מטלה 3 -רשתות תקשורת

מגישים: 318845500 , 324460682

חלק ב:

בחלק הזה מימשנו פרוטוקול RUDP

פרוטוקול שמבוסס על ה UDP והופך אותו לאמין

בחלק הזה יש 4 קבצים: RUDP_Sender.c RUDP_Receiver.c RUDP_API.c RUDP_API.h

מימשנו את ה RUDP שיעבוד על IPv4

נתחיל לעבור על ה API

בקובץ RUDP_API.h הגדרנו קודם את ה flags שבו נשתמש וגם את הכמות בייטים שנוכל לשלוח בחבילה כל פעם (כמות הדאטה שנשלח)

בנוסף יש פה את מבנה הפאקטה של RUDP כולל ה header וגם ה data

במימוש שלנו הגדרנו את הheader לכלול את הנתונים הבאים:

Length- אורך הדאטה שנשלח

Checksum- מספר שבודק את תקינות הדאטה בעזרת פונקציית checksum

Seq_num- מספר החבילה ברצף השליחה (נשתמש בזה כדי לוודא שכל החבילות מגיעות)

Flags – מסמן לנו איזה סוג חבילה נשלח (SYN, ACK, FIN, SYNACK, FINACK)

```
#ifndef NETWORK3_RUDP_API_H
#define NETWORK3_RUDP_API_H
#define SYN_FLAG 0x1
#define ACK_FLAG 0x2
#define FIN_FLAG 0x4
#define SYN_ACK_FLAG 0x3
#define FIN_ACK_FLAG 0x6
#define BUFFER_SIZE 16384

typedef struct _RUDP_PACKET {
    int length;
    int checksum;
    int seq_num;
    int flags;
    char data[BUFFER_SIZE];
}RUDP_PACKET;
```

הגדרנו גם את כל הפונקציות שיהיו לנו ב API:

```c
int rudp_my_socket();
int rudp_my_send(int sockfd, const RUDP_PACKET *rudp_packet);
int rudp_my_sendto(int sockfd, const RUDP_PACKET *rudp_packet, struct sockaddr_in *dest_addr, socklen_t addrlen);
int rudp_my_recv(int sockfd, RUDP_PACKET *rudp_packet);
int rudp_my_recvfrom(int sockfd, RUDP_PACKET *rudp_packet, struct sockaddr_in *src_addr, socklen_t *addrlen);
int send_ack(int sockfd, struct sockaddr_in *src_addr, socklen_t addrlen, int seq_num);
int rudp_connect(int sockfd, struct sockaddr_in *dest_addr, socklen_t addrlen);
int rudp_accept(int sockfd, struct sockaddr_in *src_addr, socklen_t *addrlen);
int rudp_dissconnect_client(int sockfd, struct sockaddr_in *dest_addr, socklen_t addrlen);
int rudp_dissconect_server(int sockfd, struct sockaddr_in *src_addr, socklen_t addrlen);
int rudp_recvfrom(int sock,RUDP_PACKET* rudp_packet, struct sockaddr_in *src_addr, socklen_t *addrlen);
int rudp_sendto(int sock, const void *data, size_t len, struct sockaddr_in *dest_addr, socklen_t addrlen);
int rudp_socket(int server, int opt, struct sockaddr_in server_addr,int server_port, int MAX_CLIENTS);
int rudp_close(int sock);
```

נעבור על כל פונקציה ואיך מימשנו אותה:

RUDP_API.c:

נגדיר משתנים חשובים:

```c
#define INVALID_SOCKET -1 // define a constant for an invalid socket
#define TIMEOUT_SECONDS 0 // define a constant for the timeout in seconds
#define TIMEOUT_MICROSECONDS 100000 // define a constant for the timeout in microseconds
#define NUM_RETRIES 3 // define a constant for the number of retries to connect before giving up
```

הוספנו מימוש לפונקציה checksum מתוך האפנדיקס שיש בהוראות המטלה:

```c
unsigned short int calculate_checksum(void *data, unsigned int bytes) {
    unsigned short int *data_pointer = (unsigned short int *) data;
    unsigned int total_sum = 0;
// Main summing loop
    while (bytes > 1) {
        total_sum += *data_pointer++;
        bytes -= 2;
    }
// Add left-over byte, if any
    if (bytes > 0) {
        total_sum += *((unsigned char *) data_pointer);
    }
// Fold 32-bit sum to 16 bits
    while (total_sum >> 16) {
        total_sum = (total_sum & 0xFFFF) + (total_sum >> 16);
    }
    return (~((unsigned short int) total_sum));
}
```

