Universidade do Minho
Escola de Engenharia

Aurras: Processamento de ficheiros de áudio

Beatriz da Silva Rodrigues
a93230

Índice

Introdução

Este documento descreve a implementação de uma ferramenta capaz de transformar ficheiros de áudio através da aplicação sequencial de filtros a um ficheiro de áudio.

Era pretendido que a ferramenta permitisse a submissão de pedidos de processamento de ficheiros de áudio e a consulta de informações relevantes acerca do servidor.

Para a comunicação entre o servidor e o cliente foram usados pipes com nome, enquanto para o encadeamento de filtros num ficheiro de áudio foram usados pipes anónimos.

Estrutura do serviço

O serviço apresenta uma arquitetura cliente servidor. A comunicação entre estes é realizada por pipes com nome. O cliente no início da sua execução cria um pipe para comunicação no sentido servidor para cliente com o nome correspondente ao seu pid, enquanto o servidor cria um pipe com sentido oposta. O cliente envia um pedido ao servidor, que o interpreta, valida e executa.

O ficheiro *aurrasd.c* está relacionado com as funcionalidades do servidor enquanto o ficheiro *aurras.c* corresponde às funcionalidades do cliente.

O serviço resultante deste projeto apresenta as seguintes funcionalidades:

- Permite a um cliente transformar ficheiros de áudio com os filtros configurados de forma concorrente, desde que respeite o número de instâncias disponíveis, atualizando-o acerca do decorrer do processo (“Pending”, “Processing”)

- Permite a um cliente obter o estado do servidor: as tarefas a ser executadas, a utilização de cada filtro associada ao máximo de instâncias e o pid do servidor

- Permite disponibilizar ao cliente um guia de como utilizar o serviço

Cliente

O primeiro passo no que toca a implementação do cliente consistiu na validação dos comandos do utilizador. Apenas se a instrução for válida, de acordo com a *prompt* de ajuda disponibilizada, esta será escrita no FIFO de comunicação com o servidor, de forma a que este responda ao pedido.

Se o utilizador inserir *"bin/aurras":*

- é imprimido para STDOUT um guia de utilização do serviço para o utilizador

Se o utilizador inserir "*bin/aurras status*":

- o cliente cria um FIFO com o nome equivalente ao valor do pid do seu processo, dentro da diretoria tmp, e abre o FIFO principal de comunicação com o servidor
- escreve uma string que contém um 0, que será interpretado como status pelo servidor, e o *path* para o FIFO que pode usar para comunicar com o cliente, separados por uma vírgula.
- abre o primeiro FIFO que criou e lê-o, obtendo a informação do status do servidor que lhe foi devolvida e escrevendo-a para STDOUT.

Se o utilizador inserir *"bin/aurras transform source_path output_path filtro1 … "* (notando que tem de ser introduzido pelo menos um filtro):

- são abertos 2 FIFOS de forma idêntica ao caso anterior e é construída a expressão que irá representar a instrução que contém os seguintes parâmetros separados por vírgulas: 1, source path, output path, filtros (separados por um espaço cada) e o path para o FIFO do cliente
- a instrução é escrita para o FIFO do servidor e o utilizador é informado que o seu pedido está pendente. Por fim, abre o seu FIFO e aguarda resposta do servidor. Quando o servidor lhe responde com '1' significa que o servidor se encontra atualmente a executar o seu pedido, o que conduz o cliente a indicar ao utilizador o facto de estar "Processing", imprimindo a informação para STDOUT. Quando o servidor lhe responde com '0', o pedido está pronto e o cliente pode terminar a sua execução e consultar o ficheiro resultante.

Servidor

Relativamente ao servidor, este irá ler o FIFO principal continuamente, aguardando instruções por parte do cliente. No entanto, para garantir que o servidor conseguirá responder de forma apropriada, este deve ser configurado com o auxílio de um ficheiro de configuração e o *path* para os executáveis de cada filtro.

Configuração

Assim, foi necessário escolher uma maneira de associar o nome dos filtros aos *paths* para o seu respetivo executável e também ao seu número de instâncias máximas. Para este fim foi criada uma lista ligada FILTERS, que contém em cada nodo o nome de um filtro, o *path* para o executável, o número de instâncias a ser usadas, o número máximo de instâncias disponíveis e um apontador para outro filtro. Esta opção foi tomada por razões centradas em simplificar a implementação o mais possível, apesar de haver consciência de que existiriam maneiras muito mais eficientes de o fazer.

