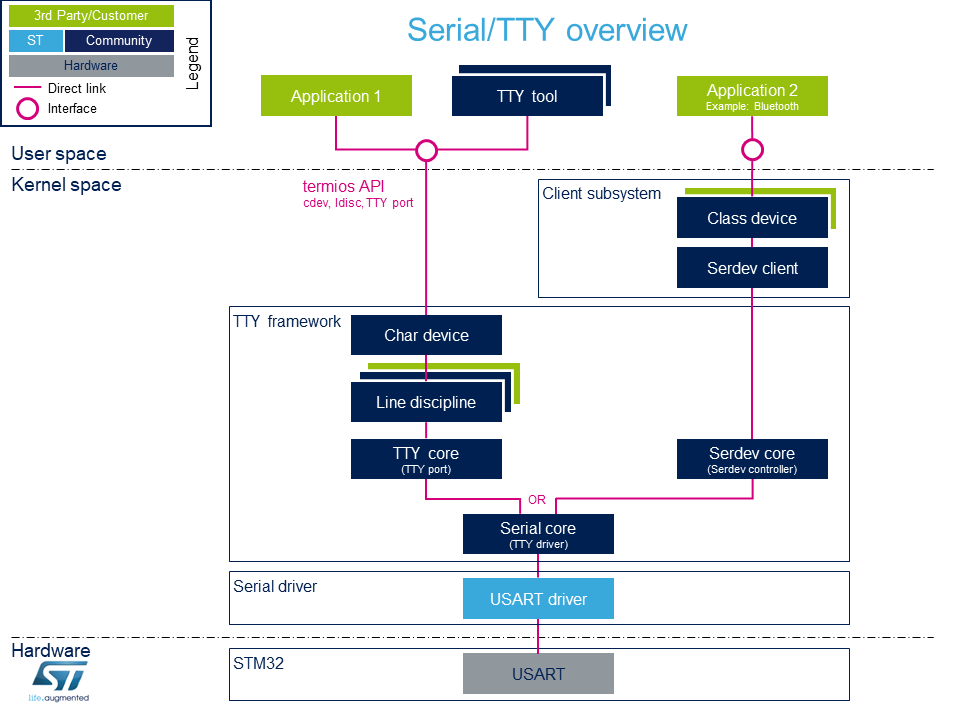
**UART Driver in Linux**

# I. Introduction

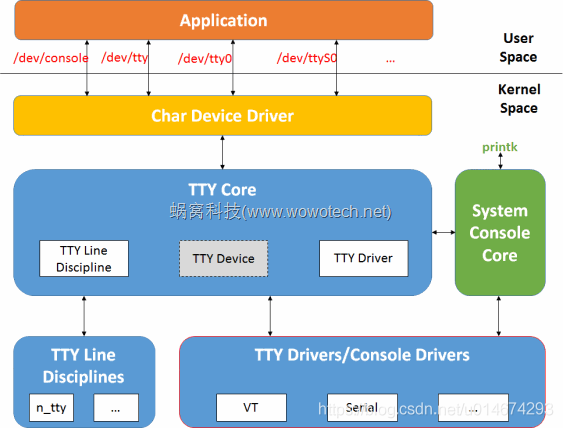
When the kernel started, serial port driver (uart driver) will initalized completely. The application can access the device node, obtain the fd handle of the serial port device. It can set related parameters (baudrate, stop bit, checkbit, etc.) from user space to communicate with other periphrals or other devices.

For example, the bluetooth chip is connected to the CPU through UART and the application program interacts with the bluetooth chip through the serial port by HCI command.

