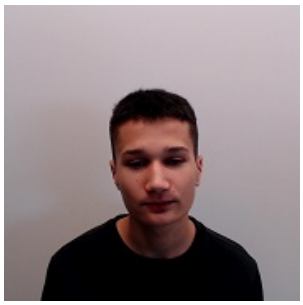
Universidade do Minho
Escola de Engenharia

Sistemas Operativos

Grupo 29

Benjamim Meleiro Rodrigues A93323
Francisco Pinto Lameirão A97504
Lara Beatriz Pinto Ferreira A95454



A93323

A97504

A95454

9 de maio de 2023

Índice

1 Introdução

Este relatório diz respeito ao trabalho prático proposto na unidade curricular de Sistemas Operativos. O objetivo do trabalho consiste no desenvolvimento de um serviço de rastreamento e monitorização da execução de programas.

As funcionalidades do programa passam por permitir que os utilizadores possam executar programas através de um cliente enviando informações sobre os mesmos para um servidor. Além disso, o serviço também oferece recursos para que seja possível consultar os programas atualmente em execução, incluindo o tempo gasto em cada um deles. Por fim, o servidor disponibiliza funcionalidades para consulta do histórico de execução de programas.

2 Arquitetura do Programa

Como referido na introdução, a aplicação é constituída por dois programas principais: o servidor (monitor.c) e o cliente (tracer.c). A comunicação entre os dois programas é feita através de um FIFO, que permite ao cliente enviar pedidos para o servidor. Caso seja necessário, também é criado um FIFO identificado pelo PID do processo tracer de forma a este conseguir receber informação enviada pelo servidor em resposta ao seu pedido.

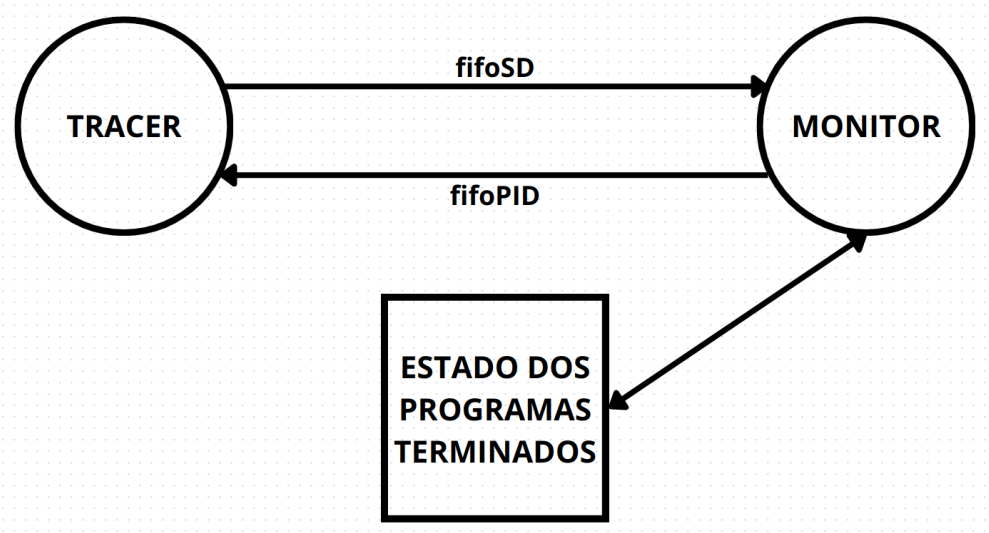


Figura 1: Arquitetura do Programa

2.1 Cliente

O cliente está responsável pela execução dos programas indicados pelo utilizador e por enviar os pedidos correspondentes ao servidor. Se o utilizador usar alguma das funcionalidades que requer comunicação com direção do servidor para o cliente (todas menos a execute) é o cliente que cria um FIFO identificado pelo seu PID.

