Tomer Ben Kereth: 305685406
Omer Hanetz: 200383578
Email: bk.tomer@gmail.com / o.hanetz@gmail.com

Communication Networks Practical Exercise #2

Protocol Description (nim_protocol.h):

The protocol is implemented inside the specified header file. In this file, we defined several structures for all the message types we send. All the structures are united together under two major structures – one for the server and one for the client, each of them holds all the types of server / client message we are sending in a specific union field:

Major Structures:

1. msg

   This is the major server message structure. It contains the following fields:

Name	Type	Description
type	Msg_type (enum)	Contains the type of the message
data	Data_union (union)	contains the structures for the possible server messages we send

   - Type – A msg_type enum field that contains the type of the message.
   - Data – a data_union field that contains the structures for the possible server messages we send.

   a. Msg_type – an enum that contains values for the possible message types we have, each type has a specific structure that we send alongside it:

Value	Description	Related Structure
INIT_MSG	Initial connection message	Init_server_msg
SERVER_MSG	Server status message	Server_msg
CHAT_MSG	Server/client chat message	Chat_msg
CLIENT_MOVE_MSG	Client move message	Client_move_msg
AM_MSG	After move message	After_move_msg
REJECTED_MSG	Client rejected message	No related structure

   b. Data_union – a union that contains fields for all the possible server messages we send:

Name	Type	Description
Init_msg	Init_server_msg	Initial connection message
S_msg	Server_msg	Server status message
Chat	Chat_msg	Server chat message
Am_msg	After_move_msg	After move message

2. Client_msg

   This is the major client message structure. It contains the following fields:

Name	Type	Description
type	Msg_type (enum)	Contains the type of the message

data	Client_data_union (union)	contains the structures for the possible client messages we send

- Type – A msg_type enum field that contains the type of the message.
- Data – a data_union field that contains the structures for the possible server messages we send.

a. Msg_type – an enum that contains values for the possible message types we have, each type has a specific structure that we send alongside it (same enum as in msg structure).

b. Client_data_union – a union that contains fields for all the possible client messages we send:

Name	Type	Description
Chat	Chat_msg	Client chat message
Client_move	client_move_msg	Client move message

Additional Structures:

1. Init_server_msg
   This type of message is used to inform the client his number when he connects.
   This structure contains only one field:

Name	Type	Description
Client_num	Short	The client number

2. Server_msg
   This type of message is used at the start of every client's turn, when the server informs the client for the pile's sizes and if there was a winner or the game is still on.
   This structure contains the following fields:

Name	Type	Description
N_a, n_b, n_c	short	Represent the heap sizes in the nim game (assuming that size is at most 1000).
Winner	char	Used for letting the client know if there was a winner in the last round. There are 3 values that can be used here: ○ `NO_WIN` (a built-in macro in the protocol for 'n') ○ `CLIENT_WIN` (a built-in macro in the protocol for 'w') ○ `CLIENT_LOSE` (a built-in macro in the protocol for 'l')
Legal	Char	Whether or not the previous move made by the opponent was legal: ○ `LEGAL_MOVE` (a built-in macro in the protocol for 'g') ○ `ILLEGAL_MOVE` (a built-in macro in the protocol for 'b')
Player_turn	Short	The current player turn
Cubes_removed	Short	The number of cubes removed by the opponent
Heap_name	char	The name of the heap that the opponent removed cubes from

3. Chat_msg

This type of message is used to send chat messages from a client to the server, and to transfer the chat messages from the server to the other client.
This structure contains the following fields:

Name	Type	Description
Sender_num	Short	The number of the client that sent the message
Msg	Char array	The message. The size of the array is MSG_MAX_SIZE, which is a macro, defaults to 255.