פונקציה שמחזירה סוקט מסוג udp:

```c
int rudp_my_socket(){
    return socket( domain: AF_INET,  type: SOCK_DGRAM,  protocol: 0);
}
```

פונקציות שמשתמשות בסוקט הבנוי של udp כדי לשלוח פאקטות שלי של rudp:

```c
int rudp_my_send(int sockfd, const RUDP_PACKET *rudp_packet){
    return send( fd: sockfd,  buf: (const char*)rudp_packet,  n: sizeof(rudp_packet)+rudp_packet->length,  flags: 0);
}
int rudp_my_sendto(int sockfd, const RUDP_PACKET *rudp_packet, struct sockaddr_in *dest_addr, socklen_t addrlen){
    return sendto( fd: sockfd,  buf: (const char*)rudp_packet,  n: sizeof(RUDP_PACKET)+rudp_packet->length,  flags: 0,  addr: (struct sockaddr*)dest_addr,  addr_len: addrlen);
}

int rudp_my_recv(int sockfd, RUDP_PACKET *rudp_packet){
    return recv( fd: sockfd,  buf: rudp_packet,  n: sizeof(rudp_packet)+rudp_packet->length,  flags: 0);
}
int rudp_my_recvfrom(int sockfd, RUDP_PACKET *rudp_packet, struct sockaddr_in *src_addr, socklen_t *addrlen){
    return recvfrom( fd: sockfd,  buf: rudp_packet,  n: sizeof(RUDP_PACKET)+rudp_packet->length,  flags: 0,  addr: (struct sockaddr*)src_addr,  addr_len: addrlen);
}
```

פונקציה שאחראית להכין חבילה של ACK ולשלוח אותה

```c
// The function to send an ACK packet.
int send_ack(int sockfd, struct sockaddr_in *src_addr, socklen_t addrlen, int seq_num){
    RUDP_PACKET ack_packet;
    ack_packet.length = 0; // no data in ack packet
    ack_packet.checksum = 0;
    ack_packet.seq_num = seq_num; // the sguence number of the received packet
    ack_packet.flags = ACK_FLAG;


    if(rudp_my_sendto(sockfd,  rudp_packet: &ack_packet, dest_addr: src_addr, addrlen) < 0){
        perror( s: "sendACK failed");
        return -1;
    }
    return 0;
}
```

עכשיו נעבור על הפונקציה connect שמטרתה להתחבר לreceiver עם לחיצת יד משולשת:

נכין את החבילת SYN הראשונה שתישלח:

```c
// The function to set up a connection with the server
int rudp_connect(int sockfd, struct sockaddr_in *dest_addr, socklen_t addrlen) {

    RUDP_PACKET syn_packet;
    RUDP_PACKET syn_ack_packet;
    int retries = NUM_RETRIES; // number of retries to connect before giving up

    //make the syn packet
    syn_packet.length = 0; // no data i syn packet
    syn_packet.checksum = 0;
    syn_packet.seq_num = 0; // only 1 packet so seq_num is 0
    syn_packet.flags = SYN_FLAG;
```

ננסה להתחבר כמה פעמים ואם לא נצליח נחזיר שגיאה (נשתמש בפונקציית select כדי לדעת אם קיבלנו SYNACK או TIMEOUT)

```
while (retries > 0) {
    if (rudp_my_sendto(sockfd, rudp_packet: &syn_packet, dest_addr, addrlen) < 0) {
        perror( s: "sendto failed");
        return -1;
    }
    printf( format: "Sent SYN packet\n");

    struct timeval timeout;
    timeout.tv_sec = TIMEOUT_SECONDS;
    timeout.tv_usec = TIMEOUT_MICROSECONDS;

    // Set up the file descriptor set for select
    fd_set read_fds;
    FD_ZERO(&read_fds);
    FD_SET(sockfd, &read_fds);

    // Use select to wait for data on the socket or timeout
    int ready = select( nfds: sockfd + 1, readfds: &read_fds, writefds: NULL, exceptfds: NULL, &timeout);
```

