

# Relatório Projeto SO

Grupo 63:

António Luís de Macedo Fernandes (a93312)

José Diogo Martins Vieira (a93251)

João Silva Torres a93231

June 16, 2021



**Universidade do Minho**

# Contents

<b>1</b>	<b>Introdução</b>	<b>3</b>
<b>2</b>	<b>Arquitetura do Projeto</b>	<b>4</b>
2.1	Servidor . . . . .	4
2.2	Comandos . . . . .	5
2.2.1	Transform . . . . .	5
2.2.2	Status . . . . .	5
2.3	Cliente . . . . .	5
<b>3</b>	<b>Testes-exemplo</b>	<b>6</b>
3.1	Concorrência de pedidos . . . . .	6
3.2	Pedido Pendente . . . . .	7
<b>4</b>	<b>Conclusão</b>	<b>8</b>

# 1 Introdução

Este projeto consistiu na criação de um serviço capaz de transformar ficheiros de áudio por aplicação de uma sequência de filtros.

Para além de executar tarefas, o servior permite também a consulta das tarefas em execução, do número de filtros disponíveis e em uos.

Inicialmente, o maior desaio passou por fazer a concorrência de pedidos, com isto, veio o problema da comunicação entre o servidor e o respetivo cliente. Mais tarde, tivemos dificuldades em conseguir obter o status das tarefas em execução tal como os pedidos pendentes, mas conseguimos superar estes obstáculos.

## 2 Arquitetura do Projeto

### 2.1 Servidor

Quando o servidor é iniciado, verifica se o program "ffmpeg" está instalado. Caso esteja, o programa procede, caso contrário, indica ao utilizador que é necessário instalar o programa. Após isto, é feito o setup do ficheiro de configurações dado. Para tal é usado um array de Filtro para guardar a informação sobre estes.

A estrutura Filtro está definida da seguinte forma:

---

```
typedef struct filtros
{
    char *name;
    char *path;
    int running;
    int max;
}*Filtro;
```

---

De início são criados dois fifos unidireccionais, uma para a comunicação CLIENTETOSERVER, (leitura), outro para a comunicação SERVERTOCLIENTE (escrita).

Para permitir que a escrita servidor-cliente seja feita para o cliente certo, criamos um pipe com nome que será temporário (eliminado no fim da execução do comando) de sentido uniderecional (servidor-cliente), diferente para cada cliente. E é onde será transmitida toda a informação acerca da execução do comando.

Para a gestão de informação sobre as diferentes tasks, temos as seguintes variáveis globais:

---

```
//registo do nr de tasks
int taskNumber = 0;
// pid do processo associado a cada task
int task_pid[1024];
// o estado da task. Pode ser: A_EXECUTAR | PENDENTE | TERMINADO
int taskStatus[1024];
// o comando associado a cada task
char* taskCommand[1024];
// Array que contem os filtros
Filtro* filtrosArray;
// o nmeoro de filtros
int numberFiltros;
// registo do nr de fifos
int fifoNumber;
```

---

## **2.2 Comandos**

### **2.2.1 Transform**

Após fazer o parsing necessário para saber quais os filtros a usar, verifica se é possível fazer a sua execução sem exceder o limite máximo dos filtros. Caso não seja possível, fica pendente até ser possível a sua execução.

Para o processamento, são usados pipes anónimos, caso seja dado mais de um filtro. É colocado o resultado final no ficheiro de output dado pelo utilizador.

### **2.2.2 Status**

Para este comando é verificado quais as tasks que ainda estão em execução, escrevendo assim o comando associado a estas. É dada também a informação sobre a utilização dos filtros, tal como o pid do servidor.

