Acústica 2020
XI Congreso Ibérico de Acústica
51º Congreso Español de Acústica
TECNIACÚSTICA 2020

TUTORIAL: CONFIGURAÇÃO DE DISPOSITIVOS DE ÁUDIO NO RASPBERRY PI – PARTE 2

Bárbara Circe[1], **William D'Andrea Fonseca**[1], **Leonardo Jacomussi**[1], **Paulo H. Mareze**[1]

[1] Engenharia Acústica, Universidade Federal de Santa Maria,
Av. Roraima nº 1000, Cidade Universitária, Santa Maria (RS), Brasil, 97105-900
{barbara.circe, will.fonseca, leonardo.jacomussi, paulo.mareze}@eac.ufsm.br

Resumo

O Raspberry Pi (Rpi) é um microcomputador completo integrado em uma única placa (*single board computer*, SBC). Ele pode ser utilizado tanto de maneira didática quanto na automação profissional de tarefas. Entre suas grandes vantagens estão o custo-benefício, a facilidade de uso e sua integração com diversos outros dispositivos. Atualmente existem diversos modelos de Rpi, como por exemplo, o Raspberry Pi 3 B+ e o Raspberry Pi 4 B, sendo que este último foi lançado em meados 2019. Assim, existem inúmeros modos de configuração e operação. Este trabalho busca auxiliar a comunidade científica, trazendo instruções e dicas. Nesta segunda etapa do tutorial, explora-se formas mais avançadas de configurações de dispositivos de entrada de sinais, além de apontar as diferenças funcionais e limitações entre os modelos 3 B+ e 4. As novidades no sistema operacional (OS) e na arquitetura do microcomputador, no que tange às configurações para dispositivos de áudio, são de acordo abordadas. Ademais, são elaborados de forma didática os passos necessários para a configuração de dispositivos utilizando mediadores como o PulseAudio e a obtenção da biblioteca PortAudio para o uso em Python.

Palavras-chave: Raspberry Pi, PortAudio, PulseAudio, processamento de sinais, tutorial, Python.

Abstract

Raspberry Pi (Rpi) is a complete microcomputer integrated onto a single board (SBC). As such, it can be used both didactically and in professional automation tasks. Its main advantages are its cost-benefit, ease of use, and integration with several other devices. There are currently a number of different Rpi models, for example, the Raspberry Pi 3 B + and Raspberry Pi 4 B, the latter launched in mid-2019. Thus, the versatile Raspberry Pi offers numerous modes of configuration and operation. This paper seeks to assist the scientific community by providing usage instructions and tips. In this second part of the tutorial, more advanced ways of configuring signal input devices are explored, in addition to pointing out the functional differences and limitations between models 3 B + and 4. What's new in the operating system (OS) as well as microcomputer architecture regarding settings for audio devices are accordingly addressed. Finally, this article didactically describes the necessary steps for configuring devices using mediators such as PulseAudio and obtaining the PortAudio library for use in Python.

Keywords: Raspberry Pi, PortAudio, PulseAudio, signal processing, tutorial, Python.

PACS no. 43.38.Md, 43.10.Pr, 43.38.Md, 43.58.Ta, 43.60.-c, 43.60.Qv.

1 Introdução

A segunda[1] parte do tutorial aborda configurações que permitem maior controle de entradas e saídas de diferentes dispositivos, fazendo uso do *servidor de áudio* PulseAudio. Serão explorados métodos que permitem a comunicação entre PulseAudio e a ALSA, de forma que um possa controlar o outro ou estabelecer uma relação específica que pode ser automatizada fazendo uso de um *script*. Ademais, serão enumerados os passos para a obtenção do PortAudio, bem como comentários sobre seu funcionamento em Python. Por fim, são apresentadas algumas informações comparativas entre os dois últimos modelos Raspberry Pi lançados até o momento, o 3 B+ e Pi 4 B [2, 3].

2 Fundamentos

Esta seção define e contextualiza brevemente o servidor PulseAudio e a biblioteca PortAudio, ferramentas importantes para diversos tipos de configuração de áudio em Raspberry Pi. É interessante o leitor ter o conhecimento prévio apresentado na Parte 1 [1].

2.1 PulseAudio

O PulseAudio[2] é um servidor de som multi-plataformas que costuma vir junto com gerenciadores de ambiente de *desktop* (como o Gnome, por exemplo) [4]. É considerado um *middleware*, pois tem a capacidade de agir mediando *hardware* e aplicativo [5]. Pode rodar em diversos modelos de sistemas operacionais (e em até alguns mais antigos, como MacOS X e Windows XP). O PulseAudio é uma ferramenta valiosa por conseguir lidar com múltiplos dispositivos simultaneamente, controlando suas entradas e saídas, além de poder operar com dispositivos de áudio *bluetooth* e diversas outras aplicações [6].

2.2 PortAudio

O PortAudio[3] é uma biblioteca gratuita para gravação, reprodução e processamento de áudio. Ele é escrito em linguagem C e tem capacidade de ser executado em vários sistemas operacionais e plataformas, possibilitando um programador escrever e compilar programas simples de áudio em "C" e/ou "C++" [7]. Alguns dos programas que utilizam essa biblioteca são o Python (para os módulos **`PyAudio`** e **`sounddevice`**, por exemplo), Audacity e o ITA-Toolbox (que roda dentro do Matlab).