לפי מה שקיבלנו ב select נדע מה הגיע קודם:

אם ready הוא 0 קיבלנו את ה timeout אז ננסה שוב לפי כמות הנסיונות:

```
if (ready < 0) {
    perror( s: "select failed");
    return -1;
} else if (ready == 0) {
    // Timeout occurred
    printf( format: "Timeout occurred. Connection failed.\n");
    retries--;
```

אם ready הוא 1 קיבלנו את חבילת ה ACK :

```
} else {
    // Data is ready to read
    if (rudp_my_recvfrom(sockfd, rudp_packet: &syn_ack_packet, src_addr: dest_addr, &addrlen) < 0) {
        perror( s: "recvfrom failed");
        return -1;
    }

    printf( format: "Received SYNACK packet\n");
```

במקרה כזה נשלח חבילת ACK בעזרת הפונקציה שבנינו כבר:

```
if (syn_ack_packet.flags == SYN_ACK_FLAG) {
    // Make Ack packet
    if (send_ack(sockfd, src_addr: dest_addr, addrlen, seq_num: 0) < 0) {
        perror( s: "sendto failed");
        return -1;
    }
    printf( format: "Sent ACK packet\n");
    return 0;
} else {
    printf( format: "Invalid SYNACK packet\n");
    return -1;
}
```

אם ניסינו כמה פעמים ולא קיבלנו תשובה אז נחזיר שגיאה:

```
}
printf( format: "Connection failed\n");
return -1;
```

הפונקציה accept אחראית על הצד של ה receiver בלחיצת יד המשולשת:

```
// The function to accept a connection from a client
int rudp_accept(int sockfd, struct sockaddr_in *src_addr, socklen_t *addrlen) {
    RUDP_PACKET syn_packet;
    RUDP_PACKET syn_ack_packet;
    RUDP_PACKET ack_packet;

    if (rudp_my_recvfrom(sockfd, rudp_packet: &syn_packet, src_addr, addrlen) < 0) {
        perror( s: "recvfrom failed");
        return -1;
    }
```

נבדוק את החבילה שהגיע שהיא SYN ואם כן נחזיר חבילה SYNACK:

```
if (syn_packet.flags == SYN_FLAG) {
    printf( format: "Received SYN packet\n");

    //make the syn_ack packet
    syn_ack_packet.length = 0; // no data i syn_ack packet
    syn_ack_packet.checksum = 0;
    syn_ack_packet.seq_num = 0; // only 1 packet so seq_num is 0
    syn_ack_packet.flags = SYN_ACK_FLAG;


    if (rudp_my_sendto(sockfd, rudp_packet: &syn_ack_packet, dest_addr: src_addr, *addrlen) < 0) {
        perror( s: "sendto failed");
        return -1;
    }
    printf( format: "Sent SYNACK packet\n");
```

עכשיו נוודא שקיבלנו חזרה חבילת ACK כדי להשלים את הלחיצת יד המשולשת:

```
if (rudp_my_recvfrom(sockfd, rudp_packet: &ack_packet, src_addr, addrlen) < 0) {
    perror( s: "recvfrom failed");
    return -1;
}

if (ack_packet.flags == ACK_FLAG) {
    printf( format: "Received ACK packet\n");
    return 0;
} else {
    printf( format: "Invalid ACK packet\n");
    return -1;
}
```

הפונקציה disconnect_client אחראית על ההתנתקות מהצד של השולח:

קודם נשלח FIN :

```
// The function to close the connection by the client
int rudp_dissconnect_client(int sockfd, struct sockaddr_in *dest_addr, socklen_t addrlen){
    RUDP_PACKET fin_packet;
    RUDP_PACKET fin_ack_packet;

    //make the fin packet
    fin_packet.length = 0; // no data i fin packet
    fin_packet.checksum = 0;
    fin_packet.seq_num = 0; // only 1 packet so seq_num is 0
    fin_packet.flags = FIN_FLAG;

    // Set socket timeout for sending
    struct timeval send_timeout;
    send_timeout.tv_sec = TIMEOUT_SECONDS;
    send_timeout.tv_usec = TIMEOUT_MICROSECONDS;
    setsockopt( fd: sockfd,  level: SOL_SOCKET,  optname: SO_SNDTIMEO,  optval: &send_timeout,  optlen: sizeof(send_timeout));

    if(rudp_my_sendto(sockfd,  rudp_packet: &fin_packet,dest_addr,addrlen) < 0){
        perror( s: "sendto failed");
        return -1;
    }
    printf( format: "Disconnecting from server, sending FIN packet\n");
```