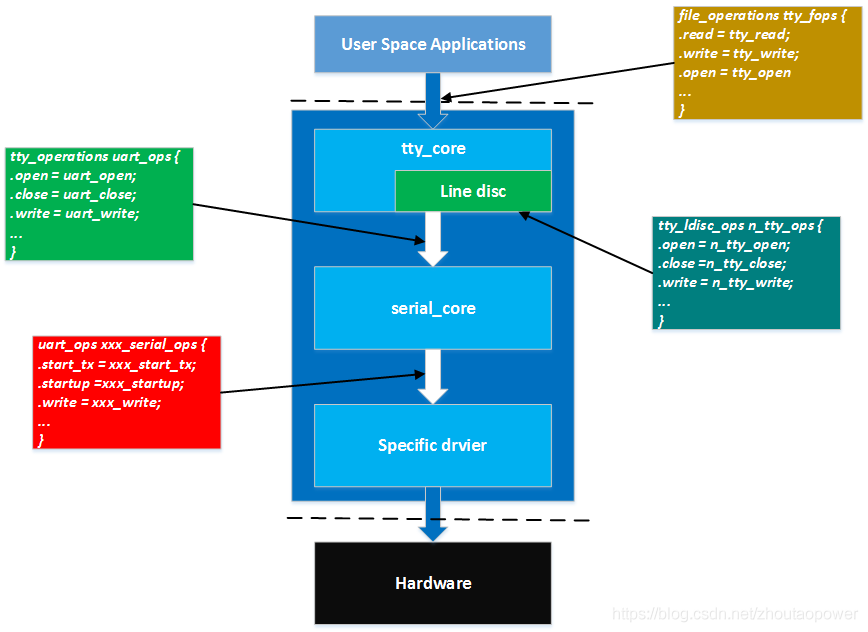
The UART serial port of the Linux kernel is devided into several layers, tty core layer, tty line descipline, serial driver core layer and the following ops for specific chips



# III. TTY Framwork



1. Operation



2. Open process

open (user space) -> tty\_open -> n\_tty\_open -> uart\_open -> xxx.startup

3. Write process

- Not use interrupt: write (user space) -> tty\_write -> n\_tty\_write -> uart\_write -> uart\_start -> xxx\_start\_tx

- Use interrupt: IRQ handler-> tty\_wakeup -> wake\_up\_interruptible\_poll

4. Read process

- Not use interrupt: read (user space) -> tty\_read ->n\_tty\_read -> add current process to waitqueue -> set the current process to be interruptible -> scheduling process, current process go to sleep -> wait to wakeup -> wakeup -> tty->disc\_data->read\_buf -> tty->disc\_data->read\_buf -> copy data to user space via tty\_put\_user

- Use interrupt: IRQ Handler -> Data coming -> tty\_insert\_flip\_char -> tty\_flip\_buffer\_push -> flush\_to\_ldisc -> n\_tty\_receiver\_buf -> tty->disc\_data->read\_buf -> wake\_up\_interruptible -> wakeup -> tty->disc\_data->read\_buf -> tty->disc\_data->read\_buf -> copy data to user space via tty\_put\_user

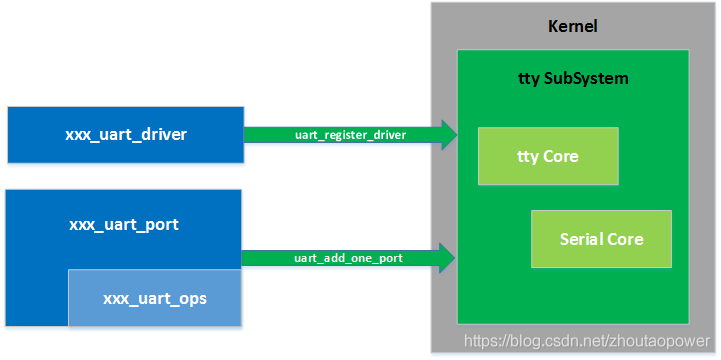
4. Close the device

close (user space) -> tty\_release -> uart\_close -> xxx\_stop\_rx -> uart\_shutdown -> xxx\_shutdown

