Universidade do Minho

Mestrado Integrado de Engenharia Informática
Sistemas Operativos

PROJETO CLIENTE/SERVIDOR – AURRAS

(MiEI-SO 20/21)

GRUPO Nº 25:

- Joana Maia Teixeira Alves (A93290)

- Vicente Gonçalves Moreira (A93296)

Data de Entrega: 17/06/2021

ÍNDICE

ESTRATÉGIA INICIAL

Inicialmente, planeamos uma lógica de servidor básica onde este controla a fila de pedidos assim como a lista de filtros. Este depois receberia os pedidos dos clientes, processando-os e efetuando de seguida um *fork* onde o processo filho trataria de efetuar o pedido e comunicar com o cliente por um *named pipe* privado. Para tal usaríamos um esquema de comunicação onde o servidor estaria à espera de entrada de novos clientes, ou seja, efetuando *reads* sucessivos no pipe principal. No entanto esta abordagem trouxe problemas pois não haveria forma de recolher os processos filhos que foram anteriormente criados.

Mudamos então a abordagem ligeiramente, optando por pôr o servidor num loop onde espera por processos filhos (caso existam) e que a entrada de novos clientes fosse sinalizada através de uma interrupção. Para isto tivemos de criar um pipe extra (Servidor -> Cliente) onde é armazenado o pid do servidor. Assim clientes que desejam comunicar com o servidor apenas têm de ler o pid armazenado e enviar um sinal (SIGUSR1).

SERVIDOR E CLIENTE

Como referido anteriormente, para evitar um espera ativa de clientes, decidimos tratar a entrada de novos clientes como interrupções no servidor pois, na sua escala de runtime, a entrada de um cliente é considerado um "evento raro".

Quando recebe esta interrupção, o servidor corre uma rotina chamada "new_client". Esta lê o pipe (Client -> Server), monta o pedido, verifica a sua validade e envia ao cliente um sinal de confirmação ou rejeição do pedido (SIGUSR1 e SIGUSR2, respetivamente). Caso este seja aprovado, adiciona-o à fila de pedidos e, se for um pedido de status ou inválido, é imediatamente processado (Mais informação sobre a estratégia de fila implementada no subtópico "Queue Manager"). Quando o servidor acaba de tratar do novo cliente, escreve novamente o seu PID no pipe (Server -> Client) e retorna a sua execução.

O loop fundamental do servidor é constituído principalmente pela espera de processos utilizando a variável global "ongoing_processes" que contabiliza todos os processos filhos em execução, ficando então à espera destes. Quando estes retornam, o seus pids são recolhidos e utilizados para eliminar o pedido respetivo da fila de espera sendo de seguida processado um novo pedido.

No outro lado, o cliente tem uma lógica mais simples, validando de forma simplificada o pedido do utilizador (verifica o número de argumentos e o comando) e envia-o para o servidor, efetuando a sinalização prévia. De seguida é efetuado "pauses" onde o cliente fica á espera da aprovação do servidor. Se receber uma aprovação positiva o cliente cria um named pipe (com o seu pid como nome) e aguarda novamente por outro sinal. Ao receber este sinal, é lhe indicado de que um processo filho lhe foi atribuído e que o seu pedido está a ser processado, logo, o cliente entra novamente em pause. Quando receber outro sinal, significa que o seu pedido foi efetuado. Assim o cliente lê do pipe o valor de sucesso ou, no caso do pedido status, lê o estado do servidor.

