serialMux

Jeroen F.J. Laros

Apr 10, 2022

CONTENTS:

This library provides a simple way to create multiple virtual serial devices that communicate over one physical serial connection. A virtual device can be used as a drop-in replacement for `Stream` like objects such as `Serial`.

A service is needed on the host for multiplexing and demultiplexing and to create virtual ports. A Python client for Linux is provided as a reference implementation.

Please see ReadTheDocs for the latest documentation.

CHAPTER

ONE

QUICK START

Create multiple virtual serial devices and use them like the standard `Serial` object.

```
#include <serialMux.h>

SerialMux mux(Serial);
VSerial serialA(mux);
VSerial serialB(mux);

void setup(void) {
  Serial.begin(9600);
}

void loop(void) {
  serialA.println("Virtual serial device A.");
  serialB.println("Virtual serial device B.");
  delay(1000);
}
```

On the host, two virtual ports are created, e.g., `/dev/pts/8` and `/dev/pts/9`. When we connect to one of these ports, we only see the messages that are sent to that port.

```
$ picocom -q /dev/pts/8
Virtual device A.
Virtual device A.
```

1.1 Introduction

This library provides a simple way to create multiple virtual serial devices that communicate over one physical serial connection. A virtual device can be used as a drop-in replacement for `Stream` like objects such as `Serial`.

1.1.1 Motivation

Suppose we we have a function `someFunction()` that takes a command and returns a status. Calling such a function from outside may look something like this.

```
if (Serial.available()) {
  Serial.write(someFunction(Serial.read()));
}
```

This approach works as long as there are no other processes using the serial connection. This means that things like debugging information can no longer be printed to `Serial`.

This is where the serialMux library comes in. By creating virtual serial devices, multiple connections can be made over the same physical serial line.

1.2 Installation

1.2.1 Arduino IDE

To install this library in the Arduino IDE, please follow these comprehensive installation instructions.

1.2.2 Arduino CLI

The latest version can be installed with the Arduino CLI interface using the following command.

```
arduino-cli lib install serialMux
```

1.2.3 Manual installation

Latest release

Navigate to the latest release and either download the `.zip` or the `.tar.gz` file and unpack the downloaded archive.

From source

The source is hosted on GitHub, use the following command to install the latest development version.

```
git clone https://github.com/jfjlaros/serialMux.git
```

1.3 Usage

Include the header file to use the serialMux library.

```
#include <serialMux.h>
```

The library provides the `SerialMux` and `VSerial` classes, which are used to create virtual serial devices.

First create a serial multiplexer.

```
SerialMux mux(Serial);
```

The serial multiplexer can be used to create multiple virtual serial devices.

```
VSerial serialA(mux);
VSerial serialB(mux);
```

The physical serial device must be configured as usual.

```
void setup(void) {
  Serial.begin(9600);
}
```

Virtual serial devices can be used just like the familiar physical serial device.

```
void loop(void) {
  serialA.println("Virtual serial device A.");
  serialB.println("Virtual serial device B.");
  delay(1000);
}
```

1.3.1 Hard coded `Serial` workaround

Some libraries will write debugging or other information directly to the physical serial device. To force this library to use one of the virtual serial devices, the following lines can be added before the library is included.

```
HardwareSerial& masterSerial = Serial;
SerialMux mux(masterSerial);
VSerial serialA(mux)
#define Serial serialA

#include <hardcodedSerialUsingLib.h>
```

Note that any communication with the physical serial device must now be done using `masterSerial` instead of `Serial`.

```
void setup(void) {
  masterSerial.begin(9600);
}
```

1.4 Protocol

In this section we describe the serialMux protocol.

1.4.1 Control messages

Communication over a shared serial connection is accomplished by using control messages. Control messages start with an escape character, followed by a virtual port number. The virtual port remains active until the next control message is received.

Table 1: Control messages.

offset	description
0	`0xff`
1	virtual serial port

The first virtual device has port number `0`, the second `1`, etc. Port number `254` is reserved for control messages, which limits the maximum number of virtual devices 253.

1.4.2 Control channel

Before any of the virtual devices can be used, the host must be informed about the number of virtual devices. To ensure that this initial communication is interference free, all virtual devices are disabled on start up.

