1 はじめに

1.1 自己紹介

こんにちは。おそらく多くの読者さんには初めまして。SunPro メンバーの @hideo54 です。灘高校で高校 1 年生をしていますが、学業の傍らいろいろやりたい放題やっています。

灘高校では 灘校パソコン研究部 (NPCA) に所属しています。今のところ、SunPro メンバーはだいたいみんな NPCA 卒業生です。よければ以後お見知り置きを。

自己紹介リンクいろいろ

- Twitter: @hideo54
- Homepage: https://hideo54.com
- Blog: https://blog.hideo54.com
- Wishlist: https://wishlist.hideo54.com
- E-mail: contact@hideo54.com

1.2 やること

先日東京に行く機会があったのですが、秋葉原でぼっちうろうろしていた時、「NFC タグ」というキーワードを含んだでかい広告を見かけて、「そういや NFC で遊んだことないなぁ…」と思い、つい衝動買いしてしまいました。20 枚 1200 円です。1 枚あたりで考えてみると安い気がしますね。そこで、早速 NFC タグでいろいろ遊んでみようと思い、第一弾としてタイトルのようなことをしようと思ったのです。

その前に、次の章では NFC の基本のごく一部について、その次では NFC タグの Python での読み書きについて書いていきます。実は、NFC タグの読み書きの方は、すでに Raspberry Pi Advent Calendar 2015 19 日目の記事 として僕のブログで公開したものを少し加筆したものにすぎません。今回の工作では読み書きは必要ないのですが、nfcpy で遊ぶにはたいてい必要になりますし、するべき準備は同じなので、ついでにこちらにも載せることにしました。よろしくお願いします。

2 NFCの基本的なこと

2.1 モード

NFC フォーラムという業界標準団体によると、NFC (Near Field Communication / 近距離無線通信) という技術では、3 つのモードがサポートされています。

- カードエミュレーションモード: NFC端末をICチップとして振る舞わせる
- リーダー/ライターモード: NFC端末と情報を読み書きする
- P2Pモード: NFC端末同士でデータのやりとりをする

今回はカードリーダー / ライターモードを使用します。

2.2 NDEF(NFC Data Exchange Format)

NDEF (NFC Data Exchange Format) は、NFC デバイス間でデータを交換したり、NFC タグ上にデータを持たせたりする時の、バイナリのフォーマットのことを指します。NDEF Message は 1 つ以上の NDEF Record で構成されており、それぞれの NDEF Record にデータを持たせていく、という風な仕組みで使われます。知識がなくても基本的なことは扱えるのでここでは省略しますが、各 Record に持たせる Header のバイナリの規則もなかなか面白いので、興味を持った方は調べてみてください。

3 NFCタグを読み書きする

3.1 必要なもの

- Raspberry Piおよびその使用に必要な最低限の環境（電源、SDカード）
  - OS: Raspbian
- NFCタグ
- NFCリーダー/ライター

Raspberry Pi は、安定性の理由から B+ 以上がおすすめです。今回、Raspbian は Debian 8 (Jessie) ベースのものを、NFC リーダー / ライターには、SONY の PaSoRi を使用しました。

3.2 環境構築

libnfc を使う手もありますが、工作の際には GPIO ピンを Python でいじった方がよりやりやすいと思うので、Python でお手軽にいじれる nfcpy を利用します。というわけで、利用にあたって必要なものをインストールします。

libusb

Raspbian に標準で入ってました。確かに入ってそう。

PyUSB

PaSoRi は USB 接続で使用しますので、Python から USB ポートにアクセスするために必要です。僕は pip で入れました。

```
sudo pip install pyusb --pre
```

nfcpy

ドキュメントに従って入れます。

```
sudo apt-get install bzr
cd ~
bzr branch lp:nfcpy trunk
```

これで、~/trunk下にいろいろ落ちたはずです。

どこでも nfcpy を使えるようにする

（ここらへん環境によって違いそうな気がするけど、よく考えたら Raspbian 前提にしてるから別に注釈いらないか。）

```
cp -R trunk/nfc /usr/local/lib/python2.7/dist-packages
```

sudo しなくても USB デバイスを見れるようにする

デフォルトだと、sudo をつけて Python プログラムを実行しないと、USB 接続のリーダーを認識してくれません。そこで、リーダー情報を登録して、sudo しなくても認識できるようにします。

```
dmesg | tail
```

