Setting up `broadcast` and `recvmsg` scripts

Louis Marais

Start: 2020-09-23
Last: 2020-09-23

1. Description of the software

These very simple scripts are part of the gasmeter distribution but their use is not limited to this application. They do what they say. `broadcast.py` send data across the network and `recvmsg.py` captures the data and stores it. The behaviour of both bits of sotfware are determined by configuration files. They can be made to run autonomously using the `kickstart` functionality of the OpenTTP distribution.

Due to the nature of our network I had to devise a system that would work within a subnet with dynamically assigned addresses. If I had control over the network I would have assigned IP addresses to the various devices and grabbed the data when I needed it.

The `broadcast.py` script expects a single value in a file; this is the value that is transmitted. The script checks the modification date of the file and when this changes it reads the content and transmits the value using a UDP broadcast message on a selected port. The `recvmsg.py` script listens on the same port and checks any received messages against its configuration. If it finds a match it captures the data and stores it locally in a data file. You can do whatever you want with the data. I have used these scripts to capture data from an environmental sensing device, from flow and from gas meters.

2. Getting hold of the software

The software is hosted on github. Clone the distribution into a suitable directory, I usually use the `~/src` directory:

```
cd ~/src/

git clone https://gthub.com/ELouisMarais/gasmeter/

cd gasmeter

git checkout develop

cd software

ls -l

total 52
-rwxrwxrwx 1 root root  6582 Aug 19 15:40 broadcast.py
-rwxrwxrwx 1 root root   880 Aug 19 15:40 build
-rwxrwxrwx 1 root root  4793 Aug 19 15:40 countpulses.c
-rwxrwxrwx 1 root root 14336 Aug 19 15:40 gasmeter.c
-rwxrwxrwx 1 root root  3300 Aug 19 15:40 intensity.c
-rwxrwxrwx 1 root root  8349 Aug 19 15:40 recvmsg.py
drwxrwxrwx 1 root root     0 Aug 19 15:40 threadedServer
```

We are interested in setting up and testing the `broadcast.py` and `recvmsg.py` scritps. Copy these to your `~/bin` directory

```
cp broadcast.py ~/bin/
cp recvmsg.py ~/bin/
```

3. Configuring

To test our application we will first set up a file with a measurement value:

```
echo 54.5 > ~/tmp/value.dat
```

Now we can create the configurations. For the `broadcast.py` script:

```
cd ~/etc/
nano broadcast.conf
```

Add the following lines to the file:

```
[main]
name = testing
targets = value
path = tmp
lockfile = broadcast.lock

[value]
file = value.dat
```

Then save (`<Ctrl>-O`) and exit (`<Ctrl>-X`). The `[main]` section requires a `name` entry. The same name will be used to identify the message for the `recvmsg.py` script. For each of the `targets` entries there needs to be a section with that name. These sections contain a single entry that is a file name containing the value to be transmitted. The `path` entry is an absolute path or a relative path from the current user's home directory where the data files are stored. The `lockfile` is the file that is created when the program runs. This file is used to check if the program is already running (multiple instances are not allowed). It is also used by the `kickstart` subsystem if that is utilised.

Configuration for the `recvmsg.py` script:

```
cd ~/etc/
nano recvmsg.conf
```

Add the following lines to the file:

```
[main]
targets = demonstration
lockfile = recvmsg.lock
lock path = logs

[demonstration]
name = testing
data = value
path = data/value
file extension = dat
file type = MJD
```

The `[main]` section contains the `targets` to be checked for. The `lockfile` is the file that is created when the program runs. This file is used to check if the program is already running (multiple

instances are not allowed). It is also used by the `kickstart` subsystem if that is utilised. The `lock path` entry shows where the `lock file` is stored. Each of the `targets` entries has its own section. In these the `name` entry MUST be the same as the one used in the `broadcast.conf` file for that parameter. The `data` entry is used as the header for the entry. The `path` is where the data file is stored, and the `file extension` is the extension used for the data file. The `file type` can be one of `MJD` (modified julian day of the current date, so daily files are produced), `YYYY-MM` (file per month, `YYYYMM` (file per month) or `YYYY` (file per year).

4. Testing

Open two terminal windows, run `broadcast.py` in one and `recvmsg.py` in the other. You can run them with debugging (use the `-d` or `--debug` switch) to see what is going on. Open another terminal window so you can modify the value file (this will force the broadcast script to send the value).

Run `recvmsg.py` first, then `broadcast.py`. The `broadcast.py` script always checks the specified file when it starts and sends the value. It then waits until the file modification date changes before it reads it again and sends the value.

If you run the scripts with debugging on, you can see what happens. If not, you will need to check the data file (`~/data/value/{MJD}.dat` in this case) to see the progress everytime you modify the value file.

If you get an error message, carefully check the configuration and try again.