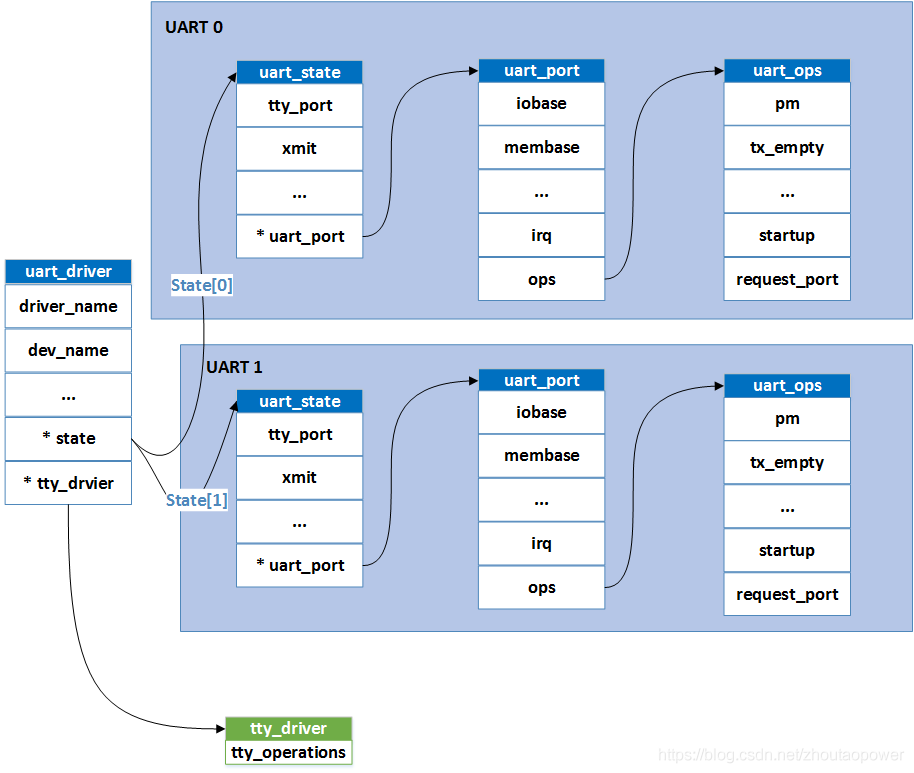
5. Cancellation process

5.1. Remove uart\_port

5.2. Unregister uart\_driver



From a structural point of view, the entire docking process

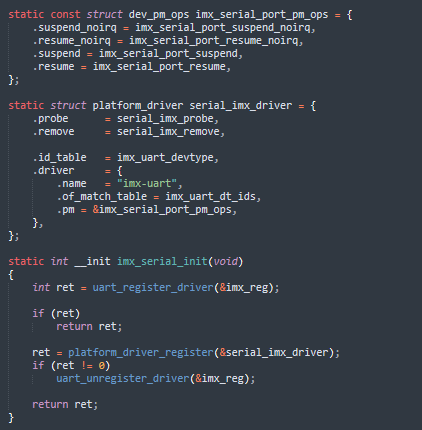


From the perspective of data structure and the relationship between

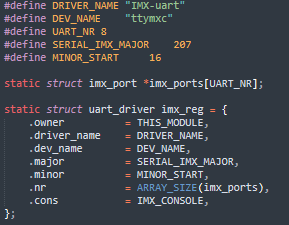
# II. UART Driver

The Linux Kernel provides a set of low-level interfaces for UART driver. Because the serial is not available in the chip, UART driver will use to implement platfrom driver way.

**1. Init UART driver**



1. The uart\_driver struct



Attributes of the uart\_driver struct:

* driver\_name: Take name of UART driver.
* dev\_name: This is the interface that the application uses to commnicate with tty layer
* major: Major number of driver
* minor: Minor number of driver
* nr: Take number of ports which vendor to implement. Number of ports depend on how many serial port supported in specific chip.

After you have prepared the uart\_driver struct, you can register uart driver by

**2. Register uart driver**

The registration process mainly does the following operation:

1. According to the maximum number of devices supported by the driver, apply for **n** uart spaces and each uart\_state has a uart\_port.
2. Allocate a tty\_driver and point uart\_driver->tty\_driver to it
3. Set the tty\_driver, including the default baudrate, inspection mode, etc. There is also an important ops, the registration of the structure tty\_operation, which is the communication interface between the tty core and the serial port driver.
4. Initialize the tty\_port of each uart\_state
5. Register tty\_driver.

Register uart\_driver is actually registering tty\_driver. The work of dealing with user space is completely handed over to tty\_driver. This part is implemented by the kernel and does not need to be modified.

**3. Add uart\_port**

IV. Refer

https://www.cnblogs.com/yikoulinux/p/14507445.html