すると、

```
[ 4225.809847] usb 1-1.3: new full-speed USB device number 4 using dwc_otg
[ 4225.914395] usb 1-1.3: New USB device found, idVendor=054c, idProduct=06c3
[ 4225.914435] usb 1-1.3: New USB device strings: Mfr=1, Product=2, SerialNumber=4
[ 4225.914453] usb 1-1.3: Product: RC-S380/P
[ 4225.914469] usb 1-1.3: Manufacturer: SONY
[ 4225.914484] usb 1-1.3: SerialNumber: *******
```

みたいな感じのが出てくるので、/etc/udev/rules.d/nfcdev.rules を以下のように編集。

```
SUBSYSTEM==”usb”, ACTION==”add”, ATTRS{idVendor}==”054c”, ATTRS{idProduct}==”06c3”, GROUP==”sudo”
```

これで、sudo グループに入ってるユーザーは sudo なしで認識できるようになりました。（ここらへん、“nfc” グループを作って、ユーザーを放り込んで、nfc グループ指定にしたほうが良い気がする。）

3.3 NFC タグ内のデータの読み書き

やっと準備完了。以下のような、NFC タグ内のデータを読む簡単な Python プログラムを作成しました。

```
 1: import nfc
 2: clf = nfc.ContactlessFrontend('usb')
 3:
 4: def connected(tag):
 5:     print tag.ndef.message.pretty()
 6:
 7: clf.connect(rdwr={'on-connect': connected})
```

実行して新品の NFC タグをあてると、以下のようなデータが出ます。

```
record 1
  type   = ''
  name   = ''
  data   = ''
```

買ったまま何も手を加えていないからね。というわけでデータを書き込んでみます。

```
 1: import nfc
 2: clf = nfc.ContactlessFrontend('usb')
 3:
 4: def connected(tag):
 5:     record = nfc.ndef.TextRecord("Hello World!")
 6:     tag.ndef.message = nfc.ndef.Message(record)
 7:     print tag.ndef.message.pretty()
 8:
 9: clf.connect(rdwr={'on-connect': connected})
```

すると…

```
record 1
  type   = 'urn:nfc:wkt:T'
  name   = ''
  data   = '\x02enHello World!'
```

無事書き込めました！一旦プログラムを終了して再度読んでみても、ちゃんとデータが書き込まれていることが確認できます。めでたしめでたし。こんな感じで、お手軽に書き込めます。上記プログラムでは `nfc.ndef.TextRecord()` 関数を用いましたが、nfcpy では他にも多くの NDEF の記

録関数が用意されています。詳しくは 公式ドキュメント を参照してください。

4　NFCタグを貼ったドアを照明のスイッチにする

本題です。

実は数カ月前に引っ越しをしまして、それに伴い部屋の構造や机なども変えたのですが、概ね快適と言えるものの、部屋の LED 照明のスイッチがわりと押しにくい感じになってしまいました。

図4.1　照明のスイッチ周りの状況

そこで、「**部屋のドアをスイッチ代わりにすればいいんじゃね ?!?!**」と思いついたので、実行しました。ドアが開いてるか閉じてるかを判定するのに、NFC タグを使うことにしました。くっついていなくても、タグとリーダーが数 cm 以内にあれば良いので、動くものの状態判定には最適ですね！

（※発想自体についてはわかりませんが、必要なことについては特に新規性はないです。）

4.1 構想の概要

- 部屋のドアにNFCタグを貼り付け、そのすぐ近くの棚にNFCリーダーを置いておく。
- タグが離れた状態から近づいた状態になったとリーダーが感知した時、フラグの値を変更する。
- フラグがTrueの時はLED照明をオンに、Falseの時はオフにする。
- LED照明のオン/オフは、Raspberry PiのGPIOピンに接続された赤外線LEDへの電流をオン/オフすることによって行う。

