CODE DOCUMENTATION

Introduction to Computer Communications
Programming Assignment 1 - Noisy Channel

Liron Cohen 207481268, Yuval Mor 209011543

Running Instructions

Running the program is as required in the assignment.

1. Running the Channel exe file with the relevant parameters.
2. The channel will print the IP addresses and ports for the sender and receiver connections.
3. Running the sender and receiver exe files with the IP addresses and ports that the channel printed.

Sender Main Flow

- Parsing arguments and initializing Winsock.
- Asking user to enter file name and read it.
- While user didn't enter "quit":
    - Opening the file.
    - Connecting to socket (using **ConnectToSocket** function).
        - Calling "socket" syscall to create a TCP socket.
        - Creating channel address struct using inet_pton for converting IP string to an IP address and using htons to convert port to network endian.
        - Connecting to the socket.
    - Creating three buffers (using **createBuffers** function):
        - rawBytesFileBuffer - buffer for raw file, size of 26 bytes.
        - originalBitsFileBuffer - buffer for original bits, size of 26*8 bytes (208 binary chars, e.g. '0' / '1').
        - encodedBitsFileBuffer - buffer for encoded bits, size of 31 bytes (binary chars).
    - While finished flag if 0 (using while true and break):
        - Initilizing buffers values to '\0' (for cases when the file length is not a divided by 26).
        - Reading section of 26 bytes from file (using **readSectionFromFile** function) and adding the number to the bytesReadTotal variable. If bytesRead is 0 then we finished reading the file and the finished flag is changed to 1.
        - If finished flag is 1, we break the loop.

        - Else, translating the section from bytes to char bits (using **translateSectionFromBytesToCharBits** function):
            - Based on answer from: https://www.dreamincode.net/forums/topic/134396-how-to-convert-a-char-to-its-8-binary-bits-in-c/
            - The translation is done using a binary calculation - going over every byte in the section and translate the bits of the byte to '1' and '0' chars, putting in originalBitsFileBuffer.
        - For every block (26 bytes that are 26 char bits) in the section, if the block starts with actual data (and not '\0'):
            - Copying data to encoded buffer (using **copyDataToEncodedBuffer** function) in order to encode the data with hamming. Also, initilazing parity check bits to '0'.
            - Adding hamming parity bits (using **addHummingCheckBits** function that calls **generateParityBit** function with the five power-of-two indexes that are the parity check bits indexes. The generation function counts the number of '1's in the relevant bits to the parity and write the parity bits to buffer (ignore '\0' that we have when the file length is not a divided by 26).
            - Writing block to socket (using **writeBlockToSocket** function) - writing 31 char bits, that are 31 encoded bits, to socket.
    - Closing the socket and the file.
    - Printing the relevant messages.
    - Asking the user to enter a new file name and reading it.
    - Initializing some of the variables.
- Cleaning up Winsock and exits successfully.

Receiver Main Flow

- Parsing arguments and initializing Winsock.
- Asking user to enter file name and read it.
- While user didn't enter "quit":
    - Opening the file.
    - Connecting to socket (using **ConnectToSocket** function).
        - Calling "socket" syscall to create a TCP socket.
        - Creating channel address struct using inet_pton for converting IP string to an IP address and using htons to convert port to network endian.
        - Connecting to the socket.
    - Creating two block buffers (using **createBlockBuffers** function):
        - encodedBitsFileBuffer- buffer for encoded noised content - size 31 bytes (bit chars).

