Universidade do Minho

Sistemas Operativos
Trabalho Prático
Grupo-02

Rui Jordão Sampaio Gonçalves (A91652)
Pedro Diogo Pinto Martins (A91681)

Índice

Introdução

Este projeto tem como objetivo implementar um serviço de monitorização de programas executados numa máquina, permitindo que os utilizadores possam executar programas e obter o seu tempo de execução através do cliente. Além disso, o administrador de sistemas poderá consultar todos os programas em execução, juntamente com o tempo dispendido pelos mesmos, através do servidor. O servidor também terá a capacidade de fornecer estatísticas sobre programas já terminados, como o tempo de execução agregado de um determinado conjunto de programas. Este serviço é valioso para monitorar o desempenho do sistema e identificar quaisquer problemas que possam surgir. Com isso, espera-se melhorar a eficiência e a produtividade do sistema como um todo.

Cliente

O cliente, denomidado por tracer, aceita dois comandos:

1. **/tracer execute -u "prog-a arg-1 (...) arg-n"**
2. **./tracer execute -p "prog-a arg-1 (...) arg-n | prog-b arg-1 (...) arg-n | prog-c arg-1 (...) arg-n"**

Caso seja enviado outro comando é imprimida uma mensagem de erro.

Para comunicar com o servidor, o cliente primeiro abre a pipe “fifo” criada previamente no servidor, com o intuito de informar o seu pid. A partir daí, são criados dois pipes, com o nome “pipe<pid>Leitura” / ”pipe<pid>Escrita” de modo a garantir que não existe dois clientes a escrever no mesmo pipe ao mesmo tempo.

A **pipe<pid>Leitura** serve para o cliente ler informação e a **pipe<pid>Escrita** serve para o cliente escrever informação.

O cliente é também responsável por proceder a inicialização da execução dos programas passado pelo utilizador na sua inicialização.

Para realizar a execução dos programas pedidos pelo cliente é mandatório fazer uso de uma das funções da família exec. Em ambos os casos foram utilizados a execvp, pois consideramos ser a mais adquada pela forma como espera os argumentos para funcionar.

Funcionalidade -u:

Para executar a funcionalidade -u, é criado um processo filho para que neste seja executado o execvp, logo após termos feito tratamento dos argumentos existentes no argv. Primeiro são criados os arrays **pidString** e **info**, respetivamente, para guardar informação referente ao pid do filho criado e para guardar a informação referente à mensagem que vai ser mandada para o servidor.

O **info** irá conter <<pid do filho> <nome do programa a executar> <tempo inicial do programa>>.

Entre o pedido do **tempo incial do programa** e a **execvp,** juntamos a informação no **info** e enviamos para o servidor, utilizando o pipe referido anteriormente(**pipe<pid>Escrita**).

Fora do processo filho recebemos o tempo anteriormente enviado para o servidor que fica alocado no variável **tempo_inicial_server.** É então calculada para a variável **duracao** a resposta final, que irá ser impressa no terminal do cliente.

Após ser impressa a informação do tempo demorado pelo programa o cliente termina.

Funcionalidade -p:

Para executar a funcionalidade -p conseguimos separar em dois casos o uso desta funcionalidade:

- 2 programas separados por 1 pipe
- N programas separados por N-1 pipes

Começamos por percorrer o argv à procura do caracter '|' para contar quantos pipes são precisos e basta somar mais um para saber o número de programas.

Fazemos uso da função **criaPipes** e **closePipes** de modo a facilitar a criação e eliminação dos pipes usados. Fazemos também uso da função **splitString** para guardar num array os argumentos que a execvp necessita.

2 programas:

No caso de o número de programas ser igual a 2 basta criar dois processos filhos, utilizando um para cada programa, tendo em atenção que no primeiro caso utilizamos a função **dup2** para redirecionar a saída padrão do processo atual (representado por **STDOUT_FILENO**) para o **fd[0][1]** (pipe). Assim o que a execvp retornar na sua chamada irá ser enviado para a ponta de escrita do pipe.

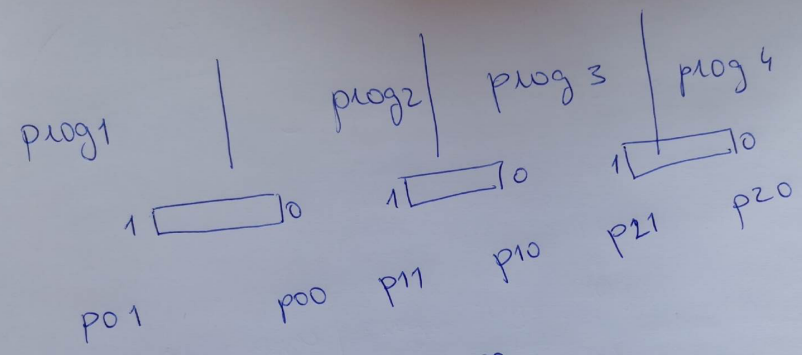
No outro processo filho, será então tratado o segundo programa da mesma forma que o primeiro, apenas precisando de ter em atenção que aqui precisamos de usar a **dup2** para redirecionar a entrada padrão (**STDIN_FILENO**) para o descritor de leitura do pipe criado.