4. After_move_msg
   This type of message is used to inform the client that just made a move whether his move was legal or not.
   This structure contains only one field:

Name	Type	Description
Legal	Char	Whether or not the previous move was legal: ○ `LEGAL_MOVE` (a built-in macro in the protocol for 'g') ○ `ILLEGAL_MOVE` (a built-in macro in the protocol for 'b')

5. Client_move_msg
   This type of message is used right after the client writes his move in the console. It is used to inform the server what is the move the client entered.
   This structure contains the following fields:

Name	Type	Description
Heap_name	char	The name of the heap that the client wants to remove cubes from
Num_of_cubes_to_remove	Short	The number of cubes that the client wants to remove from the heap he selected

In addition to the macros described before, we defined the following macros that are suitable for both client and server side:

- `DEFAULT_HOST "localhost"`
- `DEFAULT_PORT "6444"`
- `PILE_A_CHAR 'A'`
- `PILE_B_CHAR 'B'`
- `PILE_C_CHAR 'C'`
- `QUIT_CHAR 'Q'`
- `MSG_CHAR 'M'`
- `INIT_CHAR 'I'`
- `NUM_CLIENTS 2`
- `MSG_NUM 50`

Client Program Description (nim.c)

The client program is used for communication between the player and the server program. The only logic implemented in this program is for communicating with the server, submit a move, submit a chat message, display the status of the game (current heap sizes, winner information if needed, inform the user if its move was legal), and display a chat message from the other client.
First, we checked if we received any details about the server IP and port. If not, we used the default options. Then, by calling **getaddrinfo** we got a list of the server's addrinfo's, each of which contains an Internet address that can be specified in a call to connect. For each of this addresses, we tried to establish a connection. When we succeed connecting one of them, we stop iterating over the list. After the establishment of the connection, we communicated with server as described in the application protocol.
When the first client connects, he waits for an indication from the server that a second client connected. When a second client connects, the game begins and the first client gets the turn. If a client tries to connect after 2 clients are already connected, he gets rejected by the server and quits.
In order to run the program, we need to make sure that the server program is already running, otherwise we won't be able to connect and the application will crash. If the server program is already running, the running command will be:
nim [hostname [port]] (giving a hostname and a port is not mandatory. But if a port was given we must give also a hostname).

Server Program Description (nim-server.c)

The server program handles only one game at a time. The server "remembers" the sizes of the heaps, getting the client's move and making a counter move after it.
First, we checked if we received any details about the listening port. If not, we used the default option. In addition, we save the initial sizes of the heaps. Then, by calling **getaddrinfo** we got a list of the listening hostname's addrinfo's, each of which contains an Internet address that can be specified in a call to connect. For each of this addresses, we tried to establish a binding. When we succeed binding one of them, we stop iterating over the list. After the establishment of the binding, we try to listen to the socket we establish a bind with. Then, when we successfully started listening to the listening socket, we try to accept an incoming request from clients. Once a client connects, we check if there are already 2 connected clients, and if so, we send the client a message to reject his connection. When one of the first 2 clients connect, we send him an init message to tell him his number. Once two clients are connected, we send them both initial server message with heaps status, and the first clients gets to start. Then we start communicating with the clients as described in the application protocol and the exercise instructions:

- Once a client sends his move, we switch turns, send him an indication whether his move was legal, and sending the other client a server message with the new status.
- Once a client sends a chat message, we transfer it to the other client.
- Once a client quits, the other client gets a winner indication.

In order to run the server program, we need to execute the following command**: nim-server n_a n_b n_c [port]** (where n_a, n_b, n_c stand for the 3 heaps' initial sizes and the port is an optional attribute. If not provided, default value is used).

Edge Cases

The following edge cases are handled in the program:

- In both server and client program, when we call **getaddrinfo** we get a linked list of **struct addrinfo**, each one represents an internet address that we can connect to, based on the given parameters. Sometimes addresses may not work (and indeed, in practice, when we used only the first address in the list we sometimes failed to connect), and therefore we are iterating over the list and trying to bind each address, until we find one that binds successfully.