General Program Structure and implementation

1. netreqchannel.cpp:
    a. client-side constructor

    The client-side constructor is fairly simple. First, the constructure input _port_no needs to casted to string to be the correct type for function calls. After, we create a struct sockaddr_in to assign the transport address to out sockIn. Next, we create a servant structure using tcp at the desired port. We update our sockIn port to match. We assign the hostname to the struct as well. Finally, we create a socket, verify connection, and assign the socket file descriptor.

    b. Server-side constructor

    First a quick note about the given function parameters: Here we need to add an additional backlog parameter to allow the dataserver listen(). Additionally, in order to use pthread_create we need the start routine (in our case connection_handler) to be of the form "void * (*start_routine) (void *)" instead of "void * (*connection_handler) (int *). Lastly, as fd is a private member, a read_fd() function is needed.

    The server-side constructor takes a little more involvement. Again, we need to stringify the given _port_no. Like before, we create a sockaddr_in serverIn struct to manage the connection and set the serverIn structure. As this is the server-side we bind the socket. We place the socket in a listening statue using the desired backlog. Lastly, we create an infinite while loop to accept and create threads for connections until forcing a stop.

    c. Deconstructor

    All the deconstructor does is close the socket file descriptor

    d. Cread()

    Cread() reads a max 255 bytes from the socket file descriptor into a buffer and returns said buffer

    e. Cwrite

    Cwrite() c_strings the given message and writes it to the socket file descriptor

    f. Read_fd()

    Simply returns the private member fd

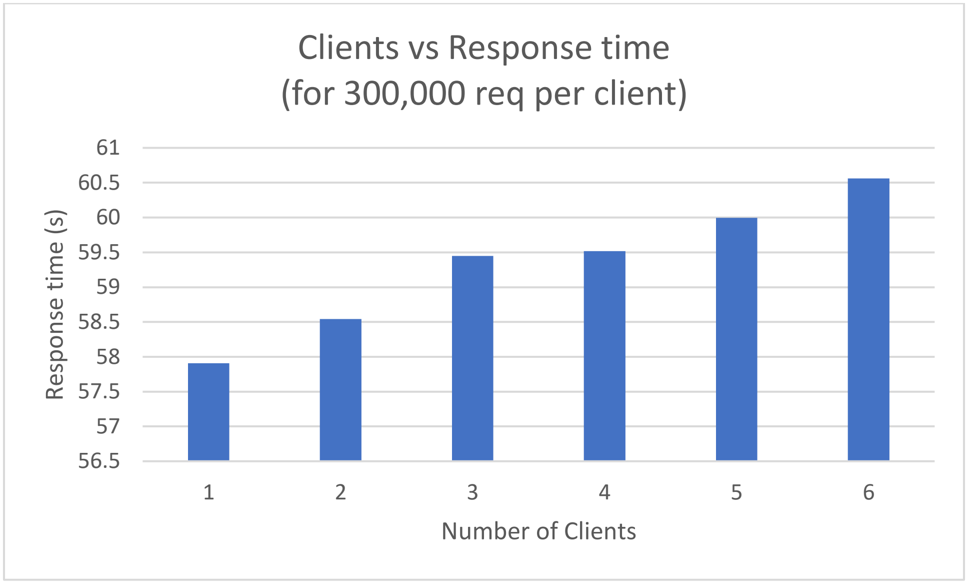
2. client.cpp:

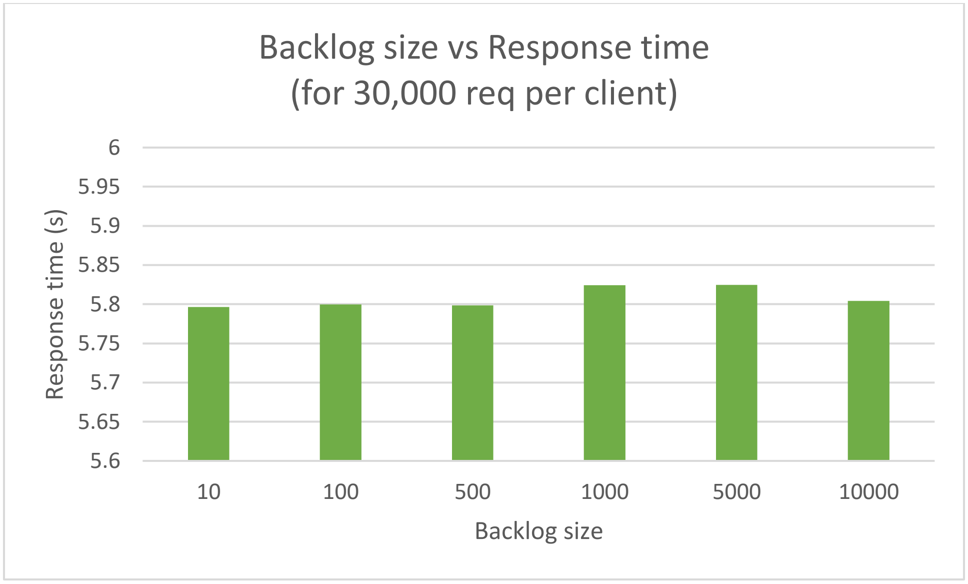
    Client.cpp uses MP4s implementation. The only change is the removal of the control channel, and the syntax for calling the NetworkRequestChannel inside of the event_thread_fnct()

3. dataserver.cpp

    dataserver functions are similar to previous machine problem dataserver uses with updated parameters and helper functions to suit the new network request channel. However, dataserver main() has been updated to intake command line backlog and port number, now creates the updated network request channel and has to explicitly call the deconstructor.

Performance





At least two terminals are needed to run this implementation. One for the dataserver and one per client. As testing with few data requests returned too quickly to allow for multiple clients to be run simultaneously by me, so I sent 100,000 data requests per person (300,000 total) per client. Per each client addition, about 1 second was added to the request returns. Incrementing the datserver backlog from 10 – 10000 did not show considerable performance difference ~0.02s max discrepency.