fand

Release 0.1.0

Sep 26, 2020

Contents

Simple daemon to control fan speed.

CHAPTER 1

About

The main executable of this program is the `fand-server` daemon. There are 3 main modules: server, clientrpi and fanctl. They can be accessed through their respective entry points: `fand-server`, `fand-clientrpi` and `fanctl`. They can also be accessed with `fand <module-name>`.

A server monitor the hardware and clients connect to it to get data (e.g. fan speed or override a fan speed).

```
$ fanctl get-rpm shelf1
1500
$ fanctl get-pwm shelf1
50
$ fanctl set-pwm-override shelf1 100
ok
$ fanctl get-pwm shelf1
100
$ fanctl get-rpm shelf1
3000
```

1.1 Server

The server module provide a daemon which monitor devices temperatures and find a corresponding fan speed. It listens for connections from clients, and answers to requests.

1.2 Fan clients

A client is assigned a shelf and will regularly request the server for the fan speed (percentage). It will then ajust the fan to use this speed.

Clients also send the actual fan speed in RPM to the server. This will allow other client to have access to the data from the server.

1.2.1 Raspberry Pi client

The clientrpi module will connect to a server and get a fan speed from it. It will then set the fan speed with a PWM signal through the GPIO interface of the Pi. It will also tell the server the current real speed of the fans in rpm.

1.3 Command-line interface

The fanctl module is a command line interface to interact with the server. It provides commands to get the fan speed and rpm, and also allow to override the fans speed.

1.4 Documentation

The fand documentation is available at https://fand.readthedocs.io/. The installation chapter provides install instructions and compatibility informations.

CHAPTER 2

Table of contents

2.1 Server

The server is a daemon monitoring devices temperatures.

Devices are separated in shelves. Each shelf contains a set of devices. Each device has a type (HDD, SSD, CPU).

Fan speed is determined from the temperature of the device which is in most need of cooling.

For each type of device, an effective temperature is determined from the maximum temperature of all the devices of this type. With this temperature, an effective fan speed is determined. We then have an effective fan speed for each device type, the highest fan speed is then defined as the speed for the entire shelf.

2.1.1 Examples

Start the server:

```
# fand server
```

Start and listen on `0.0.0.0:1234`:

```
# fand server -a 0.0.0.0 -P 1234
```

Start and show very verbose logging:

```
# fand server -vvv
```

2.1.2 Configuration file

Default configuration file is read from either `/etc/fand.ini`, the `FAND_CONFIG` environment variable, or `./fand.ini`. There is also a `-c` parameter to specify the config file path.

The configuration is in the `ini` format.

It must have a `[DEFAULT]` section wich contain a `shelves` key listing shelves names to use. This section also contains default configuration for fan speed.

For each shelf, a section with its name has to be defined. It will contain a `devices` key listing devices assigned to this shelf. It can also override fan speed defined in `[DEFAULT]`.

Example configuration file:

```
# Example configuration file for fand

# DEFAULT section, mandatory
[DEFAULT]

# List shelves, comma separated, mandatory
# Each shelf must have a section with its name
shelves = shelf1, shelf2

# Default hdd_temps configuration
# Dictionnary: `temperature in deg C: speed in percentage`
# Example here: if the drive is at 37 deg C, corresponding speed is 30%
#               if the drive is at 30 deg C, corresponding speed is 25%
#               if the drive is < 37 deg C, corresponding speed is 25%
hdd_temps = 0: 25, 37: 30, 38: 40, 39: 50, 40: 75, 41: 100

# Default ssd_temps configuration
# Same format as hdd_temps, but the values are for SSD rather than HDD
ssd_temps = 0: 25, 60: 40, 62.5: 50, 65: 70, 67.5: 90, 70: 100

# Default cpu_temps configuration
# Same format as hdd_temps, but the values are for the CPU rather than HDD
cpu_temps = 0: 25, 75: 40, 80: 60, 85: 80, 90: 100

# Configuration for shelf shelf1
[shelf1]

# List of devices in shelf shelf1, mandatory, newline separated
# Each line is `serial; position`
# serial: serial number of the device
# position: position information about the drive, used to help locate it
devices = AD7E4EE5B03693D6; drive 1,1
          F63414EF35A424FB; drive 1,2
          C198DB33426BE180; system drive

# Configuration for shelf shelf2
[shelf2]

# List of devices
devices = 062110D2532377B5; small drive 2,1
          FFBEB97C6ED5953B; system drive

# Override hdd_temps for this shelf
# This configuration will be used for this shelf only
hdd_temps = 0: 40, 37: 50, 38: 60, 39: 75, 40: 85, 41: 100
```