## **2.3 Cliente**

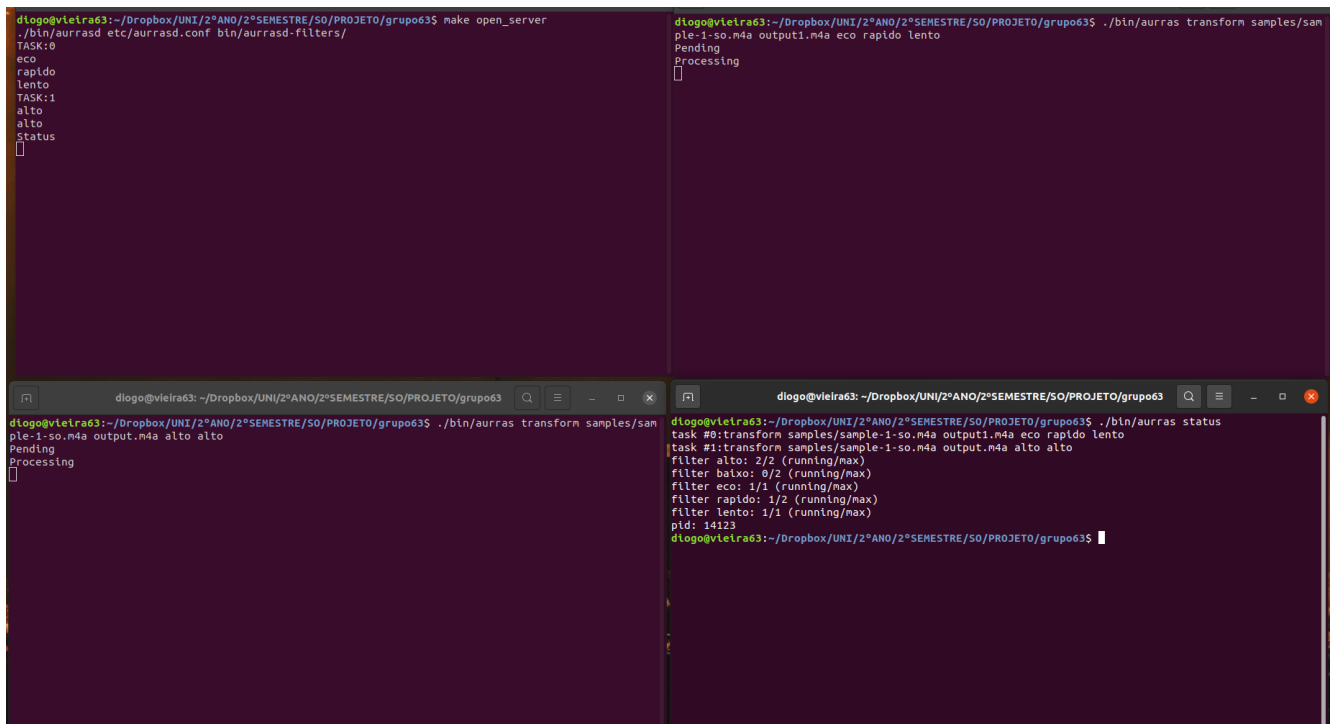
O Cliente abre os pipes CLIENTETOSERVER (escrita) e SERVERTOCLIENTE (leitura), por esta ordem. É escrito o comando do utilizador, e espera-se a leitura do nome do novo pipe para a comunicação com este cliente apenas. É aberto este pipe para leitura e espera-se por informação do servidor até este pipe ser fechado.

### 3 Testes-exemplo

Para estes testes, a ordem feita dos comandos encontra-se apresentada no terminal do servidor (topo-esquerda).