נפעיל טיימר על ה socket כדי לשלוח שוב אם לא קיבלנו תשובה:

```
// Reset socket timeout for receiving
struct timeval recv_timeout;
recv_timeout.tv_sec = TIMEOUT_SECONDS;
recv_timeout.tv_usec = TIMEOUT_MICROSECONDS;
setsockopt( fd: sockfd,  level: SOL_SOCKET,  optname: SO_RCVTIMEO,  optval: &recv_timeout,  optlen: sizeof(recv_timeout));

if(rudp_my_recvfrom(sockfd,  rudp_packet: &fin_ack_packet, src_addr: dest_addr, &addrlen) < 0){
    perror( s: "recvfrom failed");
    return -1;
}

printf( format: "Received FINACK packet\n");
```

אם קיבלנו את ה FINACK נשלח חזרה ACK:

```
if(fin_ack_packet.flags == FIN_ACK_FLAG){
    // send Ack packet
    if(send_ack(sockfd, src_addr: dest_addr,addrlen, seq_num: 0) < 0){
        perror( s: "sendto failed");
        return -1;
    }
    printf( format: "Sent ACK packet\n");
    return 0;
}else{
    printf( format: "Invalid FINACK packet\n");
    return -1;

}
```

הפונקציה disconnect_server אחראית על לנתק את השרת הרגע שקיבלנו FIN:

היא שולחת FINACK ומחכה לתשובה עם טיימר:

```
int rudp_dissconect_server(int sockfd, struct sockaddr_in *src_addr, socklen_t addrlen){
    RUDP_PACKET fin_ack_packet;
    RUDP_PACKET ack_packet;

    //make the fin_ack packet
    fin_ack_packet.length = 0; // no data i fin_ack packet
    fin_ack_packet.checksum = 0;
    fin_ack_packet.seq_num = 0; // only 1 packet so seq_num is 0
    fin_ack_packet.flags = FIN_ACK_FLAG;

    // Set socket timeout for sending
    struct timeval send_timeout;
    send_timeout.tv_sec = TIMEOUT_SECONDS;
    send_timeout.tv_usec = TIMEOUT_MICROSECONDS;
    setsockopt( fd: sockfd,  level: SOL_SOCKET,  optname: SO_SNDTIMEO,  optval: &send_timeout,  optlen: sizeof(send_timeout));


    if(rudp_my_sendto(sockfd,  rudp_packet: &fin_ack_packet,  dest_addr: src_addr, addrlen) < 0){
        perror( s: "FIN_ACK failed");
        return -1;
    }
    printf( format: "Sent FINACK packet\n");
```

אם היא קיבלה ACK בזמן היא מחזירה 0

```
  // Reset socket timeout for receiving
  struct timeval recv_timeout;
  recv_timeout.tv_sec = TIMEOUT_SECONDS;
  recv_timeout.tv_usec = TIMEOUT_MICROSECONDS;
  setsockopt( fd: sockfd,  level: SOL_SOCKET,  optname: SO_RCVTIMEO,  optval: &recv_timeout,  optlen: sizeof(recv_timeout));


  if(rudp_my_recvfrom(sockfd,  rudp_packet: &ack_packet, src_addr, &addrlen) < 0){
      perror( s: "recvfrom failed");
      return -1;
  }
  printf( format: "Received ACK packet\n");
  return 0;
```

פונקציה rudp_recvfrom אחראית על קבלת חבילות.