3 Metodologia

Os passos para a configuração e exposição do PulseAudio à ALSA serão abordados nesta seção. Serão explorados comandos importantes do servidor e exemplos de sua configuração, possibilitando o controle de entradas e saídas de diversos dispositivos e até automação com uso de *scripts*. Esta seção também aborda a biblioteca PortAudio, apresentando o passo a passo e os comandos necessários para sua instalação através do terminal.

3.1 Configurando o PulseAudio

O Código 1 apresenta, repectivamente, como checar se o PulseAudio está instalado e qual sua versão, seguido dos comandos para atualizar e instalar caso necessário. Atualmente a última versão estável do PulseAudio é a 13.0, lançada em 13 setembro de 2019, entretanto, a versão 9 do Raspbian Strech necessita de construção manual para versões posteriores a 10.0.

Código 1 – Comandos para checar, instalar ou atualizar o PulseAudio.

```
$ pulseaudio --version
pulseaudio 10.0
$ sudo apt-get upgrade pulseaudio
$ sudo apt-get install pulseaudio
```

[1] Este artigo continua do artigo prévio *Tutorial: configuração de dispositivos de áudio no Raspberry Pi – Parte 1* [1].
[2] Informações oficiais no site: https://www.freedesktop.org/wiki/Software/PulseAudio/.
[3] Mais informações em http://www.portaudio.com/.

Há dois principais comandos do PulseAudio que nos permitem configurá-lo com o próposito de alterar dispositivos de áudio, perfil dos *sound cards* e seus canais de entrada e saída. São eles:

i. `pacmd`: reconfigura o servidor de áudio durante seu tempo de execução, muitas vezes sendo necessária a reinicialização do servidor.

ii. `pactl`: controla o servidor de áudio ativo (sessão em execução). É uma parte do `pacmd` que altera configurações de áudio, sem a necessidade de reiniciar o PulseAudio.

Para conhecer um pouco mais sobre a infinidade de opções de ambos os comandos citados acima, pode-se utilizar os comandos: `pacmd -help` e `pactl -help`. Para ler os manuais, use os seguintes códigos: `man pulseaudio`, `man pacmd` e `man pactl`.

Código 2 – Comandos para exibir informações e para listar dispostivos, respectivamente.

```
$ pactl info
$ pacmd list-cards
```

A Figura 1 (a) exemplifica o resultado da primeira linha do Código 2 (`pactl info`), exibindo na tela que as configurações estão sendo seguidas e que dispositivo é o escolhido como padrão para o PulseAudio. Neste caso temos o mesmo nome do *card* destacado na Figura 1 (b) e algumas outras informações, como o formato `ps16le` (sinal amostrado com 16-bits e padrão *little-endian*[4]), dois canais de saída (`2ch`) e frequência de amostragem de 44.100 Hz. A segunda linha do Código 2 apresenta em forma de lista um dos dispositivos de áudio (*cards*) em detalhes, emque índices, nome, perfis, saídas e entradas são as principais informações. Os termos *sink* e *source* são para *output* e *input*, respectivamente, e serão bastante utilizados. O comando `pacmd list-cards` imprime na tela extensas informações, como já mencionado anteriormente, fazendo com que a Figura 1 (b) mostre apenas parte do que é exibido. De acordo com o número do índice (`index: 1`), podemos concluir que o PulseAudio lê esse dispositivo como o segundo *card*, diferentemente da ALSA. Em "*properties*" podemos ver listados o número do *card* para a ALSA e o nome do dispositivo, que é outra diferença entre os dois. Como o HiFiBerry não possui entrada, suas características como *profile*, *sink* e *sources* são preenchidas com informações apenas sobre a saída (analógica e estéreo).

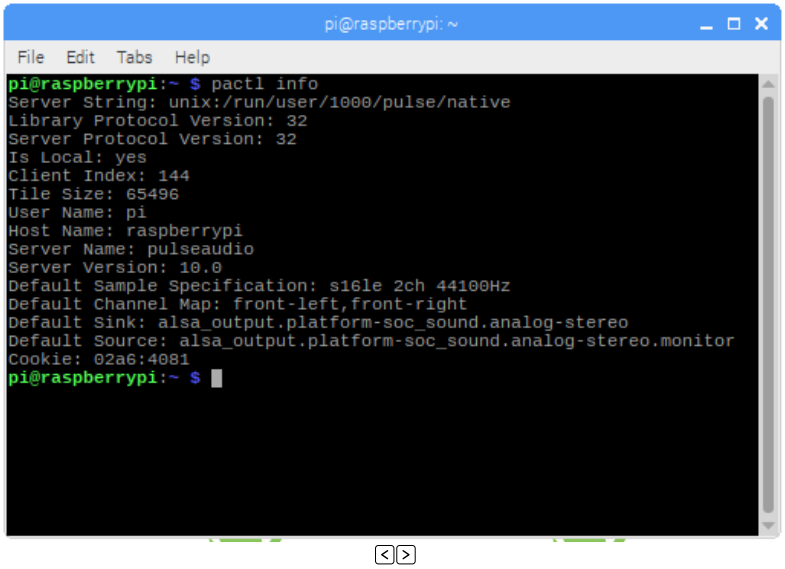

Figura 1 – (a) Comando `pactl info`: informações sobre parâmetros padrão para a sessão atual do PulseAudio. (b) Parte do resultado do comando `pacmd list-cards` , apresentando informações sobre o segundo *card*. (*Utilize as setas para alternar entre figuras, com o Acrobrat Reader, por exemplo.*)

[4] *Little-endian* é uma maneira ordenada de armazenar *bytes* na memória do computador. Neste caso (*little)* começando a armazenar pelo *byte* menos significativo.

