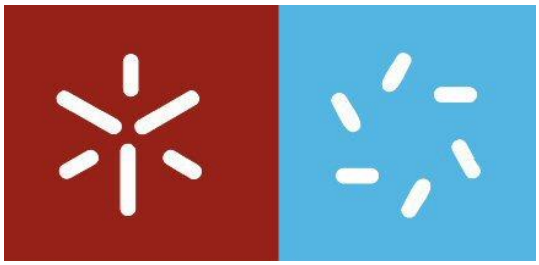

UNIVERSIDADE DO MINHO

LICENCIATURA EM CIÊNCIAS DE COMPUTAÇÃO

SO – Trabalho Prático

2020/2021

Grupo 27

Joel Magalhães Pinto - A91645

Francisco José Pereira Teófilo - A93741

Rui Pedro Magalhães Torres - A84588

Conteúdo

1. Introdução 3
2. Servidor 4
   2.1. Chamadas 4
   2.2. Funcionalidade 4
3. Cliente 6
   3.1. Chamadas 6
   3.2. Funcionalidade 6
4. Conclusão 7

1. Introdução

Este projeto consistiu na criação de um serviço de execução de filtros sobre ficheiros áudio, no qual o cliente é capaz de enviar pedidos para aplicar um ou mais filtros a um ficheiro áudio, ou pedidos para verificar o estado do servidor.

Por sua vez, o servidor recebe os pedidos de números clientes, e resolve-os adequadamente, seja isto através da aplicação de filtros ou enviar o seu estado.

Inicialmente não houve muitos desafios na implementação da estrutura base e comunicação Cliente->Servidor e comunicação privada Servidor->Cliente, mas rapidamente se tornou num projeto complexo acompanhando o aumento de funcionalidades. Tendo por destaque a execução de filtros, que apenas recebiam o ficheiro de input pelo seu STDIN e apenas escreviam no seu STDOUT, forçando o redireccionamento de descritores de ficheiro, o que dificultou exponencialmente o encadeamento destes. Este encadeamento forçou, para preencher também o requisito desejável de não criar ficheiros temporários, a utilização de pipes anónimos entre processos-filho.

É de valor também mencionar a funcionalidade de colocar um pedido em espera enquanto não há filtros suficientes disponíveis, uma vez que foi bastante difícil implementar.

Apesar de tudo pensamos que temos um projeto bem conseguido, com todas as funcionalidades obrigatórias e desejadas, tudo em código estruturado ao melhor das nossas habilidades.

2. Servidor

2.1. Chamadas

O servidor tem 2 chamadas distintas:

- ./bin/servidor
  Age como um -h habitual, imprime para o STDOUT como se deve chamar este programa.

- ./bin/servidor *Ficheiro de configuração* *Ficheiro com filtros*

  Exemplo: *etc/aurrasd.conf* *bin/aurrasd-filters*

  É importante não colocar '/' no fim do ficheiro com filtros.

2.2. Funcionalidade

O servidor quando é inicializado verifica se o ficheiro de configuração e válido e lê deste os parâmetros que deve assumir para o seu funcionamento.

Após ler os parâmetros cria um pipe com nome que será por onde irá receber pedidos dos clientes, abre ambas as pontas (começando pela de leitura), de modo a não receber EOF quando não há clientes ligados ao FIFO.

Posteriormente entra num ciclo de ler pedidos que apenas termina quando recebe o sinal SIGTERM (altera a variável *podeReceber* para 0) e a lista de tarefas a executar está vazia. Se *podeReceber* estiver a 0 mas ainda houver itens na lista, o servidor recebe na mesma pedidos, mas se estes pedidos forem *transformar*, diz a esse cliente que já não aceita pedidos para *transformar* (apenas aceita pedidos de status).

Dentro deste ciclo o servidor segue os seguintes passos:

Analisa(*parse*) o comando recebido » cria processo-filho que irá tratar desse pedido.