כל חבילה היא בודקת את ה flags אם היא קיבלה FIN היא מפעילה
disconnect_server אם היא קיבלה ללא flags היא מחזירה ACK לפי החבילה שהיא בדיוק קיבלה:

```
int rudp_recvfrom(int sock, RUDP_PACKET* rudp_packet ,struct sockaddr_in *src_addr, socklen_t *addrlen){

    int receive_bytes = rudp_my_recvfrom( sockfd: sock, rudp_packet, src_addr, addrlen);
    if(receive_bytes < 0){
        perror( s: "recv failed");
        return -1;
    }
```

אם אין flags נבדוק את ה checksum ואת ה seq_num:

```c
// regular send pack - always return an ACK paket
if(rudp_packet->flags == 0) {
    int seq_num = rudp_packet->seq_num;

    if (rudp_packet->data == NULL) {
        fprintf( stream: stderr,  format: "Error: Received NULL data.\n");
        return -1;
    }
    // Calculate the checksum of the received packet
    int checksum = calculate_checksum(rudp_packet->data,  bytes: rudp_packet->length);
    char buffer[BUFFER_SIZE];
    memset( s: buffer,  c: 0,  n: BUFFER_SIZE);
```

אם הכל תקין נחזיר ack לפי ה seq_num של החבילה שבדיוק קיבלנו:

```c
// Check if the checksum is correct
if(checksum == rudp_packet->checksum){
    if(seq_num>=0){
          printf("Received packet number %d. Checksum = True. sending ACK for packet...\n", seq_num);
        send_ack( sockfd: sock, src_addr, *addrlen, seq_num);
        memset( s: buffer,  c: 0,  n: rudp_packet->length);

    } else{
          printf("Received last packet!\n");
          printf("last packet length: %d\n", rudp_packet->length);
        send_ack( sockfd: sock, src_addr, *addrlen, seq_num);
        memset( s: buffer,  c: 0,  n: rudp_packet->length);
        return rudp_packet->length;
    }
} else{
    printf( format: "Checksum is incorrect\n");
    send_ack( sockfd: sock, src_addr, *addrlen,  seq_num: seq_num-1);
}
```

במקרה שקיבלנו חבילה FIN נבצע התנתקות בעזרת הפונקציה שהגדרנו מקודם:

```c
  // if the packet is a FIN packet
  else if(rudp_packet->flags==FIN_FLAG){
      printf( format: "FIN packet received. Disconnecting...\n");
      if(rudp_dissconect_server( sockfd: sock, src_addr, *addrlen) < 0){
          perror( s: "dissconect failed");
          return -1;
      }
      printf( format: "Disconnected from the client\n");
      return -2;
  }
  else{
      printf( format: "Invalid packet\n");
      return -1;
  }
  return rudp_packet->length;
}
```

פונקצית ה rudp_sendto שולחת חבילה לשרת

אם הגודל של החבילה גדולה יותר מהקיבולת שהגדרנו ל RUDP נחלק את החבילה לחבילות יותר קטנות. נשחל אותם אחד אחר ועבור כל אחד נחכה ל ACKעם אותו

מספר סידורי שיחזור אלינו. אם לא קיבלנו את ה ACK עד ל timeout נשלח את אותה החבילה שוב.

פונקציה זו מממשת את הפרוטוקול של stop and wait:

```
int rudp_sendto(int sock, const void *data, size_t len, struct sockaddr_in *dest_addr, socklen_t addrlen){
    RUDP_PACKET rudp_packet;
    size_t num_packets = (len + BUFFER_SIZE - 1) / BUFFER_SIZE;
    char* data_ptr = (char*)data;
    int total_bytes_sent = 0;
    printf( format: "Sending the data in %zu packets\n", num_packets);
```

נשלח את החבילות בלולאת for:

נכין את החבילה (נסמן את החבילה האחרונה במספר הסידורי 1- כדי שהמקבל ידע מתי הגיע אליו החבילה האחרונה)

```
for (size_t i = 0; i < num_packets; i++) {
    data_ptr = (char*)data + (i * BUFFER_SIZE);
    rudp_packet.seq_num = i;
    rudp_packet.flags = 0;
    rudp_packet.checksum = calculate_checksum( data: data_ptr,  bytes: BUFFER_SIZE);
      rudp_packet.data = buffer;
    memcpy( dest: rudp_packet.data,  src: data_ptr,  n: BUFFER_SIZE);
    rudp_packet.length = BUFFER_SIZE;
    if (i == num_packets - 1) {

        rudp_packet.seq_num = -1;
        rudp_packet.length = len - i * BUFFER_SIZE;
        rudp_packet.checksum = calculate_checksum( data: data_ptr,  bytes: rudp_packet.length);
    }
    if (rudp_my_sendto( sockfd: sock, &rudp_packet, dest_addr, addrlen) < 0) {
        perror( s: "sendto failed");
        return -1;
    }
    if(i==(num_packets/2)){
        printf( format: "Sent half the packets\n");
    }
```