Table 2: Control channel messages.

message	response	description
0	`serialMux`	Protocol identifier.
1	`\x01\x00\x00` (example)	Version (major, minor, patch).
2	Number of virtual serial ports.	Get the number of virtual serial ports.
3	`0x00`	Enable multiplexer.
4	`0x00`	Disable multiplexer.
5	`0x00`	Reset.

A typical initialisation procedure looks as follows.

1. The host asks for the protocol identifier (`0xff, 0xfe, 0x00`).
2. The device responds with the protocol identifier.
3. The host asks for the protocol version (`0x01`).
4. The device responds with the protocol version.
5. The host requests the number of virtual serial ports (`0x02`).
6. The device sends the number of virtual serial ports (e.g., `0x02`).
7. The host sets up pseudo terminals that connect to the virtual serial ports.
8. The host send the enable command (`0x03`).
9. The device responds with an acknowledgement (`0x00`).

After initialisation the first pseudo terminal can be used to communicate with the virtual device on port `0`, the second pseudo terminal can be used to communicate with the virtual device on port `1`, etc.

1.5 API documentation

1.5.1 Ring buffer

```
#include "buffer.tcc"
```

Class definition

template<uint8_t **bits**>

class **Buffer**

Ring buffer.

Public Functions

Buffer(void)

Constructor.

uint8_t **available**(void)

Get the number of bytes available for reading.

Returns Number of bytes.

uint8_t **read**(uint8_t*, uint8_t)

Read `size` bytes of data.

Parameters

- **data** – *Buffer* to receive `size` bytes of data.
- **size** – Number of bytes to read.

Returns Number of bytes read.

int16_t **read**(void)

Read one byte of data.

Returns The first byte of incoming data or `-1` if no data is available.

void **write**(uint8_t*, uint8_t)

Write `size` bytes of data.

Parameters

- **data** – *Buffer* containing `size` bytes of data.
- **size** – Number of bytes to write.

Returns Number of bytes written.

void **write**(uint8_t)

Write one byte of data.

Parameters **data** – Data.

int16_t **peek**(void)

Return the next byte of incoming data without removing it from the buffer.

Returns The first byte of incoming data or `-1` if no data is available.

1.5.2 Serial multiplexer

```
#include "serialMux.tcc"
```

Class definition

template<uint8_t **bits** = 6>

class **SerialMux**

Serial multiplexer.

Public Functions

SerialMux(Stream&)

Constructor.

Parameters **serial** – Serial device.

uint8_t **add**(void)

Add a virtual serial device.

Returns New virtual serial port.

size_t **available**(uint8_t)

Get the number of bytes available for reading.

Parameters **port** – Virtual serial port.

Returns Number of bytes.

int **read**(uint8_t)

Read one byte of data.

Parameters **port** – Virtual serial port.

Returns The first byte of incoming data or -1 if no data is available.

void **write**(uint8_t, uint8_t)

Write one byte of data.

Parameters

- **port** – Virtual serial port.
- **data** – Data.

Returns Number of bytes written.

int **peek**(uint8_t)

Return the next byte of incoming data without removing it from the buffer.

Parameters **port** – Virtual serial port.

Returns The first byte of incoming data or -1 if no data is available.

1.5.3 Virtual serial device

```
#include "vSerial.tcc"
```

Class definition

template<uint8_t **bits** = 6>

class **VSerial** : public Stream

Virtual serial device.

Public Functions

VSerial(*SerialMux*<*bits*>&)

Constructor.

Parameters **mux** – Serial multiplexer.

int **available**(void)

Get the number of bytes available for reading.

Returns Number of bytes.

int **read**(void)

Read one byte of data.

Returns The first byte of incoming data or -1 if no data is available.

size_t **write**(uint8_t)

Write one byte of data.

Parameters **data** – Data.

Returns Number of bytes written.

int **peek**(void)

Return the next byte of incoming data without removing it from the buffer.

Returns The first byte of incoming data or -1 if no data is available.

1.6 Contributors

- Jeroen F.J. Laros <jlaros@fixedpoint.nl> (Original author, maintainer)

Find out who contributed:

```
git shortlog -s -e
```

INDEX