2.1.3 Python API

`fand.server.`**`REQUEST_HANDLERS`**
Dictionnary assigning a Request to a function

Device

class fand.server.**Device**(*serial: str*, *position: str*)
Class handling devices to get temperature from

Parameters

- **serial** – Device serial number
- **position** – Device positionning information

find() → fand.server.Device._DeviceWrapper
Search device on the system

position = None
Device positionning information

serial = None
Device serial number

temperature
Current drive temperature

type
DeviceType

update() → None
Update device informations

class Device.**DeviceType**
Bases: enum.Enum

Enumeration of device types, to identify Device objects

CPU = 3
System CPU

HDD = 1
HDD

NONE = 0
Unknown device

SSD = 2
SSD

Shelf

class fand.server.**Shelf**(*identifier: str, devices: Iterable[fand.server.Device], sleep_time: float = 60, hdd_temps: Optional[Dict[float, float]] = None, ssd_temps: Optional[Dict[float, float]] = None, cpu_temps: Optional[Dict[float, float]] = None*)
Class handling shelf data

Parameters

- **idenifier** – Shelf identifier (name)
- **devices** – Iterable of Device objects
- **sleep_time** – How many seconds to wait between each shelf update

- **hdd_temps** – Dictionnary in the format `temperature: speed`, temperature in Celcius, speed in percent, must have a 0 deg key
- **ssd_temps** – Dictionnary in the format `temperature: speed`, temperature in Celcius, speed in percent, must have a 0 deg key
- **cpu_temps** – Dictionnary in the format `temperature: speed`, temperature in Celcius, speed in percent, must have a 0 deg key

Raises ***ShelfTemperatureBadValue*** – One of the temps dictionnary is invalid

identifier = None
Shelf identifier (name)

pwm
Get shelf PWM value

Reading get the effective PWM value. Changing override the PWM value.

Raises ***ShelfPwmBadValue*** – Invalid value

pwm_expire
Set the PWM override expiration date, defaults to local timezone

Raises ***ShelfPwmExpireBadValue*** – Invalid value

rpm
Shelf fan speed RPM

Raises ***ShelfRpmBadValue*** – Invalid value

sleep_time = None
How many seconds to wait between each shelf update

update() → None
Update shelf data

add_shelf

fand.server.**add_shelf**(*shelf: fand.server.Shelf*) → None
Add a Shelf to the dictionnary of known shelves

Parameters **shelf** – Shelf to add

listen_client

fand.server.**listen_client**(*client_socket: socket.socket*) → None
Listen for client requests until the connection is closed

Parameters **client_socket** – Socket to listen to

read_config

fand.server.**read_config**(*config_file: Optional[str] = None*) → Iterable[fand.server.Shelf]
Read configuration from a file, returns an iterable of shelves

Parameters **config_file** – Configuration file to use, defaults to the `FAND_CONFIG` environment variable or `./fand.ini` or `/etc/fand.ini`

Raises ***ServerNoConfigError*** – Configuration not found

shelf_thread

fand.server.**shelf_thread**(*shelf: fand.server.Shelf*) → None
Monitor a shelf

Stops when *fand.util.terminating()* is True or when an unexpected exception occur.