Ainda em *sources*, o "*monitor*" no final do nome representa um *device* (assim como um *hardware device* e um *virtual device* [8]). Apesar de estar em *sources*, o monitor está associado às *sinks*, e atua copiando, duplicando e salvando todo sinal que vai para os alto-falantes [8]. O *card* de índice 0 também foi mostrado na tela e apesar de ser o dispositivo que desejamos configurar (placa de áudio USB), utilizamos outro (HiFiBerry) apenas com intuito de ilustrar a estrutura geral das informações obtidas a partir do comando.

Agora que conhecemos um pouco a forma que nosso PulseAudio está configurado, podemos modificar, comparar e testar. As próximas linhas mostram como utilizar os comandos para alterar em diferentes níveis as entradas e saídas padrão do PulseAudio. A sintaxe de dois comandos importantes é descrita no Código 3. O comando `pacmd` permite a configuração do servidor, enquanto o `pactl` controla e modifica a sessão atual do PulseAudio. Na prática isso quer dizer que, para que o `pacmd` faça efeito, deve-se reiniciar o servidor, o que leva à finalização de uma sessão e início de outra (os comandos `pulseaudio --kill` e `pulseaudio --start` fazem isso, veja mais detalhes na Nota 1).

Código 3 – Sintaxe para alteração de *inputs* e *outputs* para `pacmd` e `pactl`.

```
$ # Sintaxe
$ pacmd set-default-(sink|source) NAME|#N
$ pactl [options] set-default-(sink|source) NAME
```

Alguns exemplos de uso dos dois comandos podem ser vistos no Código 4. O `pacmd` possui dois modos de declaração, sendo estes utilizando o número ou o nome e `pactl` apenas o nome. Utilizando o `pactl` as modificações serão feitas no mesmo momento, porém ao reiniciar o servidor, podem ser perdidas. Recomenda-se portanto, a configuração de ambos os comandos para evitar perdas.

Código 4 – Exemplos de como modificar *sinks* e *sources* com `pacmd` e `pactl`.

```
$ # Exemplos
$ pacmd set-default-sink 0
$ pacmd set-default-source alsa_input.usb-0d8c_USB_Sound_Device-00.analog-stereo
$ pactl set-default-sink alsa_output.usb-0d8c_USB_Sound_Device-00.analog-stereo
$ pactl set-default-source alsa_input.usb-0d8c_USB_Sound_Device-00.analog-stereo
```

Os nomes de *sinks* e *sources* utilizados acima no exemplo podem ser encontrados com o comando `pacmd list-cards`, já mencionado anteriormente. A Figura 2 exemplifica esses nomes e destaca também outra característica: "*ports*", que representa uma de cada vez as portas físicas de saída ou entrada do dispositivo [8]. Há ainda "*profile*", que reúne combinações de entrada e saída que podem ser bastante específicas, como habilitar um *surround* 7.1 ou deixar a entrada digital e a saída analógica, por exemplo.

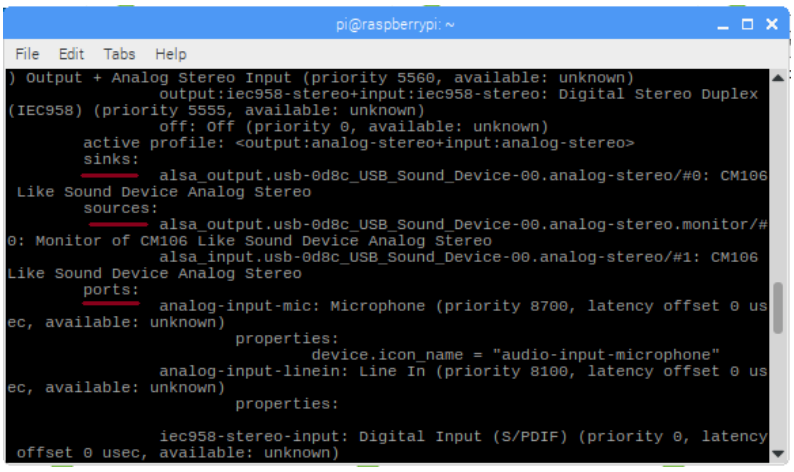

Figura 2 – Características do *card* destacadas em vermelho após o uso do comando `pacmd list-cards`.

Nota 1 – Configuração de `pacmd` e/ou `pactl`.