#### 3.1 Concorrência de pedidos

Para este teste pretende-se mostrar a concorrência de pedidos. O tempo que o pedido demora a executar é independente dos outros. Neste caso, o comando status é instantâneo apesar de ter sido o último a executar. A task 0 por conter 3 filtros, o mais provável, é que será a última a terminar, apesar de ter sido a primeira a efetuar o pedido.



```
diogo@vleira63: ~/Dropbox/UNI/2ºANO/2ºSEMESTRE/SO/PROJETO/grupo63$ make open_server
./bin/aurrasd etc/aurrasd.conf bin/aurrasd-filters/
TASK:0
eco
rapido
lento
TASK:1
alto
alto
Status
█

diogo@vleira63:~/Dropbox/UNI/2ºANO/2ºSEMESTRE/SO/PROJETO/grupo63$ ./bin/aurras transform samples/sample-1-so.m4a output1.m4a eco rapido lento
Pending
Processing
█

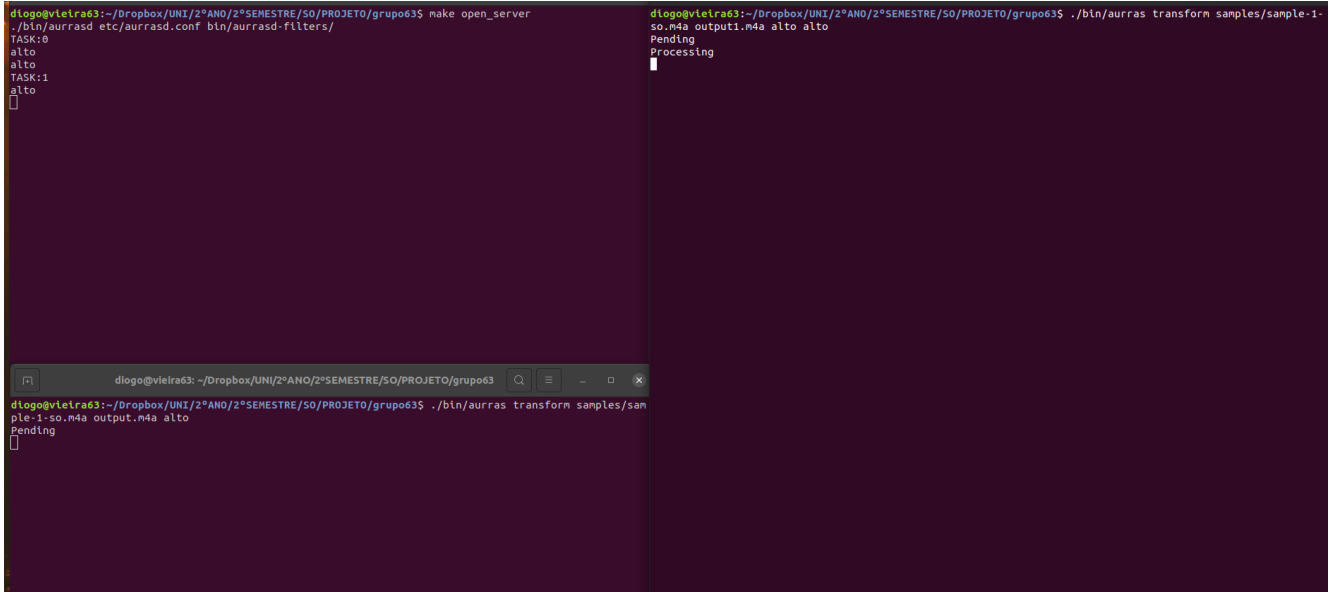
diogo@vleira63:~/Dropbox/UNI/2ºANO/2ºSEMESTRE/SO/PROJETO/grupo63$ ./bin/aurras transform samples/sample-1-so.m4a output1.m4a eco rapido lento
Pending
Processing
█

diogo@vleira63:~/Dropbox/UNI/2ºANO/2ºSEMESTRE/SO/PROJETO/grupo63$ ./bin/aurras status
task #0:transform samples/sample-1-so.m4a output1.m4a eco rapido lento
task #1:transform samples/sample-1-so.m4a output1.m4a alto alto
filter alto: 2/2 (running/max)
filter baixo: 0/2 (running/max)
filter eco: 1/1 (running/max)
filter rapido: 1/2 (running/max)
filter lento: 1/1 (running/max)
pid: 14123
diogo@vleira63:~/Dropbox/UNI/2ºANO/2ºSEMESTRE/SO/PROJETO/grupo63$
```

