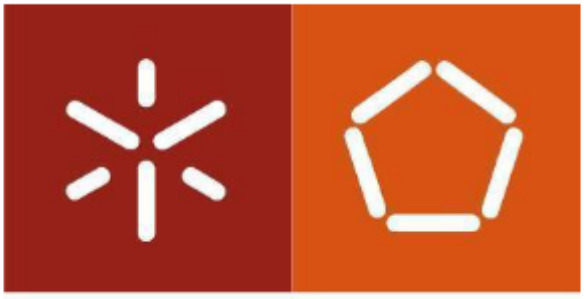

Universidade do Minho

A100547 - José Eduardo Silva Monteiro Santos Oliveira

A100824 - Gonçalo Daniel Machado Costa

A100759 - Pedro Afonso Moreira Lopes

Índice

1.Introdução

No âmbito da disciplina de Sistemas Operativos, foi pedido que fosse realizado um projeto de monitorização de programas executados numa máquina.

Para isso, os utilizadores irão executar programas através dos clientes(tracer), e obter o tempo de execução do mesmo. Por outro lado, um administrador, através do servidor, deverá conseguir consultar informações sobre os programas que estão a ser executados no momento, incluindo o tempo de execução que passou até ao momento da consulta.

Outras funcionalidades opcionais para o servidor são o armazenamento de informações sobre processos já terminados. A comunicação entre o servidor e cliente deve ser efetuada através de um *pipe com nome.*

1.1.Explicação geral da abordagem ao projeto

Para a resolução do problema apresentado, foi criado um FIFO geral para a comunicação entre o servidor e o cliente, e a criação de FIFOs especializados para realizar comunicações que devem ser efetuadas de modo seguido entre o servidor e o cliente, ou seja, às quais qualquer escrita efetuada pode afetar o armazenamento ou cálculo efetuados pelo servidor.

O tracer ficou responsável por executar os programas pedidos pelo utilizador, enquanto que o monitor ficou encarregado de receber as informações enviadas pelo cliente, agrupá-las diretamente no servidor enquanto os programas estivessem a correr e a guardar/libertar as informações relativas aos mesmos quando o programa acabasse de correr.

O projeto possui duas estruturas importantes, conhecidas por ambos clientes e servidor:

- A *struct processo* é utilizada para enviar informações do cliente para o servidor, que possui espaço para um inteiro, o número do processo, para duas strings de 50 caracteres, uma para armazenar o nome do ficheiro, e outra para mandar o caminho para o FIFO que vai ser utilizado e uma *struct timeval* com o tempo atual.
- A *struct status* é utilizada para enviar informações do servidor para o cliente, que possui espaço para um inteiro, o número do processo, para uma string de 50 caracteres, uma para armazenar o nome do ficheiro e um long int com o tempo passado até o momento de envio.

Para além disso, o servidor irá armazenar as informações relativas a cada processo num array de processos alocado dinamicamente, para ser possível aumentar o array caso este esteja a ser inundado de pedidos.

2.Funcionalidades

2.1.Execute

2.1.1. -u

O execute -u é uma funcionalidade básica do programa. O utilizador corre um único programa, e as suas informações são enviadas para o servidor duas vezes. Na primeira, serão enviadas as informações para colocar no array de processos do servidor, e na segunda estas informações serão enviadas para poderem ser removidas desse mesmo array. Para enviar essas informações utilizamos a *struct processo,* porém não preenchemos a string com o FIFO, pois as informações serão enviadas pelo FIFO principal, visto não ser necessário enviar várias vezes a mesma informação.

2.1.2. -p

O execute -p tem uma execução muito semelhante ao execute -u. Assim como o primeiro, irá enviar as informações duas vezes através do FIFO principal, sem a necessidade de preencher o espaço na *struct processo* relativo ao path para o FIFO.

Porém, a maior distinção vem da execução dos programas. Para criar uma pipeline de processos vão ser necessários N-1 *pipes* anónimas, sendo N o número de programas. Cada processo filho criado para executar cada comando vai ler do descritor de read do pipe anterior e irá escrever no pipe atual, excepto o primeiro comando, que irá ler do *standart input*, e o último, que irá escrever para o *standart output*. Ao longo da execução do programa, os *pipes* que não irão ser mais utilizados vão sendo fechados para evitar problemas de comandos que esperam por novos inputs.