2.2 Servidor

O servidor está responsável por receber pedidos vindos do lado dos clientes e enviar a informação correspondente. É também nele que é criado o FIFO que irá ser utilizado para estabelecer comunicação entre o servidor e o cliente.

Este também guarda o estado dos programas em execução no momento e é responsável por guardar o estado dos programas executados aquando do fim da sua execução em ficheiros.

3 Funcionalidades Implementadas

3.1 Execute

A função execute recebe como argumento uma string que define o modo em que vai operar (-u para executar um só programa ou -p para suportar a execução encadeada de programas do utilizador), o nome dos programas que o Cliente vai executar e os possíveis argumentos dos mesmos. Dependendo do número de processos fornecidos como argumento, a função é executada de forma diferente.

Antes de executar os programas, é enviada a informação relacionada com o execute através do FIFO ligado ao servidor, incluíndo o timestamp mesmo antes do início da execução do programa. Ao receber esta informação, o servidor preenche uma struct com a informação presente na string (PID, programas a correr e o tempo de início da execução) e insere-a na Hashtable, usando o seu PID como chave.

Se houver apenas um programa a executar (-u), os argumentos são analisados usando a função strtok para obter o nome do programa e seus argumentos individuais. Em seguida, é exibida uma mensagem indicando o PID em execução e é feito um fork para criar um processo filho. No processo filho, é chamada a função execvp para executar o programa especificado pelos argumentos. Se ocorrer um erro, uma mensagem de erro é exibida e o processo filho é terminado. O processo pai aguarda a conclusão do processo filho usando a função wait.

Se houver mais do que um programa a executar (-p), é criado um array de pipes anónimos usando a função pipe. Em seguida, os argumentos são analisados para cada programa individual usando a função strtok. É feito um loop para criar um novo processo filho para cada programa. No primeiro processo filho, a saída é redirecionada para o pipe anónimo usando dup2, e em seguida, o programa é executado usando execvp. Nos outros processos filhos, a entrada é redirecionada do pipe anterior usando dup2, a saída é redirecionada para o pipe atual e o programa é executado. O último processo filho apenas redireciona a entrada do pipe anterior e executa o último programa. O processo pai aguarda a conclusão de todos os processos filhos usando a função wait.

Depois de concluir a execução dos programas, é registado o tempo do fim da execução e é enviada outra vez uma string através do FIFO, contendo o PID do processo e o tempo em que terminou a sua execução. Ao ler esta string, o servidor vai procurar a informação guardada anteriormente na Hashtable através do PID do processo e cria um fork onde o processo filho tem o trabalho de escrever a informação num ficheiro binário. Também é removido o processo da HashTable.

Finalmente, é apresentado o tempo que o(s) programa(s) demoraram a executar subtraindo os tempos anteriormente registados.

3.2 Status

A função status, quando é executada pelo utilizador, abre o FIFO usado anteriormente mas também cria um FIFO associado ao PID do processo para poder comunicar quais os processos em execução do servidor para o cliente.

É enviada uma string para o servidor contendo o PID do processo que fez o pedido através do FIFO público, que é usado para conseguir abrir o FIFO anteriormente criado. Depois deste envio, a ligação é logo fechada pois já não é necessário transmitir mais informação do cliente para o servidor

Se não ocorrerem nenhuns problemas com as aberturas dos FIFOS, é enviado através do novo FIFO uma string com os valores de cada struct guardada na Hashtable (PID, nome do(s) processos(s) e tempo passado, que é calculado através das timestamps antes de ser enviado para o cliente).

O cliente, ao receber informação de cada processo, dá print desta para o terminal até ao cliente ler toda a informação enviada pelo servidor. Quando isso acontecer, é fechado o FIFO.

3.3 Stats-time

Esta função tem como objetivo apresentar o tempo total que um conjunto de programas (identificados pelos seus PIDs passados como argumento) demoraram a executar.

