Aplicativo de Streaming de Áudio em Rede Local

Gustavo F. dos Santos[1], **Eduardo F. Model**[1]

[1]Centro de Desenvolvimento Tecnológico – Universidade Federal de Pelotas (UFPEL)

{gfdsantos,efmodel}@inf.ufpel.edu.br

1. Apresentação da Aplicação

Como trabalho final da disciplina de Redes de Computadores, este grupo escolheu fazer um aplicativo de transmissão de áudio em rede local. A motivação por trás do tema se dá ao crescente uso de serviços de streaming de conteúdo digital, como Netflix, Spotify, Youtube, entre outros. O grupo achou instigante investigar o funcionamento de tais serviços e sua dificuldade de implementação.

Como ferramenta de implementação foi usado a linguagem de programação Python, na sua versão 3.6, juntamente com o módulo externo chamado PyAudio, que pode ser baixado através da ferramenta pip (exemplo: pip3 install pyaudio). Aqui, recomenda-se o uso do comando:

python3 -m pip install pyaudio –user

Onde é forçado o uso do pip pelo Python3, ou seja, o pacote instalado será disponibilizado para a distribuição do Python3 instalado. Isto ocorre pois módulos são implementados para diferentes versões do Python, mais comummente o Python 2.7.x e o Python 3.x.

A aplicação desenvolvida consta de duas implementações diferentes: o cliente e o servidor. O servidor procura uma porta disponível no computador que servirá como servidor de transmissão e, então, inicia a execução do servidor, onde o mesmo fica aguardando a conexão de clientes. Um cliente ao iniciar uma conexão com o servidor, abre uma nova thread no servidor, ou seja, um servidor de streaming pode ser conectado a vários clientes distintos simultaneamente.

O aplicativo do cliente funciona via uma interface CLI – Command Line Interface, isto é, via linha de comando. Quando um cliente inicia uma transmissão de uma música, essa transmissão fica executando de fundo e o prompt de comandos do cliente é liberado para que o cliente faça outras coisas enquanto a música requisitada é transmitida e tocada.

A solução implementada consta com funções básicas de um player de músicas: tocar, pausar e trocar de música. A forma de utilização será discutida a seguir.

A transmissão da música se dá em sequências de bytes ao longo de uma conexão TCP. O servidor envia blocos de 1024 bytes ao cliente e o cliente escreve estes bytes em sequência no buffer de reprodução do PyAudio, então a música é reproduzida ao usuário.

2. Forma de Utilização

A utulização do aplicativo foi desenvolvida para ser simples. O primeiro passo é iniciar o servidor, o que pode ser feito executando o comando:

Tabela 1. Descrição dos comandos disponíveis

Comando	Descrição
listar	Requisita ao servidor a relação de músicas disponível para serem transmitidas e forma uma lista de reprodução.
transmitir	Requisita ao servidor uma dada música e inicia a transmissão e a reprodução, se a música estiver disponível.
pausar	Pausa a reprodução da música (a transmissão de pacotes da determinada música é também pausada).
parar	Pára a reprodução da música e finaliza a thread de transmissão.
prox	Pára a reprodução da música e inicializa uma nova thread de transmissão com a próxima música da lista de reprodução.
ant	Pára,a reprodução da música e inicia uma nova thread de transmissão,com a música anterior.
aleat	Liga ou desliga o modo de reprodução aleatório.
sair	Finaliza a execução do programa.

python servidor.py

Nesse momento o servidor inicia tentando escutar a porta 9992, mas se esta porta estiver ocupada, o servidor consegue se recuperar e escolher outras portas até que consiga estar disponível para conexão com clientes. Todas as etapas são mostradas ao usuário, para que o mesmo possa acompanhar o funcionamento do servidor.

