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

Communication Networks Practical Exercise #1

Protocol Description (nim_protocol.h):

The protocol is implemented inside the specified header file. In this file, we defined three structures, each one of them represents a type of a message sent by the client or the server. The messages are:

1) **Server_msg :**
   This structure contains the following fields:
   - Three **short** integers (n_a, n_b, n_c), which represent the heaps' sizes in a nim game (assuming a pile's size is at most 1000).
   - A **char**, named "winner", 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 'c')
     - `SERVER_WIN` (a built-in macro in the protocol for 's')

   This type of message is used at the start of every client's turn, when the server inform the client for the pile's sizes and if there was a winner or the game is still on.

2) **Client_msg :**
   This structure contains the following fields:
   - A **char** (heap_name), that represents the client's selected heap, to remove cubes from.
   - A **short** integer (num_cubes_to_remove), that represent the number of cubes the client asked to remove from the heap he selected.

   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.

3) **After_move_msg :**
   This structure contains only one field:
   - A **char** (legal), representing whether the client's move was legal or not. There are two optional values:
     - `LEGAL_MOVE` (built-in macro for 'g')
     - `ILLEGAL_MOVE` (built-in macro for 'b')

   This type of message is used to inform the client whether his move was legal or not.

In addition, 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'`
- `The five macros described before.`

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 and display the status of the game (current heap sizes, winner information if needed, inform the user if its move was legal).
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.
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 a client. Afterwards, we start communicating with client as described in the application protocol (and in-between making a server's move, as described in the exercise instructions.). 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.

- In the client program, when the user wants to quit, the client should send a message to the server and then close the socket. However, we cannot be sure that the server got the message from the client, and if the message wasn't delivered the server may not function correctly (because it assumes that the client is still connected). Therefore, we are using the **shutdown** command to close the connection only after all the data was sent, and then we try to get data from the server using **recv**, and if we get 0, that means that the server has closed the connection as we expected, and we can safely close the client socket now, and we do that.