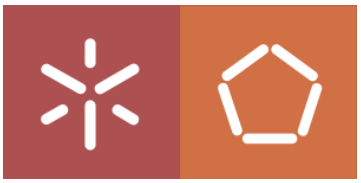

Universidade do Minho

Departamento de Informática

Sistemas Operativos

Aurras: Processamento de Ficheiros de Áudio

João Freitas (A83782) Rui Fernandes (A89138)

Junho 2021

Resumo

O presente relatório descreve o trabalho prático realizado no âmbito da disciplina de *Sistemas Operativos*, ao longo do segundo semestre do segundo ano do Mestrado Integrado em Engenharia Informática da Universidade do Minho.

Com a realização deste trabalho prático, pretende-se implementar um serviço capaz de transformar ficheiros de áudio por aplicação de uma sequência de filtros, sendo que este deverá ser constituído por um servidor e por um cliente que deverão comunicar por *pipes* com nome.

Neste documento descreve-se sucintamente a aplicação desenvolvida e discutem-se as decisões tomadas durante a realização do projeto.

Conteúdo

Lista de Figuras

1 Introdução

No âmbito da Unidade Curricular de *Sistemas Operativos*, foi proposta a implementação de um serviço capaz de transformar ficheiros de áudio por aplicação de uma sequência de filtros, sendo que este deverá ser constituído por um servidor e um por cliente, que deverão comunicar por *pipes* com nome. Tanto o cliente como o servidor apenas disponibilizam uma interface de linha de comandos.

Estrutura do Relatório

O presente relatório encontra-se dividido em quatro partes distintas.

No capítulo 1, Introdução, foi feito um enquadramento e uma breve contextualização do trabalho prático.

De seguida, no capítulo 2, Conceção da Solução, é feita uma descrição detalhada de todo o processo de desenvolvimento do trabalho prático até se obter a solução final.

No capítulo 3, Análise de Resultados, são apresentados alguns testes realizados e os resultados obtidos.

Por fim, no capítulo 4, Conclusão, termina-se o relatório com uma síntese a análise crítica do trabalho desenvolvido.

2 Conceção da Solução

2.1 Implementação da Solução

No desenvolvimento do presente trabalho prático, foi implementada uma arquitetura Cliente-Servidor com o intuito de distribuir processamento da informação entre o fornecedor do serviço – servidor – e os requerentes dos serviço – clientes. Assim, os clientes enviam pedidos para o servidor e este, por sua vez, processa e envia os resultados dos respetivos pedidos. Note-se que os clientes e servidor comunicam através de *pipes* com nome.

Desta forma, um dos pontos centrais serão os mecanismos de comunicação entre processos (IPC), assim como a utilização das diferentes *System Calls*.

Para controlo de erros foi também utilizada diversas vezes a função `perror` que imprime, se for o caso, uma mensagem de erro para o `stderr`.

Posto isto, importa também salientar que foram utilizados como ponto de partida a seguinte estrutura do código e a `Makefile` disponibilizados.

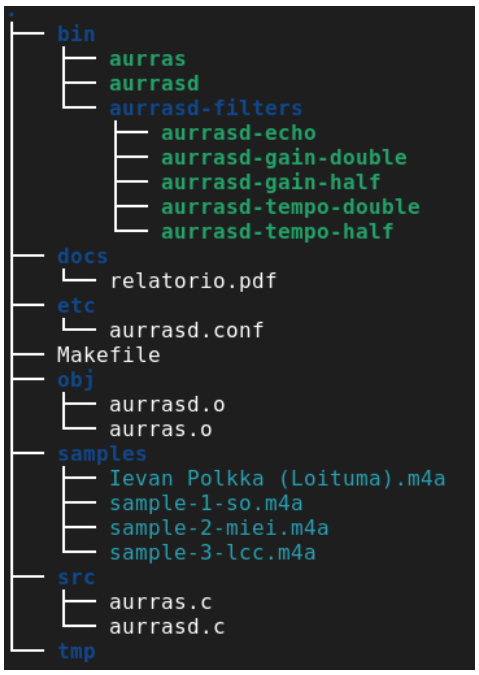

Figura 1: Estrutura do código

Servidor

No arranque do servidor, são criados dois *pipes* com nome que serão conhecidos por todos os clientes, um destinado aos pedidos dos clientes, e outro para o servidor enviar o seu estado atual.

