Alta Libreria Documentación

- structures.h
    - Enums:
        - MessageType: Nombre del header de la comunicación. Cada mensaje de cada módulo que se envíe tendría que tener un nombre en este enum
    - Structs:
        - MessageHeader: Estructura que contiene el nombre del header de la comunicación y el tamaño de la comunicación en sí.
        - t_paquete: Estructura que contiene a un MessageHeader y a un void*, esta es la estructura que luego se serializara y se enviara por sockets.

- connections.h
    - Encabezado de connections.c, nada especial, salvo el límite máximo de conexiones posibles en un servidor(Magic number?).

- connections.c (Solo tiene funciones, Ambos = Cliente y Servidor)
    - create_socket(): creó un socket para comunicarse,
      Return (Int) : **-1** ante error o el **valor del socket** en cuestión en caso de éxito.

    - bind_socket(Socket, Port): primero creo un socket con la función anterior y luego asignar un puerto al socket de escucha.
      Socket (Int): Número de socket Servidor que se va a conectar y se va a crear con la función creat_socket().
      Port (int): Puerto que se usa de coneccion.
      Return (Int): **-1** ante error o **0** ante exito.

    - connect_socket(N_socket, Ip, Puerto_servidor): Analoga de bind pero del lado del cliente.
      N_socket (int): un socket creado con la función create socket.
      Ip (char *): un puntero a una dirección ip.
      Puerto_servidor (int) : y un puerto
      Return (int): devuelve un **-1** en caso de error o un **0** en caso de éxito.

    - close_socket(Socket): Cierra los sockets creados en la función create_socket(), **SIEMPRE** cerrarlos luego de usados, de lo contrario, memory leak y otras cosas feas.
      Socket (Int): El socket que va a morir(Vas a morir socket wiiiii).
      Return (int) : **0**.

    - receive header(Socket, Buffer):
      Es una función bloqueante, por lo que se queda esperando hasta que reciba todo.
      Socket (Int): Socket que va a recibir el mensaje.

Buffer (MessageHeader *) : Es una estructura que describe al mensaje que vas a recibir (Ver arriba).
Return (Int): si es **positivo**, representa la cantidad de bytes recibidos, si es **0**, el server se desconecto y si es **-1**, hubo un error.

- start_server(Socket , **void** (*new_connection) ,**void** (*lost_connection), **void** (*incoming_message)):
  Por cada módulo las funciones se van a encargar de hacer algo distinto
  Socket (Int): Socket creado por creat_socket() y bindeado por bind_socket().
  void (*new_connection): Parámetros de función (**int** socket_cliente, **char** * ip_cliente, **int** port_de_coneccion).
  void (*lost_connection): Parámetros de función (**int** socket_cliente, **char** * ip_cliente, **int** port_de_coneccion).
  void (*incoming_message): Parámetros de función (**int** socket_cliente, **char** * ip_cliente, **int** port_de_coneccion, MessageHeader * header).

  Explicación en detalle: Creo un servidor al que le paso el socket creado y bindeado anteriormente, 3 funciones, una para para las conexiones nuevas, otra para las conexiones perdidas y otra para los mensajes nuevos, y finalmente una estructura MessageHeader, la función se queda en un bucle infinito escuchando las conexiones entrantes. Básicamente la función tiene un array de un tamaño máximo de conexiones que está predefinido(en el archivo connections.h), en cada iteración del bucle infinito, se llama a la función select, la cual me dice si hay alguna conexión nueva(en este caso se guarda en el array, y se llama a la función encargada de las nuevas conexiones), alguna conexión perdida o algún mensaje de alguna conexión anterior, luego de llamar a la función select, se itera sobre el array de conexiones(sockets), si hay algún mensaje nuevo sobre alguna conexión, se llama a la función que se encarga de los mensajes nuevos(pasada por parámetro), y finalmente, si hay algún alguna conexión que se perdió, se llama a la función correspondiente también pasada por parámetro y se quita a la misma del array. El MessageHeader pasado por parámetro sirve para saber el tamaño de los headers que va a recibir el servidor.

- create_package(MessageType):
  Esta funcion retorna un puntero a una estructura vacia del tipo t_paquete * a la que se van a cargar los campos a enviar por socket.