Para fazer isto, o cliente começa por abrir o FIFOSD e criar um FIFO identificado pelo seu PID. Depois, junta todos os PIDs indicados como argumento numa string, que envia junto com o seu PID e a quantidade de PIDs inseridos. Esta string é enviada para o servidor através do FIFOSD anteriormente aberto.

Ao receber esta string, o servidor vai dividi-la e tentar abrir todos os ficheiros binários com os PIDs inseridos como nome. Para fazer isto, é utilizado um fork para cada PID que, após abrir uma ligação de leitura com o ficheiro binário que pretendia encontrar, devolve o tempo gasto guardado no ficheiro.

Depois de serem acedidos todos os ficheiros, o servidor soma todo o tempo gasto, envia-o através do FIFO identificado pelo PID anteriormente criado e fecha a ligação.

O cliente vai ler o tempo que o servidor enviou, vai fazer print do mesmo e vai também fechar a ligação.

3.4 Stats-command

A função stats-command pretende apresentar a quantidade de vezes que um certo programa foi executado para um conjunto de PIDs que também são dados como argumento.

Para fazer isto, o cliente faz os mesmos passos que fez na anterior (criar/abrir FIFOs) e volta a enviar uma string pela FIFOSD, desta vez incluindo o seu PID, o número de processos, o programa que pretendemos procurar, os PIDs que iremos usar para procurar e a quantidade deles.

Com esta informação, o servidor vai abrir os ficheiros binários com os PIDs indicados da mesma forma que fez na stats-time, mas desta vez, depois de ler o ficheiro, verifica se o nome do programa executado é igual ao do inserido como argumento e, se for, incrementa uma variável que, no final, é somada tal como o tempo no função anterior. No final de tudo, o servidor envia a quantidade de vezes que o programa foi executado pelo FIFO identificado pelo PID do cliente e fecha a ligação.

O cliente vai receber a quantidade de vezes que o programa foi executado e vai dar print dessa informação no Standard Output e de seguida vai fechar a ligação com o FIFO pelo qual recebeu a informação.

3.5 Stats-uniq

Esta função tem como objetivo apresentar todos os programas diferentes que foram executados na lista de PIDs que é dada como argumento.

O funcionamento desta função é muito parecido com o das outras duas Stats, ou seja, depois de abrir e criar os FIFOs, o cliente vai enviar uma string com o seu PID, a lista de PIDs e a quantidade de PIDs.

O servidor vai receber essa string e, como nas outras Stats, vai fazer fork para ler todos os ficheiros binários com os PIDs inseridos como argumento. Como é necessário enviar a informação lida (nome do programa executado), é criado um pipe onde os processos filhos vão escrever a informação que lerem dos ficheiros binários.

Depois de todos os processos filhos terem acabado, o processo pai vai concatenar toda a informação que recebeu do pipe numa string.

De seguida, cria um par de pipes e redireciona o descritor de leitura do pipe anterior para o descritor de entrada padrão do processo filho.

Toda a informação que foi colecionada anteriormente na string é escrita no pipe que foi criado e é criado outro processo filho que serve para executar o comando "uniq", para eliminar qualquer programa repetido que esteja na string.

Quando o processo filho terminar, o processo pai vai fechar as pipes desnecessárias e ler o resultado do processo filho que foi escrito numa das pipes. Depois de ler, vai fechar esse pipe e enviar a string lida pelo FIFO associado ao PID anteriormente criado.

O cliente, ao receber a string, vai dar print dela no Standard Output antes de fechar as ligações restantes.

4 Conclusão

Durante o desenvolvimento deste projeto, tivemos a possibilidade de implementar alguns dos conhecimentos que foram adquiridos nas aulas de Sistemas Operativos. Contudo, sentimos algumas dificuldades que nos obrigaram a ser mais autónomos e a procurar soluções diferentes para os problemas que foram propostos.

Deste modo, e para concluir, consideramos ter cumprido todos os requisitos pedidos, implementando uma aplicação cliente/servidor através de comunicação entre processos.

Assim sendo, o grupo considera que o seu projeto tenha sido bem sucedido.