O segundo passo é executar os clientes, neste momento o servidor pode se conectar com diversos clientes diferentes. Para iniciar um cliente, é apenas preciso digitar o comando:

python cliente.py

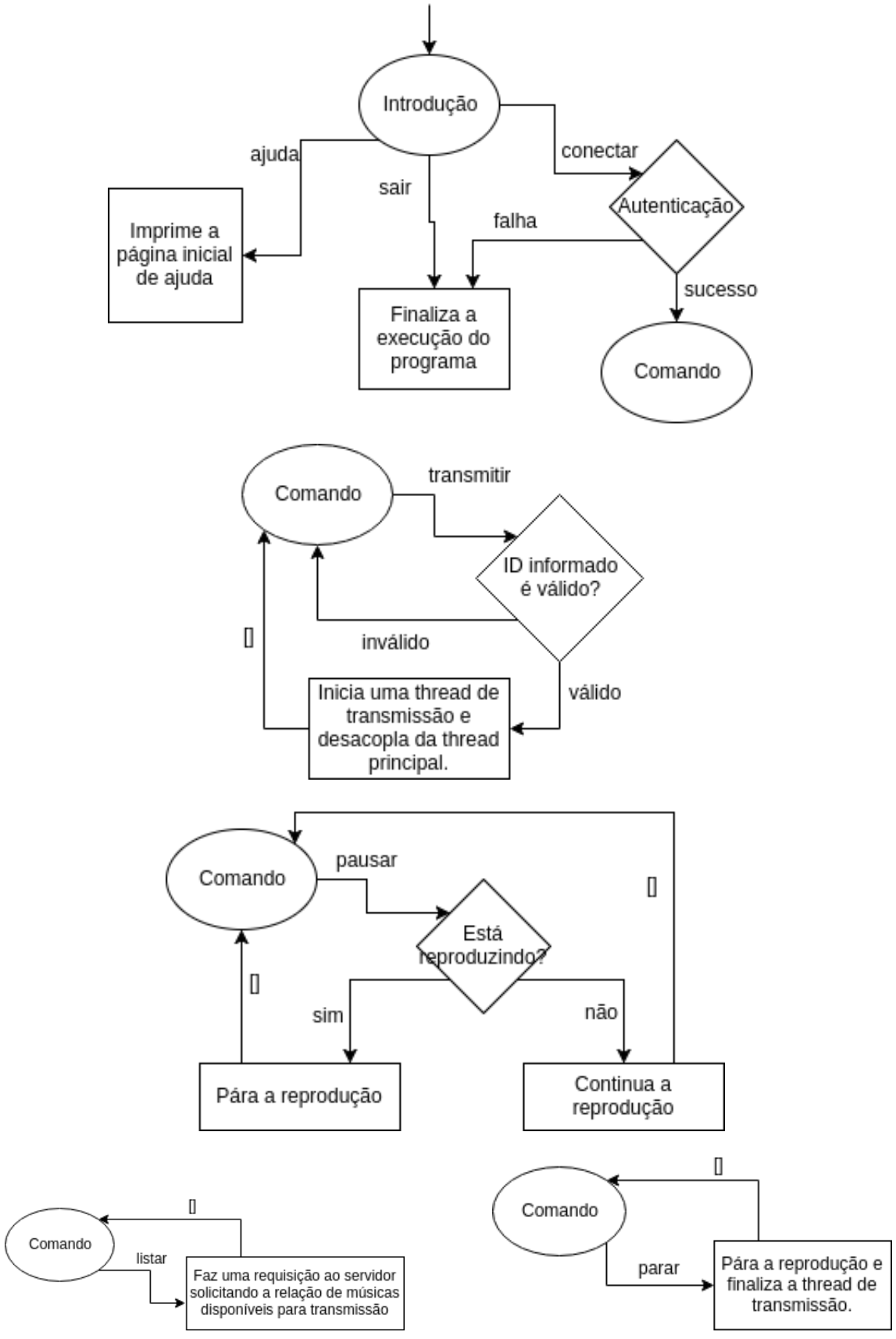
Ao executar o cliente, será emitido uma mensagem onde mostra ao usuário que, se desejado, ele pode digitar o comando 'ajuda', onde será mostrado os comandos disponíveis no primeiro momento da execução.