נגדיר timeout עבור ה ACK אם הוא לא מגיע נשלח שוב את אותה החבילה:

```
// Set up a timeout
struct timeval timeout;
timeout.tv_sec = TIMEOUT_SECONDS;
timeout.tv_usec = TIMEOUT_MICROSECONDS;

while (1) {
    // Set up the file descriptor set for select
    fd_set read_fds;
    FD_ZERO(&read_fds);
    FD_SET(sock, &read_fds);

    // Use select to wait for data on the socket or timeout
    int ready = select( nfds: sock + 1,  readfds: &read_fds,  writefds: NULL,  exceptfds: NULL, &timeout);
```

אם הגענו לסוף הזמן, נשלח את החבילה שוב ונאתחל את השעון:

```
if (ready < 0) {
    perror( s: "select failed");
    return -1;
} else if (ready == 0) {
    // Timeout occurred, resend the packet
      printf("Timeout occurred, resending packet\n");
    if (rudp_my_sendto( sockfd: sock, &rudp_packet, dest_addr, addrlen) < 0) {
        perror( s: "sendto failed");
        return -1;
    }
      printf("Resent packet, waiting for ACK\n");
    timeout.tv_sec = TIMEOUT_SECONDS;
    timeout.tv_usec = TIMEOUT_MICROSECONDS;
    continue;
```

אם קיבלנו את ה ACK נוודא שהוא האחד הנכון ונמשיך לחבילה הבאה:

```
} else {
    // Data is ready to read, check if it's an ACK packet
    RUDP_PACKET received_packet;
    if (rudp_my_recvfrom( sockfd: sock,  rudp_packet: &received_packet,  src_addr: dest_addr, &addrlen)){

        if ((received_packet.flags == ACK_FLAG)&&(received_packet.seq_num == rudp_packet.seq_num )){
            // ACK received, break the loop and continue to the next packet
            if (received_packet.seq_num != -1) {
                  printf("Received ACK for packet number %d\n", received_packet.seq_num);
            }else{
                printf( format: "Received ACK for last packet\n");
            }
        }
        break;
    }
}
```

בסוף כל התהליך נדפיס ששלחנו את כל הדאטה:

```
    }
    printf( format: "All data sent!\n");
    return 0;
}
```

הפונקציה ליצירת socket חדש לפי ה RUDP :

אם אנחנו השרת אז נכין סוקט

```
int rudp_socket(int server, int opt, struct sockaddr_in server_addr,int server_port, int MAX_CLIENTS){
    if(server){
        int sock = socket( domain: AF_INET,  type: SOCK_DGRAM,  protocol: 0);
        if (sock == INVALID_SOCKET){
            perror( s: "rudp_socket(2)");
            return -1;
        }
```

נכניס בו את הנתונים שלנו ונעשה לו bind עם ה IP וה PORT :

```
// Set the server's address to "0.0.0.0" (all IP addresses on the local machine).
server_addr.sin_addr.s_addr = INADDR_ANY;
// Set the server's address family and port.
server_addr.sin_family = AF_INET;
server_addr.sin_port = htons( hostshort: server_port);

// 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;
}
fprintf( stream: stdout,  format: "Listening for incoming connections on port %d...\n", server_port);
return sock;
```

אם אנחנו הלקוח אז לא צריך לעשות את על זה רק צריך את הסוקט:

```
}else {
    int sock = socket( domain: AF_INET,  type: SOCK_DGRAM,  protocol: 0);
    if (sock == INVALID_SOCKET) {
        perror( s: "rudp_socket(2)");
        return -1;
    }
    return sock;
}
```

פונקציה לסגירת הסוקט :

```
int rudp_close(int sock){
    return close( fd: sock);
}
```

עכשיו נעבור על ה RUDP_Sender:

