Speak Python with Devices

Petertc Chu @ EuroPython 2020
Why this session?

We will see:

- How Python can be applied in IoT/infrastructure automation tasks

You might:

- wish to know how Python can be used beyond data analysis and web dev
- a Pythonista who interested in craft some touchable things
- want to acquire something new into your Python skillset
Outline

- Devices in Linux/UNIX-like system
- How to manipulate a device
- Manipulate a device in Python, a mini example
- A more attractive example
User space

Your **Python** interpreter, packages and code here

Kernel space **Driver** of LED panel, camera, sensors and other devices...

Hardware LED panel, camera and sensors on bluetooth/USB/serial/parallel ports...

**Computer organization, briefly**
Everything is a file, so is a device
Manipulate a device

with common file operations:

- open()
- write()
- read()
- close()

Example: blink an LED on Raspberry Pi

```python
import time

def led_on():
    with open('/sys/class/leds/led0/brightness', 'w') as brightness:
        brightness.write('255')
        print('on')

def led_off():
    with open('/sys/class/leds/led0/brightness', 'w') as brightness:
        brightness.write('0')
        print('off')

def blink(sec: int):
    disable_trigger()
    for i in range(sec):
        led_on()
        time.sleep(1)
        led_off()
        time.sleep(1)
```

and a more interesting one...
IOCTL()
Input Output Control

Read/write is not enough for a device which is more complex than an LED

Example: a modem

- **READ** - receive data
- **WRITE** - send data
- **IOCTL** - talk to the modem itself, e.g., set bitrate, get config
IOCTL - Parameters

- file descriptor
- request
  - a.k.a. (device-dependent) request code or command
  - composed of:
    - type (8 bits, a~z)
    - number (8 bits, 1~255)
    - argument size (14 bits, max 16KB)
    - direction (2 bits, R/W/RW/NONE)
- argument (string, a C struct or anything)
An analogy

request (direction, type, number, argument size)

direction

type

number

PUT /myBucket/my-object.jpg HTTP/1.1
Host: s3.amazonaws.com
Date: Fri, 24 Jul 2020 06:00:00 GMT
Authorization: authorization string
Content-Type: text/plain
Content-Length: 11434
x-amz-meta-author: Janet
Expect: 100-continue
[11434 bytes of object data]
“DON’T PANIC!”
just like RESTful APIs we use every day!
IOCTL ♡ Python
Let’s start from a mini example:

Get the name of input devices

```python
if __name__ == '__main__':
    for i in range(7):
        with open('/dev/input/event' + str(i)) as fd:
            print(get_device_name(fd))

Power Button
Sleep Button
AT Translated Set 2 keyboard
Video Bus
ImExPS/2 Generic Explorer Mouse
VirtualBox USB Tablet
VirtualBox mouse integration
```
Do IOCTL() from Python

Things to do:

- Create an IOCTL request (header)
- C<>Py type conversion (body)
- Do IOCTL system call

(At least) two approaches:

- C extension module
- Pure Python solution
Approach 1: C extension module

Step 1: IOCTL call

```c
char* getDeviceName(char* device, char* name, size_t len)
{
    int fd = -1;
    int result = 0;

    fd = open(device, O_RDONLY);

    // EVIOCGNAME(len) _IOC(_IOC_READ, 'E', 0x06, len) (linux/input.h)
    int request = EVIOCGNAME(len);

    result = ioctl(fd, request, name);

    close(fd);
}
```

Create IOCTL request (header) by macros
Approach 1: C extension module

Step 2: C<->Py type conversion (req/resp body)

```c
static PyObject * get_device_name(PyObject *self, PyObject *args) {
    // parse input
    const char *device;
    if (!PyArg_ParseTuple(args, "s", &device))
        return NULL;

    char name[256] = "Unknown";
    getDeviceName(device, name, 256);

    // return name
    return Py_BuildValue("s", name);
}
```
Approach 1: C extension module

Step 3: packaging

```c
static PyMethodDef EVIOCG_Methods[] = {
    {"get_device_name",  get_device_name, METH_VARARGS,
        "Get device name."},
    {NULL, NULL, 0, NULL} /* Sentinel */
};

static struct PyModuleDef EVIOCG_MODULE = {
    PyModuleDef_HEAD_INIT,
    "eviocg", /* name of module */
    NULL, /* module documentation, may be NULL */
    -1,    /* size of per-interpreter state of the module,
           or -1 if the module keeps state in global variables. */
    EVIOCG_Methods
};

PyMODINIT_FUNC PyInit_eviocg(void) {
    PyObject* m = PyModule_Create(&EVIOCG_MODULE);
    if (m == NULL) {
        return NULL;
    }
    return m;
}
```
Approach 1: C extension module

Install and use it as usual

```
pyeviogc# pip list|grep eviogc
evliocg 0.0.0

IPython 7.9.0 -- An enhanced Interactive Python. Type '?' for help.
In [1]: import eviogc
In [2]: eviogc.get_device_name('/dev/input/event3')
Out[2]: 'Video Bus'
```
Approach 2: Pure Python solution