Concorrência de pedidos

## 3.2 Pedido Pendente

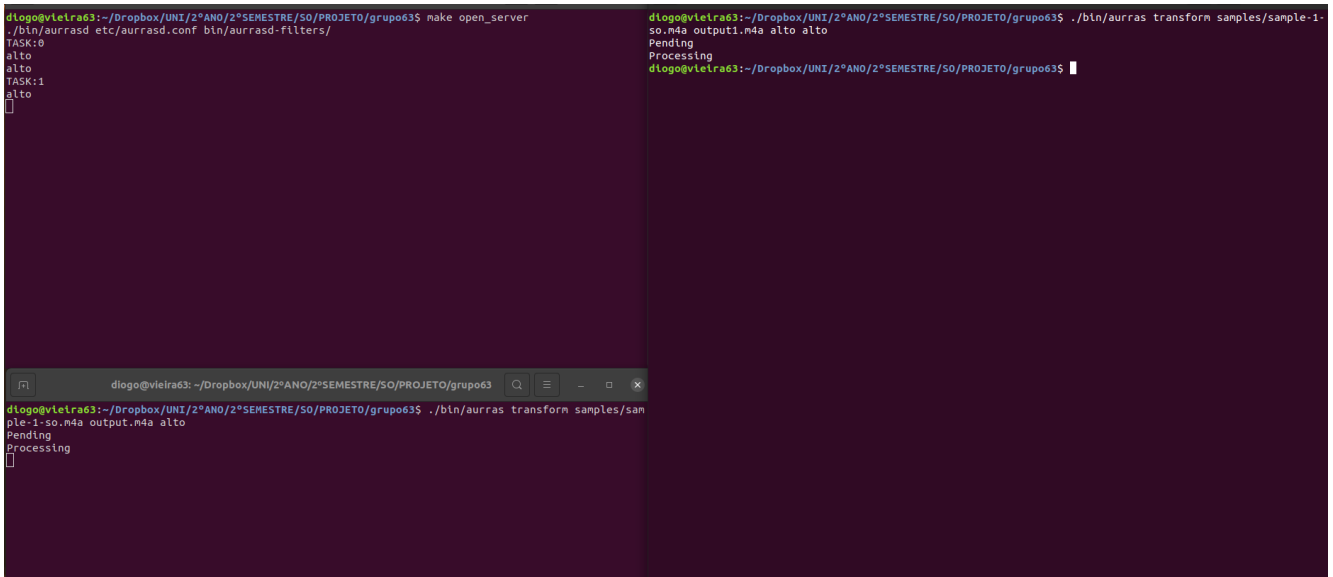
Para este teste pretende-se mostrar um pedido a ficar pendente, caso esteja à espera de recursos para o seu pedido. Neste caso, na task 0 estão 2 filtros "alto" a correr sendo 2 o máximo permitido. A task 1 quer usar um filtro alto, mas tem de esperar que a task 0 acabe.



The image shows a terminal window with two panes. The left pane shows the output of the 'make open\_server' command, which starts two tasks. Task 0 is shown as 'TASK:0' and Task 1 as 'TASK:1'. Both tasks are in a 'Pending' state. The right pane shows the output of the './bin/aurras transform samples/sample-1-so.m4a output1.m4a alto alto' command, which is also in a 'Pending' state. A third terminal window is open below the left pane, showing the same command and its 'Pending' status.

```
diogo@vleira63: ~/Dropbox/UNI/2ºANO/2ºSEMESTRE/50/PROJETO/grupo63$ make open_server
./bin/aurrasd etc/aurrasd.conf bin/aurrasd-filters/
TASK:0
alto
alto
TASK:1
alto
Pending
diogo@vleira63: ~/Dropbox/UNI/2ºANO/2ºSEMESTRE/50/PROJETO/grupo63$ ./bin/aurras transform samples/sample-1-so.m4a output1.m4a alto alto
Pending
Processing
```

Task 0 a executar | Task 1 pendente



The image shows the same terminal window as before, but now Task 0 is terminated and Task 1 is executing. The left pane shows 'TASK:0' as 'alto' and 'TASK:1' as 'alto'. The right pane shows the output of the './bin/aurras transform samples/sample-1-so.m4a output1.m4a alto alto' command, which is now in a 'Processing' state. The third terminal window below the left pane also shows the same command and its 'Processing' status.

```
diogo@vleira63: ~/Dropbox/UNI/2ºANO/2ºSEMESTRE/50/PROJETO/grupo63$ make open_server
./bin/aurrasd etc/aurrasd.conf bin/aurrasd-filters/
TASK:0
alto
alto
TASK:1
alto
Pending
diogo@vleira63: ~/Dropbox/UNI/2ºANO/2ºSEMESTRE/50/PROJETO/grupo63$ ./bin/aurras transform samples/sample-1-so.m4a output1.m4a alto alto
Pending
Processing
```

Task 0 terminada | Task 1 a executar

## 4 Conclusão

Desta forma, conseguimos implementar todas os requisitos obrigatórios: o setup das configurações, as transformações com os diversos filtros, o status com a informação dos filtros, o processamento concorrente dos pedidos, o processo a ficar pendente.

Reconhecemos que houve certos aspetos que poderíamos ter feito melhor, como é o caso de, quando um pedido fica pendente não ser possível processar mais pedidos mesmo que haja recursos disponíveis para o fazer, o pedido tem de sair do estado pendente.

Era dito que se devia evitar a criação de ficheiro temporários, mas acabamos por usar um fifo temporário para a comunicação servidor-cliente, de modo a que estas não se baralhassem.

No entanto, consideramos que tivemos um bom aproveitamento neste trabalho, que nos permitiu consolidar e elevar os conhecimentos que fomos adquirindo ao longo desta UC.