

In nodejs I can detect close event for a socket but not in Rust

[help](#)

[emirbugra](#) August 1, 2025, 12:02pm 1

Hi folks. I'm struggling a strange problem. I can detect the close event in nodejs for a TCP socket connection but I can't do same thing in Rust (Tokio). For example:

```
// Javascript
let connection = new net.Socket();
connection.on("end", (data) => {
  console.log("Connection finished.")
})
```

For example I can connect to this `connection` with CURL and when connection finishes the `end` function invokes. But I can't do same thing in `tokio::net::TcpStream`

Yes I know that we can't really understand that is the connection live. All network connections actually isn't exist but at least I want to detect the `FIN` flag sent from the peer. How can I do this?

[kornel](#) August 1, 2025, 12:25pm 2

I think you must be trying to read from the socket to observe it closing. Node probably has a reader by default collecting data into a buffer.

1 Like

[emirbugra](#) August 1, 2025, 12:39pm 3

Thanks for your message but I need code

[alice](#) August 1, 2025, 12:42pm 4

I suggest you share the code you tried.

In general, the way to detect a close event is to call `read`. The call to `read` returns a length, and if the length is zero, then the stream is closed.

1 Like

[emirbugra](#) August 1, 2025, 12:55pm 5

`read()` function returns `Ok(0)` even if the connection is closed. My code is like that:

```
use tokio::sync::Mutex;
use tokio::net::TcpStream;
use std::sync::Arc;

pub type TcpStreamThreaded = Arc<Mutex<TcpStream>>;

async fn transfer_stream(
    read_stream: TcpStreamThreaded,
    write_stream: TcpStreamThreaded,
    direction: String,
) -> std::io::Result<usize> {
    let mut buffer = vec![0; 1024 * 256];
    let bytes_cnt = read_stream.lock().await.try_read(&mut buffer)?;

    if bytes_cnt == 0 {
        return Ok(bytes_cnt);
    }

    buffer.truncate(bytes_cnt);

    write_stream.lock().await.write_all(&buffer).await?;
    tracing::info!("Bytes transferred to direction {direction}: {bytes_cnt}");

    Ok(bytes_cnt)
}
```

Important thing is that I can't wait any connection. `read()` function is blocks everything.

[mroth](#) August 1, 2025, 1:15pm 6

emirbugra:

`read()` function returns `Ok(0)` even if the connection is closed

Check the docs: https://docs.rs/tokio/latest/tokio/net/struct.TcpStream.html#method.try_read

`Ok(0)` means, the stream is closed.

1 Like

[alice](#) August 1, 2025, 1:34pm 7

emirbugra:

`read()` function is blocks everything.

Your mistake is putting a `TcpStream` inside a `Mutex`. That is almost always a design mistake. Don't do it.

Instead, you can [split the TcpStream](#), which makes it possible to read and write at the same time.

For more complex cases, you can give ownership of the `TcpStream` to [an actor](#).

6 Likes

[emirbugra](#) August 1, 2025, 4:32pm 8

You're awesome. `split` function returns reference with lifetime specifier. This causes problems which difficult to solve. But I used `into_split` and moved the values to `Arc<Mutex<>>s`. This is easier then struggling with references and lifetime specifiers when moving values to async functions, `tokio::spawns` and loops. Let me show the code:

```
async fn handle_sockets(
    socket1: TcpStream,
    socket2: TcpStream
) -> anyhow::Result<()> {
    // ...

    let (socket1_read_half, socket1_write_half) = socket1.into_split();
    let (socket2_read_half, socket2_write_half) = socket2.into_split();

    let arc_socket1_read_half = Arc::new(Mutex::new(socket1_read_half));
    let arc_socket1_write_half = Arc::new(Mutex::new(socket1_write_half));
    let arc_socket2_read_half = Arc::new(Mutex::new(socket2_read_half));
    let arc_socket2_write_half = Arc::new(Mutex::new(socket2_write_half));

    let _ = timeout(
        Duration::from_secs(90),
        tokio::spawn(async move {
            tokio::spawn(transfer_stream(
                arc_socket1_read_half,
                arc_socket2_write_half,
                "s1 to s2"
            ));
            tokio::spawn(transfer_stream(
                arc_socket2_read_half,
                arc_socket1_write_half,
                "s2 to s1"
            ));
        })
    )
    .await;
    // ...
}
```

```
async fn transfer_stream(
    read_stream: Arc<Mutex<OwnedReadHalf>>,
    write_stream: Arc<Mutex<OwnedWriteHalf>>,
    direction: &str,
) -> std::io::Result<()> {
    tracing::info!("{direction} starting.");
```

```
loop {
    let mut buffer = vec![0; 1024 * 256];
    let bytes_cnt = read_stream.lock().await.read(&mut buffer).await?;

    if bytes_cnt == 0 {
        break;
    }

    buffer.truncate(bytes_cnt);

    write_stream.lock().await.write_all(&buffer).await?;
    tracing::info!("Bytes transferred to direction {direction}: {bytes_cnt}");
}

Ok(())
}
```

[ProgramCrafter](#) August 2, 2025, 9:29pm 9

Have you checked out `tokio::io::copy_bidirectional` for that exact problem of forwarding contents as-is?

Anyways, you do not want to allocate `vec![0; 1024 * 256]` upon each loop iteration, because it is interacting with a global heap - it would be faster to move allocation out of the loop.

2 Likes

[emirbugra](#) August 3, 2025, 6:47am 10

ProgramCrafter:

Anyways, you do not want to allocate `vec![0; 1024 * 256]` upon each loop iteration, because it is interacting with a global heap - it would be faster to move allocation out of the loop.

I like this community. People suggest awesome information. Thanks for this.

Also I'm looking to `copy_bidirectional` library now. This seems awesome thing.

[system](#) Closed November 1, 2025, 6:48am 11

This topic was automatically closed 90 days after the last reply. We invite you to open a new topic if you have further questions or comments.

Related topics

Topic		Replies	Views	Activity
Event on TcpStream?	3	1262	January 12, 2023	
help				
How to detect TCP close?	13	8062	January 30, 2021	
Tokio TCP connection not closed when sender is dropped (Futures 0.3 + compat layer)	6	3143	July 3, 2019	
help				
tokio::io::BufStream.read_line() hangs forever when connection lost / closed?	5	114	November 13, 2024	
help				
Clean way to terminate a TcpStream::read	30	4632	April 2, 2022	

- [Home](#)
- [Categories](#)
- [Guidelines](#)
- [Terms of Service](#)