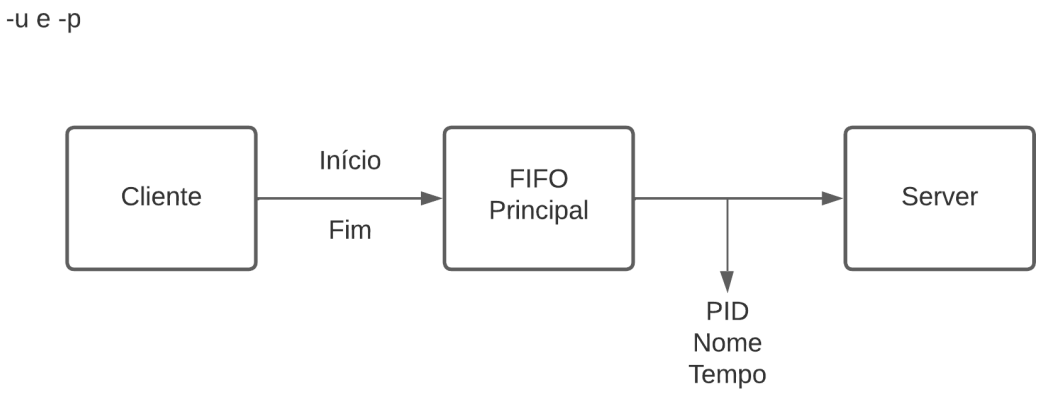

Figura 1. Estrutura básica do funcionamento do execute -u e execute -p

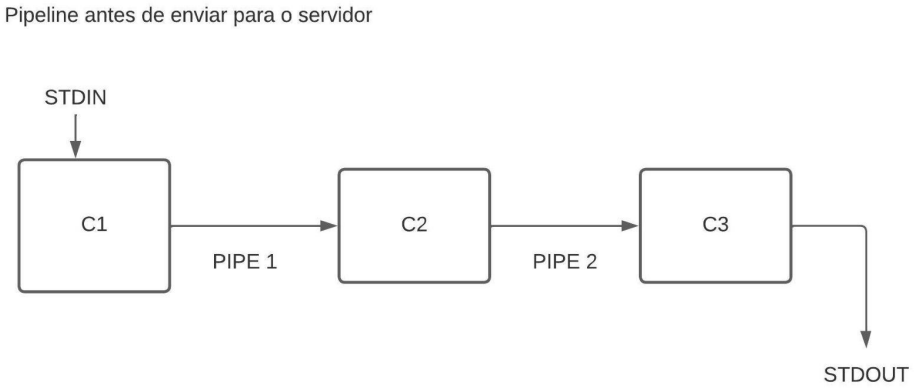

Figura 2. Esquema de execução de programas encadeados

2.2.Status

A função status pede ao servidor que devolva as informações que estão atualmente a serem executadas pelos diferentes clientes. Esta função corre em paralelo aos outros programas, pois, quando recebe, através do FIFO principal, uma *struct processo* cujo número é igual a 0, esta cria um processo filho para lidar com este pedido. Esse processo irá ler do array de processos e identificar quais são os processos cujo número é diferente de zero, ou seja, estão ativos, e devolvê-los através de um FIFO diferente para evitar que outros clientes possam receber as informações destinadas ao cliente que executou a função status.

2.3.Armazenamento de informação sobre programas terminados

De modo a poder ser armazenada a informação relativa aos programas terminados, verificou-se qual era o número de argumentos passados na chamada ao servidor, e caso fosse maior que 1, era certo que existia mais do que um argumento, então, sempre que o servidor recebesse uma mensagem para apagar um processo, ele cria um ficheiro com o caminho indicado, e lá irá escrever uma *struct status* com as informações do final da execução do programa, com o pid, o nome e o tempo.