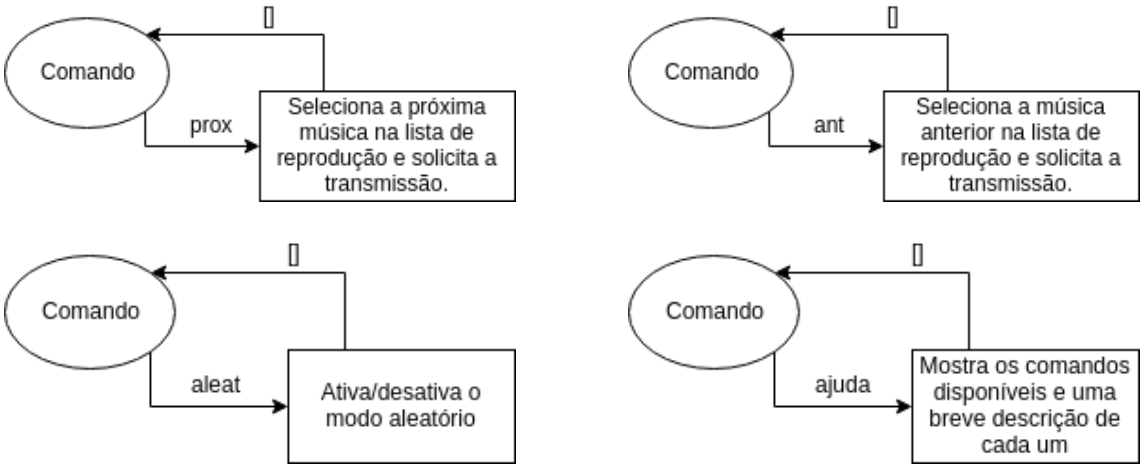
Em seguida é preciso configurar o cliente com o endereço IP do servidor, junto da porta em que o servidor está escutando as conexões com clientes. Note que o primeiro e o segundo passo podem ser executados de forma inversa, entretanto o usuário receberá uma mensagem de erro, pois o servidor ainda não está escutando clientes. Ainda, caso o usuário desejar dar aos scripts em Python a autorização para serem executados, através do comando chmod, por exemplo, em sistemas Linux, é permitido e o aplicativo deve funcionar normalmente.

Agora, com o cliente devidamente conectado ao servidor, podem ser executados diversos comandos: listar, transmitir, pausar, parar, prox, etc. A descrição de todos os comandos disponíveis são vistos na Tabela 1.

3. Protocolo

O protocolo pode ser visto nas máquinas de estado a seguir. A fim de melhor visualização do funcionamento do cliente e do servidor, as máquinas de estados foram fracionadas em partes, mantendo os estados anteriores no qual derivaram o estado atual.

