# CITS3002 Report

| Name        | Student Number

| ------------- |:-------------:|

| Thanh Nguyen       | 22914578 |

| Anfernee Pontilan Alviar      | 22886082      |

| Ethan Pui | 22704879      |

Date Submitted:

Lecturer: Dr Chris McDonald

-------------------------

1. the protocol you have designed and developed for all communication between your client and server programs,

2. a 'walk-through' of the execution sequence employed to compile and link an multi-file program, and

3. the conditions under which remote compilation and linking appears to perform better (faster) than just using your local machine.

<!-- ## Success criteria

- Create connection between local host and remote hosts.

- Use protocols to ensure client and servers are able to communicate via integers and strings.

- Send meaningful data between local hosts and remote hosts.

- Have multiple instances of server programs to service the client.

- Perform compilation of a C program on a remote host.

- Perform compilation of a C program on multiple remote hosts concurrently.

- Have remote hosts spawn child processes to services connections.

- Have the client send and receive transmissions in a non block fashion using select()

# -->

-------------------------

## Protocol Design

##### The Protocol we have designed for all communication between client and server programs

### Encoding type

Integers will be encoded into 8 bytes using the big edian byte order, where the most significant bytes will be stored first, at the lowest storage address. Strings will be encoded using the utf-8 standard.

When communications are made between client and server, the client must make the initial connection, the first type of communication must be represented as a 4 byte integer followed by any number of payloads. Once the entire datagram is sent by the client, the client will be in wait mode, until an acknowledgement has been received from the server, then more datagrams can be sent, in the same order of a 4 byte integer followed by one or more payloads.

The 4 byte integer will prompt the server or the client what kind of payload to expect, this will set the receiving connection into a state in which it can properly accept the incoming payload.

Standard strings will be encoded in utf-8 bytes before sending and decoded on the otherside.

### Cost Request Protocol

When the initial connection is accepted by the server the client will send a integer represented by CMD\_QUOTE\_REQUEST padded to 4 bytes in big edian byte order.

Once the preamble is accepted by the server it will return a an integer represented by CMD\_QUOTE\_REPLY followed by the cost also in an integer and all padded to 4 bytes and using the big edian byte order.

<div style="text-align: center;">

    <img src="/diagrams/cost\_protocol.png" alt="cost protocol" width="400" height="400">

</div>

### Send File Protocol

Once the appropriate file is located and buffered into memory, the sending connection will first start by

1. sending the preamble CMD\_SEND\_FILE if the file is a text file or CMD\_BIN\_FILE if the file is a binary file.

2. The size of the name of the file, followed by the name of the file, formated in utf-8 encoding

3. The size of the actual file following our established 4 byte big edian standard.

4. Send the contents, if in text format it will be encoded in utf-8 otherwise the binary will be sent.

5. Wait for an acknowledgement from the server - CMD\_ACK - to prompt the client the sver is ready for the next file.

6. If more files need to be sent repeat 1-4

### Execute Command Protocol

Once the server has all required files, the client will do the following.

1. Send the preamble CMD\_EXECUTE

2. The command will be in string format so we first encode it in utf-8 bytes and send the size, followed by the command.

3. Wait for a return code, this will be received as an integer

4. If the return code does not equal zero, something went wrong and the server will send the error message.

    - The server will send the size of the message as an integer followed by the message, the message will have the utf-8 encoding

5. If the return code equals zero, the client will wait the file. Protocol for receive file below.

<center>

    <img src="/diagrams/execute-cmd-protocol.png" alt="execute cmd protocol" width="400" height="500">

</center>

### Receive File Protocol

The receiving socket will be in a state to receive incoming data.