Parameters **shelf** – Shelf to monitor

main

fand.server.**main**() → NoReturn
Entry point of the module

daemon

fand.server.**daemon**(*config_file: Optional[str] = None*, *address: str = 'build-11968335-project-622900-fand'*, *port: int = 9999*) → None
Main function

Parameters

- **config_file** – Configuration file to use, defaults to the FAND_CONFIG environment variable or ./fand.ini or /etc/fand.ini
- **address** – Address of the interface to listen on, defaults to hostname
- **port** – Port to listen on

Raises ***ListeningError*** – Error while listening for new connections

2.2 Client: Raspberry Pi

The Raspberry Pi client control the fan speed by sending a PWM signal through the GPIO pins.

Two pins are used:

- The PWM pin used to output the PWM signal regulating the fan speed. It should be connected to the PWM input of the fans.
- The RPM pin used to read the actual fan speed in RPM. It should be connected to the tachometer output of the fans.

2.2.1 PWM backend

By default, gpiozero will use whatever supported library is installed.

To manually set which backend to use, you can use the GPIOZERO_PIN_FACTORY environment variable.

See the gpiozero.pins documentation for more information.

2.2.2 Examples

Start the client:

```
# fand clientrpi
```

Start and use GPIO pin 17 for PWM, and pin 18 for tacho:

```
# fand server -W 17 -r 18
```

Start with verbose output and connect to server at server-host:1234:

```
# fand server -v -a server-host -P 1234
```

2.2.3 Python API

fand.clientrpi.**SLEEP_TIME = 60**
How much time to wait between updates

GpioRpm

class fand.clientrpi.**GpioRpm**(*pin: int*, *managed: bool = True*)
Class to handle RPM tachometer input from a fan

Parameters

- **pin** – GPIO pin number to use
- **managed** – set to true to have the GPIO device automatically closed when *fand.util.terminate()* is called

Raises ***GpioError*** – Received a gpiozero.GPIOZeroError

close() → None
Close the GPIO device

Raises ***GpioError*** – Received a gpiozero.GPIOZeroError

rpm = None
RPM value

update() → None
Update the RPM value

GpioPwm

class fand.clientrpi.**GpioPwm**(*pin: int*, *managed: bool = True*)
Class to handle PWM output for a fan

Parameters

- **pin** – GPIO pin number to use
- **managed** – set to true to have the GPIO device automatically closed when *fand.util.terminate()* is called

Raises ***GpioError*** – Received a gpiozero.GPIOZeroError

close() → None
Close the GPIO device

Raises ***GpioError*** – Received a gpiozero.GPIOZeroError

pwm
PWM output value, backend is gpiozero.PWMLED.value

Raises ***GpioError*** – Received a gpiozero.GPIOZeroError

add_gpio_device

fand.clientrpi.**add_gpio_device**(*device: Union[GpioRpm, GpioPwm]*) → None
Add a GPIO device to the set of managed GPIO devices

Parameters **device** – GPIO device to add

Raises ***TerminatingError*** – Trying to add a socket but *fand.util.terminating()* is True

main

fand.clientrpi.**main**() → NoReturn
Module entry point

daemon

fand.clientrpi.**daemon**(*gpio_pwm: fand.clientrpi.GpioPwm*, *gpio_rpm: fand.clientrpi.GpioRpm*, *shelf_name: str = 'build-11968335-project-622900-fand'*, *address: str = 'build-11968335-project-622900-fand'*, *port: int = 9999*) → None
Main function of this module

Parameters

- **gpio_pwm** – GPIO device to use for PWM output
- **gpio_rpm** – GPIO device to use for RPM input
- **shelf_name** – Name of this shelf, used to communicate with the server
- **address** – Server address or hostname
- **port** – Port number to connect to

2.3 Command-line interface

fanctl is a CLI allowing to interract with the server.

It is basically a fand client, but does not act as a fan controller.

The user can get the assigned fan speed in percentage, the real fan speed in rpm.

The user can override the assigned fan speed. The override can also be set to expire in a given amount of time, or expire at a given date and time.