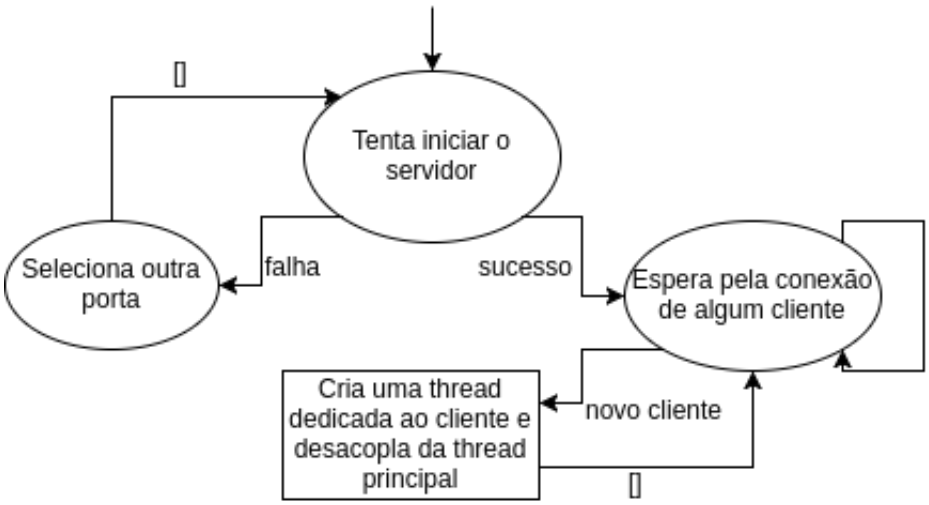

Introdução
ajuda
sair
conectar
Autenticação
falha
sucesso
Imprime a página inicial de ajuda
Finaliza a execução do programa
Comando
Comando
transmitir
ID informado é válido?
inválido
válido
Inicia uma thread de transmissão e desacopla da thread principal.
Comando
pausar
Está reproduzindo?
sim
não
Pára a reprodução
Continua a reprodução
Comando
listar
Faz uma requisição ao servidor solicitando a relação de músicas disponíveis para transmissão
Comando
parar
Pára a reprodução e finaliza a thread de transmissão.

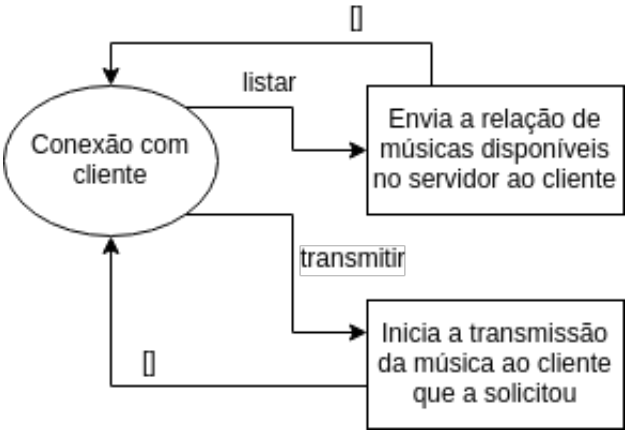


3.1. Cliente

Nas imagens das máquinas de estado, o simbolo [] representa uma transição vazia.

3.2. Servidor





4. Exemplos de Uso

Nesta sessão serão apresentados alguns exemplos de uso e uma breve explicação do funcionamento do cliente e servidor.

4.1. Transmissão com um cliente conectado

Este exemplo demonstra o funcionamento do cliente e servidor quando somente um cliente está conectado ao servidor.

Primeiramente, o servidor deve ser executado, ao fazê-lo, o prompt do servidor mostrará a seguinte mensagem:

```
$ python servidor.py

[!] Servidor iniciando em 127.0.0.1:9992...

[+] Aguardando conexões. Pressione Ctrl+C para parar.
```

O usuário comandante do servidor pode parar a execução do servidor com a combinação de teclas Ctrl e C. Ao executar o cliente, o usuário receberá a seguinte mensagem:

```
$ python cliente.py

[!] Entre com 'ajuda' para obter ajuda na execução.
>
```

Neste prompt o usuário poderá digitar comandos e interagir com o servidor. Ao entrar com **ajuda**, será mostrada a mensagem:

```
> ajuda

COMANDO     DESCRIÇÃO
-------------------------------------------------------------------------
ajuda       Imprime esta mensagem de ajuda.

conectar    Conecta a um servidor de transmissão usando o
            número de IP e o número de porta.

sair        Finaliza a execução do programa.
```

Como ilustrado, se o usuário tentar efetuar a conexão com o servidor, isto é, digitar o comando **conectar**, então restará apenas ao usuário entrar com as informações do servidor - os números de IP e porta:

```
> conectar

[!] Endereço IP do servidor:
> 127.0.0.1
```

```
[!] Número de porta do servidor:
> 9992

[!] Conectando com 127.0.0.1:9992

[+] Conectado com 127.0.0.1:9992
```

Da mesma forma, quando o servidor identifica a conexão com um cliente, este imprime a seguinte mensagem:

```
[!] Novo cliente ('127.0.0.1', 34708)
```

Onde mostra o número de IP do cliente conectado e o identificador para este cliente.