> Este processo-filho é criado para que o servidor não gaste tempo a processar pedidos enquanto que outros clientes estão a tentar enviar pedidos (colocando estes à espera).

Servidor volta ao início do ciclo à espera de pedidos.

Entretanto no processo-filho, chamemos-lhe **manager de pedido**, caso o pedido seja do tipo *status*, abre o FIFO privado de resposta a cliente, e envia o estado atual. Caso o pedido seja do tipo *transform*, o **manager** verifica se há recurso suficientes e/ou disponíveis para executar este pedido.

Se, porventura, não existirem filtros suficientes para processar o pedido, responder adequadamente ao cliente, caso haja, cria um processo-filho, que chamaremos de **gerente de execs**, e fica à espera que este termine depois de adicionar este pedido à **lista de tarefas**.

No **gerente de execs**, sabemos que há filtros suficientes para correr o pedido, mas se estes estão ocupados, informa o cliente que o pedido está em espera (*Pending…*).

Quando sair do estado de espera, informa o cliente que o pedido está a ser processado e inicia a execução do pedido através de processos-filho que correm concorrentemente e comunicam entre si através de pipes anónimos, sendo que cada processo-filho executa um dos filtros desejados.

As execuções de filtros terminam todas e o **gerente** termina, retomando então o **manager de pedido**, este último remove o pedido da **lista de tarefas** e termina.

É necessária esta arquitetura para tratamento de um pedido *transform* pois para receber a informação de que um pedido terminou é necessário fazer *wait*, e colocar um *wait* no servidor iria colocar todos os outros clientes à espera, daí o **manager de pedido** englobar o **gerente de execs**, o **gerente de execs** é obviamente necessário para realizar os *execs* concorrentemente.

Para finalizar, a **lista de tarefas** é uma *linked-list* em que cada nodo contém:

- o pid do **gerente de execs** que está a tratar dos filtros desse pedido;
- uma *string* (*char string[150]*) que contém em forma de texto esse pedido;
- apontador para o próximo nodo.

São adicionados/removidos nodos a esta lista através de um pedido enviado ao servidor pelo **gerente de execs** (adicionar) ou pelo **manager de pedido** (remover), esse pedido é enviado para o mesmo *FIFO* em que o servidor recebe pedidos de clientes. Estes pedidos internos não são executados dentro de um processo-filho.

O servidor apenas permite pedidos com até 15 filtros, algo que pode ser facilmente alterado no código (é um define MAX_FILTROS) ou até mesmo implementado no ficheiro de configuração.

3. Cliente

3.1. Chamadas

O Cliente tem 3 chamadas distintas:

- ./bin/cliente
  Age como um -h habitual, imprime para o STDOUT como se deve chamar este programa.

- ./bin/cliente *status*
  Pede ao servidor o seu estado

1..1.1. ./bin/cliente *transform* *FileIn* *FileOut* *[FilterID1…FilterID2]*
Exemplo: *transform* *samples/sample-1-so.m4a* *output* *alto*

3.2. Funcionalidade

O Cliente é um programa bastante simples, ao ser inicializado verifica se o servidor está operacional, caso esteja e o argumento seja *status*, envia ao servidor um pedido de status, caso o argumento seja de *transform* verifica se o *FileIn* existe, se o *FileOut não existe e* se os filtros têm todos IDs válidos. Após estas confirmações todas, envia ao servidor um pedido de transform com a informação necessária, principalmente o ID do *FIFO* privado.

4. Conclusão

Durante a realização deste trabalho, foi possível observar que todas as funcionalidades pedidas estão a funcionar na perfeição, para além das funcionalidades extra que foram pedidas. Sendo assim, pensamos que o trabalho foi bem conseguido.

Este trabalho permitiu que conseguíssemos praticar e consolidar a matéria lecionada ao longo do semestre na Unidade Curricular de Sistemas Operativos de forma útil e consistente.