        - decodedBitsFileBuffer- buffer for decoded content - size 26 bytes (bit chars).
    - Creating two section buffers (using **createSectionBuffers** function):
        - sectionFileBuffer - buffer for section file content, size 208 (26 bytes * 8 bits per bytes, 26 char bytes).
        - bytesFileBuffer - buffer for file content, size 26 bytes.
    - While finished flag if 0 (using while true and break):
        - Reading section of 31 bytes from socket (using **readBlockFromSocket** function) and adding the number to the bitsReadTotal variable. If bitsRead is 0 then we finished reading and the finished flag is changed to 1.
        - If finished flag is 1, we break the loop.
        - Else, Decoding the block (using **hummingDecode** function) by checking parity bits (using **IsCheckBitWrong** function) which compares check parity bit value it should have based on calculation. If error was found we get it's index using hamming code calculations, and we correct it accordingly (using **flipBit** function).
        - Copying the (corrected, if needed) buffer to decodedBitsFileBuffer (using **copyToDecodedBuffer** function), writing block to section buffer (using **writeBlockToSectionBuffer** function) and initializing the values of the block buffers for file who are not divided by 26 (using **createBlockBuffers** function).
        - After that, if the section is not empty, translating the decoded char bits ('0'/'1') to bytes (using **translateSectionFromCharBitsToBytes** function):
            - Based on answer from: https://www.dreamincode.net/forums/topic/134396-how-to-convert-a-char-to-its-8-binary-bits-in-c/
            - The translation is done using a binary calculation - going over every byte in the section and translate the char bit to bytes, putting in bytesFileBuffer.
        - Writing section to file (using **writeSectionToFile** function) and initializing the values of the section buffers for file who are not divided by 26 (using **createSectionBuffers** function).
    - Closing the socket and the file.
    - Printing the relevant messages.
    - Asking the user to enter a new file name and reading it.
    - Initializing some of the variables.
- Cleaning up Winsock and exits successfully.

Channel Main Flow

- Parsing arguments (using **parseArguments** function) and initializing Winsock.
- Initializing listen sockets (using **initSenderSocket** and **initReceiverSocket** functions):
    - Calling "socket" syscall to create a TCP socket for listening.
    - Getting IP addresses (using **getIPAddress** function):
        - Based on answer from: https://www.geeksforgeeks.org/c-program-display-hostname-ip-address/
        - allocating memory for host name and IP address, calling gethostname function and then using it to call gethostbyname and get the host entry. After that, calling inet_ntoa to get the first IP address in the list received.
    - Creating address struct using inet_pton for converting IP string to an IP address and using htons of 0 (default value for port that will be changed in bind) to convert port to network endian.
    - Binding socket to port and listening to it.
    - Printing IP and port of socket.
- While user wants to continue:
    - Accepting sender and receiver connection to sockets (using **acceptConnections** function).
    - Creating buffer for data (using **CreateBuffer** function). Its size is 31 bytes (for 31 bit chars) and its name is dataBuffer.
    - While finished flag is not 1:
        - Reading 31 bytes from sender socket to dataBuffer (using **readOriginalDataFromSocket** function).
        - If set to random noise, adding random noise (using **addRandomNoise** function):
            - generating a random number using rand function and diving it by RAND_MAX to get a random number between 0 and 1.
            - Comparing it to the noise probability (value entered in cmd divided by 2^{16}).
            - If the number generated is smaller, flipping the bit (using **flipBit** function) and incrementing the relevant counter.
            - else adding deterministic noise (using **addDeterministicNoise** function) - For every cycle length received from cmd, flipping the relevant bit (using **flipBit** function) and incrementing the relevant counter.
        - Writing noised data (31 bytes) from dataBuffer to receiver socket (using **writeNoisedDataToSocket** function).
    - Closing connection sockets.
    - Printing the relevant messages.
    - Asking the user if they want to continue.
    - Initializing some of the variables.
- Cleaning up Winsock and exits successfully.

Bonus - Implementation for every file length

- Hereby listed the relevant additions and changes that were made to support every file length (and not just file length that is divided by 26 bytes):
  - Sender and Receiver
    - Changing buffers values to '\0' after every iteration to prevent using partially-right buffers (i.e. writing to the first x bytes of the buffer and the rest are left from previous iteration).
    - Before writing a block to socket, checking if it's not empty (i.e. first char is not '\0'). For file length that is divided by 26 bytes, all section is full in every iteration.
    - When reading section from file, allowing reading of less than 26 bytes, and checking EOF instead of allowing only reading of 26 bytes.
    - When generating parity bits, calculate result only based on non-'\0' char, for supporting partially-full buffers.
    - When writing section to file in receiver, we added using of strlen function in order to find the first occurrence of '\0' in buffer and getting the real number of bytes that will be written to file.
  - Channel
    - No changes were needed.