Ao receber um pedido de transformação de um ficheiro de áudio, o servidor primeiro verifica se o pedido foi enviado de forma correta e, nesse caso, verifica se todos os filtros solicitados se encontram disponíveis. Feitas as devidas verificações, cria-se um processo filho que será responsável por processar o pedido, *i.e.*, por cada pedido recebido no servidor, é criado um processo filho que irá efetivamente tratar o pedido. Assim, este processo filho,

recorrendo à *system call* `execl` e a um correto redirecionamento de descritores de ficheiros, garante que os filtros são aplicados sequencialmente. Aqui, importa salientar que no caso do primeiro filtro, o `stdin` será o ficheiro de *input* e, por oposição, no caso do último filtro, o `stdout` será o *pipe* com nome criado pelo cliente para o efeito, sendo este responsável por guardar o ficheiro final. Por outro lado, ao receber um pedido sobre o seu estado atual, o servidor envia um sinal `SIGUSR1` a todos os processos filhos que, por sua vez, deverão enviar para o cliente, via *pipe* com nome, informação relativa ao pedido que estão a processar. No caso do processo pai, este deverá enviar informação relativa à capacidade e disponibilidade dos filtros de áudio.

Por fim, ao receber um sinal `SIGTERM`, o servidor espera que o processamento atual dos ficheiros de áudio termine e, posteriormente, elimina os *pipes* criados aquando do arranque.

Cliente

Tal como referido anteriormente, o cliente apenas tem duas formas de interação com o servidor – efetuar um pedido de transformação de um ficheiro de áudio ou pedir a informação do estado do mesmo.

Para implementar estas funcionalidades recorreu-se, em ambos os casos, ao uso de *pipes* com nome. Assim, no caso da transformação de ficheiros de áudio, quando o cliente efetua um pedido, é criado um *pipe* com nome, utilizando, para isso, o seu PID como identificador. Submetido o pedido, este espera a resposta do servidor e, posteriormente, cria o ficheiro final, eliminando, por fim, o *pipe* criado.

Por outro lado, quando o cliente envia um pedido relativo ao estado do servidor, irá receber esta resposta através do *pipe* com nome criado especificamente para este efeito. Após imprimir para o `stdout` esta informação, fecha o *pipe* e termina a execução.

2.2 Limitações da Solução Desenvolvida

No seguimento da secção anterior, as limitações encontradas da solução implementada passam por:

— Na eventualidade de o servidor não ter todos os filtros disponíveis, o pedido é descartado em vez de ficar em fila de espera até haver disponibilidade no servidor;

— Limitação quanto ao número máximo de pedidos a ser executados em simultâneo, assim como do número de filtros.

2.3 Considerações Finais

Apesar limitações anteriormente apresentadas, importa salientar que foram implementadas funcionalidades que oferecem grandes recursos ao servidor no que diz respeito a concorrência e desempenho, pelo que, de um modo geral, faz-se um balanço bastante positivo do trabalho desenvolvido. De facto:

— O servidor inicia o tratamento dos pedidos de forma sequencial, suportando o processamento concorrente dos mesmos;

— Ao receber o sinal `SIGTERM`, o servidor termina de forma graciosa o seu funcionamento, isto é, termina os pedidos que em espera e só no fim termina o processo;

— Evita a criação de ficheiros temporários;

— O processamento de um pedido não é iniciado pelo servidor se não existirem instâncias disponíveis de todos os filtros necessários.

3 Análise de Resultados

Informação de Utilização

```
~/SO/grupo-032 (Linux Intel) bin/aurras
USAGE:
./aurras status
./aurras transform input-filename output-filename filter id-1 filter id-2 ...
~/SO/grupo-032 (Linux Intel) bin/aurrasd
USAGE:
./aurrasd config-filename filters-folder
~/SO/grupo-032 (Linux Intel)
```