要するに、**ドアが閉まるたびに電気をつけたり消したりする**、ということです。
とりあえず準備したらこうなりました。

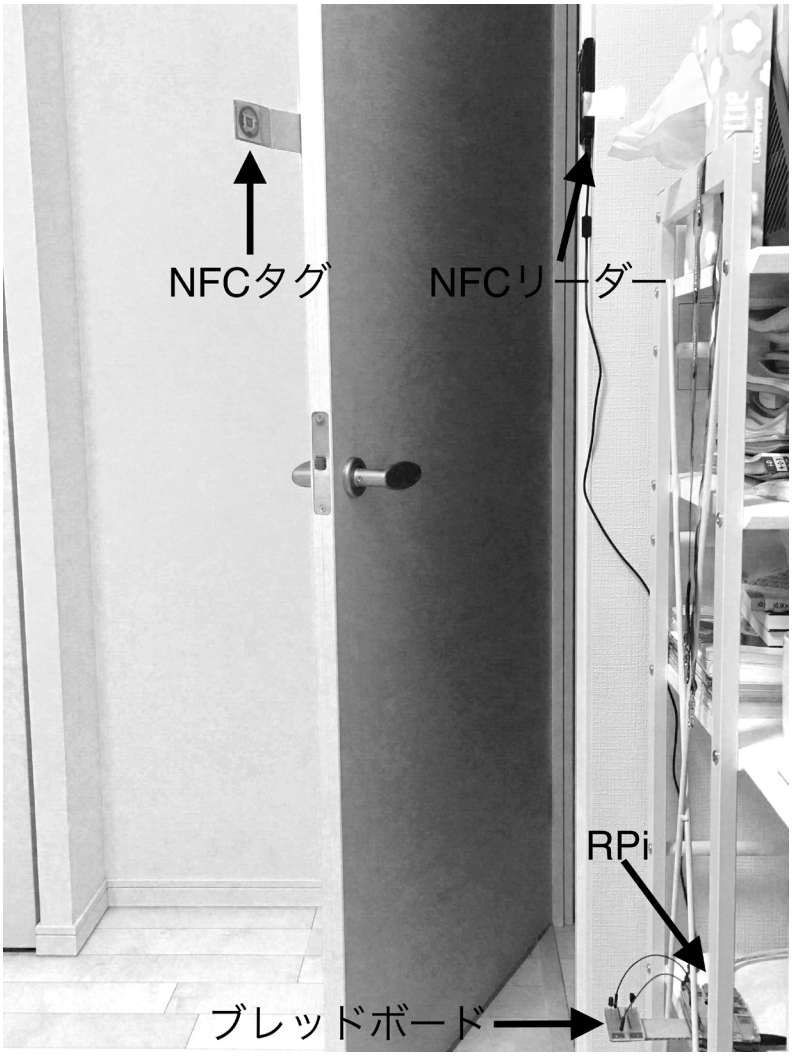

図4.2 配置の結果

ブレッドボード部分を延ばしているのは、赤外線 LED が天井に向くようにするためです。

めっちゃ「なんとかした」感が強くてあまりスマートに見えませんが、まあ動くからいいでしょう（適当）

以下、やったことです。

4.2 照明を操作する

必要なもの

- Raspberry Pi (上記と同じなので省略)
- 赤外線LED
- 赤外線受光器
- ブレッドボード、ジャンパワイヤ

環境構築

lirc をインストールします。

```
sudo apt-get install lirc
```

使用する GPIO ピン番号を決めます。僕は今回、入力を 20、出力を 21 としました。

決めた通りに配線します。

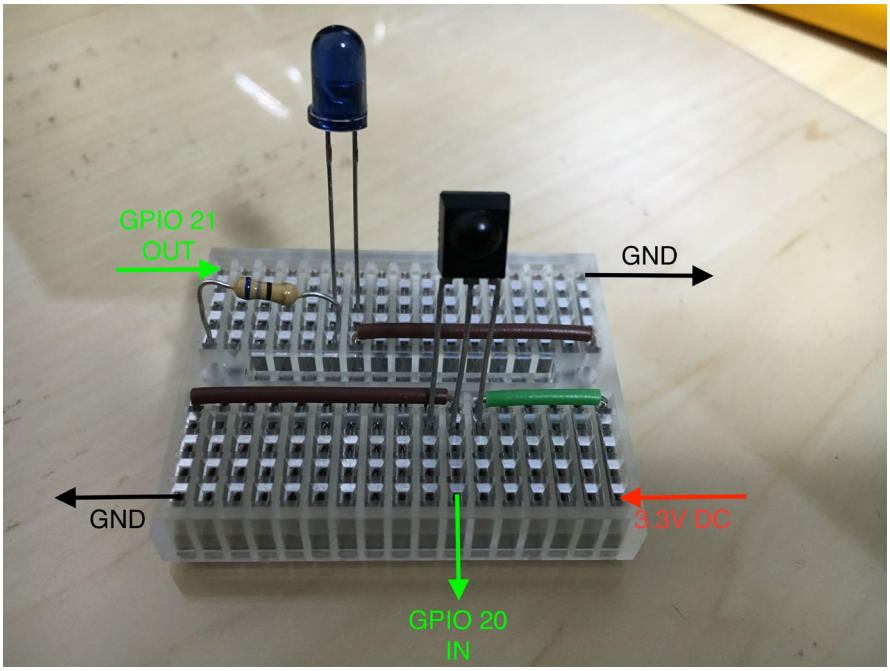

図4.3 配線

写真上は赤外線 LED、下は赤外線受光器です。あとは矢印の通りにジャンパワイヤを配線します。

/etc/lirc/hardware.conf を以下のように編集します :

```
 1: # /etc/lirc/hardware.conf
 2: #
 3: # Arguments which will be used when launching lircd
 4: LIRCD_ARGS="--uinput"
 5:
 6: #Don't start lircmd even if there seems to be a good config file
 7: #START_LIRCMD=false
 8:
 9: #Don't start irexec, even if a good config file seems to exist.
10: #START_IREXEC=false
11:
12: #Try to load appropriate kernel modules
13: LOAD_MODULES=true
14:
15: # Run "lircd --driver=help" for a list of supported drivers.
16: DRIVER="default"
17: # usually /dev/lirc0 is the correct setting for systems using udev
18: DEVICE="/dev/lirc0"
19: MODULES="lirc_rpi"
20:
21: # Default configuration files for your hardware if any
22: LIRCD_CONF=""
23: LIRCMD_CONF=""
```

次に、lirc モジュールを追加します。(**やや古い資料では /etc/modules に追加する方法が見つかりますが、仕様変更により以下の方法でないと失敗するようになったので、気をつけてください！**)

/boot/config.txt に以下の 1 行を追加します :

```
dtoverlay=lirc-rpi, gpio_in_pin=20, gpio_out_pin=21
```

再起動してモジュールを有効にします :

```
sudo reboot
```

以下のコマンドを実行して、lirc_rpi, lirc_dev, rc_core が表示されたら成功です。

