Relatório de SO

Grupo 109

Curso: MIEI

Unidade Curricular: Sistemas Operativos (H504N1)

Coordenador: Professor Doutor Francisco Coelho Soares Moura

Ano Letivo 2020/2021

Nome: Carlos Filipe Almeida Dias
Número: 93185
Contacto: a93185@alunos.uminho.pt

Nome: José Pedro Martins Magalhães
Número: 93273
Contacto: a93273@alunos.uminho.pt

Nome: Francisco Reis Izquierdo
Número: a93241
Contacto: a93241@alunos.uminho.pt

Índice

Introdução

No âmbito da disciplina de Sistemas Operativos tivemos como projeto o desenvolvimento de um servidor cuja função é estabelecer a ligação com o(s) cliente(s) de forma a processar o(s) seu(s) pedido(s). Estes clientes podem chegar em simultâneo e/ou sequencialmente, sendo que tivemos de escolher uma abordagem apropriada de forma a lidar com estes casos. Além disso, os clientes podem enviar ao servidor um dos dois pedidos seguintes:

- Verificar o estado do servidor, isto é, o servidor enviar ao cliente a lista de pedidos pendentes e/ou a processar de outros clientes. O cliente consegue mandar este pedido ao correr o executável *./aurras status*.
- Transformar ficheiros de áudio por aplicação de uma sequência de filtros. O cliente consegue este pedido ao correr o executável *./aurras transform file_input file_output filtro1 filtro2…*, onde *file_input* denomina o ficheiro de áudio que o cliente deseja que o servidor transforme, *file_output* denomina o ficheiro áudio de saída já transformado e por fim a sequência de filtros.

Como foi supramencionado, tivemos de escolher uma abordagem apropriada, sendo que esta engloba várias vertentes como ligação cliente – servidor e gestão de pedidos de clientes. Estas vertentes foram implementadas com recurso às *system calls* que foram abordadas nas aulas práticas da disciplina. Por fim tivemos de ter em conta o uso de estruturas de dados apropriadas, de modo a ter a informação quer da parte do cliente quer da parte do servidor guardada de forma concisa e eficiente. Todo o projeto teve como linguagem de programação a indicada no enunciado do mesmo, a linguagem C.

Assim, iremos detalhar a abordagem que escolhemos de forma a implementar o projeto que nos foi proposto.

Estratégias Abordadas

Uma vez que o cliente terá de estabelecer uma ligação com servidor como foi acima mencionado, uma das estratégias que implementámos foi o uso de *fifos (pipes* com nome), matéria abordada no guião 6 das aulas práticas da disciplina. Desta forma, o cliente estabelece a ligação com o servidor que comunica com o mesmo, enviando os seus pedidos por um *fifo* e este recebe o resultado do seu pedido através também de um *fifo* único a cada cliente.

Tal como referimos acima, tivemos de ter em atenção à gestão de pedidos dos clientes uma vez que, consoante o número de filtros no pedido de um cliente, teríamos de processar o ficheiro de *input* de forma sequencial de filtro em filtro, fazendo uso da família de funções de *system call exec*, matéria abordada no guião 5. Assim, para cada filtro tem de ser invocado o executável correspondente, através de uma *system call* da família de funções anteriormente referida. Estes executáveis são invocados sequencialmente, passando a informação entre si através de ficheiros temporários que são posteriormente apagados, quando o pedido do cliente é finalizado. Além disso, de forma que o servidor consiga executar todos os pedidos e não haja clientes que não tenham o seu pedido processado, estes são postos numa fila de espera. Posteriormente, o pedido do cliente que está na fila de espera será processado por processos filhos, fazendo uso da *system call fork*, matéria abordada no guião 1.

Outra estratégia foi a implementação de estruturas de dados, tal como supramencionado. Ora uma vez que teríamos vários pedidos e dado o problema acima mencionado, tivemos de implementar uma estrutura de dados que fosse guardando estes pedidos, desempenhando o papel de fila de espera acima referido. Como temos um número limitado de filtros que podem estar a ser usados ao mesmo tempo, tivemos de implementar uma estrutura de dados que tivesse guardada os filtros existentes, bem como as suas capacidades máximas e a sua disponibilidade.

