6ri

Lecteur de fichiers simple

Nous allons voir deux méthodes:

- en MIDI avec un pad MPX8 et PureData
- en Python avec des boutons physiques de récupération

En MIDI, avec un MPX8 et PureData

L'idée de ce projet est de pouvoir déclencher la lecture de fichiers son (wav) avec un MPX8 de chez AKAI (ou tout autre appareil MIDI pouvant envoyer des notes MIDI) vers un Raspberry Pi 3B, pour ensuite les diffuser sur un système son composé d'une radio à transistors... À noter que l'envoie d'un fichier son doit faire s'arrêter la lecture de celui qui est en cours s'il y en a un.

Schéma de principe

Nous aurions donc:

MPX8 → envoie de note MIDI via USB → **Raspberry Pi 3B** avec PatchboxOS → **PureData** → Patch

Patch PureData

La patch va être développé sur une autre machine pour la bonne raison que je n'ai pas d'écran HDMI à disposition. Le principe: On utilise des fichiers wav. Les objets suivant sont utilisés : Pour charger les fichiers:

- [loadbang] → permet de recharger les morceau au lancement du patch
- [tabplay~] → lit un tableau donné
- [soundfiler] → transforme un fichier en données via le tableau
- tableau → permet de visualiser un tableau

Pour la partie MIDI:

- [notein] → reçoit les notes MIDI et dispense: Note, Vélocité et Channel dans les outlets
- [sel] → exhausteur de goût, écoute ce qui entre et si une valeur correspond à l'un des arguments, envoie un bang sur le outlet correspondant à la place de l'argument dans la liste. Ex: La note MIDI 40 est reçue, celle-ci est en 1ère position dans la liste d'arguments, un bang sera envoyé sur le outlet 1 en partant de la gauche.

La note MIDI 41 est reçue, celle-ci est en 2ème position dans la liste d'arguments, un bang sera envoyé sur le outlet 2, etc.

- [dac~]→ dirige les flux audios vers la sortie son.

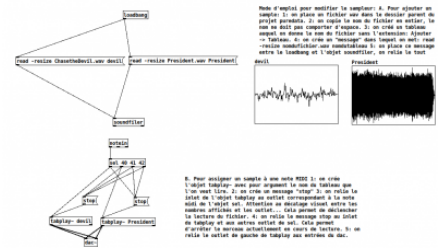
—- Mode d'emploi pour modifier le sampleur: —-

A. Pour ajouter un sample:

1. : on place un fichier wav dans le dossier parent du projet PureData.
2. : on copie le nom du fichier en entier, le nom ne doit pas comporter d'espace.
3. : on créé un tableau auquel on donne le nom du fichier sans l'extension: Ajouter → Tableau.
4. : on crée un “message” dans lequel on met: read -resize nomdufichier.wav nomdutableau
5. : on place ce message entre le loadbang et l'objet soundfiler, on relie le tout

B. Pour assigner un sample à une note MIDI

1. : on crée l'objet tabplay~ avec pour argument le nom du tableau que l'on veut lire.
2. : on relie le inlet de l'objet tabplay au outlet correspondant à la note midi de l'objet sel. Attention au décalage visuel entre les nombres affichés et les outlet... Cela permet de déclencher la lecture du fichier.
3. : on crée un message “stop”
4. : on relie le message stop au inlet du tabplay et aux autres outlet de sel. Cela permet d'arrêter le morceau actuellement en cours de lecture.
5. : on relie le outlet de gauche de tabplay aux entrées du dac.