Figura 2: Informação de utilização

Processamento de Ficheiros de Áudio

```
~/SO/grupo-032 (Linux Intel) make run
bin/aurrasd etc/aurrasd.conf bin/aurrasd-filters/
Server Staring   (PID = 423769)
Config file etc/aurrasd.conf parsed      5 filters

Request received: 423771 samples/sample-1-so.m4a tmp/sample-1.mp3

PID: 423771
Input: samples/sample-1-so.m4a
Output: tmp/sample-1.mp3
Filtro: N/A
Request #423771 is invalid

Request received: 423909 samples/sample-1-so.m4a tmp/sample-1.mp3 alto

PID: 423909
Input: samples/sample-1-so.m4a
Output: tmp/sample-1.mp3
Filtro: alto
```

Figura 3: Processamento de ficheiros de áudio

Sinal `SIGTERM` & Terminação do Servidor

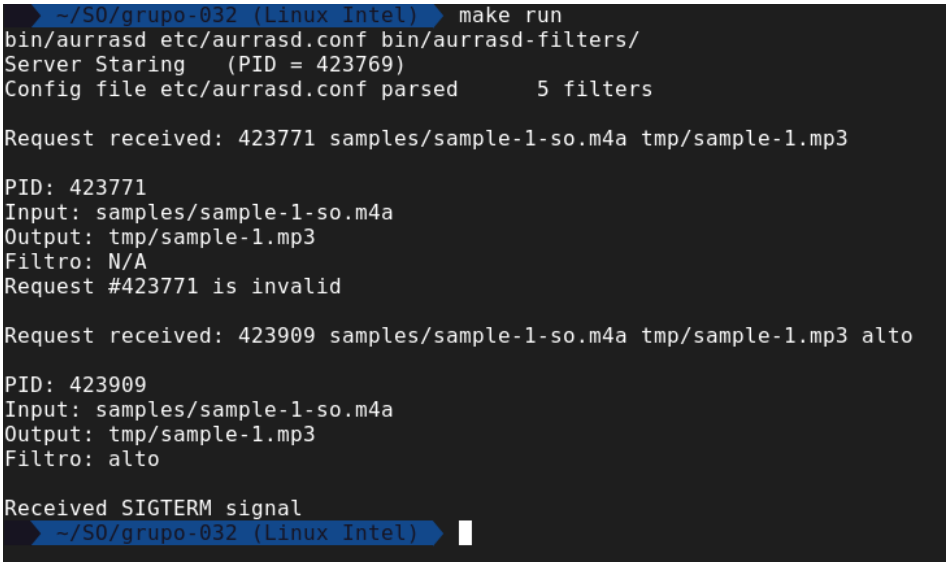

Figura 4: Sinal `SIGTERM` & terminação do servidor

Estado do Servidor

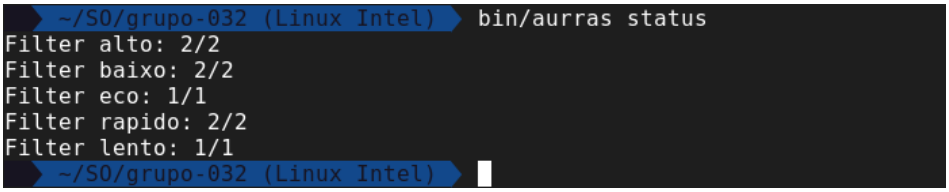

Figura 5: Estado do servidor

Figura 6: Estado do servidor

4 Conclusão

Face ao problema apresentado e analisando criticamente a solução proposta concluímos que cumprimos as tarefas propostas, conseguindo atingir os objetivos definidos. De facto, foram implementadas todas as funcionalidades apresentadas. Assim, após a elaboração deste trabalho prático, pode-se afirmar que aprimoramos o nosso domínio sobre a linguagem de programação `C`, nomeadamente o uso das *System Calls*, que foram a base do nosso trabalho.

No entanto, o grupo consegue reconhecer que, com uma melhor gestão do tempo, teria sido possível aprimorar as funcionalidades feitas até ao presente.

Como trabalho futuro, poderia ser implementado um histórico associado a um dado cliente e as tarefas executadas por esse mesmo cliente, entre outras. Neste sentido, importa salientar que a base de cliente e servidor implementada permite futura escalabilidade.

Lista de Siglas e Acrónimos

IPC *Inter-process Communication*

PID *Process ID*

`stderr` *Standard Error*

`stdin` *Standard Input*

`stdout` *Standard Output*