```
lsmod | grep lirc
```

以下のコマンドを入力したあと、受光器にリモコンを向けボタンを押し、リモコンコードを受信できているか確認します :

```
sudo /etc/init.d/lirc stop
sudo mode2 -d /dev/lirc0
```

LED 照明のリモコンの記憶

次のコマンドを実行して、出てきた英文に従ってセットアップを済ませます :

```
sudo irrecord -n -d /dev/lirc0 room-led
```

ちょっとわかりにくい感じの文でしたがまあ頑張って登録を済ませました（適当）。
ちなみに、こんな感じの room-led.conf が生成されます。

```
 1: # Please make this file available to others
 2: # by sending it to <lirc@bartelmus.de>
 3: #
 4: # this config file was automatically generated
 5: # using lirc-0.9.0-pre1(default) on Wed Dec 23 19:15:29 2015
 6: #
 7: # contributed by
 8: #
 9: # brand:                       room-led.conf
10: # model no. of remote control:
11: # devices being controlled by this remote:
12: #
13:
14: begin remote
15:
16:   name  room-led.conf
17:   bits           16
18:   flags SPACE_ENC|CONST_LENGTH
19:   eps            30
20:   aeps          100
21:
22:   header       9087  4503
23:   one           594  1661
24:   zero          594   546
25:   ptrail        592
26:   repeat       9087  2247
27:   pre_data_bits   16
28:   pre_data       0x41B6
29:   gap          108695
30:   toggle_bit_mask 0x0
31:
32:       begin codes
33:           on                      0x659A
34:           off                     0x7D82
35:       end codes
36:
37: end remote
```

こいつをデフォルトの設定ファイルとされている /etc/lirc/lircd.conf に追記します。

```
sudo sh -c "cat room-led.conf > /etc/lirc/lircd.conf"
```

LED 照明のコントロール

```
irsend SEND_ONCE room-led.conf off
irsend SEND_ONCE room-led.conf on
```

これで電気が消えたりついたりしました！あとは NFC タグと組み合わせるだけです !!