Se um novo cliente conectar-se com o servidor, então o servidor informaria a nova conexão. Cada conexão entre o servidor e um cliente é encapsulada em uma thread e esta thread se mantém em execução no servidor até que o cliente se desconecte, ou seja, que envie o comando **sair**. Neste exemplo, quando o segundo cliente se conecta, o servidor informa:

```
[!] Novo cliente ('127.0.0.1', 34720)
```

Onde este novo cliente é identificado pelo número e IP e o identificador **34720**.

Quando um cliente desejar listar as músicas disponíveis no servidor, utilizando o comando **listar**, o servidor irá identificar as músicas que estão presentes na pasta músicas, então irá encapsular em uma lista e enviar ao cliente. No lado do cliente a seguinte mensagem de exemplo será mostrada:

```
> listar

MÚSICAS DISPONÍVEIS
------------------------------------------------------------
ID      NOME
------------------------------------------------------------
0       Jack Johnson - Cupid
1       Gojira - Stranded
2       Guitarra Humana
3       Jack Johnson - Gone
4       Highly Suspect - Claudeland
------------------------------------------------------------
```

Neste exemplo, o servidor contém 5 músicas.

Agora, com o cache de músicas atualizado, é possível efetuar uma transmissão, para isso o cliente deve digitar o comando **transmitir** e em seguida, o ID da música, como no exemplo:

```
> transmitir
```

```
Entre com o ID da música (número):
> 3

[!] Conectando com 127.0.0.1:9992

[+] Conectado com 127.0.0.1:9992

[!] Solicitando a música Jack Johnson – Gone...

[!] Abrindo canal para transmissão...
>
```

Então a transmissão inicia em ambos os lados, o servidor enviando blocos de 1024 bytes e o cliente recebendo, anexando n último bloco recebido e reproduzindo. No lado do servidor, a seguinte mensagem é mostrada:

```
[!] Novo cliente ('127.0.0.1', 34728)

[!] $('127.0.0.1', 34728): transmitir:Jack Johnson – Gone

[!] Iniciando transmissão...

[!] Transmitindo a musica Jack Johnson – Gone.wav para ('127.0.0.1', 34
```

Quando o servidor termina de enviar os blocos da música, são mostradas as seguintes mensagens:

```
[+] Transmissão de Jack Johnson – Gone.wav finalizada.
```

As mensagens de *conectando* e *conectado* são mostradas porque, quando uma transmissão começa, o cliente cria uma thread especialmente para lidar com a transmissão e reprodução da música - chamada aqui de **thread de transmissão**, liberando o prompt de comando para que o usuário digite outros comandos, como ligar ou desligar o modo aleatório.

Durante uma transmissão para um cliente de qualquer música presente no servidor, outro cliente conectado pode solicitar transmissões e será atendido.

O usuário pode também pausar e parar a reprodução, onde o *pausar* pausa a thread de transmissão, pausando consequentemente a transmissão dos pacotes e a reprodução da música. O comando *parar* pára a execução da thread de transmissão e o Python se encarrega de finalizar a thread. Um exemplo de utilização, no mesmo cliente de exemplo, pode ser visto abaixo:

```
> pausar
> pausar
> parar
>
[+] Transmissão finalizada
```

```
> pausar

[-] Thread de transmissão não está executando.
> parar

[-] Thread de transmissão não está executando.
>
```

Pode ocorrer do cliente e servidor exibirem erros de transmissão quando usar o comando *parar*, mas estes erros são normais. Quando o comando *parar* é executado, uma exceção na thread de reprodução é atingida, que é a exceção de finalização da execução. Esta exceção é atingida também quando há um erro na transmissão, por este motivo esta mensagem é exibida e deve ser ignorada dependendo das vezes.

Quando o cliente desejar se desconectar do servidor, o cliente apenas precisa digitar o comando **sair**, então a execução do cliente é finalizada e o servidor finaliza a thread do cliente, pode ser acompanhado no exemplo abaixo no lado do servidor:

```
[!] $('127.0.0.1', 34824): sair

[!] Conexão fechada de ('127.0.0.1', 34824)
```