This report is based on 4.19.68. At first, I will follow the way that server-side binds socket on address and port. From the code in server side, the program will create a socket, then bind the socket in an address and a port and next listen on the socket. After that, the program will wait until the incoming network connection trigger the accept function. Next, the incoming connection will trigger the write and read. At last, the socket will be closed.

The sys\_socket is the first network related syscall and this syscall uses \_\_sys\_socket function (/net/socket.c:1326). This function call \_\_sock\_create(/net/socket.c:1205) in order to allocate the socket in sock\_alloc() (/net/socket.c:543) and allow the family to set things up, and also call \_\_sock\_map\_fd (/net/socket.c:408) and sock\_alloc\_file (/net/socket.c:387) which create file structures and maps them to fd space of the current process in order to obtain the first available file descriptor and sets it up for use.

After that, the program will bind the socket in an address and a port, which is done by \_\_sys\_bind() (/net/socket.c:1468) function. This function will call sockfd\_lookup\_light() (/net/socket.c:468) to get the socket from fd, call move\_addr\_to\_kernel() (/net/socket.c:182) to move the socket address to kernel space before we call the protocol layer and then call the sock->ops->bind() in the returned socket struct pointer. I have found the definitions of socket operations on /include/linux/net.h, and the implementation of bind operation on inet\_bind() (/net/ipv4/af\_inet.c:431). In inet\_bind(), it will check the sk pointer, which is internal networking protocol agnostic socket representation, and if the socket has its own function then calls it or calls lock\_sock\_nested() (/net/core/sock.c: 2695) to occupy the address and port.

Next, the program will perform a listen syscall by \_\_sys\_listen() (/net/socket.c:1502). This function will also call sockfd\_lookup\_light() to get socket pointer by fd and then sock->ops->listen(), which will call inet\_listen()(/net/ipv4/af\_inet.c: 194) () . In inet\_listen(), the state and type of socket will be checked, and then the program will call sysctl\_tcp\_fastopen() to listen on the socket.

Furthermore, the program will call \_\_sys\_accept4() (/net/socket.c:1540). After creating the new socket, this function will call sock->ops->accept(), which is inet\_accept() (/net/ipv4/af\_inet.c:729) and will wait for the TCP connection. And then, the server will set up the link with the client, wake up the client, then return the new connected fd.

O the client side, after creating the socket, the client will call \_\_sys\_connect() (/net/socket.c:1650). This function is a little similar with bind() because it calls sockfd\_lookup\_light() and then move\_addr\_to\_kernel(). The most important part is sock->ops->connect() which invokes \_\_inet\_stream\_connect() (/net/ipv4/af\_inet.c:603). \_\_inet\_stream\_connect() will check the state of socket, if the socket is SS\_UNCONNECTED, then this function will try to connect it by sk->sk\_prot->connect() (I have tried but I couldn’t find where it point to. I only found the struct (/include/net/sock.h:1058) and the constructor sk\_alloc()(/net/core/sock.c:1515).

On the server side, the message will be sent by \_\_sys\_sendto() (/net/socket.c:1756). And this function will get the socket by sock\_fd\_lookup\_light(), move\_addr\_to\_kernel(), and then call sock\_sendmsg() (/net/socket.c:627) in order to send message. On the client side, the program calls \_\_sys\_recvfrom (/net/socket.c:1817). This function get the socket by sock\_fd\_lookup\_light() , then call sock\_recvmsg() (/net/socket.c:798) in order to get the message from server, and next call move\_addr\_to\_user().

At last, the program will call \_\_sock\_release() (/net/socket.c:572) to release the socket. This function will call inet\_sock\_destruct()(/net/ipv4/af\_inet.c:134) to do destruct, which reclaim the memory, queue and counter.