**CMPSC 311 – Technical Report**

**Introduction:**

The code is an implementation of a client-server system designed for a JBOD (Just a Bunch of Disks) storage system. The client connects to the server and sends operations to be performed on the JBOD, such as reading or writing data blocks. The code is written in C and utilizes various libraries and system calls. This report aims to explain the flow of the code and how it performs its functions.

**Main Function:**

The code's main function is jbod\_client\_operation, which takes an operation code (op) and a block of data. It sends the op and block to the server using the send\_packet function and receives a response from the server using the recv\_packet function. It then returns the response code (ret).

**Connection:**

Before the main function is executed, the client must connect to the server using the jbod\_connect function. This function takes an IP address and port number as input, creates a socket and connects to the server using the specified IP and port. If the connection is successful, the cli\_sd variable is set to the socket descriptor for the connection.

**Disconnection:**

After the client has completed its operations, it should disconnect from the server using the jbod\_disconnect function. This function simply closes the connection by calling the close function and sets the cli\_sd variable to -1.

**Packet Sending:**

The send packet function is responsible for packaging the operation code and data block into a packet and sending it to the server. The packet consists of a 2-byte length field, a 4-byte operation code field, and a variable-length data block field (if the operation code is for writing data). The length field includes the length of both the header and data block. The operation code is converted to network byte order (big-endian) using the htonl function, and the length field is converted using the htons function. If the operation code is for writing data, the data block is included in the packet. The nwrite function is used to write the packet to the server's socket descriptor.

**Packet Receiving:**

The recv\_packet function is responsible for receiving a packet from the server and unpackaging it into the operation code, response code, and data block (if the operation code is for reading data). The packet is read in chunks using the nread function until the entire header has been received. The length, operation code and response code fields are then extracted from the header and converted to host byte order (little-endian) using the ntohs and ntohl functions. If the length field indicates that there is a data block to be received, the nread function is used to receive the data block.

**The library defines the following global variable**: cli\_sd is an integer that holds the client socket descriptor for the connection to the server. It is initialized to -1.

**Below is a detailed explanation of all my functions:**

1. bool nread(int fd, int len, uint8\_t \*buf): This function attempts to read n bytes from the file descriptor fd. It takes in the fd, the number of bytes to be read len, and a buffer buf to store the read data. The function reads len bytes of data from fd and stores it in buf and returns true on success and false on failure.
2. bool nwrite(int fd, int len, uint8\_t \*buf): This function attempts to write n bytes to the file descriptor fd. It takes in the fd, the number of bytes to be written len, and a buffer buf containing the data to be written. The function writes len bytes of data from buf to fd and returns true on success and false on failure.
3. bool recv\_packet(int fd, uint32\_t \*op, uint16\_t \*ret, uint8\_t \*block): This function attempts to receive a packet from the file descriptor fd. It takes in the fd, a pointer to op, a pointer to ret, and a buffer block. The function reads a packet from fd, and unpacks the packet's header to retrieve the operation op, the return value ret, and the packet's length. If the packet's length is equal to HEADER\_LEN, the function returns true. If the packet's length is equal to HEADER\_LEN+JBOD\_BLOCK\_SIZE, it reads an additional 256 bytes of data from fd and stores it in block. The function returns true on success and false on failure.
4. bool send\_packet(int sd, uint32\_t op, uint8\_t \*block): This function attempts to send a packet to the file descriptor sd. It takes in the sd, the operation op, and a buffer block. The function creates a packet with a header that includes the length of the packet and the operation op. If the command is a JBOD\_WRITE\_BLOCK, the packet also includes the data to be written block. The function then sends the packet to sd and returns true on success and false on failure.
5. bool jbod\_connect(const char \*ip, uint16\_t port): This function attempts to connect to a server with the given IP address ip and port port. It creates a socket and attempts to connect to the server. If the connection is successful, it sets the global variable cli\_sd to the socket descriptor and returns true. If the connection is not successful, it returns false.
6. void jbod\_disconnect(void): This function disconnects the client from the server by closing the socket descriptor cli\_sd and resetting it to -1.
7. int jbod\_client\_operation(uint32\_t op, uint8\_t \*block): This function sends a JBOD operation op to the server and receives and processes the response. It sends the operation to the server using send\_packet() and receives the response using recv\_packet(). It then extracts the return value from the response and returns it to the caller.

**Conclusion:**

The code implements a client-server system for a JBOD storage system. The client connects to the server, sends operations to be performed on the JBOD, and receives responses from the server. The flow of the code is as follows: connect to the server, send an operation packet, receive a response packet, and disconnect from the server. The send\_packet function packages the operation code and data block into a packet and sends it to the server, and the recv\_packet function receives a packet from the server and unpackages it into the operation code, response code, and data block.