2.3.1 Examples

Ping the server at 192.168.1.10:1234:

```
$ fanctl -a 192.168.1.10 -P 1234 ping
```

Get the assigned fan speed for shelf 'shelf1':

```
$ fanctl get-pwm shelf1
```

Override fan speed of 'myshelf' to 100%:

```
$ fanctl set-pwm-override myshelf 100
```

Remove override in 1 hour and 30 minutes:

```
$ fanctl set-pwm-expire-in myshelf 1h30m
```

Remove override now:

```
$ fanctl set-pwm-override myshelf none
```

2.3.2 Python API

`fand.fanctl.`**`DATETIME_DATE_FORMATS`**
List of accepted string formats for `datetime.datetime.strptime()`

`fand.fanctl.`**`DATETIME_DURATION_FORMATS`**
List of accepted regex formats for `datetime.timedelta`

`fand.fanctl.`**`ACTION_DICT`**
Dictionnary associating action strings to their corresponding functions

main

`fand.fanctl.`**`main`**`()` → NoReturn
Entry point of the module

send

`fand.fanctl.`**`send`**`(`*action: str*, **args*, *address: str = 'build-11968335-project-622900-fand'*, *port: int = 9999*`)` → None
Main function of this module

Parameters

- **`action`** – Action to call
- **`args`** – Arguments to send to the action
- **`address`** – Server address
- **`port`** – Server port

Raises ***`FanctlActionBadValue`*** – Invalid action name or arguments

2.4 Communication module

The communication module handles the low level communication between the server and the clients.

It provides functions to send and receive a request with arguments to a given socket.

It can start a connection with the server and close a socket. It also keep track of sockets to close them automatically at the end of the program.

2.4.1 Examples

```
from fand.communication import *
s = connect('myserver.example.com', 9999)
send(s, Request.GET_PWM, 'myshelf1')
req, args = recv(s)
if req == Request.SET_PWM:
    print("The fan speed of", args[0], "is", args[1])
else:
    print("The server did not answer the expected request")
```

2.4.2 Python API

Request

class fand.communication.**Request**

Bases: enum.Enum

Enumeration of known requests

ACK = 'ack'
Acknowledge a previously received *Request*

DISCONNECT = 'disconnect'
Notification of disconnection

GET_PWM = 'get_pwm'
Request for a *Request.SET_PWM* to get current PWM

GET_RPM = 'get_rpm'
Request for a *Request.SET_RPM* to get current RPM

PING = 'ping'
Request for a *Request.ACK*

SET_PWM = 'set_pwm'
Give the current PWM

SET_PWM_EXPIRE = 'set_pwm_expire'
Set the expiration date of the PWM override

SET_PWM_OVERRIDE = 'set_pwm_override'
Override the PWM value

SET_RPM = 'set_rpm'
Give the current RPM

add_socket

fand.communication.**add_socket**(*sock: socket.socket*) → None
Add sock to the set of managed sockets

It can be removed with *reset_connection()* and will automatically be when *fand.util.terminate()* is called.

Parameters **sock** – Socket to add

Raises ***TerminatingError*** – Trying to add a socket but *fand.util.terminating()* is True

is_socket_open

fand.communication.**is_socket_open**(*sock: socket.socket*) → bool
Returns True if sock is currently managed by this module

This will be False after a socket has been closed with *reset_connection()*.

Parameters **sock** – Socket to test

send

fand.communication.**send**(*sock: socket.socket*, *request: fand.communication.Request*, **args*) → None
Send a request to a remote socket

Parameters

- **sock** – Socket to send the request to
- **request** – Request to send
- **args** – Request arguments

Raises

- ***UnpicklableError*** – Given data cannot be pickled by pickle
- ***FandTimeoutError*** – Connection timed out
- ***SendReceiveError*** – Error while sending the data

recv

fand.communication.**recv**(*sock: socket.socket*) → Tuple[fand.communication.Request, Tuple]
Receive a request from a remote socket, returns (request, args)

