLIENTS 1 // The maximum number of clients that can be connected to the server
#define BUFFER_SIZE 1024 // The buffer size to store the received message
#define FILENAME "stats.txt" // A file to store the statistics
#define INVALID_SOCKET (-1) // define a constant for an invalid socket
#define FILE_LENGTH 2000000 // A variable to store the length of the file to send
```

נקבל את הארגומנטים מהמשתמש (אם אין מספיק ארגומנטים הוא מכניס default)

```c
int main(int args, char** argv){
    printf( format: "TCP Receiver starting\n");
    int receiver_port = 0;
    char *CC_algo = 0;
    if (args != 5) {
        printf( format: "Invalid number of arguments, inserting default values\n");
        receiver_port = SERVER_PORT;
        CC_algo = "reno";
    } else if (args ==5) {
        receiver_port = atoi( nptr: argv[2]);
        CC_algo = argv[4];
    }
```

אתחול של המשתנים:

1) ה struct של שמירת הזמן (בשביל למדוד את הזמנים שלוקח לחבילה להגיע)

2) הסוקט שלנו

3) ה struct של כתובות ה sender וה receiver (גם נאתחל אותם להיות 0)

4) אורך הstruct של כתובת ה sender

```c
struct timeval start, end; // The timeval structures to store the start and end times.
int sock = -1; // The variable to store the socket file descriptor.
struct sockaddr_in server_addr; // The server's address structure.
struct sockaddr_in client_addr; // The client's address structure.
socklen_t client_len = sizeof(client_addr); // The client's structure length.
// Reset the server and client structures to zeros.
memset( s: &server_addr,  c: 0,  n: sizeof(server_addr));
memset( s: &client_addr,  c: 0,  n: client_len);
```

נבנה את הסוקט שלנו:

```c
// Try to create a TCP socket (IPv4, stream-based, default protocol).
sock = socket( domain: AF_INET,  type: SOCK_STREAM,  protocol: 0);
if (sock == -1) {
    perror( s: "socket(2)");
    return 1;
}
```

נכניס את כל הנתונים שלנו ל struct (ip, גרסא של ה ip, והפורט)

```c
// Convert the server's address from text to binary form and store it in the server structure.
// This should not fail if the address is valid (e.g. "127.0.0.1").
if (inet_pton( af: AF_INET,  cp: SERVER_IP,  buf: &server_addr.sin_addr) <= 0) {
    perror( s: "inet_pton(3)");
    close( fd: sock);
    return -1;
}
// Set the server's address family and port.
server_addr.sin_family = AF_INET;
server_addr.sin_port = htons( hostshort: receiver_port);
printf( format: "receiver_port set to: %d\n", receiver_port);
```

נגדיר את אלגוריתם בקרת הגודש:

```
// Try to change the Congestion Control algorithm
socklen_t lenCC = sizeof(CC_algo);
if (setsockopt( fd: sock,  level: IPPROTO_TCP,  optname: TCP_CONGESTION,  optval: CC_algo,  optlen: lenCC) != 0) {
    perror( s: "setsockopt(2)");
    close( fd: sock);
    return 1;
}
printf( format: "CC_algo set to: %s\n", CC_algo);
```

נעשה את ה bind (כלומר הסוקט שלנו "יקשיב" לכל מי שמגיע לip ולפורט שהגדרנו)

ואז נעשה listen (כלומר נפתח את הסוקט למצב הקשבה)

```
// Try to bind the socket to the server's address.
if (bind( fd: sock,  addr: (struct sockaddr *) &server_addr,  len: sizeof(server_addr)) < 0) {
    perror( s: "bind(2)");
    close( fd: sock);
    return 1;
}
// Try to listen for incoming connections.
if (listen( fd: sock,  n: MAX_CLIENTS) < 0){
    perror( s: "listen(2)");
    close( fd: sock);
    return 1;
}
fprintf( stream: stdout,  format: "Listening for incoming connections on port %d...\n", SERVER_PORT);
```

נאשר את ההתחברות של ה sender לסוקט שלנו

```
// Try to accept a new client connection.
int client_sock = accept( fd: sock,  addr: (struct sockaddr *) &client_addr,  addr_len: &client_len);
if (client_sock < 0) {
    perror( s: "accept(2)");
    close( fd: sock);
    return 1;
}
// Print a message to the standard output to indicate that a new client has connected.
fprintf( stream: stdout,  format: "Sender %s:%d connected, beginning to receive file\n", inet_ntoa( in: client_addr.sin_addr), ntohs( netshort: client_addr.sin_port));
```

נפתח קובץ חדש לשמור את כל הנתונים על כל חבילה שמגיעה.

ונאתחל משתנים שנשתמש בהן להדפסה של הנתונים והסטטיסטיקות

