close vs shutdown socket?

Asked 14 years, 1 month ago Modified 4 years, 3 months ago

Viewed 330k times

260

In C, I understood that if we close a socket, it means the socket will be destroyed and can be re-used later.

How about shutdown? The description said it closes half of a duplex connection to that socket. But will that socket be destroyed like `close` system call?

c sockets network-programming

Share

Improve this question

Follow

edited Dec 12, 2013 at 22:28

tshepang

12.4k • 25 • 95 • 139

asked Nov 11, 2010 at 23:38

user188276

2 It can't be reused later. It's closed. Finished. Done. – user207421 Feb 1, 2020 at 10:06

@user207421 They probably mean the socket handler variable can be reused (to hold a new socket) without leaking memory. – Daniel Chin Aug 1, 2023 at 20:07

9 Answers

Sorted by: Highest score (default)

233

This is explained in Beej's networking guide. `shutdown` is a flexible way to block communication in one or both directions. When the second parameter is `SHUT_RDWR`, it will block both sending and receiving (like `close`). However, `close` is the way to actually destroy a socket.

With `shutdown`, you will still be able to receive pending data the peer already sent (thanks to Joey Adams for noting this).

Share Improve this answer

Follow

edited Feb 14, 2020 at 9:08

red0ct

5,055 ● 3 ● 21 ● 45

answered Nov 11, 2010 at 23:39

Matthew Flaschen

284k ● 53 ● 521 ● 552

26 Keep in mind, even if you close() a TCP socket, it won't necessarily be immediately reusable anyway, since it will be in a TIME_WAIT state while the OS makes sure there's no outstanding packets that might get confused as new information if you were to immediately reuse that socket for something else. – alesplin Nov 11, 2010 at 23:46

118 Big difference between shutdown and close on a socket is the behavior when the socket is shared by other processes. A shutdown() affects *all* copies of the socket while close() affects only the file descriptor in one process. – Zan Lynx Nov 12, 2010 at 0:24

7 One reason you may want to use `shutdown` both directions but not `close` is if you made a `FILE` reference to the socket using `fdopen`. If you `close` the socket, a newly opened file could be assigned the same fd, and subsequent use of the `FILE` will read/write the wrong place which could be very bad. If you just `shutdown`, subsequent use of the `FILE` will just give errors until `fclose` is called. – R.. GitHub STOP HELPING ICE Nov 12, 2010 at 4:22

34 The comment about TIME_WAIT is incorrect. That applies to ports, not to sockets. You can't reuse sockets. – user207421 Nov 12, 2010 at 8:53

61 -1. Both this post and the link omit an important conceptual reason to want to use `shutdown`: to signal EOF to the peer and still be able to receive pending data the peer sent. – Joey Adams Jan 13, 2013 at 23:39

218

None of the existing answers tell people how `shutdown` and `close` works at the TCP protocol level, so it is worth to add this.

A standard TCP connection gets terminated by 4-way finalization:

1. Once a participant has no more data to send, it sends a FIN packet to the other
2. The other party returns an ACK for the FIN.
3. When the other party also finished data transfer, it sends another FIN packet

4. The initial participant returns an ACK and finalizes transfer.

However, there is another "emergent" way to close a TCP connection:

1. A participant sends an RST packet and abandons the connection
2. The other side receives an RST and then abandon the connection as well

In my test with Wireshark, with default socket options, `shutdown` sends a FIN packet to the other end but it is all it does. Until the other party send you the FIN packet you are still able to receive data. Once this happened, your `Receive` will get an 0 size result. So if you are the first one to shut down "send", you should close the socket once you finished receiving data.

On the other hand, if you call `close` whilst the connection is still active (the other side is still active and you may have unsent data in the system buffer as well), an RST packet will be sent to the other side. This is good for errors. For example, if you think the other party provided wrong data or it refused to provide data (DOS attack?), you can close the socket straight away.

My opinion of rules would be:

1. Consider `shutdown` before `close` when possible

2. If you finished receiving (0 size data received) before you decided to shutdown, close the connection after the last send (if any) finished.
3. If you want to close the connection normally, shutdown the connection (with SHUT_WR, and if you don't care about receiving data after this point, with SHUT_RD as well), and wait until you receive a 0 size data, and then close the socket.
4. In any case, if any other error occurred (timeout for example), simply close the socket.