נגדיר את הנתונים :

```
#define INVALID_SOCKET -1
#define TIMEOUT_SECONDS 2
#define SERVER_IP "127.0.0.1"
#define SERVER_PORT 5060
#define FILE_LENGTH 2500000
```

נקבל את הארגומנטים של ה IP וה PORT:

```
int main(int args, char** argv){
    printf( format: "RUDP Sender starting\n");
    char *server_ip = NULL;
    int server_port = 0;

    if (args != 5) {
        printf( format: "Invalid number of arguments, inserting default values\n");
        server_ip = SERVER_IP;
        server_port = SERVER_PORT;
    } else if (args == 5) {
        server_ip = argv[2];
        server_port = atoi( nptr: argv[4]);
    }
```

נבנה את הסוקט שלנו בעזרת API_RUDP

ונבנה את החבילה בעזרת הפונקציה שנתונה לנו באפנדיקס

```
int sock = -1;
struct sockaddr_in server;
char *message = util_generate_random_data( size: FILE_LENGTH);

memset( s: &server, c: 0, n: sizeof(server));
sock = rudp_socket( server: 0, opt: 1, server_addr: server, server_port, MAX_CLIENTS: 1);
if (sock == INVALID_SOCKET){
    perror( s: "rudp_socket(2)");
    return 1;
}
```

נכניס בו את הנתונים שלנו וננסה להתחבר לשרת:

```
server.sin_family = AF_INET;
server.sin_port = htons( hostshort: server_port);

fprintf( stream: stdout, format: "Connecting to %s:%d ...\n", server_ip, server_port);
if(rudp_connect( sockfd: sock, dest_addr: (struct sockaddr_in *)&server, addrlen: sizeof(server)) < 0){
    perror("rudp_connect(2)");
    rudp_close(sock);
    return -1;
}
fprintf( stream: stdout, format: "Successfully connected to the server!\n"
                "Sending message to the server\n");
```

אם התחברנו בהצלחה נשלח את ההחבילה בפעם הראשונה:

```
int bytes_sent = rudp_sendto(sock, data: message, len: FILE_LENGTH, dest_addr: (struct sockaddr_in *)&server, addrlen: sizeof(server));
if (bytes_sent < 0) {
    perror( s: "rudp_sendto(2)");
    rudp_close(sock);
    return -1;
}
fprintf( stream: stdout, format: "Message sent successfully!\n");
```

עכשיו נשאל את המשתמש אם הוא רוצה לשלוח שוב:

```
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;
    }
```

אם הוא רוצה להתנתק נשלח חבילה FIN ונתנתק בעזרת הפונקציות שמוגדרות ב API:

```
if(choice == 0){
    if(rudp_dissconnect_client( sockfd: sock, dest_addr: (struct sockaddr_in *)&server, addrlen: sizeof(server)) < 0){
        perror( s: "rudp_dissconnect_client(2)");
        rudp_close(sock);
        return -1;
    }
    printf( format: "Disconnected from the server\n");
    rudp_close(sock);
    break;
}
```

אחרת נשלח את החבילה שוב:

```
if(choice == 1){
    bytes_sent= rudp_sendto(sock, data: message, len: FILE_LENGTH, dest_addr: (struct sockaddr_in *)&server, addrlen: sizeof(server));
    if (bytes_sent < 0) {
        perror( s: "rudp_sendto(2)");
        rudp_close(sock);
        return -1;
    }
    fprintf( stream: stdout, format: "Message sent successfully!\n");
    bytes_sent=0;
}
```

בסוף נשחרר את הזיכרון ונסיים :

```
    free( ptr: message);
    fprintf( stream: stdout, format: "Connection closed!\n");
    return 0;
}
```

נעבור על ה RUDP_Receiver:

נתחיל בלקבל את הארגומנטים של ה PORT:

```
int main(int args, char** argv){
    printf( format: "RUDP Receiver starting\n");
    int server_port = 0;
    if (args != 3) {
        printf( format: "Invalid number of arguments, inserting default values\n");
        server_port = SERVER_PORT;
    } else if (args ==3) {
        server_port = atoi( nptr: argv[2]);

    }
```

