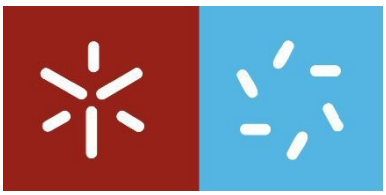

UNIVERSIDADE DO MINHO

LICENCIATURA EM CIÊNCIAS DA COMPUTAÇÃO

SISTEMAS OPERATIVOS

TRABALHO PRÁTICO

GRUPO TP-LCC 32

José Pedro Gomes Ferreira

A91636

Rui Jordão Sampaio Gonçalves

A91652

Tiago André Oliveira Leite

A91693

Ano Letivo 2020/2021

Table of Contents

1. Introdução

Com este projeto foi nos proposto o desenvolvimento em linguagem C de um serviço capaz de transformar ficheiros de áudio através da aplicação de um sistema de filtros.

Para tal, seria necessário implementar um servidor e um cliente, em que o servidor recebe os pedidos de um ou mais clientes e consoante a disponibilidade dos filtros executa os pedidos.

Consideramos que o principal desafio deste projeto seria conseguir gerir a ocupação dos fitros de maneira a que estes nunca excedam o seu limite de utilização bem com a aplicação simultânea de vários filtros à mesma música.

2. Cliente (aurras)

O cliente, denomidado por aurras, aceita dois comandos:

- ./aurras status
- ./aurras transform <ficheiro de input> <ficheiro de output> <filtro 1> <filtro 2> … <filtro n>

Caso seja enviado outro comando ou o nome do filtro enviado esteja incorreto, é imprimida uma mensagem de erro.

Para comunicar com o servidor, o cliente primeiro cria duas pipes com nome na pasta tmp cujos nomes são o seu pid concatenado com as letas R ou W. De seguida envia o nome destas pipes através de uma pipe com nome localizada na pasta tmp e denominada **principal**, pipe esta que foi criada pelo servidor para o estabelecimento de contacto entre cliente e servidor. Toda a comunicação seguinte entre cliente e servidor é feita atraves das pipes **pidR** e **pidW**.

A pipe **pidR** serve para o cliente ler informação do servidor, e a pipe **pidW** serve para o cliente enviar informação para o servidor.

3. Servidor (aurrasd)

O servidor, denomidado por aurrasd, aceita apenas um comando:

- ./aurarsd <ficheiro de configuração dos filtros> <localização dos filtros>

Se o número de parâmetros enviados for diferente ou se o ficheiro de configuração dos filtros estiver incorreto, é imprimida uma mensagem de erro. Por outro lado, se a localização dos filtros não estiver correta, a posterior aplicação dos filtro não irá funcionar.

Depois de configurado, a informação dos filtros fica guardada numa lista ligada de filtros , *struct Filter*, que contém:

- tipo de filtro (alto, baixo, eco, lento, rápido) – *type*;
- nome do executável do filtro – *name;*
- capacidade máxima – *max*;
- ocupação – *max*.

Para estabelecer o primeiro contacto com os clientes, tal como foi dito no tópico anterior, o servidor cria uma pipe com nome denominado **principal**, através do qual recebe o nome das pipes de comunicação específicos de cada cliente. Sempre que há um novo cliente, é feito um fork e o novo processo (filho) criado fica responsável por responder ao pedido do cliente.

Quando recebe um pedido de aplicação de filtro, a primeira tarefa que o servidor faz é verificar se o número de filtros a aplicar é viável. Por exemplo, se a ocupação máxima do filtro “rápido” for 1 e o cliente pedir para aplicar duas vezes este filtro, é enviada uma mensagem de erro ao cliente e a comunicação termina. Caso o pedido seja viável, o processo que está a tratar do pedido tem que verificar a disponibilidade dos filtros. Para tal existe um ficheiro na pasta tmp no qual está guardado a informação da ocupação dos filtros. Através deste ficheiro, cada processo filho obtém a informação atual da ocupação dos filtros e verifica se pode proseguir com a execução do seu pedido. Se sim, o processo filho atualiza o ficheiro consoante os filtros que vai utilizar, informa o cliente que o seu pedido está a ser processado e prossegue para a aplicação dos filtros. Caso contrário envia mensagem ao cliente a informa que o pedido está pendente e fica a fazer leitura do ficheiro até os filtros necessários ficarem disponíveis.

Para guardar a informação do número de tarefas em execução existe um ficheiro na pasta tmp que tem armazenado todas a tarefas executadas ou em execução até ao momento assim como o estado da tarefa. Cada processo antes de proceder à execução dos filtros, atualiza esse ficheiro.

Antes de aplicar um filtro, é verificado se o ficheiro de origem existe. Caso não exista é enviada uma mensagem de erro ao cliente e a comunicação é encerrada.