Para configurarmos com `pacmd` ou `pactl` devemos seguir os seguintes passos:

1) verificar entradas saídas e outros parâmetros definidos como padrão pelo PulseAudio com o comando `pactl info`;
2) conhecer o nome, índice e outras características dos *cards* existentes com `pacmd list-cards`;
3) definir *sinks* e *sources* tanto com o `pacmd` quanto para `pactl` usando os Códigos 3 ou 4;
4) reiniciar o servidor com `pulseaudio --kill` e `pulseaudio --start` (opcional); e
5) testar em diferentes aplicativos, como o Youtube no Chromium, ou no Python (com PortAudio) por exemplo.

3.2 Expondo o PulseAudio à ALSA

Enquanto a ALSA, no primeiro momento, age escolhendo diretamente qual dispositivo de áudio (chamado pelo programa de *card*) será utilizado e controlado, o PulseAudio escolhe as saídas e entradas, que podem ser inclusive de diferentes dispositivos. Para que isso possa acontecer, o servidor deve se sobrepor à ALSA, podendo controlá-la. O Código 5 mostra em sua primeira e segunda linha como acessar o ponto de interação entre PulseAudio e ALSA.

Código 5 – Códigos para checar a relação "pulse-alsa".

```
$ cd /usr/share/alsa/
$ sudo nano pulse-alsa.conf
$ cd /usr/share/alsa/alsa.conf.d/
$ sudo nano pulse.conf
```

Abrindo o arquivo `pulse-alsa.conf` (Figura 3) vemos uma estrutura muito semelhante à estrutura do arquivo `asound.config`, mas que neste caso, define o PulseAudio como padrão, mesmo quando as aplicações utilizam a ALSA. Para ter este arquivo, deve-se ter o pacote `pulseaudio-alsa` que faz esse trabalho. Veremos mais à frente como personalizar a interação "pulse-alsa" a partir do `asound.config`.

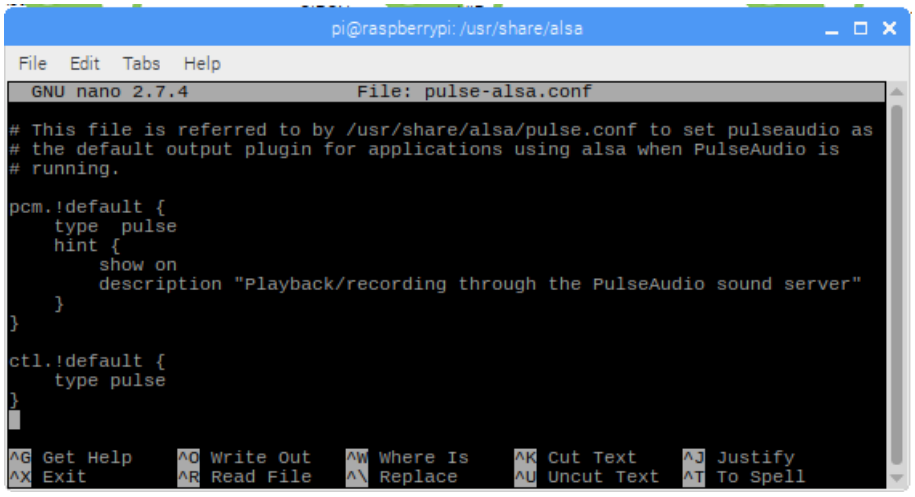

Figura 3 – Configuração do arquivo "`pulse-alsa.conf`": definindo o PulseAudio como padrão para os parâmetros `pcm` e `ctl`.

O arquivo `pulse.conf` tem a capacidade de fazer com que o PulseAudio sirva como *plugin* para a ALSA. Na Figura 4 vemos que esse arquivo chama outro de nome `pulse-alsa.conf` (já mencionado acima), e então define os dispositivos e portas padrão do PulseAudio como sendo também o padrão da ALSA. Para manter a autonomia da ALSA, esses dois arquivos devem ser modificados, porém não recomenda-se desacoplar o servidor da arquitetura, pois se por um lado o PulseAudio tem maior capacidade de gerir os dispositivos, por outro a ALSA trabalha diretamente no núcleo do Linux, e juntos fornecem uma ferramenta mais poderosa de configuração de áudio.

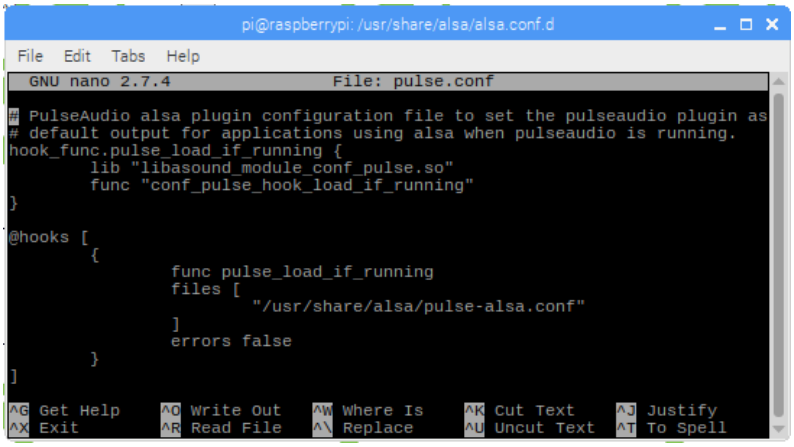

Figura 4 – Configuração da definição do PulseAudio como *plugin* para sobrepor a ALSA.