Parameters **sock** – Socket to receive the request and its arguments from

Raises

- ***FandTimeoutError*** – Connection timed out
- ***SendReceiveError*** – Error while receiving the data
- ***FandConnectionResetError*** – No data received or *Request.DISCONNECT* received
- ***CorruptedDataError*** – Invalid data received

connect

fand.communication.**connect**(*address: str*, *port: int*) → socket.socket
Connect to server and returns socket

Parameters

- **address** – Server address
- **port** – Server port

Raises

- ***FandTimeoutError*** – Connection timed out
- ***ConnectionFailedError*** – Failed to connect to remote socket

reset_connection

fand.communication.**reset_connection**(*client_socket: socket.socket*, *error_msg: Optional[str] = None*, *notice: bool = True*) → None
Closes a connection to a client

Parameters

- **client_socket** – Socket to close
- **error** – Error to send
- **notice** – Send a notice about the reset to the remote socket

2.5 Utilities module

The util module provides some functions used by most fand modules.

It provides a *terminate()* function to make the daemon and its threads terminate cleanly. It provide a *when_terminate()* function decorator to add functions to call when *terminate()* is called, allowing custom cleanup from modules.

It also has a default signal handler, and a default argument parser.

2.5.1 Python API

terminate

fand.util.**terminate**(*error: Optional[str] = None*) → None
Function terminating the program

Sets the terminate flag (see *terminating()*), and does some cleanup (see *when_terminate()*)

Parameters **error** – Error message to print

sys_exit

fand.util.**sys_exit**() → NoReturn
Exit the program with the error from *terminate()* if any

terminating

fand.util.**terminating**() → bool
Returns True if the program is terminating, else False

when_terminate

fand.util.**when_terminate**(*function: Callable*, **args*, ***kwargs*) → None
Add function to call when terminating

Parameters

- **function** – Function to call
- **args** – Arguments to call the function with
- **kwargs** – Keyworded arguments to call the function with

sleep

fand.util.**sleep**(*secs: float*) → None
Sleep some time, stops if terminating

Parameters **secs** – Number of seconds to sleep

default_signal_handler

fand.util.**default_signal_handler**(*sig: signal.Signals*, *_: Any*) → None
Default signal handler

parse_args

fand.util.**parse_args**(*parser: argparse.ArgumentParser*) → argparse.Namespace
Add common arguments, parse arguments, set root logger verbosity

Parameters **parser** – Argument parser to use

2.6 Exceptions

The exception module provides fand-specific exceptions. All exceptions raised by fand descend from *FandError*.

Certain errors have multiple parents. For instance, *ShelfNotFoundError* is a *FandError*, but also a ValueError.

2.6.1 Python API

Common fand errors

exception fand.exceptions.**FandError**
Bases: Exception

Base class for all exceptions in fand

exception fand.exceptions.**TerminatingError**
Bases: *fand.exceptions.FandError*

Daemon is terminating

Communication related errors

exception fand.exceptions.**CommunicationError**
Bases: *fand.exceptions.FandError*

Base class for communication related errors

exception fand.exceptions.**SendReceiveError**
Bases: *fand.exceptions.CommunicationError*, ConnectionError

Error while sending or receiving data

exception fand.exceptions.**ListeningError**
Bases: *fand.exceptions.CommunicationError*, ConnectionError

Error when listening to clients

exception fand.exceptions.**ConnectionFailedError**
Bases: *fand.exceptions.CommunicationError*, ConnectionError

Cannot connect to given address and port

exception fand.exceptions.**CorruptedDataError**
Bases: *fand.exceptions.CommunicationError*, ConnectionError

Invalid data received

exception fand.exceptions.**FandConnectionResetError**
Bases: *fand.exceptions.CommunicationError*, ConnectionResetError

Connection reset

exception fand.exceptions.**FandTimeoutError**
Bases: *fand.exceptions.CommunicationError*, TimeoutError

Connection timed out

exception fand.exceptions.**UnpicklableError**
Bases: *fand.exceptions.CommunicationError*, ValueError

Given value cannot be pickled

Shelf related errors

exception fand.exceptions.**ShelfTemperatureBadValue**
Bases: *fand.exceptions.FandError*, ValueError

