# Serieller Port – Florian Boemmel

## Generelles

In unserem Projekt nutzen wir eine serielle USB-Verbindung zwischen Arduino und Raspberry Pi, um Daten und Befehle zwischen den beiden Geräten auszutauschen. Dieser Abschnitt beschäftigt sich ausschließlich nur mit dem Seriellen Port für die USB-Verbindung zwischen Raspberry Pi und Arduino.

## Grundlagen

Die Grundlage jeder seriellen Kommunikation auf einem linuxbasiertem Betriebssystem ist das Öffnen und Konfigurieren eines Seriellen Ports. Serielle Ports werden unter Linux durch eine Datei repräsentiert.

## Seriellen Port bestimmen

Zunächst muss der Port festgestellt werden, an dem der Arduino am Pi erkannt wird. Dazu kann entweder die Arduino IDE benutzt werden, oder über das Terminal.

* 1. Möchte man das Terminal nutzen, muss die Verbindung zum Arduino unbedingt getrennt werden und folgendes Kommando ausgeführt werden:

pi@raspberrypi:~$ ls /dev/

Nun muss zunächst überprüft werden, ob bereits ein ttyUSB oder ttyACM existiert. Jetzt muss der Arduino verbunden werden. Eine erneute Ausführung des Kommandos sollte jetzt einen weiteren Eintrag liefern (z.B. ttyUSB0). Dieser Eintrag ist nun der Serielle Port zu unserm Arduino.

* 1. Möchte man die Arduino IDE benutzen, öffnet man diese und verbindet den Arduino mit dem Pi. Anschließend wählt man im Menü:

Tools 🡪 Serieller Port

Hier wird nun der Port angezeigt. Jedoch muss beachtet werden, dass weitere angeschlossene Geräte unter Umständen auch angezeigt werden.

## Seriellen Port implementieren

Das Implementieren des Seriellen Ports erfolgt mit C und unter der Verwendung der [terminos API](https://linux.die.net/man/3/termios). Die terminos API unterstützt unterschiedliche Modi um einen Seriellen Port anzusprechen. Die zwei wichtigsten sind:

* **Cannonical Mode:** Dieser Modus ist Zeilenorientiert. Dies bedeutet, dass Eingaben gepuffert und durch den Benutzer bearbeitet werden können, bis ein carriage return (unter Linux CTRL-C) oder ein line feed (Zeilenumbruch) erkannt wird.Anschließend kann ein [read(2)](https://linux.die.net/man/2/read) ausgeführt werden. Wird von Terminals verwendet.

* **NonCannonical Mode:** Dieser Modus ist im Gegensatz zum Cannonical Mode weder Zeilenorientiert noch werden Eingaben gepuffert oder können vom Benutzer bearbeitet werden. Dies bedeutet, dass ein Input sofort zur Verfügung steht. Zusätzlich muss hier eine Einstellung vorgenommen werden, unter welchen Umständen ein [read(2)](https://linux.die.net/man/2/read) aufgerufen wird und wie sich dieses verhält.

Ausführliche Informationen über die Seriellen Ports und deren Programmierung können im [“**The Serial Programming Guide for POSIX Operating Systems**”](https://www.cmrr.umn.edu/~strupp/serial.html#2_4) nachgelesen werden.

## Seriellen Port öffnen und schließen

Zum Öffnen eines Seriellen Ports unter Linux wird der Systemaufruf [open(2)](http://man7.org/linux/man-pages/man2/open.2.html) verwendet:

int fd;

fd = open("/dev/ttyUSB0", O\_RDWR | O\_NOCTTY);

fd: File-Deskriptor

/dev/ttyUSB0: Serieller Port im Verzeichnis /dev

O\_RDWR: Serieller Port wird geöffnet für schreiben und lesen

O\_NOCTTY: Kein Terminal wird das öffnen kontrollieren

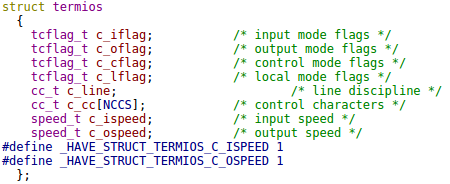
Wurde der Port erfolgreich geöffnet, erhält fd einen positiven Wert. Im Fehlerfall liefert open -1 zurück.

Zum schließen wird [close(2)](https://linux.die.net/man/2/close) verwendet:

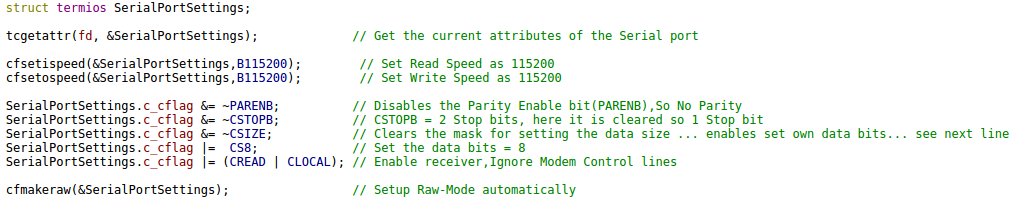
close(fd);

## Seriellen Port konfigurieren

Zum Konfigurieren des Seriellen Ports wird, wie schon beschrieben, die terminos API benutzt. Die terminos Struktur sieht wie folgt aus:



Nun werden die spezifischen Einstellungen für unser Projekt gesetzt.



Für weitere Informationen und einer detaillierten Beschreibung der verwendeten sowie möglichen weiteren Einstellungen kann das unter [Punkt 4](#_Seriellen_Port_implementieren) referenzierte Dokument verwendet werden.