Step 1: Create an IOCTL request (header)

- porting IOC* macros from asm-generic/ioctl.h => Someone has already done it!
  - olavmrk/python-ioctl
  - vpelletier/python-ioctl-opt
- porting driver specific macros

```python
from ioctl_opt import IOC, IOC_READ

# porting from linux/input.h
EVIOCNAME = lambda length: IOC(IOC_READ, ord('E'), 0x06, length)
```

Courtesy of
vpelletier/python-ioctl-opt/blob/master/README.rst
**Approach 2: Pure Python solution**

Step 2: ioctl call and C<-> data type conversion

```python
def get_device_name(fd, length=1024):
    name = bytearray(length)
    actual_length = fcntl.ioctl(fd, EVIOCNAME(length), name, True)
    return name[:actual_length - 1].decode('UTF-8')
```

- Use built-in module
- byte array <-> str
- Macro we implemented in the first step
Approach 2: Pure Python solution

Same result we saw before

```python
if __name__ == '__main__':
    for i in range(7):
        with open('/dev/input/event' + str(i)) as fd:
            print(get_device_name(fd))
```
OK now I know how these things work but...

Question: any use case? 🤔
a cat food feeder?
The CERN Advanced STORage system (CASTOR)

http://castor.web.cern.ch/castor/


http://castor.web.cern.ch/castor/

What kind of data?

- Digitized tracks of particles in detectors
- Data must be collected as it is generated (~18 months uninterrupted)
- One event similar to the others
- Volume: 15-20 PB/year
- Transfer rates: ~0.5 – 1.5 GB/s
- Keep for > 10 years (forever)


https://youtu.be/IDgXa0ioVTs
Explore the universe 🚀 with what we learn today!
Quick start

- Device: mhVTL simulated
- Driver: Linux SCSI tape (st) driver
Quick start

Typical tape write procedure:

1. Find the cartridge by barcode scanner
2. Load the cartridge by a robot arm
3. Check the cartridge status is ready
4. Rewind the cartridge by a tape drive
5. Write data on the cartridge
6. Unload the cartridge
Get tape status by C extension

```c
static PyObject *
do_status(PyObject *self, PyObject *args) {
    // parse input
    const char *device;
    if (!PyArg_ParseTuple(args, "s", &device))
        return NULL;
    // open device file
    int fd;
    if ((fd = open(device, O_RDONLY)) < 0) {
        PyErr_SetFromErrno(PyExc_OSError);
        return NULL;
    }
    // execute ioctl command
    struct mtget status;
    if (ioctl(fd, MTIOCGET, (char *)&status) < 0) {
        PyErr_SetFromErrno(PyExc_OSError);
        return NULL;
    }
    if (status.mt_type != MT_ISSCSI2) {
        PyErr_SetString(PyExc_NotImplementedError, "Unsupported tape type");
        return NULL;
    }
    close(fd);
    // return status info in dict
    return Py_BuildValue("{s:i,s:i,s:i}",
        "file number", status.mt_fileno,
        "block number", status.mt_blkno,
        "partition", (status.mt_resid & 0xff)
    );
}
```
def rewind(device):
    MTREW = 6
    mt_com = struct.pack('hi', MTREW, 1)
    MTIOCTOP = IOW(ord('m'), 1, len(mt_com))

    with open(device, 'r') as fd:
        fcntl.ioctl(fd, MTIOCTOP, mt_com)

def status(device):
    long_size = 8
    int_size = 4
    status = bytearray(long_size * 5 + int_size * 2)
    MTIOCGET = IOR(ord('m'), 2, len(status))

    with open(device, 'r') as fd:
        fcntl.ioctl(fd, MTIOCGET, status)
    status = struct.unpack('lllllii', status)
    return {
        "file number": status[-2],
        "block number": status[-1],
        "partition": status[1] & 0xff
    }
**Bonus:**
rewind cartridge by ctypes

Define input/output/buffer data structure by extending ctypes.Structure

class mtop(ctypes.Structure):
    _fields_ = [
        ('mt_op', ctypes.c_short),
        ('mt_count', ctypes.c_int)
    ]

def rewind(device):
    MTIOCTOP = ioctl.linux.IOW('m', 1, ctypes.sizeof(mtop))
    MTREW = 6
    mt_com = mtop(MTREW, 1)
    with open(device, 'r') as fd:
        ioctl.ioctl(fd.fileno(), MTIOCTOP,
                    ctypes.byref(mt_com))
Takeaway

- You can manipulate a device like a file
- IOCTL is just like RESTful APIs we use every day
- Yes, we can speak Python while working on IoT and infra automation tasks

Example code is available on https://github.com/hrchu/playioctl
Thank you! 🙏🙏🙏

@petertc_chu