En Python avec des boutons physiques de récupération

Le script python 3.11 qui fonctionne est le suivant (à adapter à vos GPIO, noms de fichiers à déclencher, etc...):

```
#Always comment your code like a violent psychopath will be maintaining it and they know where you live  ;)

from pygame import mixer                                        #importe la classe Mixer depuis le module "pygame" pour gérer les
fichiers son
from gpiozero import Button                                     #importe que l'élément "Button"depuis la bibliothèque GPIOzero

ChasetheDevil = Button(13)                                      #désigne la broche GPIO 13, l'active et lui donne un nom
President = Button(21)                                          #désigne la broche GPIO 21, l'active et lui donne un nom
mixer.pre_init(44100,-16,2, 2048)                               #configure les différents paramètres de alsa pour lire le fichier son. En
deça de 2048, il y a des                                                                    décrochages et la lecture
des fichiers n'est pas fluide.
mixer.init()                                                    #créé une instance de la classe "mixer" de la bibliothèque "Pygame"

def chase():                                                    #déclare une fonction pour gérer l'arrêt du morceau en cours, son volume,
son chargement e>
    mixer.music.stop()                                          #stoppe le fichier actuellement en lecture
    mixer.music.set_volume(1.0)                                 #règle le volume
    mixer.music.load('/home/waxmonde/samples/ChasetheDevil.wav') #charge le fichier son en indiquant le chemin entier pour pouvoir
démarrer le script depui>
    mixer.music.play()                                          #trouve les bons numéros du loto de hier

def pres():                                                     #déclare une fonction pour gérer l'arrêt du morceau en cours, son volume,
son chargement e>
    mixer.music.stop()                                          #stoppe le fichier actuellement en lecture
    mixer.music.set_volume(1.0)                                 #règle le volume
    mixer.music.load('/home/waxmonde/samples/President.wav')    #charge le fichier son en indiquant le chemin entier pour pouvoir
démarrer le script depui>
    mixer.music.play()                                          #répare ta gazinière


while True:                                                     #initialise une boucle "while" (conditionnelle)
    ChasetheDevil.when_pressed = chase                          #lorsque le boutton auquel est assigné ChasetheDevil est pressé, cela
lance la fonction "c>
    President.when_pressed = pres                               #lorsque le boutton auquel est assigné President est pressé, cela lance
la fonction "pres"

pause()                                                         #mets le script en pause, il attend donc une nouvelle instruction en
permanence
```

Prochaine étape: faire en sorte que ce script se lance automatiquement au démarrage du Raspberry Pi!

Lancement automatique du script python au démarrage du Raspberry Pi

On va utiliser la méthode “systemd” (si j'ose dire...) qui gère le lancement de “services” au démarrage du Raspberry Pi (en gros...).

Tout est décrit ici: https://www.youtube.com/watch?v=kQjnesqBMzQ et tout se passera en ligne de commande.

- On crée un nouveau fichier pour le nouveau service:

```
sudo nano /etc/systemd/system/waxmonde.service
```

→ vous pouvez donner le nom du service que vous voulez bien entendu...

- Dans nano, on édite le fichier avec ce qui suit:

```
 [Unit]
Description=Lecteur de samples Waxmonde v1.0
After=alsa-restore.service
[Service]
ExecStart=/usr/bin/python3 /home/waxmonde/samples/gpio_music_box08.py
WorkingDirectory=/home/waxmonde/samples
User=waxmonde

[Install]
WantedBy=multi-user.target
```

Vous pouvez retrouver les explications sur les différentes sections du code sur les internets.

Il y a par exemple la documentation officielle ici: https://systemd.io/

Et la documentation pour systemd sur debian ici (PatchboxOs étant basée sur Debian) : https://wiki.debian.org/systemd Vous pouvez sauvegarder votre fichier, sachant qu'il est possible qu'il y ait des réglages à faire ultérieurement.

- À la suite de quoi, on donne les droits qu'il faut à ce fichier:

```
sudo chmod 644 /etc/systemd/system/waxmonde.service
```

Vous pouvez vous référer au man de chmod pour plus de détails sur cette commande.

- On démarre le service:

```
sudo systemctl start waxmonde.service
```

systemctl est la commande qui permet de gérer les lancements, arrêts, etc. des différents services à chaud.

- On rend le servie actif au démarrage:

```
sudo systemctl enable waxmonde.service
```

- On vérifie le statu du service, à savoir s'il a été lancé, si cela a fonctionné et si il est actuellement en route ou à l'arrêt:

```
sudo systemctl status waxmonde.service
```

