# TCP RST: Calling close() on a socket with data in the receive queue

Consider two peers, A and B, communicating via TCP. If B closes a socket and there is any data in B's receive queue, B sends a TCP RST to A instead of following the standard TCP closing protocol, resulting in an error return value from recv( ).

| **A** | | **B** |
|---|---|---|
| send( ) | data → | |
| | data → | |
| | data → | |
| recv( ) →ERROR | ← RST | close( ) |

When might this situation occur? Consider a simple protocol where A sends 5 prime numbers to B, and B responds with "OK" and closes the connection. If B receives a nonprime number, it assumes A is confused, responds with "ERROR", and closes the connection. If A sends 5 numbers and the second number is not prime, then B will send "ERROR" and call close( ) with data (3 more numbers) in its receive queue. This will cause a TCP RST to be sent to A. Meanwhile, A is blocked on recv( ) awaiting word from B. The RST from B causes A's recv( ) to return an error (-1) so A never receives the "ERROR" message from B.

So why not just have B read all 5 prime numbers before closing? Consider the case where A sends 6 prime numbers. Here B reads the first 5 prime numbers and closes the socket with the 6th prime still in its receive queue, resulting in a RST.

We can view this RST behavior with a minor modification to the TCP Echo Server. To make this modification, simply compile the server with the following version of HandleTCPClient( ):

```
#include <stdio.h>      /* for printf() */
#include <sys/socket.h> /* for recv() */
#include <unistd.h>     /* for close() */

void DieWithError(char *errorMessage);  /* Error handling function */

void HandleTCPClient(int clntSocket)
{
    char recvChar;          /* Character to hold received data */

    /* Receive a single byte from the client */
    if (recv(clntSocket, &recvChar, 1, 0) < 0)
        DieWithError("recv() failed");

    printf("Received a byte...now closing\n");

    /* shutdown(clntSocket, SHUT_WR); */
    close(clntSocket);
}
```

You may download this code from http://cs.baylor.edu/~donahoo/practical/CSockets/code/HandleTCPClientRST.c. The new HandleTCPClient( ) receives a single byte and closes the connection. First, try sending a single byte. It works great! However, if the client sends more than one byte, this new server will read the first byte and call close with data in its receive queue, causing a RST to be sent to the client. The client will be blocked on a call to recv( ) awaiting the response from the server. Upon receiving the RST, the client's recv( ) returns –1 and DieWithError( ) reports the problem as follows:

```
recv() failed or connection closed prematurely: Connection reset by peer
```

Note the “Connection reset by peer”. This is the system’s explanation for the failure of the call to recv( ). That is, the peer (server) sent a TCP RST to the client, terminating the connection.

So how do we fix this? You might have already guessed it from the source code. We can use shutdown( ). If we shutdown the write direction of the connection before calling close, we avoid the RST behavior. (See the man page and Chapter 6 for an explanation of shutdown( )). To see this, uncomment the call to shutdown( ) in the new HandleTCPClient( ). **Do not remove the close( )**. If we run the same experiment as before, the client’s DieWithError( ) reports:

```
recv() failed or connection closed prematurely: Success
```

In this case, the system’s message is “Success”. **This means that recv( ) returned 0.** Recall that the echo client wants to read as many bytes as it sent to the echo server so this error message indicates that the server sent back fewer bytes than the client sent, a failure of the echo protocol and not recv().

Note that this RST problem does not occur in all operating systems. Some simply ignore the receive queue.