Lab Project 6: **Stop & Wait Protocol (reliable data transmission over UDP)**

- **Number of members per group: 1 or 2**
- **Use C**

This project consists of building and Stop and Wait reliable protocol on top of UDP. Messages are sent one at a time, and each message needs to be acknowledged when received, before a new message can be sent.

The program consists of a client and a server. The client transmits a file to the server. Communication is unidirectional, i.e., data flows from the client to the server. The server starts first and waits for messages. The client starts the communication. Messages have sequence number 0 or 1 (start with zero).

Before sending each message, a 1-byte checksum is calculated and added to the header. After sending each message, the client waits for a corresponding ACK. When it arrives, if it is not the corresponding ACK or if the checksum does not match, the message is sent again. If it is the corresponding ACK, the client changes state and returns to the application (`main()` function here), which can now send one more message. This means that the client blocks on writes until an ACK is received.

Use `select()` function for timed wait.

- If `select()` returns zero, there is no data, and the client needs to retransmit, restart the timer, and call select again.
- If `select()` returns 1, there is data, so the client calls `recvfrom()` to receive the ACK and then processes it. If ACK is not corrupted and the ack number is right, the client can now send one more message.

The server, after receiving a message, checks its checksum. If the message is correct and the sequence number is right, the server sends an ACK message (according to the sequence number) to the client, and the data is ready to be written in the file. Otherwise the server repeats the last ACK message and waits to receive a message again.

The protocol should deal properly with duplicate data messages and duplicate ACK messages.

Each message contains the header and the application data. No reordering is necessary, since the client is sending the exact message given by the application, one by one.

The checksum must be calculated for messages from the server and client. To calculate the checksum, calculate the XOR of all the bytes (header + data, or just header if there is no data) when member cksum field (see below) is zero.

To verify your protocol, use the result of a random function to decide whether to send the right checksum or just zero. This will fake the error effect. Follow a similar approach to fake packet loss.

Packet format is defined as follows:

```c
typedef struct
{
    int seq_ack;
    int len;
    int checksum;
} HEADER;
```

```
typedef struct
{
    HEADER     header;
    char       data[SIZE];
} PACKET;
```

SENDER

- Member `seq_ack` is used as sequence number, and the data is in member `data`.
- Each packet may have 100 or less bytes of data.
- **After transmitting the file, a packet with no data (`len` = 0) is sent to notify the receiver that the file is complete.**

RECEIVER

- Member `seq_ack` is used as ACK, and `data` is empty (`len` = 0).

Note

The server closes the file and terminates execution after the message with zero bytes arrives. If the ack sent for that last message does not make it to the client, the client will keep resending it forever. To avoid that, the client will start a counter after sending a message with zero bytes and will only resend that last message 3 times. After 3 times, it will return to the main function.

Using select() function

This is an example on how to use select (check the man page for includes):

```
// local variables needed
// set it up, in the beginning of the function fd_set readfds;
fcntl (sock, F_SETFL, O_NONBLOCK);
...

// start before calling select
FD_ZERO (&readfds);
FD_SET (sock, &readfds);

// set the timer
tv.tv_sec = 10;
tv.tv_usec = 0;
```

```
struct timeval tv;
// timer
int rv;
// select returned value
// call select
rv = select (sock + 1, &readfds, NULL, NULL, &tv);

if (rv == 0)
{
    // timeout, no data
}
else if (rv == 1)
{
    // there is data to be received
}
```

Deliverables:

- **Demo your project to the TA in the lab**
- **Submit your code to Camino**