Há vários *front-ends* para PulseAudio, sendo alguns deles pacmixer, pamixer, pavucontrol, paprefs etc. Na referência [8] é possivel encontrar informação sobre alguns deles, além de *prints* detalhados com explicações (em se tratando do PulseAudio, essa é uma das melhores referências encontradas). A referência [5] também apresenta informações muito boas sobre o PulseAudio (inclusive o modo de configuração a seguir). Vimos acima que é possível tornar as escolhas de saídas e entradas do PulseAudio padrão também para a ALSA a partir dos arquivos `pulse.conf` e `pulse-alsa.conf`. Segundo o *site* da ArchWiki, embora possamos definir o padrão ALSA como sendo o padrão PulseAudio (o que já fizemos), temos uma maneira mais versátil de configurar essa interação [5].

Para personalizar a interação "pulse-alsa", podemos criar estilos e definições editando `asound.config`. O Código 6 relembra como chegar ao arquivo, enquanto o Código 7 exemplifica essas personalizações. Podemos criar configurações exclusivas para os parâmetros `pcm` e `ctl`, utilizando "`type pulse`". O efeito disso é utilizar comandos do PulseAudio, como *device*, *sink* e *source* dentro das configurações da ALSA. Essa personalização deve ser usada caso não existam os arquivos mostrados nas Figuras 3 e 4 (pode-se conseguir o arquivo instalando o pacote `pulseaudio-alsa`). Veja o passo a passo na Nota 2.

Para automatizar a escolha do dispositivo de áudio e sua combinação de *inputs* (*source*) e *outputs* (*sink*), podemos fazer uso de um *script*. Isso não será detalhado aqui, mas pode-se resumir que o caminho é criar um arquivo como "`meu-script.sh`", contendo comandos que definem entradas, saídas etc. e então adicionar ao terminal (*bash*) e executar o *script*. Diversos exemplos podem ser encontrados *online* e a referência [9] pode dar alguma ideia de como funciona.

Código 6 – Comandos para acessar o arquivo `asound.config`.

```
1 $ cd \etc\
2 $ sudo nano asound.config #para abrir arquivo
```

Código 7 – Perfis de exposição do PulseAudio à ALSA, usando o *plugin* "pulse" (adaptado de [5]).

```

# Cria um input ou output ALSA usando sources ou sinks específicos do PulseAudio

 pcm.pulse-exemplo1 {
     type pulse
     device alsa_card.usb-0d8c_USB_Sound_Device-00
     fallback "pulse-exemplo2" #caso não esteja disponível
    }

 pcm.pulse-exemplo2 {
     type pulse
     device alsa_card.platform-soc_sound

 }

# Controlando um source ou uma sink separadamente com um mixer:
 ctl.pulse-example {
     type pulse
     sink "HAT"
     source "placa-de-audio-USB"

     #exemplo: saida do HAT e entrada da placa
     sink alsa_output.platform-soc_sound.analog-stereo
     source alsa_input.usb-0d8c_USB_Sound_Device-00.analog-stereo
 }

# Para se sobrepor ao mixer padrão
 ctl.!default {
     type pulse
     sink alsa_output.platform-soc_sound.analog-stereo
     source alsa_input.usb-0d8c_USB_Sound_Device-00.analog-stereo
 }
```

Nota 2 – Expondo o PulseAudio aos comandos da ALSA.

Para expor o PulseAudio aos comandos da ALSA, devemos:

1) verificar se os arquivos `pulse-alsa.conf` e `pulse.conf` existem e se estão configurados de maneira semelhante às Figuras 3 e 4;

2) caso não existam, verificar com o comando `pactl info` entradas saídas e outros parâmetros definidos como padrão pelo PulseAudio;

3) então usar essas informações para configurar o arquivo `asound.config` conforme Código 7; e

4) ainda, pode-se criar um *script* para automatizar suas escolhas dentro do PulseAudio.

3.3 Obtendo a biblioteca PortAudio

A versão mais recente da biblioteca (até o dado momento) pode ser obtida executando a sequência de comandos contidos no Código 8. No entanto, outras versões podem ser encontradas no site oficial da biblioteca.

Código 8 – Comando para instalar atualizar o PortAudio.

```
$ sudo apt-get install portaudio19-dev #instalar
$ sudo apt-get upgrade portaudio19-dev #atualizar
```

Para instalar outras versões do PortAudio, recomenda-se seguir as etapas da Nota 3.

Nota 3 – Instalando versões diferentes do PortAudio.

1) Faça o download do arquivo no site: http://portaudio.com/download.html;
2) extraia os arquivos do *download*, abra a pasta e encontre o arquivo "`README.configure.txt`", nele terá todas as instruções de como instalar; e
3) instale conforme Código 9 e, logo após, reinicie.