N programas:

Neste caso colocamos um processo filho para realizar o primeiro e o último programa. E os programas intermédios são tratados num ciclo for, criando um processo filho para cada programa intermédio.
Tendo em atenção que cada processo filho intermédio necessita de dois **dup2**, um de modo a redirecionar a entrada padrão (**STDIN_FILENO**) para o descritor de leitura do pipe usado pelo filho anterior, e o outro de modo a redirecionar a saída padrão(**STDOUT_FILENO**) para o descritor de escrita do pipe usado por este filho.

Precisamos também de ter em atenção que no processo filho que realiza o último programa ao chamar a função **dup2,** iremos redirecionar a entrada padrão (**STDIN_FILENO)** para o **fd[nComandos-2][0].**

A parte da comunicação entre cliente e servidor é igual à usada na **funcionalidade -u.**



p01 p00 p11 p10 p21 p20

px y representa o pipe x no descritor y

4 programas
são necessários 3 pipes
0, 1, 2
logo, último seria
nComando - 2

Servidor

O servidor, denominado por **monitor** é sempre iniciado da seguinte forma:

- ./monitor

Para comunicar com os clientes começámos por criar um pipe com nome que serve para que cada cliente envie o seu pid. Com essa informação criamos um processo filho e dentro deste, abrimos os pipes únicos do cliente para garantir que através daqueles pipes só o servidor e este cliente podem comunicar.

No caso, o servidor recebe a informação enviada a partir do processo filho do cliente, sendo esta guardada numa variável chamada **buffer,** que contem o seguinte:

- **<<pid do filho> <nome do programa a executar> <tempo inicial do programa>>**.

Com as funcionalidades que foram implementadas neste projeto, o que o servidor faz a seguir é separar a informação recebida por um cliente para um array **infoOutput,** e enviar para o pipe correspondente a informação referente ao tempo, de modo que seja recebida no lado do cliente.

Arquitetatura

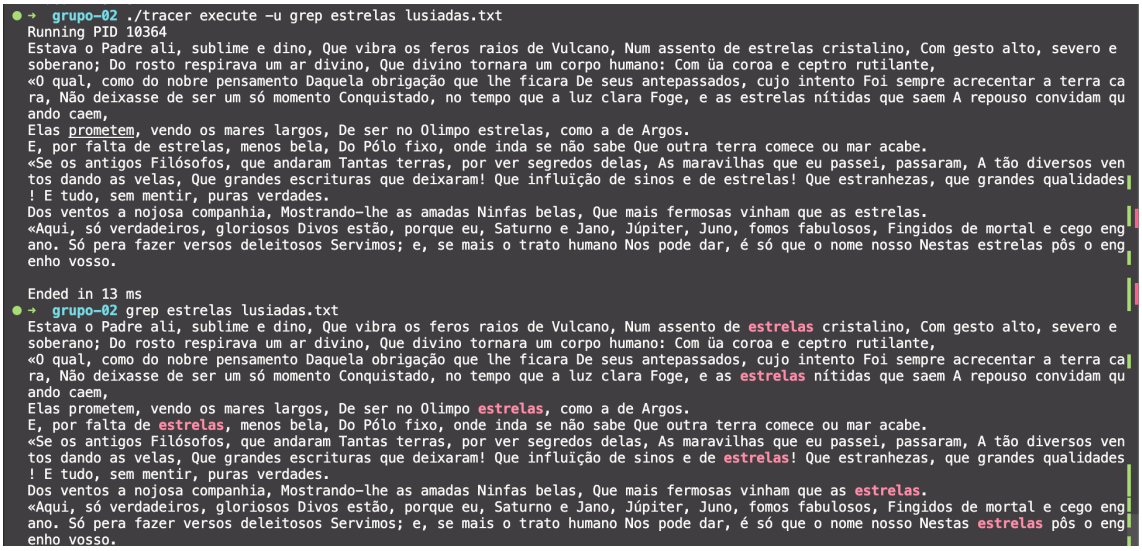
A arquitetura utilizada, com uma pipe com nome para o estabelecimento do primeiro contacto entre cliente e servidor e posterior utilização de pipes específicas para cada comunicação cliente - processo filho do servidor, pareceu-nos a escolha mais viável de forma a evitar sobreposições de comunicação e erros de leitura e escrita.

Para poder realizar a funcionalidade -u decidimos fazer um fork e um execvp.

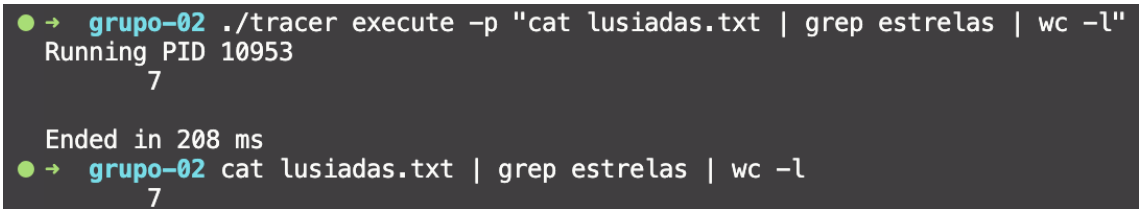
Para poder realizar a funcionalidade -p decidimos utilizar uma sequência de forks, dup2 e execvp.

Implementação de funcionalidades

Funcionalidade -u:



Funcionalidade -p:



Conclusão

Apesar de não termos sido capazes de concluir todas as funcionalidades pretendidas, gostámos de trabalhar com as chamadas do sistema e aprender mais sobre o assunto.