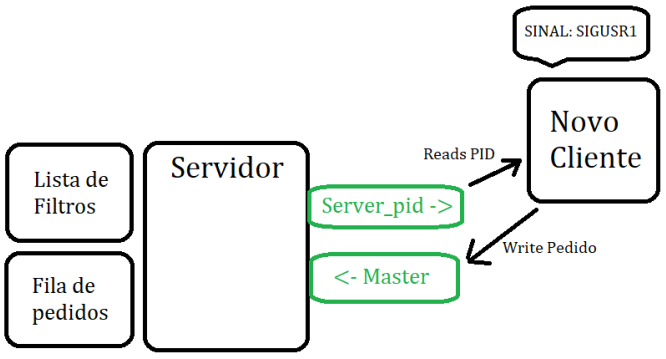

FIGURA 1 – EXEMPLO: NOVO CLIENTE SINALIZA O SERVIDOR

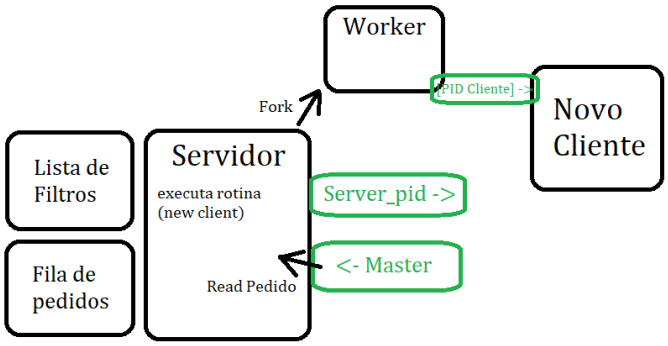

FIGURA 2 – EXEMPLO: SERVIDOR RESPONDE AO PEDIDO

SERVIDOR E AS SUAS COMPONENTES

FILTERMGR.H

Este módulo é responsável pela leitura, armazenamento e controlo dos vários filtros do servidor. Este contém funções para a leitura do ficheiro de configuração, verificar a existência de filtros e a sua disponibilidade, ocupar/desocupar os filtros e obter o código executável de cada filtro.

REQUESTLOGIC.H

Este módulo armazena toda a informação dos pedidos enviados pelos clientes. Estes pedidos têm informação como: PID do cliente e do "trabalhador" (processo filho que efetua o pedido), a validade do pedido e se este já está a ser processado, o seu tempo de espera e por fim, ficheiro de input, output e os filtros a serem usados.

QUEUEMGR.H

Este módulo é responsável por armazenar os vários pedidos e controlar a ordem de entrada e saída destes mesmos. A função "rQ_next_request" é a sua função principal e é responsável por decidir qual o pedido que será processado a seguir.

Esta função utiliza uma mistura da estratégia FCFS (First Come First Serve), onde os pedidos são armazenados numa lista por ordem de chegada e da estratégia SJF (Shortest Job First), em que pedidos com menos filtros são considerados mais curtos ("Less Filters First") e, mesmo sendo mais recentes, podem ser enviados para processamento caso os seus filtros estejam disponíveis, favorecendo pedidos "pequenos". Por cima dessa implementação, para evitar "starvation" utilizamos algo semelhante a Preemptive Priority, onde cada pedido contém o seu "tempo de espera" que é incrementado sempre que algum pedido é processado. Quando esta espera é excedida (limite definido) estes pedidos tornam-se de alta prioridade, ou seja, o servidor não envia mais pedidos para processo até que todos os pedidos de alta prioridade na lista sejam terminados (evitando que pedidos "grandes" sejam ultrapassados constantemente).

SERVER.H

Este é a parte principal do servidor onde corre a função main, este módulo contém também funções como "execute_request" e "execute_transform" onde ocorre o fork, com a lógica do processo filho e a transformação dos ficheiros de áudio. Este processo de transformação ocorre utilizando um array de pipes anónimos que ligam os vários executáveis em série e os ficheiros de entrada e saída.

CONCLUSÃO

O desenvolvimento deste projeto ajudou-nos a perceber a importância da comunicação entre processos e o planeamento necessário para desenvolver aplicações de servidor que sejam robustas. Acreditamos que conseguimos alcançar todos os objetivos necessários neste projeto, apesar de uma implementação ligeiramente mais atípica.