Dans notre cas, on avait ceci en réponse:

```
× waxmonde.service - Lecteur de samples Waxmonde v1.0
     Loaded: loaded (/etc/systemd/system/waxmonde.service; enabled; preset: enabled)
     Active: failed (Result: exit-code) since Sun 2025-05-25 13:18:30 CEST; 8h ago
   Duration: 19.682s
    Process: 585 ExecStart=/usr/bin/python3 /home/waxmonde/samples/gpio_music_box08.py (code=exited, status=1/FAILURE)
   Main PID: 585 (code=exited, status=1/FAILURE)
        CPU: 1.999s

May 25 13:18:29 patchbox python3[585]: pygame 2.1.2 (SDL 2.26.5, Python 3.11.2)
May 25 13:18:29 patchbox python3[585]: Hello from the pygame community. https://www.pygame.org/contribute.html
May 25 13:18:29 patchbox python3[585]: Traceback (most recent call last):
May 25 13:18:29 patchbox python3[585]:   File "/home/waxmonde/samples/gpio_music_box08.py", line 9, in <module>
May 25 13:18:29 patchbox python3[585]:     mixer.init()                                                        #créé une instance de la
classe "mixer" de la b>
May 25 13:18:29 patchbox python3[585]:     ^^^^^^^^^^^
May 25 13:18:29 patchbox python3[585]: pygame.error: ALSA: Couldn't open audio device: Unknown error 524
May 25 13:18:30 patchbox systemd[1]: waxmonde.service: Main process exited, code=exited, status=1/FAILURE
May 25 13:18:30 patchbox systemd[1]: waxmonde.service: Failed with result 'exit-code'.
May 25 13:18:30 patchbox systemd[1]: waxmonde.service: Consumed 1.999s CPU time.
lines 1-18/18 (END)
```

Ce qui signifie que le service a été lancé, que le script s'est lancé mais qu'il a rencontré une erreur.

L'erreur ici est que alsa ne sait pas sur quel carte son se lancer, alors que le script lorsqu'il est lancé à la main, en ssh, s’exécute sans soucis.

La question a été posée sur le forum de la distribution PatchboxOs, utilisée ici sur le raspberry pi:

https://community.blokas.io/t/how-to-start-a-python-script-at-the-boot-of-patchboxos/5852

Et par ce biais des explications ont été trouvées ici:

https://forums.raspberrypi.com/viewtopic.php?t=360496

Il faut donc créer un fichier de conf dans /etc/, le tout en sudo donc:

Dans un premier temps on va chercher la bonne dénomination de carte son à utiliser par alsa. La commande est la suivante (pas besoin d'être sudo): aplay -l

```
**** List of PLAYBACK Hardware Devices ****
card 0: vc4hdmi [vc4-hdmi], device 0: MAI PCM i2s-hifi-0 [MAI PCM i2s-hifi-0]
Subdevices: 1/1
Subdevice #0: subdevice #0
card 1: Headphones [bcm2835 Headphones], device 0: bcm2835 Headphones [bcm2835 Headphones]
```

Dans le fichier de conf que l'on va créer, il faudra donc spécifier la carte son ainsi que la connectique (device) qui nous intéressent avec “hw:1,0”. Ici, le premier chiffre est pour la carte utilisée, le second, après la virgule, est pour le 'device' utilisé. Il pourrait y avoir plusieurs devices par carte dans le cas d'une carte son externe avec entrées et sorties multiples.

```
sudo touch /etc/asound.conf
```

← la commande sudo permet d'avoir les droits d'écriture, la commande touch créé le fichier dont le nom est précisé après avec son emplacement, ici /etc/asound.conf. C'est à cet endroit que alsa viendra chercher la configuration qui va bien pour lui permettre de diffuser la lecture de fichiers son.

```
sudo nano /etc/asound.conf
```

← la commande nano ouvre un éditeur de texte basique mais puissant en ligne de commande et permet de mettre ce qui suit dans le fichier asound.conf:

```
pcm.!default {
        type asym
        playback.pcm {
        type plug
        slave.pcm "hw:1,0"
        }
        }
```

On sauvegarde le fichier, dans nano il faut faire : ctrl+O, puis entrée, puis ctrl+X On peut rebooter le raspberry pi (sudo reboot)

On teste sans se connecter en ssh (au cas où et puisque c'est comme cela que cela fonctionnera en production) en appuyant sur un des boutons pour déclencher la lecture d'un fichier son.

On est content.e et on va fêter ça dans un endroit sympa comme son jardin puisqu'on a du linge à étendre!!!

Article extrait de : https://lesporteslogiques.net/wiki/ - **WIKI Les Portes Logiques**
Adresse :
https://lesporteslogiques.net/wiki/openatelier/projet/sampler_puredata_pour_raspberry_pi_sans_ecran
Article mis à jour: **2025/05/26 11:45**