```
// open a file to document the stats
FILE *stats_file = fopen( filename: FILENAME,  modes: "w");
if (stats_file == NULL) {
    perror( s: "fopen");
    return 1;
}

int total_bytes_received = 0;
int bytes_received=0;
int counter = 0;
int packages_received = 0; // The number of received packages
double speed_sum = 0.0; // The sum of the speeds
double time_sum = 0.0; // The sum of the elapsed times
//create a new buffer to store the received message.
char buffer[BUFFER_SIZE] = {0};
memset( s: buffer,  c: 0,  n: BUFFER_SIZE);
```

ניכנס ללולאה העיקרית של ה receiver ובכל איטרציה של הלולאה הזאת הוא יקבל חבילה מה sender. נאתחל את הנתונים שלנו:

```
// The server's main loop.
while (1) {

    // Reset the variables to their initial values.
    bytes_received=0;
    counter = 0;
    total_bytes_received = 0;
```

בתוך הלולאה המרכזית ניכנס ללולאה פנימית

הלולאה הפנימית תרוץ עד שקיבלנו את כל הדאטה מהשולח.

אחרי שהתחיל לקבל את החבילה הוא מפעיל טיימר כדי למדוד את הזמן שלוקח לחבילה להגיע.

אם הוא מקבל חבילה ריקה זה אומר שהשולח מנסה להתנתק אז הוא מסיים את הלולאה.

בסוף הלולאה נעצור את הטיימר

```
while(total_bytes_received< FILE_LENGTH){ // 2MB (2000000 bytes)
    bytes_received = recv( fd: client_sock,  buf: buffer,  n: sizeof(buffer),  flags: 0);
    if(counter == 0){ // start the timer
        gettimeofday( tv: &start,  tz: NULL);
    }
    if ((bytes_received <= 0)&&(counter==0)) { // if received an exit message (an empty packet)
        close( fd: client_sock);
        printf( format: "exit message received\n");
        break;
    }

    total_bytes_received = total_bytes_received+ bytes_received;
    counter++;
    memset( s: buffer,  c: 0,  n: sizeof(buffer));
}
gettimeofday( tv: &end,  tz: NULL); // end the timer
if(total_bytes_received == 0){ // if received an exit message (an empty packet)
    break;
}
fprintf( stream: stdout,  format: "File transfer complete. Recieved %d bytes \n", total_bytes_received);
```

עכשיו נחשב את כל הנתונים ונוסיף אותם לקובץ שלנו שיהיו מוכנים להדפסה בסוף:

```
double this_elapsed_time = ((end.tv_sec - start.tv_sec) * 1000.0) + ((end.tv_usec - start.tv_usec) / 1000.0);
double this_speed = (total_bytes_received / 1024.0 / 1024.0) / (this_elapsed_time / 1000.0);
speed_sum = speed_sum + this_speed;
time_sum = time_sum + this_elapsed_time;
packages_received++;
fprintf( stream: stats_file,  format: "- Run #%d Data: Time = %.2f ms; Speed = %.2f MB/s\n", packages_received, this_elapsed_time, this_speed);

printf( format: "waiting for the Senders response...\n");
```

נמשיך בלולאה הזאת עד שנקבל חבילה ריקה ואז נצא מהלולאה, נסגור את הסוקט, ונדפיס את הנתונים:

```
fclose( stream: stats_file);
close( fd: sock);
printf( format: "closed the client socket\n");
// ... (Print statistics and times)
printf( format: "---------------------------------\n");
printf( format: "-            * Statistics *       -\n");
stats_file = fopen( filename: FILENAME,  modes: "r");
if (stats_file == NULL) {
    perror( s: "fopen");
    return 1;
}
while (fgets( s: buffer,  n: sizeof(buffer),  stream: stats_file) != NULL) {
    printf( format: "%s", buffer);
}
printf( format: "-\n");
printf( format: "- Average time: %.2fms\n", time_sum / packages_received);
printf( format: "- Average bandwidth: %.2fMB/s\n", speed_sum / packages_received);
printf( format: "---------------------------------\n");
printf(  format: "Receiver end.\n");
fclose( stream: stats_file);
return 0;
```