Código 9 – Ordem a ser executada na linha de comando para instalação do PortAudio via *download* manual pelo site.

```
$ sudo apt-get install autoconf
$ sudo apt-get install build-essential
$ sudo apt-get install pkg-config
$ sudo apt-get install libtool libasound-dev
$ cd /home/pi/Downloads/portaudio #exemplo de diretório
$ autoreconf -if
$ sudo make install
```

4 PortAudio em Python

De forma indireta, o PortAudio vem sendo bastante utilizado em Python para aquisição e reprodução de áudio. Os módulos **`SoundDevice`** [10] e **`PyAudio`** [11] são bons exemplos de ligações entre a biblioteca desenvolvida em C e o Python. Ambos módulos apresentam métodos com princípios de "*chamada de retorno*" para aplicações que fazem reprodução ou aquisição de áudio com diferentes quantidades de amostras (conhecido como *streaming*), possuindo também métodos com bloqueios, em que a sequência do *script* fica bloqueada até que o processo de geração ou gravação de áudio termine.

Em aplicativos de *streaming* com processamento de áudio em *tempo real*, deve-se tomar o cuidado de não sobrecarregar o sistema. Como mencionado, os módulos que acessam invólucros do PortAudio em Python utilizando chamadas de retorno, conhecidas como funções de *callback* [10], em que as amostras são recolhidas ou lançadas ao PortAudio de tempos em tempos, com acesso livre às variáveis da função (para serem exibidas em interfaces gráficas, por exemplo). A forma mais prática e rápida de capturar essas amostras para realizar o processamento é por meio da inserção dos métodos de cálculo (ou então fazer chamadas de funções externas que realizam o processamento) dentro da função de *callback*, as duas ações geram bloqueio da chamada de retorno até que a ação do processamento do sinal de áudio seja realizada. Neste contexto, deve-se atentar ao tempo de processamento do sinal, uma vez que, se solicitada, a função de *callback* deve de fato retornar uma resposta ao PortAudio, seja para receber ou para enviar amostras. Quando o tempo de processamento é maior do que o intervalo de tempo da chamada de retorno, há um bloqueio do fluxo de dados, ocorrendo em perdas de amostras. Em sistemas de desempenho limitado (como o apresentado por sistemas embarcados), os recursos de processamento tendem a ser menores, se comparados a capacidade e velocidade de processamento encontradas em computadores convencionais, agravando ainda mais a situação.

Por natureza, o interpretador CPython possui um mecanismo de bloqueio que assegura que apenas um *bytecode* Python seja executado em uma única *thread*, o GIL[5][12]. No entanto, exitem dois módulos nativos do Python que permitem o desbloqueio do compilador, possibilitando a execução do *script* em múltiplas *threads* e/ou múltiplos núcleos do processador, o *threading* [13] e o *multiprocessing* [14]. *Threads* são como miniprogramas sendo executados dentro de um programa maior, ou seja, possui uma

[5] *Global Interpreter Lock*.

lógica sequencial de execução de tarefas, contendo início, meio e fim [15]. Eles permitem fracionar tarefas que podem ser executadas por pequenos programas dentro de um *software* completo. Processos, no entanto, são programas executados em paralelo [16], em outras palavras, um *script* python sendo executado em um outro núcleo do computador de forma paralela e compartilhando dados. A diferença entre ambas as ferramentas consiste da real capacidade de processamento em paralelo: o *threading* divide grandes tarefas em pequenas tarefas, que são executadas sequencialmente, porém, sem bloquear o processo principal do *software* (salvar um arquivo enquanto mantém uma interface gráfica responsiva, por exemplo), enquanto que o *multiprocessing* de fato realiza processos (cálculos e ações específicas) em um núcleo diferente do processador de forma paralela.

Tanto *threading* quanto *multiprocessing* são ferramentas importantes para aplicações de execução e processamento de tarefas em tempo real. Na prática, podem auxiliar na distribuição de tarefas dentro de um contexto complexo de processamento enquanto realiza aquisição e/ou reprodução de amostras de áudio simultaneamente, evitando a perda de amostras durante a *chamada de retorno* do PortAudio. Nas referências [17, 18] há aplicações de *streaming* de áudio para implementação de um *medidor de nível de pressão sonora* (sonômetro) em Python utilizando diferentes sistemas embarcados. Além disso, o módulo PyTTa [19, 20], um projeto desenvolvido por alunos e professores do curso de graduação em Engenharia Acústica (UFSM), possui inúmeras ferramentas implementadas em Python para medições e processamento de sinais em acústica, incluindo *streaming* de áudio com soluções *multi-threads*.

5 Raspberry Pi 4 B

Até o dado momento, não há diferenças entre os sistemas operacionais dos modelos[6] do Raspberry Pi [21], desta forma, os modos de configurações dos dispositivos de áudio apresentados são os mesmos em todos os modelos. Atentando-se apenas para aqueles que não possuem microchip DAC[7] *onboard*, nesse caso, as configurações restringem-se a saída de áudio via porta HMDI e dispositivos externos conectados via porta USB (ou *bluetooth*). No entanto, como mostra o teste de *benchmark*[8] apresentado na Figura 5, há grandes diferenças entre os modelos 3 B+ e 4 B no concerne ao desempenho dos dispositivos. Para consulta, em [17] há uma interessante tabela comparativa entre modelos de Rpi e outros sistemas embarcados.