- add_to_package(t_paquete*, void*, int):
  Esta funcion recibe 3 parametros, el primero es un puntero al paquete creado con anterioridad, el segundo es lo que queramos agregarle al paquete(tiene que estar casteado a un void*), y finalmente el largo de lo que queremos agregar. No retorna nada.

- send_package(t_paquete*, int):

Esta funcion recibe 2 parametros, el primero es el paquete a enviar, el segundo es el socket al cual enviarlo. Retorna -1 en caso de error o la cantidad de bytes enviados en caso contrario.

- serialize_package(t_paquete*, int)**FUNCION SOLO DE USO INTERNO**:
  Esta funcion recibe 2 parametros, el primero es un puntero a un paquete, el segundo es el largo del mismo. Retorna un void* con la estructura del paquete serializada.

- free_package(t_paquete*):
  Esta funcion recibe un un puntero a un paquete, cuando el paquete ya no es necesario, se **DEBE ELIMINAR** para liberar la memoria solicitada por el mismo. No retorna nada.

- receive_package(int, MessageHeader *):
  Esta funcion recibe un socket y un puntero a un MessageHeader, se debe utilizar en conjunto con la funcion receive_header, primero se utiliza receive_header(En el caso del servidor, cuando se va a utilizar la funcion incoming, la misma ya recibe el header como parametro, porque internamente el servidor lo obtiene para realizar ciertas verificaciones, por lo que no es necesario utilizar la funcion anterior, con pasarle a esta funcion el encabezado recibido ya alcanza), y el puntero a un header recibido por la misma se debe pasar a esta funcion para recibir el resto de la estructura. La funcion devuelve un puntero a una estructura de las commons del tipo t_list, en la misma cada elemento de la lista es uno de los elementos agregados al paquete original.

Metodo de implementacion

- Primero, obviamente, importar la biblioteca a nuestros distintos proyectos
- Cliente(libmuse, libsuse/hilolay)
  a. Llamar a create_socket()
  b. Llamar a connect_socket(), pasarle el socket anterior, la ip y el puerto de envio
  c. Llamar a create_package() para crear un paquete vacio, pasarle el MessageType correspondiente al mensaje a enviar.
  d. Agregar todos los campos necesarios al paquete mediante la funcion add_to_package(), pasarle el paquete al cual agregar, el elemento a agregar casteado a un void* y el largo del elemento.
     - Enviar datos: llamar a send_package(),pasarle el paquete y el socket al cual enviar.
     - Recibir datos:

            1. Llamar a receive_header(), pasarle el socket de recepcion, y un puntero a MessageHeader para saber que header voy a recibir en el siguiente mensaje.
            2. Llamar a receive_package(), pasarle el socket de recepcion y el puntero al header recibido en la operacion anterior.
        - Hacer lo que se tenga que hacer con los datos anteriores, repetir si es necesario.
    e. Llamar a free_package() para liberar la memoria solicitada para almacenar el paquete.
    f. Llamar a close_socket() para cerrar el descriptor de archivo.
- Servidor(Muse, Suse, Fuse?)
    a. Llamar a create_socket()
    b. Llamar a bind_socket(), pasarle el socket anterior y el puerto de escucha.
    c. Crear las funciones de nueva conexion, nuevo mensaje y conexion perdida, las mismas tienen las siguientes firmas:
        - void (*incoming_message)(int fd, char * ip, int port, MessageHeader * header))
        - void (*lost_connection)(int fd, char * ip, int port)
        - void (*new_connection)(int fd, char * ip, int port)
        - fd es el socket correspondiente a la conexion, ip es la IP del proceso que está del otro lado de la conexión(?), port es el puerto de la conexión anterior y MessageHeader, es la estructura que expliqué arriba.
        - Dentro de cada una de estas 3 funciones se puede usar, o no, las funciones para recibir o enviar datos explicadas en el cliente. La unica variacion es la funcion encargada de recibir un paquete, ya que como el servidor utiliza al header de los mensajes entrantes(funcion incoming) para hacer unas verificaciones, no es necesario pedirlo mediante receive_header(), directamente utilizar receive_package().
    d. Crear un hilo nuevo con pthread o lo que sea y pasarle como funcion a start_server(), a la misma se le debe pasar el socket creado y bindeado, pasarle las tres funciones creadas en el punto anterior y el puntero a los MessageHeader a recibir.
- Finalmente, divertirse.