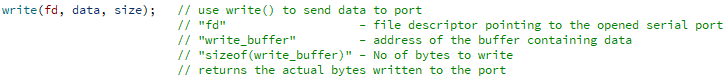
Ein letzter Schritt setzt die Einstellungen in der terminos Struktur zu dem Seriellen Port:

tcsetattr(fd,TCSANOW,&SerialPortSettings));

Die Funktion liefert im Erfolgsfall eine 0 zurück. Danach ist der Serielle Port konfiguriert und für die Übertragung und das Empfangen von Daten eingerichtet.

## Serieller Port schreiben

Das Schreiben auf dem Seriellen Port, wird durch den Systemaufruf [write(2)](https://linux.die.net/man/2/write) realisiert.



Dabei wird write() der File-Deskriptor, einem Puffer vom Typ const void \* und eine Größe der zu schreibenden Daten in Bytes übergeben.

Die Größe der zu schreibenden Daten in Bytes gibt an, wie viele Bytes auf dem Puffer geschrieben werden sollen.

write() liefert im Erfolgsfall die Anzahl der geschriebenen Bytes zurück. Im Fehlerfall wird -1 zurückgegeben und 0 bedeutet, dass keine Daten geschrieben wurden.

## Serieller Port lesen

Das Lesen auf dem Seriellen Port, wird durch den Systemaufruf [read(2)](https://linux.die.net/man/2/read) realisiert.



Dabei wird read() der File-Deskriptor, einem Puffer vom Typ void \* und eine Größe der zu lesenden Daten in Bytes übergeben. Die Größe der zu lesenden Daten in Bytes gibt an, wie viele Bytes aus dem File-Deskriptor in den übergebenen Puffer gelesen und anschließend geschrieben werden sollen.

read() liefert im Erfolgsfall die Anzahl der gelesenen Bytes zurück. Im Fehlerfall wird -1 zurückgegeben und 0 bedeutet, das Ende der Datei ist erreicht.

## Probleme

### Wechsel des Seriellen Ports

Zu Beginn des Projekts verwendeten wir einen Arduino Uno. Im späteren Verlauf wechselten wir jedoch auf einen Arduino Mega. Während der Debug-Tätigkeiten fiel immer wieder das Problem mit dem Seriellen Port auf. Manchmal wurde der Arduino unter ttyUSB0 erkannt und ein paar Tage später wieder unter ttyACM0 oder andersherum.

Schlussendlich stellte sich heraus, dass der Grund hierfür der Wechsel der Arduinos war. Einmal wurde mit dem Uno gearbeitet und ein anderes Mal mit dem Mega. Der Uno bekommt durch den Raspberry Pi den Port ttyUSB0 zugewiesen. Im Gegensatz dazu bekommt der Mega den Port ttyACM0 zugewiesen. Mögliche Gründe hierfür konnte ich noch nicht ermitteln.

### Öffnen des Seriellen Ports

Zu Beginn der Implementierung des Seriellen Ports, kam es sporadisch vor, dass der Port zwar ordnungsgemäß geöffnet wurde, jedoch anschließend nicht fehlerfrei arbeitete. Nach intensiver Recherche wurde ich fündig. In einigen Fällen, kann es vorkommen, dass das Programm zu schnell weiterarbeitet und z.B. ein [write(2)](https://linux.die.net/man/2/write) zu früh ausführt. Ein [usleep(3)](http://man7.org/linux/man-pages/man3/usleep.3.html) von 200 Millisekunden direkt nach dem Öffnen löste das Problem dauerhaft.

### Neustart des Arduino

Während der ersten Tests viel auf, dass keine Daten an den Arduino gesendet werden konnten. Ich untersuchte dies ausgiebig und stellte anschließend fest, dass nach dem unter [Punkt 6](#_Seriellen_Port_konfigurieren) beschriebenen Tätigkeiten der Arduino neustartet. Genauer gesagt geschieht dies direkt nach dem Funktionsaufruf:

tcsetattr(fd,TCSANOW,&SerialPortSettings));

Eine Lösung hierfür ist denkbar einfach und mehrfach in den Foren als einzige Lösung bekannt. Man muss auf den Neustart des Arduinos warten. Eine sichere Zeitspanne ist dabei drei Sekunden. Realisierbar unter Linux mit [sleep(3)](https://linux.die.net/man/3/sleep).

### Schließen des Seriellen Ports

Es ist unabdingbar den Seriellen Port am Ende des Programms wieder zu schließen, um Ihn anschließend beim erneuten Starten des Programms wieder öffnen zu können. Das bereitete gerade während der Entwicklungsphase der Benutzeroberfläche einige Probleme. Stürzte die Benutzeroberfläche während eines Testlauf ab, konnte der Serielle Port nicht mehr geöffnet werden und es blieb nichts anderes übrig, als jedes Mal die USB-Verbindung zum Arduino zu trennen und wiederherzustellen.

## Ausblick

Die geforderten Anforderungen wurden umgesetzt und der Serielle Port erweist sich als robust und bereitete keine Probleme. Dies wurde durch ausgiebige Tests bestätigt.

Jedoch wäre eine genauere Untersuchung der gesamten terminos Struktur ein weiterer möglicher Schritt. In diesem könnten eventuelle Verbesserungen in dem Gebiet der Performance erreicht werden.

Zusätzlich könnte noch eine Logik eingebaut werden, die die vorhandenen Ports überprüft und so selber den richtigen für den Arduino auswählt. Aktuell ist dieser im Code fest implementiert und muss bei einem Wechsel des Ports geändert und neu kompiliert werden.