Connection refused されたときは

```
1: irsend: could not connect to socket
```

```
2: irsend: Connection refused
```

とか言われたら、lircd が動いていないということなので、sudo systemctl restart lircd.service してあげましょう！

僕はこれで長時間つまづきまくっていて、危うく原稿出せなくなるところでした…

4.3 NFC タグ認識によるフラグの管理

前述の構想から、ドアが閉まるたび、つまり NFC タグがリーダーに認識されるたびにフラグ (should_on) の値を更新すれば良さそうです。

nfc.clf モジュールの connect は、terminate オプションで関数を指定された時、「タグが認識されるか、指定された関数が True を返すとき、終了する」という振る舞いをします。（公式ドキュメント参照）

タグが離れた時に終了する、といった便利なメソッドは残念ながら用意されていなかったので、各秒について、前秒 (was_door_closed) は開いていて、現在の秒 (is_door_closed) は閉まっている状態、つまりたった今閉じられた状態に、フラグ (should_on) を更新する、といった挙動をさせました。

```
 1: import nfc
 2: import time
 3:
 4: clf = nfc.ContactlessFrontend('usb')
 5:
 6: was_door_closed = False
 7: is_door_closed = False
 8: should_on = False
 9:
10: while True:
11:     was_door_closed = is_door_closed
12:     is_door_closed = False
13:
14:     after1s = lambda : time.time() - started > 1
15:     started = time.time()
16:     clf.connect(rdwr={'on-connect': connected}, terminate=after1s)
17:     if is_door_closed == True and was_door_closed == False:
18:         if should_on == True:
19:             should_on = False
20:             # Turn on the switch.
21:         else:
22:             should_on = True
23:             # Turn off the switch.
```

4.4 完成品

前述のフラグ (should_on) を関数に渡して、関数内で on/off させれば良さそうです。on/off は、Python 側でシェルのコマンドを実行させるのが最も楽だと思いましたので、そうしました。

```
 1: import nfc
 2: import subprocess
 3: import time
 4: import sys
 5:
 6: clf = nfc.ContactlessFrontend('usb')
 7:
 8: was_door_closed = False
 9: is_door_closed = False
10: should_on = False
11:
12: def change_led_value(flag):
13:     if flag == True:
14:         subprocess.call('irsend SEND_ONCE room-led.conf on', shell=True)
15:         print 'turned on'
16:     else:
17:         subprocess.call('irsend SEND_ONCE room-led.conf off', shell=True)
18:         print 'turned off'
19:
20: def connected(tag):
21:     global is_door_closed
22:     is_door_closed = True
23:
24: while True:
25:     was_door_closed = is_door_closed
26:     is_door_closed = False
27:
28:     after1s = lambda : time.time() - started > 1
29:     started = time.time()
30:     clf.connect(rdwr={'on-connect': connected}, terminate=after1s)
31:     if is_door_closed == True and was_door_closed == False:
32:         if should_on == True:
33:             should_on = False
34:             change_led_value(should_on)
35:         else:
36:             should_on = True
37:             change_led_value(should_on)
```

完成です !!

ちなみに動かしたらこうなりました。 https://vine.co/v/iAi5pzO2mOd

```
nohup python nfced-room.py &
```

でバックグラウンド実行するなりして運用したいと思います。

4.5 感想

部屋を出る時に必ず電気が消えるため、電気の消し忘れもなくなり、なかなか便利になるのではないかと思います。(現在原稿提出の締め切り直前なので、まだ運用歴ないです…)

NFC タグの技術的な感想としては、セットアップは面倒でしたが、実装は易しかったので、今後は色々手を広げたいなぁという感じです。

せっかくだし、NPCA の部誌も NFC 関連でやっちゃおうかなあと思っています(1 つアイデアがある)。5/2-3 の文化祭にてオンラインで公開されるので、その際はぜひ御覧ください !! (宣伝)

5 終わりに

読んでいただきありがとうございました！

何か質問などありましたら、冒頭に書きました連絡先にご連絡ください。また、僕もプロではないので、投げるマサカリがありましたらご気軽にお投げください。

5.1 参考文献

- nfcpy公式ドキュメント (http://nfcpy.org/latest/index.html) : インストール、読み書きなど
- http://plaza.rakuten.co.jp/kugutsushi/diary/201402230000/ : `/etc/udev/rules.d/nfcdev.rules` の記法
- http://make.bcde.jp/category/27/ : lircまわりいろいろ
- LIRC manpages (http://www.lirc.org/html/programs.html) : lircまわりいろいろ
- http://wbbwbb.blog83.fc2.com/blog-entry-189.html : lircモジュールの最新の追加方法
- その他いろいろ