1. Evaluate the preamble.

    - CMD\_BIN\_FILE

    The receiver will expect to write bytes to a file

    - CMD\_SEND\_FILE

    The receiver will expect to write ut8-8 chars to a file

    - CMD\_RETURN\_FILE

    The client will expect a binary file and write bytes to a file

2. The preamble will prompt the receiver what kind of file to expect.

3. The size in bytes of the name of the file will be sent, then the name will be sent encoded in utf-8 bytes

4. The size in bytes of the contents, followed by the contents.

    - if the receiver is expecting text (CMD\_SEND\_FILE) it will simply decode and write to a text file.

    - otherwise the receiver is expecting a binary file, in that case the receiver is write the byte directly to the file.

5. If the receiver is in CMD\_RETURN\_FILE mode it will simply close the connect, as this mode is only ever each at the end of a action set. Otherwise a CMD\_ACK will be sent and repeat step 1-5

#

-------------------------

## Walkthrough execution sequence employed to compile and link and multi-file program

TODO: maybe remove this section when not needed it just repeats everything

1. Request for cost

2. Pick the server

3. Start sending files

    1. Client first sends filename and size

    2. Server receiving and writing txt files

    3. Once all files are sent the client will send the execute command

    4. Server will execute the command sent, and the reuturn status will be sent to the client. If the return staus is zero the server will initiate the protocol to send the output file, otherwise it will return the error message from executing the command.

    5. Sending the output file, since the output file

---------------------

### Walkthrough of server and client interactions

(Mention functions being called and what it returns

just follow one socket and its journey

first do just the client

then do the server)

1. Servers are created by running (python3 rakeserver.py 6328). Opening an port based on the command line third argument and listens for connections on that port number and IP address.

2. Client is created by running (python3 rake-p.py Rakefile). Client creates a connection to a given host on a given port given to the function create\_socket as an parameter. Client then connects the socket and returns the connection object in the create\_sock function for quote while listening for servers.

3. When connection is established with an server the client will initate the connection by sending a preamble represented in 4 bytes and in edian byte order (CMD\_QUOTE\_REQUEST).

4. The server that is waiting for this preamble recieves (CMD\_QUOTE\_REQUEST) and sends back an preamble (CMD\_QUOTE\_REPLY). If the preamble is not correctly converted on the client side the server will not be able to recognize the preamble and thus do nothing.

5. The client waiting for an reply recieves the preamble (CMD\_QUOTE\_REPLY) amd waits for an second reply of the cost in edian byte order. Server sends cost and closes the connection, client recieves the cost and closes the connection.

6. Steps 3 to 5 are repeated until the client has found server with lowest cost.

7. Client creates and connects sockets for executing commands on the server with lowest cost. Client determines if it requires files by reading the Rakefile. If it does require files, client sends required files to the server by sending an preamble (CMD\_SEND\_FILE) in big edian byte order.

8. Client will return code status detailing if there was an error or sucess when executing commands. On success the server will send a file and the client will receive the output file from command. If it was an error, client will output an error message and exit the program immediately. The execution of commands process is shown in the diagram (execute-cmd-protocol.png).

<center>

    <img src="/diagrams/client\_flow.png" alt="client flow" width="400" height="500">

</center>

## Performance

##### conditions under which remote compliation and linking appears to perform better (faster) than just using your local machine

TODO: EDIT rambling just some ideas. we can talk about LANs vs WLANs vs internet. like would it be quicker on a LAN or over the internet where hosts can be on the other side of the contienent. Limitations of bandwidth connections. security maybe?

When compiling sufficiantly large programs on a local machine, the compilation process completes in a sequential order. Therefore, if an object file does not have any dependencies, it still has to wait for its turn to compile. When remote compilation is possible, we can decompose the requirements into "action sets", and have a network of computers run in parallel the compilation process for individual object files. For example, assuming a compilation where object file do not have many dependencies such as the program.c example then we can send just the files need create the object file and have the local machine perform the final compilation.

## Observations and improvements

I have observed that garbled signals are possible, this could be due to our protocols not waiting or checking for errors as it just sends streams of bytes.

Add timers and check for errors, though ethernet is reliable, it could be too reliable by sending transmissions so quickly.

## Conclusion