FIFO principal e gestão dos pedidos

Logo a seguir à configuração, é criado o FIFO principal, que os clientes irão utilizar para comunicar os seus pedidos e é aberto com permissões de leitura. Estes pedidos serão armazenados numa estrutura REQUEST, que se trata de uma lista ligada onde cada nodo representa um pedido e contém a sua informação organizada em: tipo da operação, *path* para o ficheiro a alterar, *path* para colocar o ficheiro resultante, filtros separados por espaços, string com o path para o FIFO do cliente, número da *task* e o pid do processo filho responsável pela execução do pedido.

Finalização de um pedido

No início do ciclo responsável por manter o servidor a executar continuamente, a estrutura de requests é percorrida e procura-se, para cada um, se o processo que o executou já terminou, através da *system call* waitpid, com os argumentos correspondentes ao valor do pid do processo a verificar (contido no nodo correspondente da estrutura REQUEST), um apontador para colocar o status e WNOHANG para o programa não bloquear à espera desse processo, caso não tenha acabado. Se esta *system call* devolver um valor diferente do valor do pid, significa que não acabou, avançando para a verificação do pedido seguinte. Caso contrário, é necessário verificar que filtros é que esse pedido estava a monopolizar e sinalizar que esses filtros estão agora livres, atualizando a estrutura FILTERS, no campo usage. É também removido esse request da estrutura.

Receção de um novo pedido

Para receber um novo pedido, o servidor lê do FIFO principal para um buffer chamado instruction. De seguida, diferencia que tipo de pedido foi enviado. Se o primeiro caractere corresponder a um 1, trata-se de uma instrução do tipo transform que é adicionada à estrutura REQUEST, enquanto que se corresponder a um 0 trata-se de uma instrução do tipo status que irá carregar o estado atual do servidor.

Transform: O buffer que contém a instrução é parsed na função *add_request* e adicionado à estrutura para ser processado logo que possível.

Status: Foi desenvolvida uma função *load_status* que, recorrendo à estrutura REQUEST e à estrutura FILTERS, imprime toda a informação necessária. As tarefas pendentes encontram-se na primeira estrutura referida enquanto a utilização e instâncias máximas de cada filtro se encontram na segunda.

Processamento de um pedido

Para processar um pedido, recorre-se ao nodo para onde a estrutura REQUEST estiver a apontar atualmente. O primeiro passo é abrir o FIFO do cliente, de acordo com o *path* incluído dentro da estrutura na variável *pid_str*, e escrever o caractere '1', que é interpretado pelo cliente como o sinal de que o seu pedido entrou em processamento. De seguida, os filtros são contados e são abertos os descritores para o ficheiro fonte e para o ficheiro de destino.

O passo seguinte corresponde a criar um array com o tamanho correspondente ao número de filtros que irá conter o *path* para o executável responsável pela aplicação de cada filtro contido no REQUEST. No entanto, se algum executável não estiver disponível, o processo é travado numa espera ativa até que passe a estar disponível.

A este ponto, é finalmente possível iniciar a execução, logo é criado um processo filho para o fazer, criando assim também a possibilidade de executar pedidos de forma concorrente. Dentro do processo filho é feita uma separação em dois casos: se apenas for necessário aplicar um filtro, não há qualquer necessidade de criar pipes anónimos para redirecionamento de IO; caso seja necessário aplicar mais do que um filtro, estes são essenciais.

Para o primeiro caso referido, basta fazer, dentro de outro processo filho, a duplicação do ficheiro fonte e destino para, respetivamente, STDIN e STDOUT e, de seguida, usar execlp onde os argumentos correspondem ao path para o executável do filtro correspondente.

Para o segundo caso referido, nota-se que será preciso um número de pipes correspondente ao número de filtros menos uma unidade. Realiza-se o redirecionamento de IO para, no final, ser originado um ficheiro resultante da execução sequencial de cada filtro pedido.

Os processos filhos usados para tudo isto são recolhidos no final e escreve-se no FIFO do cliente o caractere '0' que significa que o processo terminou e o ficheiro está disponível. Entretanto, o processo pai avançou para um pedido diferente e a execução do programa continua.

Conclusões

O serviço implementado é funcional no que toca aos comandos requeridos. No entanto, realiza-o de formas pouco eficientes que poderiam ser otimizadas. Um exemplo seria a estrutura FILTERS que, ao invés de ser uma lista ligada, poderia ser um array de estruturas. Outro exemplo corresponde à aplicação da espera ativa (que não foi corretamente implementada), um dos maiores problemas nesta solução. Deveria também ser utilizado um método mais apropriado para fazer a espera, como por exemplo realizar outros processos na fila enquanto não seria possível a execução do pretendido. Por fim, também teria sido ideal a implementação do requisito do enunciado relacionado com o sinal SIGTERM.

Porém, considera-se que corresponde a uma solução funcional que satisfaz a maior parte dos requisitos necessários.