Ideal implementations for SHUT_RD and SHUT_WR

The following haven't been tested, trust at your own risk. However, I believe this is a reasonable and practical way of doing things.

If the TCP stack receives a shutdown with SHUT_RD only, it shall mark this connection as no more data expected. Any pending and subsequent `read` requests (regardless whichever thread they are in) will then returned with zero sized result. However, the connection is still active and usable -- you can still receive OOB data, for example. Also, the OS will drop any data it receives for this connection. But that is all, no packages will be sent to the other side.

If the TCP stack receives a shutdown with SHUT_WR only, it shall mark this connection as no more data can be sent. All pending write requests will be finished, but subsequent write requests will fail. Furthermore, a FIN

packet will be sent to another side to inform them we don't have more data to send.

Share Improve this answer Follow

edited Jul 29, 2018 at 23:29

answered May 5, 2014 at 23:19

Earth Engine
10.4k ● 7 ● 50 ● 82

3 "if you call close whilst the connection is still alive" is tautological, and it doesn't cause an RST to be sent. (1) isn't necessary. In (4), timeouts aren't necessarily fatal to the connection and don't invariable indicate you can close it. – user207421 May 6, 2014 at 0:32

7 @EJP No, it isn't tautological. You can `shutdown()` the connection and then it is no longer alive. You still have the file descriptor. You can still `recv()` from the receiving buffer. And you still need to call `close()` to dispose the file descriptor. – Pavel Šimerda Jan 21, 2016 at 9:58

1 This is the best answer regarding the wire format. I'd be interested about more details on shutting down with SHUT_RD. There's no TCP signalling for not expecting more data, right? Isn't there only FIN for signalling for not sending more data? – Pavel Šimerda Jan 24, 2016 at 13:36

4 @PavelŠimerda Yes TCP does not signalling for not expecting more data. This should be considered in the high level protocols. In my opinion, this in general not necessory. You can close your door, but you cannot stop people putting gifts in front of the door. This is THEIR decision, not yours. – Earth Engine May 31, 2016 at 22:29

1 @EJP Do you have any actual arguments? Otherwise I'm not interested. – Pavel Šimerda Nov 10, 2016 at 22:41

41

There are some limitations with `close()` that can be avoided if one uses `shutdown()` instead.

`close()` will terminate both directions on a TCP connection. Sometimes you want to tell the other endpoint that you are finished with sending data, but still want to receive data.

`close()` decrements the descriptors reference count (maintained in file table entry and counts number of descriptors currently open that are referring to a file/socket) and does not close the socket/file if the descriptor is not 0. This means that if you are forking, the cleanup happens only after reference count drops to 0. With `shutdown()` one can initiate normal TCP close sequence ignoring the reference count.

Parameters are as follows:

```
int shutdown(int s, int how); // s is socket descripto
```

`int how` can be:

`SHUT_RD` or `0` Further receives are disallowed

`SHUT_WR` or `1` Further sends are disallowed

`SHUT_RDWR` or `2` Further sends and receives are disallowed

Share Improve this answer

Follow

edited Jan 17, 2014 at 0:50

naXa stands with Ukraine

37.7k ● 24 ● 202 ● 272

answered Nov 12, 2010 at 9:05

Milan

15.8k ● 20 ● 59 ● 65

10 The two functions are for completely different purposes. The fact that the final close on a socket initiates a shutdown sequence if one hasn't already been initiated doesn't change the fact that close is for cleaning up the socket data structures and shutdown is for initiating a tcp level shutdown sequence. – Len Holgate Nov 12, 2010 at 10:08

28 You can't 'use shutdown instead'. You can use it *as well.* but you must close the socket some time. – user207421 Nov 13, 2010 at 0:44

19

This may be platform specific, I somehow doubt it, but anyway, the best explanation I've seen is here on this msdn page where they explain about shutdown, linger options, socket closure and general connection termination sequences.

