

Universidade do Minho

LICENCIATURA EM CIÊNCIAS
DA COMPUTAÇÃO

SISTEMAS OPERATIVOS

Carlos Beiramar, a84628

Jorge Silva, a80931

João Ancieto, a80292

Grupo:

7

17 de junho de 2021

Conteúdo

1	Introdução	3
2	Entidades do serviço	4
2.1	Servidor	4
2.2	Cliente	4
3	Estruturas Utilizadas	5
3.1	Client	5
3.2	Filter	5
3.3	Task	5
4	Arquitetura	6
4.1	aurrasd.c	6
4.2	aurras.c	6
4.3	structs.h	7
5	Conclusão	8

Lista de Figuras

1	<i>Status</i>	6
2	<i>Estrutura do Cliente</i>	7
3	<i>Tasks</i>	7
4	<i>Filtros</i>	7

1 Introdução

Este trabalho consiste no desenvolvimento de um serviço capaz de transformar ficheiros de áudio por aplicação de uma sequência de filtros. Este serviço permite a submissão de pedidos de processamento de ficheiros de áudio, a consulta das tarefas em execução e o número de filtros disponíveis e em uso.

O trabalho foi desenvolvido na linguagem C, tendo em mente a utilização das várias system calls presentes em ambiente Linux abordadas durante a duração da Unidade Curricular.

2 Entidades do serviço

2.1 Servidor

O **Servidor** tem como principal objetivo receber os pedidos de cada um dos clientes, assim, para que isto aconteça, é necessário criar um **FIFO** que estabelece a comunicação entre *Servidor* e o *Cliente*.

O **Servidor** é capaz de suportar e satisfazer vários clientes ao mesmo tempo. Após cada pedido do cliente, é criado um pipe cujo o nome será o pid do cliente para estabelecer uma comunicação individual. Assim, quando um cliente faz um pedido, por exemplo, para ver o **status** do servidor, só esse cliente é que vai obter essa informação.

2.2 Cliente

O **Cliente** vai interagir com o **Servidor** solicitando a execução de instruções. Como é possível vários clientes comunicarem com o servidor em simultâneo, foi necessário usar um formato específico para cada mensagem enviada por cada cliente, ou seja, quando um cliente faz um pedido, a sua mensagem chega ao servidor no formato *pid : mensagem*.

O **Cliente** pode fazer as seguintes intruções:

- Pedir as instruções de como executar o servidor.
- Pedir o estado atual do servidor.
- Executar uma transformação aplicando filtros a um ficheiro áudio.

3 Estruturas Utilizadas

3.1 Client

Cria um objeto **Client** que contém o *pid* do cliente, a *mensagem* e, posteriormente, é criado um array que irá guardar todos os clientes que se conectam ao servidor.

```
typedef struct client{
    char *pid;
    char *message;
}Client;
```

3.2 Filter

Cria um objeto **Filter** que contém o *nome do filtro*, o *nome do executável*, o *máximo de filtros possíveis* e os filtros que estão a ser executados em tempo real(*running*) e guarda a informação num array dinâmico.

```
typedef struct filter{
    char *name;
    char *exec;
    int max;
    int running;
}Filter;
```

3.3 Task

Cria um objeto **Task** que contém os ficheiros de *input*(ficheiro que o cliente pretende aplicar o filtro) e *output*(ficheiro já com os filtros), os *filtros* que o cliente aplicou, o *pid do cliente* e o *número de filtros* que o cliente pediu.

```
typedef struct task{
    char *input;
    char *output;
    char *filters;
    char *client_pid;
    int n_filters;
}Task;
```

4 Arquitetura

4.1 aurrasd.c

Neste ficheiro é onde está implementado o servidor e todas as suas funcionalidades, desde ler o pedido do cliente através do **FIFO**, até executar esse mesmo pedido.

No servidor são implementados 3 arrays:

```
Client *client_array;  
Filter *filters_array;  
Task *task_array
```

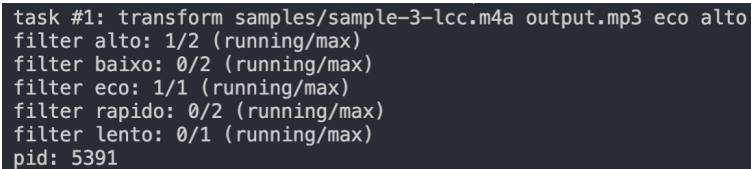
No primeiro array são guardados todos os clientes que se conectarem ao **FIFO** do servidor. No segundo array são guardados todos os filtros que são apresentados no ficheiro *aurrasd.conf*. E por fim, o array **task_array** permite guardar as tasks que irão ser pedidas pelos clientes.

4.2 aurras.c

Neste ficheiro o cliente pode executar os seguintes comandos:

```
./aurras  
./aurras status  
./aurras transform
```

O primeiro comando serve para mostrar as instruções de como se pode fazer transformações num ficheiro áudio. O segundo comando serve para mostrar os status do servidor, como por exemplo:

A terminal window with a dark background showing the output of the 'status' command. The text is as follows:

```
task #1: transform samples/sample-3-lcc.m4a output.mp3 eco alto  
filter alto: 1/2 (running/max)  
filter baixo: 0/2 (running/max)  
filter eco: 1/1 (running/max)  
filter rapido: 0/2 (running/max)  
filter lento: 0/1 (running/max)  
pid: 5391
```

Figura 1: *Status*

O terceiro comando é para fazer transformações no ficheiro áudio, como por exemplo:

```
./aurras transform samples/sample-1.m4a output.mp3 alto eco rapido
```

4.3 structs.h

Neste *headerfile* são implementadas todas as *structs* usadas no trabalho e, para além disso, também são colocados todos os *includes* necessários para a realização do trabalho.

```
typedef struct client{  
    char *pid;  
    char *message;  
}Client;
```

Figura 2: *Estrutura do Cliente*

```
typedef struct task{  
    char *input;  
    char *output;  
    char *filters;  
    char *client_pid;  
    int n_filters;  
}Task;
```

Figura 3: *Tasks*

```
typedef struct filter{  
    char *name;  
    char *exec;  
    int max;  
    int running;  
}Filter;
```

Figura 4: *Filtros*

5 Conclusão

Ao longo do desenvolvimento do trabalho, o grupo deparou-se com algumas decisões em termos de implementação, sendo uma das primeiras a forma de comunicação entre as várias entidades do programa, nomeadamente Servidor/Cliente. Inicialmente começámos por usar a estrutura do *Servidor/Cliente* implementada nas aulas práticas e fomos fazendo alterações de acordo com as necessidades para a realização do projeto.

Em relação às dificuldades enfrentadas, estabelecer uma comunicação individual entre o servidor e um cliente específico foi um obstáculo que conseguimos ultrapassar com sucesso.

No entanto, a maior das dificuldades foi a implementação das transformações com vários filtros pois, estas dependiam da implementação de pipes anónimos e das suas ligações.

Em suma, achámos que existem vários aspetos que poderiam ser melhorados, tais como, a implementação de um sinal para que o servidor termine de forma graciosa.