Arquitetura: Cliente – Servidor

A arquitetura implementada por detrás do projeto, tem como base a comunicação entre cliente e servidor, no qual o cliente envia o seu pedido ao servidor e este comunica de volta depois de o processar enviando-lhe o resultado obtido.

O pedido do cliente pode ser um pedido de consulta, isto é, consultar o estado do servidor ou um pedido de transformação de um ficheiro áudio por uma sequência de filtros. Ambos os tipos de pedido por parte do cliente são enviados para o único *fifo* de comunicação unidirecional com o servidor, sendo os pedidos constantemente lidos pelo seu processo principal.

Uma vez que temos dois tipos distintos de pedidos teremos de os abordar de forma separada e distinta.

Quando um cliente envia ao servidor um pedido de consulta de estado do mesmo, o processo principal cria um processo filho que tratará de enviar as informações paralelamente ao cliente. Este processo acede às duas estruturas de dados que contêm a lista de pedidos em execução ou espera até ao momento do pedido do cliente assim como a lista que contem a informação correspondente aos filtros.

Após o processo filho agregar a informação relevante ao estado do servidor e pretendida pelo cliente aquando do seu pedido, este envia-a ao cliente através de um *fifo* único criado pelo cliente antes de enviar o pedido. Este *fifo* é único uma vez que o seu nome será o *pid* do processo cliente e é enviado ao servidor juntamente com o pedido para que este o possa abrir.

Quando um cliente envia ao servidor um pedido de transformação de um ficheiro áudio por sequência de filtros, este pedido é redirecionado para a função *transform* que verifica se o processo pode ser executado de imediato. Um pedido só pode ser executado de imediato se todos os filtros do pedido estão disponíveis, isto é, o número de filtros a serem usados no momento é menor que a capacidade máxima permitida. Caso contrário, o pedido terá de ficar *pending*, sendo inserido na fila de espera, evitando assim a criação de processos que criem esperas ativas.

Caso um processo possa ser executado, este é enviado à função *executaPedido* que cria um processo filho que tratará da execução do pedido paralelamente ao servidor. Dentro deste processo é chamada a função *executa*, que invocará os executáveis associados a cada filtro sequencialmente, gerindo o redireccionamento do *stdin* e *stdout* bem como os processos envolventes. Esta gestão é explicada na secção Gestão de Processos.

Estrutura de Dados

De forma a complementar e a guardar a informação presente no servidor, como os pedidos pendentes, a executar e a informação relacionada aos filtros existentes, de forma eficiente e de fácil manipulação, tivemos de criar duas estruturas de dados auxiliares.

- Lista ligada com os pedidos pendentes e em execução:

  Esta estrutura de dados permite guardar a informação relativa ao pedido de transformação de um ficheiro de áudio feito por cada cliente, bem como o *pid* do processo cliente e, caso este já esteja em execução, o *pid* do processo que o executa (caso contrário tem um valor por *default -5*).

- Lista ligada com a informação relativa aos filtros existentes:

  Esta estrutura de dados permite guardar os diferentes filtros existentes no ficheiro disponibilizado pelos docentes bem como a informação de cada um. Para cada filtro é guardado o seu nome, o nome do executável associado, a capacidade máxima e a quantidade que está a ser utilizada a cada momento.

Como ambas as estruturas de dados referidas são acedidas apenas pelo processo principal do servidor, garantimos que apenas são acedidas para verificação de um pedido de cada vez. Estas são atualizadas também pelo processo principal do servidor antes da execução de cada pedido de transformação ou através de envio de sinais por parte de processos filhos no final da execução dos mesmos.