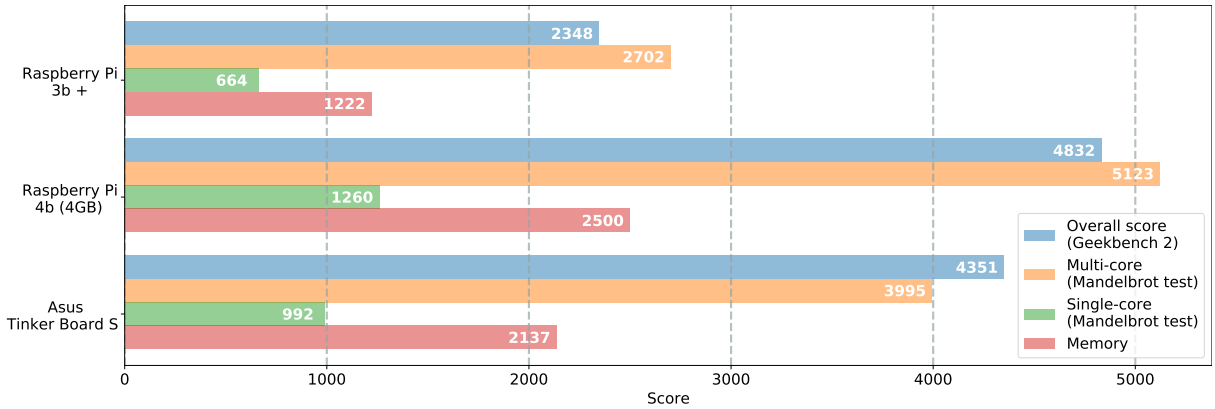

Figura 5 – Teste de desempenho desenvolvido pela Primate Labs [22] realizado com os modelos Raspberry Pi 3 B+ [23], Raspberry Pi 4 B (4 GB) [24] e Asus Tinker Board S [25] (este último é forte concorrente no mercado de sistemas embarcados).

[6] Na ocasião da construção deste tutorial o modelo mais avançado era o Rpi 3 B+.
[7] *Digital-to-Analog Converter* ou conversor digital para analógico.
[8] Em computação, *benchmark* é o ato de executar um ou mais programa(s) em um computador com o intuito de avaliar o desempenho do dispositivo.

Para projetos que envolvam aquisição de curta duração e processamento de áudio por etapas (primeiro grava e depois processa), não há grandes implicações quanto ao desempenho dos modelos, exceto o tempo que o minicomputador levará para realizar os cálculos. Deve-se considerar também, o tamanho do sinal a ser gravado e/ou processado em relação ao espaço disponível na memória RAM e no disco de armazenamento do dispositivo. Aplicações de alta exigência de *hardware* como geração de malhas e cálculos complexos, como aqueles realizados em simulações BEM[9] e FEM[10], podem sobrecarregar o sistema rapidamente. Portando, deve-se dimensionar corretamente o problema para as ferramentas computacionais disponíveis e, sempre que possível, otimizar a utilização de cada recurso de *hardware*, iniciando pela implementação do *software* desenvolvido para o projeto.

6 Considerações finais

O PulseAudio mostrou-se um servidor de áudio adequado, uma vez que possui inúmeras configurações que podem fazê-lo comunicar-se com a arquitetura ALSA, além de ser capaz de gerenciar diversos dispositivos de áudio no Raspberry Pi.

O desempenho do PortAudio como invólucro em módulos Python é bastante satisfatório, apresentando diversos recursos para aquisição e reprodução de áudio com baixa latência. Considerações devem ser feitas quanto ao processamento de áudio em *tempo real* utilizando *chamadas de retorno*, o tempo do processamento deve ser inferior ao tempo de chamada da função de *callback*.

Novos modelos de sistemas embarcados vêm apresentando melhorias significativas em relação ao desempenho geral dos minicomputadores, como os recursos computacionais oferecidos pelo novo Raspberry Pi 4 B (8 GB, versão de 2020). Ainda assim, recursos de processamento em *multi-threads* e *multi-core* podem auxiliar na velocidade de processamento e melhorar aproveitamento dos recursos de *hardware* disponíveis.

Ao final, espera-se que com a leitura desses tutoriais (Partes 1 e 2) o leitor/usuário possa economizar tempo e logo medir suas próprias *respostas impulsivas da sala* e/ou colocar uma boa música com o Rpi.

Agradecimentos

Os autores gostariam de agradecer ao Curso de Engenharia Acústica da Universidade Federal de Santa Maria (UFSM) por todo apoio institucional e infraestrutura. Ademais, agradecemos também ao Joseph Lacey pela revisão de inglês do *abstract* e à Laís Smeha pela revisão textual.

Referências

[1] Circe, Bárbara; Fonseca, William D'A.; Jacomussi, Leonardo e Mareze, Paulo H. Tutorial: configuração de dispositivos de áudio no Raspberry Pi – Parte 1. In *Acústica 2020 (XI Congreso Ibérico de Acústica, Tecniacústica 2020 e 51º Congreso Español de Acústica)*, páginas 1–12, Faro, Portugal, Outubro 2020.

