Lab exercises

The provided materials include:

```
Makefile
read_line.h
read_line.c
test.c
```

The file `read_line.c` includes the following functions:

```
int send_msg(int socket, char *message, int length);
```

This function includes code for sending a message of a determined length by a socket.

```
int recv_msg(int socket, char *message, int length);
```

This function includes code to receive a message from a determined length by a socket.

```
ssize_t readLine(int fd, void *buffer, size_t n);
```

This function reads form a file or socket descriptor a string with a maximum length of n bytes. If this number is exceeded, the remaining characters are discarded. The function returns a string that ends with the ASCII code 0. If the source is performed by the screen, the function returns when the line break is introduced. The newline character is not included in the string returned by the function.

The `test.c` includes code that uses the `readLine` function. Compile and test this program to understand how the above function.

Task 1. The objective of the first part of the lab is to implement in C code a client and a server using stream sockets.

The client performs the following actions::

1. It connects to the server in a given port as an argument passed to the program.
2. Run an infinite loop (which he gets away with Ctrl-c). In each iteration reads a string using the ReadLine function and sent to the server (the server will return the string back to the client).
3. The server receives the string and prints it.

The server (in this first task will be sequential) accepts a connection from a client and enter a loop in which the client will receive a string and return the same string back to the client. The loop will exit when the server receives the string "EXIT".

Task 2. The aim of the second part is to convert the server code using concurrent lightweight processes.

Task 3. In the third part of the laboratory, you have to implement a JAVA client that uses the server previously implemented. To do this, note that a client can create a stream socket executing the following lines:

```
Socket sc = new Socket(host, 4200);
```

The previous call creates a socket and connects with the machine running. Two data streams can be used to send and receive data from the socket. For example:

```
    DataOutputStream out = new DataOutputStream(sc.getOutputStream());
    DataInputStream  in  = new DataInputStream(sc.getInputStream());
```

The following code fragment sends a string through a socket:

```
    String message = new String("Hello");
    out.writeBytes(message);
    out.write('\0');      // insert the ASCII 0 at the end
```

The fragment used to read a string of the previous server is:

```
    byte[] aux = null;
    aux = new byte[256];

    in.read(aux);
    String s = new String(aux);
    System.out.println(s);
```