2.4.Stats

Cada uma das funções de stats possuem algumas semelhanças, visto que todas são parecidas. Cada uma delas cria um FIFO único para se comunicarem com o servidor, para onde enviam os números dos processos realizados e por onde recebem as informações para a resposta à função. Todas elas, de modo a impedir que o mesmo FIFO seja aberto para ler antes de ser tudo escrito, tem um processo filho a realizar a escrita dos *pids,* enquanto que o processo pai espera pela finalização desta escrita. Para além disso, para descrever o nome do caminho do FIFO, e para este ser igual no cliente e no servidor, o caminho é enviado através da *struct processo*, juntamente com o pedido ao servidor para realizar a função adequada. No servidor, cada uma destas funções é executada num processo filho.

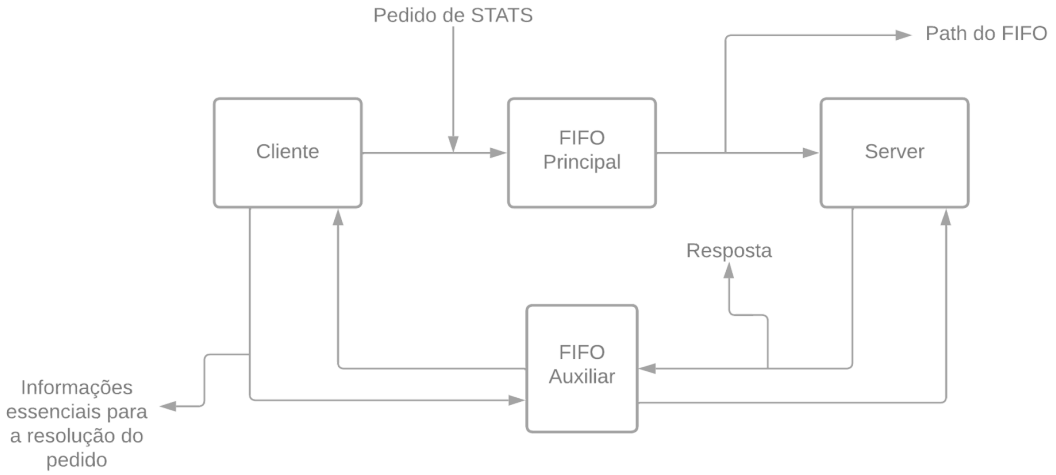

Figura 3. Realização de uma função stats genérica

2.4.1 -time

Para executar a stats-time, o servidor irá ler os ficheiros passados e ver a duração da execução daquele processo. Depois, vai adicionando no servidor a um contador, e repete até ler todos os ficheiros correspondentes aos processos enviados.Por fim devolve ao cliente o resultado da soma de todos os tempos.

2.4.2 -command

Para executar a stats-command, o servidor irá ler os ficheiros passados e ver o nome de cada processo. Depois, irá dar *parse* dessa string e separá-la em tokens, e comparar cada um deles ao nome passado, e quando o token é igual ao nome pedido pelo cliente, incrementa o contador, e repete até ler todos os ficheiros correspondentes aos processos enviados.No final ira devolver ao cliente a quantidade de vezes que aquele comando foi executado.

2.4.3 -uniq

Para executar a stats-uniq, o servidor irá ler os ficheiros passados e ver o nome de cada processo. Depois, irá dar *parse* dessa string e separá-la em tokens, e vai criar um array de strings. Sempre que aparece um token que não pertence ao array, irá adicioná-lo. Por fim, depois de ler todos os processos irá escrever cada token através do FIFO, e o cliente irá imprimi-lo.

3.Conclusão

O projeto propôs várias dificuldades, porém, as mais importantes de se realçar foram a construção da pipeline na função execute -p, visto que obrigou a um pensamento diferente do comum, visto que era necessário não só ter cuidado com o fecho das *pipes* anónimas, bem como controlar quando elas eram lidas e se existiam outros problemas com a manipulação da memória. Para além disso, existem áreas onde o código poderia ser melhorado, como, por exemplo, a criação de funções para aumentar não só a legibilidade como também a organização do código, e também permitiria evitar algumas repetições de código que possam existir. Com a conclusão do projeto, acreditamos ter concluído este trabalho com sucesso, implementado todas as funcionalidades pedidas, e com isso aprender sobre a manipulação de *pipes* anónimas e com nome, bem como a manipulação de ficheiros binários.