נאתחל את המשתנים של המדידת זמן והכתובות של הלקוח והשרת

```
struct timeval start, end;
int sock = -1;
struct sockaddr_in server_addr;
struct sockaddr_in client_addr;
socklen_t client_len = sizeof(client_addr);
// 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 בעזרת הפונקציות שלנו:

```
sock = rudp_socket( server: 1,opt, server_addr,server_port, MAX_CLIENTS: MAX_CLIENTS);
if (sock == INVALID_SOCKET) {
    perror( s: "rudp_socket_setup(2)");
    return 1;
}
```

נקבל את הקשר החדש של הלקוח בעזרת הפונקציה ב API:

```
int client_sock = rudp_accept( sockfd: sock,  src_addr: (struct sockaddr_in *)&client_addr,  addrlen: &client_len);
if (client_sock == INVALID_SOCKET){
    perror( s: "rudp_accept(2)");
    close( fd: sock);
    return 1;
}
printf( format: "Accepted connection from %s:%d\n", inet_ntoa( in: client_addr.sin_addr), ntohs( netshort: client_addr.sin_port));
```

נאתחל את המשתנים שצריך כדי למדוד את הזמן והמהירות של כל השליחת קבצים:

נאתחל את הקובץ הדפסות

ונאתחל את המיקום בזיכרון שאליו נקבל את הפקטה:

```
int total_bytes_received = 0;
int bytes_received=0;
int counter = 0;
int packages_received = 0;
double speed_sum = 0.0;
double time_sum = 0.0;
char buffer[BUFFER_SIZE];

FILE *stats_file = fopen( filename: FILENAME,  modes: "w");
if (stats_file == NULL) {
    perror( s: "fopen");
    return 1;
}
RUDP_PACKET *packet = (RUDP_PACKET *) malloc( size: sizeof(RUDP_PACKET));
```

ניכנס ללולאה של קבלת חבילות:

```
while(1){

    bytes_received=0;
    counter = 0;
    total_bytes_received = 0;
    memset( s: buffer,  c: 0,  n: BUFFER_SIZE);
    int sequence =-1;
```

עכשיו עבור כל חבילה שמגיע נמדוד את הזמן שלוקח לכל הדאטה להגיע

בנוסף, נקבל רק חבילות שבאות ברצף של המספר הסידורי (על מנת לא לכלול חבילות שנשלחו פעמיים והגיעו באיחור)

```
while(total_bytes_received<FILE_LENGTH){
    bytes_received=0;
    bytes_received = rudp_recvfrom(sock, rudp_packet: packet, src_addr: (struct sockaddr_in *)&client_addr, addrlen: &client_len);
    if(packet->seq_num > sequence||packet->seq_num ==-1) {
        if (bytes_received == -1) {
            perror( s: "rudp_recvfrom(2)");
            close( fd: sock);
            free( ptr: packet);
            return -1;
        }
        if (counter == 0) {
            gettimeofday( tv: &start, tz: NULL);
              printf("Started receiving file...\n");
        }
        if (bytes_received == -2) {
            close( fd: sock);
            break;
        }
        sequence ++;
        total_bytes_received = total_bytes_received + bytes_received;
        counter++;
          printf("including packet %d, Received total of %d bytes from the client\n", packet->seq_num,total_bytes_received);
    }
```

אם קיבלנו חבילה כפולה נתעלם ממנה

נרוקן את ה buffer ונסיים למדוד זמן. אם קיבלנו 2- זה סימן שלא נקבל עוד חבילות וצריך להתנתק

```
    }
    else{
          printf("Received a duplicate packet, ignoring it\n");
    }
    if((counter%50==0)&&(counter!=0)){
        printf( format: " -> Received %d packets from the sender\n",counter);
    }
    memset( s: buffer, c: 0, n: sizeof(buffer));
    memset( s: packet, c: 0, n: sizeof(RUDP_PACKET));
}

gettimeofday( tv: &end, tz: NULL);
if(bytes_received==-2){
    break;
}
```

נוסיף את כל הנתונים לקובץ הדפסות ונחכה לחבילה הבאה:

```
printf( format: "File transfer complete. Received a total of %d bytes\n",total_bytes_received);
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);

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

בסוף (אחרי שהלקוח מתנתק) נדפיס את כל הנתונים שלנו ונשחרר את הזיכרון:

```c
free( ptr: packet);
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");

return 0;
```