A aplicação dos filtros é realizada com recurso a pipes anónimos. Para tal é feito um fork do processo, sendo o proceso filho gerado quem vai realizar a aplicação. Na aplicação propriamente dita, em primeiro lugar é feito um dup2(<ficheiro origem>,0) e dup2(<ficheiro destino>,1). Se for para aplicar apenas um filtro, procede-se à utilização da função execl com o filtro. Se for necessário aplicar mais do que um filtro, é utilizado um ciclo de forks em profundidade com dup2 e excel dos filtros de uns processos para os outros. Finalizada a execução dos filtros, o processo pai atualiza os ficheiros de ocupação dos filtros e de tarefas e termina a comunicação com o cliente fechando os pipes que estavam a ser utilizados para a comunicação.

Se o pedido enviado pelo cliente for no sentido de saber qual o estado do servidor, o processo que foi gerado para tratar do pedido do cliente lê a informação atual armazenada nos ficheiros de tarefas e ocupação de filtros mencionados anteriormente, informa o cliente e termina a comunicação com o cliente fechando os pipes que estavam a ser utilizados para a comunicação.

O servidor termina quando recebe um sinal **SIGTERM**. Para tal, aquando da receção do sinal mencionado, é fechada a pipe **principal** impedindo assim que mais clientes se conectem ao servidor e permitindo que todos os pedidos pendentes ou em execução possam ser finalizados.

5. Arquitetura

A arquitetura utilizada, com uma pipe com nome para o estabelecimento do primeiro contacto entre cliente e servidor e posterior utilização de pipes específicas para cada comunicação cliente - processo filho do servidor, paraceu-nos a escolha mais viável de forma a evitar sobreposições de pedidos e erros de leitura e escrita.

A utilização de um ficheiro onde é armazenada a informação da ocupação dos filtros em tempo real, apesar de utilizar uma estratégia de “polling” para saber quando os filtros estão disponíveis, foi a única estratégia que conseguimos implementar com sucesso. Por forma a evitar que houvesse sobreposição na leitura e escitra do ficheiro que contem a ocupação dos filtros, decidimos que antes de ler o ficheiro, o processo era colocado a dormir por um segundo, dando tempo a quem já leu de atualizar o ficheiro antes que outro processo o começasse a ler.

Optámos também por fazer uma certa filtragem do input no cliente de forma a facilitar a leitura de comandos por parte do servidor.

Para a aplicação dos filtros, inicialmente a nossa estratégia passava por aplicar um filtro de cada vez, e na qual o ficheiro de output era aberto e fechado várias vezes. No entanto como essa estratégia não respeitava o número de filtros que estavam a ser usados simultaneamente, pois estava a ser usada apenas um de cada vez, decidimos que seria melhor a aplicaçao de todos os filtros de uma só vez através de uma sequência de pipes e execs de processo filho para processo pai.

5. Implementação das Funcionalidades

```
bin/aurras transform samples/Ievan-Polkka-Loituma.m4a music4.mp3 rapido rapido eco
processing
```

```
bin/aurras status
task #3 transform samples/Ievan-Polkka-Loituma.m4a music4.mp3 rapido rapido eco
filter alto: 0/2 (running/max)
filter baixo: 0/2 (running/max)
filter eco: 1/2 (running/max)
filter lento: 0/1 (running/max)
filter rapido: 2/3 (running/max)
pid: 16637
```

```
bin/aurras status
filter alto: 0/2 (running/max)
filter baixo: 0/2 (running/max)
filter eco: 0/2 (running/max)
filter lento: 0/1 (running/max)
filter rapido: 0/3 (running/max)
pid: 16872
```

```
bin/aurras transform samples/sample-1-so.m4a music1.mp3 lento
processing
```

```
bin/aurras transform samples/sample-2-miei.m4a music2.mp3 alto lento
pending
processing
```

```
bin/aurras status
task #4 transform samples/sample-1-so.m4a music1.mp3 lento
filter alto: 0/2 (running/max)
filter baixo: 0/2 (running/max)
filter eco: 0/2 (running/max)
filter lento: 1/1 (running/max)
filter rapido: 0/3 (running/max)
pid: 17033
```

```
bin/aurras status
task #5 transform samples/sample-2-miei.m4a music2.mp3 alto lento
filter alto: 1/2 (running/max)
filter baixo: 0/2 (running/max)
filter eco: 0/2 (running/max)
filter lento: 1/1 (running/max)
filter rapido: 0/3 (running/max)
pid: 17050
```

6. Conclusão

Com o projeto concluído esperamos ter cumprido todos os requisitos que nos foram propostos sem ultrapassar nenhuma das regras que foi estabelecida no enunciado.

Este trabalho foi fundamental pois premitiu-nos consolidar tudo o que aprendemos ao longo do semestre na disciplina de Sistemas Operativos.