**Define the following**

.z.a - This is the IP address of the host that the q process is running on in integer form

.z.h - The hostname of the system that the q process is running on

.z.u - Username of the currently logged in user of the q process

.z.w - Gives the value of the currently calling handle. If called dirctly on a process, it will always return 0i. However if called remotely it will give the handle of the remote process.

**Message Handling**

1. **Define what each of the following is, including any parameters which are passed to it.**

.z.pg - This function runs on a process whenever a synchronous IPC call is made to it. Takes an expression. By default it simply executes (using value) the passed in expression.  
.z.ps - Similar to .z.pg except that this function is called whenever an asynchronous call is made to a process. Works in the same way as .z.pg

.z.po - This function is called whenever a connection is made to a process via a hopen call. Doesn't take a parameter. Default behaviour is to do nothing  
.z.pw - Used to provide user authentication for a process. Is called whenever a hopen call is made from another process. (It is executed before .z.po). Inputs are userID<symbol> and password<string>. Default behaviour is to allow all connections to have access.  
.z.pc - Called whenever a connection is closed to a process via hclose. Same in all other respects to .z.po  
.z.ph - Called whenever a synchronous HTTP request is received by a q process. Passed a single argument (as a list): (requestText;requestHeaderAsDictionary). These contain the details of the request and are parsed in order to return the correct information to the calling process.  
.z.pi - Process input. Used to hande any kind of input. Is called whenever an input is given to the process (e.g. command line expression). Parameter is an expression that is to be executed. Useful for logging purposes. Default behaviour is to just display output as normal.

1. **Which message handler is to control access to a process (username and password authentication)?  Write a small example which will allow access to the process only for a connection which supplies the username “user1” or “user2” along with the password “mypass”.**.z.pw:{[username;password]   
    //Obviously this would never be done in practice but it suffices for this example

authDict:`user1`user2!2#enlist "mypass";  
if[not username in key authDict; :0b];   
if[not password like authDict username; :0b];   
1b  
}

1. **Modify the synchronous message handler such that only stored procedure calls are allowed, where the stored procedure is defined on the server.  Return the error “not a stored proc call” if the user tries to request something else, for example tries to do a freeform select statement.**

.z.pg:{  
 //Permit func calls in string form or general lists with function as first argument

$[0h = type x;  
 call: `$raze string x[0];  
10h = type x;  
 call:`$x [til 0 | first ss[x;"["]];   
 '"not a stored proc call"

];  
//Only looking in the top level namespace for stored procedures  
if[not call in value"\\f";   
 '"not a stored proc call"  
];   
value x  
}

1. **If a message is sent asynchronously from a q process to a client (q or some other technology) and the client is busy processing something else, where is the message held (buffered)?  Can you think of any problems this might cause in a standard kdb+ tick set up?**

The message is buffered on the sending process. This could be a problem in a kdb+ tick system when a process that receives live updates from a tickerplant. If the process goes down, updates will get backed up in the buffer on the tickerplant. If this goes on for too long the tickerplant could end up with memory problems and in severe circumstances, it could even crash.

1. **A client opens a connection to a server and sends a message synchronously to it.  Outline all the message handlers which are invoked (in order) and any parameters which are passed into them.**

Opening a connection  
Firstly, if the server process has the -u flag enabled, the process handle that was used in hopen is parsed and the username<symbol> and password<string> are passed into .z.pw. This checks if a user is authorised to open a connection to that server process   
Secondly, if the user passes the authentication step, .z.po is called. No arguements are passed in.  
Sending a synchronous message  
.z.pg is called. The input is an expression that is supplied from the calling process. The exresion is run and the result is returned to the calling process.

1. **Imagine you have been tasked with writing a q process which monitors the response times of other server q processes.  It must ping each server every 30 seconds and measure the response time.  All the q processes are single threaded, and if one server is slow or blocked, the monitoring process should not become blocked.  The processes-to-be-monitored should not have to have any additional code defined on them i.e. the monitor should be able to control how and when it pings the remote process, and the response it gets back.**

.bc.responseTimes:()!();

pingResponse:{[pingStartTime]

//Only the most recent ping times are recorded

.bc.responseTimes[.z.w]:.z.p - pingStartTime

}

pingServer:{[h]

neg[h]({neg[.z.w](`pingResponse;x)};.z.p)

}

.z.ts:{

pingServer each key .z.W

}

\t 30000

1. **Using the .z.pg and .z.ps functions create a loadable q script that.**

* **Logs all sync and async messages received to a table.**
* **Each message should store the senders handle in the process , the senders ip, the senders username,async/sync flag, time taken for the query and the message as a string.**

**N.B Careful not to store the active processes ip,user and handle in error.**

.bc.messageLogs:([] sendingHandle:`int$(); sendingIP:`int$(); sendingUsername: `$(); reqType:`$(); queryTime: `timespan$(); queryString: 0h$());

logFunc:{[query;queryType]

//Don't want to log the logging function!

$[`logQuery ~ query 0;

//If the logging function has been passed in, simply run it

value query;

[

//Other wise perform normal logging operations

tmp:runAndTimeQuery[query];

time:tmp`time;

res:tmp`result;

//Compile the already known info into a dictionary

logDict:`sendingHandle`reqType`queryTime`queryString!(.z.w;queryType;time;$[0h = type query;![-3;];::] query);

//Use an asynchronous call and callback to gather username and ip information from the calling process

neg[.z.w]({[otherInfo] neg[.z.w](`logQuery;otherInfo;.z.u;.z.a)};logDict);

:res

]

];

}

.z.pg:{[query]

logFunc[query;`synch]

}

.z.ps:{[query]

logFunc[query;`asynch]

}

//Callback function that adds a query with all of the relevant info to the message log

logQuery:{[initDict;username;ip]

`.bc.messageLogs upsert initDict , `sendingUsername`sendingIP!(username;ip);

}

runAndTimeQuery:{[query]

//Prepare the query for timing and running code

//-3! gives string version of (func;param1;param2;...) form queries

//String queries need literal quote marks around them in order for the run logic to work below

$[10h = type query;

queryString: "\"", query, "\"";

queryString:-3!query

];

//Time, run and save result in res

tm:`timespan$value "\\t res: value ", queryString;

`time`result!(tm; res)

}

1. **List the possible parameter combinations of the hopen function. i.e `::port .**hopen takes the following parameters in this format:  
   processDetails: `:host:port:username:password  
   timeout: <long> time to wait for connection to open in millliseconds.
2. **Using .z.po create a dictionary of all functions from the connecting process. (i.e A connects to B and B creates a dictionary of all functions and function defs from A)**

.bc.remoteFuncs:()!();

.z.po:{  
 //Only look for functions in the root namespace  
 .bc.remoteFuncs[.z.w]:.z.w"\\f";

}