Gestão de processos

Como foi mencionado anteriormente, a gestão de processos ocorre quando um pedido de cliente de transformação é executado. Assim, o processo filho responsável pela execução do pedido irá criar tantos filhos sequenciais (e esperar que cada um acabe antes de criar o próximo) quantos filtros o pedido do cliente tiver. Cada um destes filhos irá chamar a *system call execl* para cada executável associado a cada filtro. Além disso, estes filhos terão de comunicar entre si de forma a enviar o resultado da invocação de um filtro para o seguinte filtro, assim como ler o resultado da invocação do filtro anterior. Esta comunicação é feita através de n-1 ficheiros temporários (sendo n o número de filtros no pedido) cujo redireccionamento é feito através da *system call dup2*.

Caso estejamos no processo correspondente ao primeiro filtro, redirecionamos o *stdin* para o apontador do *file_input* e caso contrário redirecionamos o *stdin* para o apontador do ficheiro temporário do processo anterior. Caso estejamos no processo correspondente ao último filtro, redirecionamos o *stdout* para o apontador do *file_output* (sendo que este caso não exista tem de ser criado) e caso contrário redirecionamos o *stdout* para o apontador do ficheiro temporário do processo atual.

É de salientar que, para cada processo, temos de fechar os ficheiros após termos feitos os devidos redireccionamentos.

Tratamento e uso de Sinais

O uso e tratamento de sinais tornou-se um ponto fulcral do nosso projeto uma vez que precisamos de notificar o cliente acerca do estado do seu pedido bem como atualizar a informação presente no servidor de forma assíncrona.

Em termos de notificar o cliente sobre o seu pedido, que é de transformação, de que está em execução após se verificar que os filtros do pedido estão todos disponíveis, antes de criar o processo filho responsável pelo pedido, o processo principal do servidor envia um sinal (*SIGUSR1*) ao processo do cliente que trata de imprimir no ecrã do cliente a notificação pretendida.

De forma a notificar o cliente que o seu pedido já foi processado, o processo responsável pelo pedido, no final do mesmo envia ao processo cliente um sinal (*SIGUSR2*) que trata de imprimir no ecrã do cliente a notificação pretendida.

No que toca a atualizar a informação presente no servidor, no final de processar o pedido de um cliente do tipo transformação de ficheiro áudio, o processo responsável pelo pedido envia um sinal (*SIGBUS*) ao processo principal do servidor para que atualize a fila de espera, removendo os pedidos que já foram processados incluindo o que enviou o sinal e mandando executar o primeiro pedido que pode executar da fila de espera.

De forma a encerrar o servidor, este ao receber o sinal para tal (*SIGTERM*), encerra o *pipe* de comunicação que estabelecia entre os clientes de forma a não aceitar novos pedidos e além disso acaba todos os pedidos que estão na lista de espera e só quando a lista se encontra sem pedidos por executar é que o processo principal termina.

Conclusão

Após a implementação e conclusão do projeto conseguimos consolidar os conhecimentos adquiridos ao longo do semestre. Além disso, com a realização do projeto proposto, conseguimos perceber a importância da disciplina de Sistemas Operativos, uma vez que é através dos conhecimentos adquiridos desta que conseguimos criar um projeto como este no qual é necessária extrema atenção em várias vertentes. Estas vertentes são nomeadamente a gestão de processos, redireccionamentos de leitura/escrita, criação de processos filhos e uso de *pipes*. Ao longo do projeto, este foi bastante desafiante no qual tivemos algumas dificuldades como no redireccionamento de leitura/escrita dado que em muitos testes os processos ficavam bloqueados, bem como tivemos que usar ficheiros temporários que fizessem o papel de pipes anónimos, uma vez que tivemos limitações em relação ao transporte de grandes quantidades de dados.
Outra dificuldade foi na implementação da arquitetura do projeto pois apesar de definirmos inicialmente o que pensávamos ser uma boa arquitetura, rapidamente percebemos que teríamos que ter mais fatores em atenção como os supramencionados.

Em suma, achamos que conseguimos alcançar os objetivos pretendidos para este projeto e que fizemos um bom trabalho.