13. RPi Side as systemd Service

NOTE - December 7, 2023: I am continuing with this project pushed down onto my stack (so-to-speak) in order to give my time to several other projects and activities that have fallen a bit to far by the wayside. Do not yet have a sense of when I'll return back to this project. But in order to have the sensor running 'in the field,' in an actual flower pot, I feel the need to have the RPi side application autostart at reboot time. So I have take some time to add this feature.

Purpose: Make the RPi application, RPi_CapDataReceive, into a systemd Service that starts at reboot. Also make it so that it will automatically restart on error-exit.

Goal: Create a systemd unit file. Make any needed mods to the RPi_CapDataReceive.cpp code to support running as an autostart background service.

Design Notes:

Considerations:

1. Absolute minimal change to the code.
2. Might want to consider writing a log file for error/debug info as I won't have a terminal screen available to be able to see any program output.

Key References --

systemd Service Instructional References --

https://linuxhandbook.com/create-systemd-services/

https://www.tecmint.com/create-systemd-service-linux/

Raspberry Pi Login --

Pi user is: pi. So SSH login is:

```
ssh pi@[ip address of the RPi on my LAN]
```

password to the pi is: `pi9012`

NOTE: I have created an alias command in my .bash_profile for the login: '**`pilogin`**'

To see user custom services:

```
systemctl --user list-unit-files -a
```

And then once in, get into the working directory for the nRF24 utilities and programs:

```
cd /home/rf24libs/RF24/examples_linux/build
```

The run the RPi receiving program:

```
./RPi_CapDataReceive
```

Good reference for running the process detached from terminal: https://linuxconfig.org/detach-process-program-from-current-shell-to-keep-it-alive-after-logout

Based on this, the command to run in an SSH terminal to keep the RPi programming forever, even after detaching the SSH session, is:

```
nohup /home/rf24libs/RF24/examples_linux/build
/RPi_CapDataReceive  &
```

Working with Service --

To see user custom services: `systemctl --user list-unit-files -a`

To modify the service file follow these steps:

`nano /home/.config/systemd/user/RPi_DummyService.service` || Then save the file.

Reload services: `systemctl --user daemon-reload`

Disable Service: `systemctl --user disable RPi_DummyService`

Enable Service - in theory this will cause it to auto-start at boot: `systemctl --user disable RPi_DummyService`

To start service manually: `systemctl --user start RPi_DummyService`

To filter journal entries for the service: `journalctl --user -u RPi_DummyService.service`

To Modify RPi side .cpp code --

Use Kate on my desktop.

Then copy to the RPi using below command:

For the learning demo:

```
scp /home/jroc/Dropbox/projects/systemd-learning/RPi_DummyService.cpp
pi@10.215.100.118:/home/rf24libs/RF24/examples_linux/
```

For the 'live' program:

```
scp /home/jroc/Dropbox/projects/MoistureSensor/Software/RPi/RPi_CapDataReceive.cpp pi@[ip-address-of-the-RPi-on-my-LAN]:/home/rf24libs/RF24/examples_linux/
```

NOTE: My bash alias command for the above is: '**picodecopy**'

Then on the RPi, cd into the build directory and execute 'make'

BUT if this is the first iteration of the program, you need to first add it to CMakeLists.txt in the examples_linux directory (above the build directory).

NOTE: *All the code is under git version control*. So the executables for both the ATTiny and RPi are on my desktop.

GitHub Repo: https://github.com/JeffRocchio/Wireless-Soil-Moisture-Sensor

Next Step: Fabricate a few different capacitor sensors to test.

12/07/2023 --

Results: **Failure** in trying to get a dummy service to auto-run on the Raspberry Pi boot-up.

I created a simple CPP program to act as a dummy service. Wanted to do this before trying to configure the main RPi sensor data capture program as a service so that I could work out the