[2] Raspberry Pi 3 Model B. https://www.raspberrypi.org/products/raspberry-pi-3-model-b. Acesso em: ago. de 2020.

[3] Raspberry Pi 4 Model B. https://www.raspberrypi.org/products/raspberry-pi-4-model-b. Acesso em: set. de 2020.

[4] PulseAudio - Debian Wiki. https://wiki.debian.org/PulseAudio. Acesso em: set. de 2020.

[5] PulseAudio - ArchWiki. https://wiki.archlinux.org/index.php/PulseAudio. Acesso em: set. de 2020.

[6] Circe, Bárbara; Fonseca, William D'A.; Jacomussi, Leonardo e Mareze, Paulo H. Sistema computacional livre para síntese de voz a partir de texto. In *XXVIII Congresso Brasileiro de Fonoaudiologia & V Congresso Ibero Americano de Fonoaudiologia - Fono 2020*, São Paulo, SP, Brasil, 2020. Disponível em https://github.com/leonardojacomussi/speech-synthesizer.

[9] *Boundary Element Method* ou Método de elementos de contorno.
[10] *Finite Element Method* ou Método de elementos finitos.

[7] Burk, Phil. PortAudio - Open-Source Cross-Platform Audio API. http://www.portaudio.com. Acesso em: set. de 2020.

[8] Gaydov, Victor. PulseAudio under the hood. https://gavv.github.io/articles/pulseaudio-under-the-hood/#command-line-tools. Acesso em: set. de 2020.

[9] PulseAudio/Examples - ArchWiki. https://wiki.archlinux.org/index.php/PulseAudio/Examples#The_shell_script_method. Acesso em: set. de 2020.

[10] *Play and Record Sound with Python – python-sounddevice, version 0.3.12*. https://python-sounddevice.readthedocs.io/en/0.3.12/. Acesso em: set. de 2020.

[11] Pham, Hubert. *PyAudio: PortAudio v19 Python Bindings*. https://people.csail.mit.edu/hubert/pyaudio. Acesso em: set. de 2020.

[12] Raspberry Pi Foundation. *Glossary – Python 3.4.10 documentation*. https://docs.python.org/3.4/glossary.html. Acesso em: set. de 2020.

[13] Raspberry Pi Foundation. *threading – Thread-based parallelism — Python 3.8.5 documentation*. https://docs.python.org/3/library/threading.html. Acesso em: set. de 2020.

[14] Raspberry Pi Foundation. *multiprocessing – Process-based parallelism – Python 3.8.5 documentation*. https://docs.python.org/3/library/multiprocessing.html. Acesso em: set. de 2020.

[15] Big Java. What is a thread? (2016). Disponível em http://journals.ecs.soton.ac.uk/java/tutorial/java/threads/definition.html. Acesso em: set. de 2020.

[16] What is the difference between a system and a process? https://transition-support.com/m.faq27.html. Acesso em: set. de 2020.

[17] Fonseca, William D'A.; Jacomussi, Leonardo e Mareze, Paulo H. Raspberry Pi: A Low-cost Embedded System for Sound Pressure Level Measurement. In *49th Internoise - International Congress and Exposition on Noise Control Engineering*, páginas 1–12, Seul, Córeia do Sul, 2020. Disponível em http://bit.ly/SLM-Internoise2020.

[18] Jacomussi, Leonardo. Sound-Level-Meter: Sound pressure level meter and reverberation time for embedded systems (GitHub). Disponível em https://github.com/leonardojacomussi/Sound-Level-Meter. Acesso em: set. de 2020.

[19] PyTTa: Python in Technical Acoustics and Vibration – GitHub: PyTTAmaster. https://github.com/PyTTAmaster/PyTTa. Acesso em: set. de 2020.

[20] Fonseca, William D'A.; Paes, João Vitor; Lazarin, Matheus; Reis, Marcos Vinicius; Mareze, Paulo Henrique e Brandão, Eric. Pytta: Open source toolbox for acoustic measurement andsignal processing. In *23 International Congress on Acoustics - ICA 2019*, páginas 1–8, Aachen, Alemanha, 2019. doi: 10.18154/RWTH-CONV-239980 .

[21] Raspberry Pi Foundation. *Downloads - Software for the Raspberry Pi*. https://www.raspberrypi.org/downloads. Acesso em: set. de 2020.

[22] Primate Labs. *Geekbench - Cross-Platform Benchmark*. https://www.primatelabs.com. Acesso em: set. de 2020.

[23] Primate Labs – Raspberry Pi 3 B+. *Geekbenck Browser (Set. 2020)*. https://browser.geekbench.com/geekbench2/2689781. Acesso em: set. de 2020.

[24] Primate Labs – Raspberry Pi 4B (4 GB). *Geekbench Browser (Set. 2020)*. https://browser.geekbench.com/geekbench2/2689780. Acesso em: set. de 2020.

[25] Primate Labs – Asus Tinker Board S. *Geekbench Browser (Set. 2020)*. https://browser.geekbench.com/geekbench2/2689784. Acesso em: set. de 2020.