In summary, use shutdown to send a shutdown sequence at the TCP level and use close to free up the resources used by the socket data structures in your process. If you

haven't issued an explicit shutdown sequence by the time you call close then one is initiated for you.

Share Improve this answer Follow

edited Nov 18, 2010 at 8:33

answered Nov 12, 2010 at 9:04

Len Holgate
21.6k ● 4 ● 47 ● 94

2 Anybody trying to achieve graceful shutdown of an HTTP connection will probably want to read the venerable Apache notes on "lingering close", found in places like cluster.cis.drexel.edu/manual/misc/perf-tuning.html, and maybe a more modern commentary: apenwarr.ca/log/20090814 – Ron Burk Oct 11, 2020 at 10:11

11

I've also had success under linux using `shutdown()` from one pthread to force another pthread currently blocked in `connect()` to abort early.

Under other OSes (OSX at least), I found calling `close()` was enough to get `connect()` fail.

Share Improve this answer Follow

answered Nov 15, 2010 at 8:56

Toby
2,099 ● 1 ● 13 ● 10

"shutdown() doesn't actually close the file descriptor—it just changes its usability. To free a socket descriptor, you

10

need to use close()."[1]

Share Improve this answer Follow

answered Nov 12, 2014 at 1:22

user1944429

4

Close

When you have finished using a socket, you can simply close its file descriptor with close; If there is still data waiting to be transmitted over the connection, normally close tries to complete this transmission. You can control this behavior using the SO_LINGER socket option to specify a timeout period; see Socket Options.

ShutDown

You can also shut down only reception or transmission on a connection by calling shutdown.

The shutdown function shuts down the connection of socket. Its argument how specifies what action to perform: 0 Stop receiving data for this socket. If further data arrives, reject it. 1 Stop trying to transmit data from this socket. Discard any data waiting to be sent. Stop looking for acknowledgement of data already sent; don’t retransmit it if it is lost. 2 Stop both reception and transmission.

The return value is 0 on success and -1 on failure.

Share Improve this answer

Follow

edited Jul 21, 2016 at 11:57

J.J. Hakala

6,214 ● 6 ● 28 ● 61

answered Jul 7, 2016 at 7:39

Neha Agrawal

531 ● 4 ● 9

3

in my test.

`close` will send fin packet and destroy fd immediately when socket is not shared with other processes

`shutdown` **SHUT_RD**, process can still recv data from the socket, but `recv` will return 0 if TCP buffer is empty.After peer send more data, `recv` will return data again.

`shutdown` **SHUT_WR** will send fin packet to indicate the Further sends are disallowed. the peer can recv data but it will recv 0 if its TCP buffer is empty

`shutdown` **SHUT_RDWR** (equal to use both **SHUT_RD** and **SHUT_WR**) will send rst packet if peer send more data.

Share Improve this answer

Follow

answered Sep 17, 2015 at 6:06

simpx

184 ● 1 ● 11

I just tried it and `close()` sent RST on Linux instead of FIN. – Pavel Šimerda Jan 21, 2016 at 10:02

Also did you really want to state that after shutting down the reading side, the program will receive more data? – Pavel Šimerda Jan 21, 2016 at 10:06

@PavelŠimerda i think it may depends on the tcp state to send RST or FIN? i'm not sure. – simpx Jan 26, 2016 at 7:50

1. 'After peer send more data, `recv()` will return data again' is not correct. 2. The behaviour if the peer sends more data after `SHUT_RD` is platform-dependent. – user207421 Nov 3, 2016 at 5:38

@EJP I tried myself, sorry I forget which platform I do this test, centos or Mac. and this is what I got – simpx Nov 25, 2016 at 4:49

0

linux: shutdown() causes listener thread select() to awake and produce error. shutdown(); close(); will lead to endless wait.

winsock: vice versa - shutdown() has no effect, while close() is successfully catched.

Share Improve this answer

Follow

answered Sep 12, 2020 at 8:09

Vladimir Savchik

17 • 1

I do not understand this answer. – Ingo Apr 1, 2023 at 14:52

Thanks! This is really good hint how to wake another thread waiting in select(). Most probably poll() works in the same way. – Karlson2k Nov 25, 2023 at 11:02