Temperatures given for shelf are invalid

exception fand.exceptions.**ShelfRpmBadValue**
Bases: *fand.exceptions.FandError*, ValueError

RPM value is invalid

exception fand.exceptions.**ShelfPwmBadValue**
Bases: *fand.exceptions.FandError*, ValueError

PWM value is invalid

exception fand.exceptions.**ShelfPwmExpireBadValue**
Bases: *fand.exceptions.FandError*, ValueError

PWM override expiration date is invalid

exception fand.exceptions.**ShelfNotFoundError**
Bases: *fand.exceptions.FandError*, ValueError

Given shelf name is unknown

Server and clients specific errors

exception fand.exceptions.**ServerNoConfigError**
Bases: *fand.exceptions.FandError*, FileNotFoundError

No configuration file found

exception fand.exceptions.**GpioError**
Bases: *fand.exceptions.FandError*

Any GPIO related errors

exception fand.exceptions.**FanctlActionBadValue**
Bases: *fand.exceptions.FandError*, ValueError

No action found with this name and parameters

2.7 Installation

2.7.1 Python dependencies

Server

- pySMART (homepage, pypi, source): access drives SMART data
- psutil (homepage, pypi, source, doc): access CPU temperatures

Raspberry Pi client

- gpiozero (homepage, pypi, source, doc): access GPIO for PWM and tachometer signals
- gpiozero's native pin factory does not currently supports PWM (sept 2020), you need one of the following packages:
  - RPi.GPIO (homepage, pypi, source, doc): does not supports hardware PWM, only uses software PWM
  - pigpio (homepage, pypi, source, doc): supports hardware PWM but cannot work with Linux's lockdown LSM
  - RPIO (homepage, pypi, source, doc): unmaintained

Documentation

- sphinx (homepage, pypi, source, doc): documentation generator

Test

- tox (homepage, pypi, source, doc): test management
- pytest (homepage, pypi, source, doc): testing
- flake8 (homepage, pypi, source, doc): quality assurance
- mypy (homepage, pypi, source, doc): type checking

2.7.2 Non-Python dependencies

Server

- smartmontools (homepage, source, doc): pySMART's backend

2.7.3 Installation

Server

Install smartmontools on your system with your prefered package manager.

Install fand with:

```
$ pip install fand[server]
```

Raspberry Pi client

Install fand with one of the following commands:

- Install with RPi.GPIO:

```
$ pip install fand[clientrpi-rpi-gpio]
```

- Install with pigpio:

```
$ pip install fand[clientrpi-pigpio]
```

- Install with RPIO:

```
$ pip install fand[clientrpi-rpio]
```

Other modules

No extra dependencies required, you can install with:

```
$ pip install fand
```

Custom installation

You can cumulate extra dependencies:

```
$ pip install fand[server,clientrpi-pigpio]
```

Documentation

To build the documentation, you can install fand with:

```
$ pip install fand[doc]
```

Download the fand source code:

```
$ pip download --no-deps --no-binary fand fand
$ tar -xf <filename>
$ cd <directory>
```

And build the documentation with:

```
$ cd doc
$ make html
```

The documentation will be built in the `build` directory.

Testing

To run CI or QA tests, you can install fand with:

```
$ pip install fand[test]
```

Run the tests with:

```
$ tox
```

2.7.4 Python version support

Officially supported Python versions

fand should support any Python 3 version supported by upstream.

- Python 3.6
- Python 3.7
- Python 3.8

Officially supported Python implementations

- CPython
- PyPy

2.7.5 Operating system support

Server

- Linux
- FreeBSD
- Windows: untested, missing support for CPU temperature monitoring (`psutil.sensors_temperatures()` does not supports Windows)

Raspberry Pi client

- Linux
- Windows: untested
- FreeBSD: unsupported, missing support for any of the gpiozero's backend for PWM

Other modules

- Any OS with Python

2.8 License

fand is licensed under the MIT license.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

f

Index

R

S

T

U

W