What is a CLOSE_WAIT socket, and what does it mean?

SOLUTION VERIFIED - Updated July 23 2017 at 3:47 PM - English ()

Environment

- Red Hat Enterprise Linux, all versions
- TCP

Issue

- When running an application, it's possible that the application may produce a socket in CLOSE_WAIT state from a state such as ESTABLISHED or TIME_WAIT.
- What does this state signify, and how can it be adjusted?
- If I have an application with many of these sockets, what are the implications of this?
- Getting close wait connections on Service mix server. We have observed close wait socket connections on windows server and as per discussion from windows and network team we have found that it’s an application side issue and it was closing connections:
- Port 8081 running “JAVA.exe” has CLOSE_WAIT sockets.
- How to Kill Or Reduce TCP Close_Wait connections?
- Is there a way to terminate the tcp close_wait sessions in Linux ? In general, they go away automatically on its own . But is there a way to Speed up this process ?
- Why there are many 'CLOSE_WAIT' sockets seen in "netstat" command output ?
- When running netstat command there are many CLOSE_WAIT state sockets.
- Sockets are seen in this state even after the default of TCP connection timeout.

Resolution

- A CLOSE_WAIT socket is a socket in which the connection has received a FIN and has yet to be closed by the application. In this state, it is the responsibility of the application to close the socket, although the operating system will report the socket in this state under any of these circumstances:

  - The socket is empty, and the application needs to relinquish the file descriptor of the socket back to the operating system, which will effectively remove the socket (and will subsequently disappear from netstat output).
  - There is still data in the socket which the application needs to read() from the file descriptor of the socket in order to be closed fully.
  - The connection is still in-use for unidirectional transmission by the application.

- In all of the above scenarios, there is no tuning from an operating system perspective that can close this socket without killing the running process ID of the application, and subsequently closing the application. Otherwise, the application will need to close the socket via a call such as close().

- A socket in CLOSE_WAIT may possibly consume TCP memory if it is not closed properly, of if the received data in the buffer is not read into the application, as this space will consume the total amount of TCP memory available on the system (visible in /proc/net/sockstat).

- A buildup of sockets in CLOSE_WAIT state may indicate an application programming error.

- For Java, see also http://stackoverflow.com/questions/5636316/troubleshooting-connections-stuck-in-close-wait-status (http://stackoverflow.com/questions/5636316/troubleshooting-connections-stuck-in-close-wait-status) and http://serverfault.com/questions/160558/how-to-not-get-so-many-apache-close-wait-connections (http://serverfault.com/questions/160558/how-to-not-get-so-many-apache-close-wait-connections).

Root Cause

A CLOSE_WAIT socket is a socket state allowing the application to perform tasks on a socket that has been advised by the client that no further data will be sent by the client, before it is relinquished to the operating system. The socket is still usable for the application to send data to the client. Therefore, there is no direct way to tune this.

Possible causes include:

- deliberate use in this mode by the application
- application fault resulting in failure to close the socket
- file descriptor leak

Diagnostic Steps

- This will need to be handled from within the application, however, here is a very simple example using C wherein we have a telnet server:

```
    while(1)
    {
        connection_file_descriptor = accept(listening_file_descriptor, (struct
socket_address*)NULL, NULL);
        sleep(1);
     }
```

- In the above example, a client may connect to the server. While the client remains connected, the output of netstat will report the connection in ESTABLISHED state. However, if the client were to send a FIN packet to close the connection, the state of the socket will move into CLOSE_WAIT state as the operating system is awaiting the application to close the socket.
- In the above example, if we wanted to remedy that situation simply, we could add a line such as this:

```
    while(1)
    {
        connection_file_descriptor = accept(listening_file_descriptor, (struct
socket_address*)NULL, NULL);
    close(connection_file_descriptor);
        sleep(1);
     }
```

- The `close()` function above closes the file descriptor of the socket, which then ends the connection, removing the output from netstat. This example simply closes new connections, so it is likely that in a more complex application a number of functions would be performed on the file descriptor to ensure that data is read out of the buffer from connection clients, and ensuring that the clients are receiving data from the application appropriately.

Product(s)

Red Hat JBoss Enterprise Application Platform (/taxonomy/products/red-hat-jboss-enterprise-application-platform)

Red Hat Enterprise Linux (/taxonomy/products/red-hat-enterprise-linux)

Component **jbossas (/components/jbossas)** **kernel (/components/kernel)**

Category **Troubleshoot (/category/troubleshoot)**

Tags **eap (/tags/eap)** **jboss (/tags/jboss)** **jboss_eap (/tags/jboss_eap)** **network (/tags/network)** **tcp (/tags/tcp)**

This solution is part of Red Hat's fast-track publication program, providing a huge library of solutions that Red Hat engineers have created while supporting our customers. To give you the knowledge you need the instant it becomes available, these articles may be presented in a raw and unedited form.