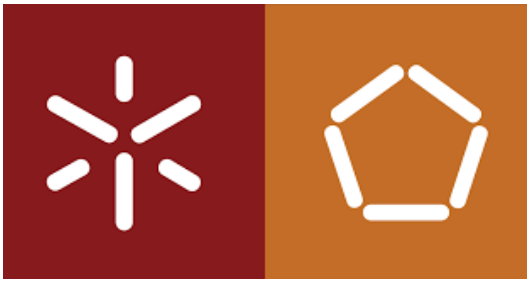

Universidade do Minho

2022/2023

Sistemas Operativos

LEI / MIEI - UMinho
Data de entrega 13/05/2023

Ema Martins[A97678]
Flávio Sousa[100715]
Henrique Malheiro[A97455]

Índice:

Introdução:

No âmbito da unidade curricular de Sistemas Operativos, foi-nos proposto desenvolver um cliente (programa tracer) que ofereça uma interface com o utilizador via linha de comando bem como um servidor (programa monitor), com o qual o cliente irá interagir.

O cliente é responsável por executar programas, tanto programas isolados, execute –u <command>, tanto como pipelines, execute –p <command / command /...>. Também é responsável por mostrar os programas que estão a ser executados naquele momento, através da opção status.

Através de interrogações ao servidor, o cliente deve ser capaz de responder a questões como o tempo que um dado programa demorou na sua execução, stats-time, quantos programas de uma dada lista de pids são do tipo passado como argumento, stats-command e dada uma lista de pids, listar todos os programas diferentes que foram executados, stats-uniq.

Servidor:

Este ficheiro cria o FIFO onde circula a maioria dos dados. Cria ainda o ficheiro que contém a informação sobre as os programas que estão a ser executados nesse momento “txt.txt” e um seguindo ficheiro “aux.txt” que será utilizado para atualizar esse mesmo estado. Em seguida tem um loop infinito que espera que dados estejam disponíveis para ler do FIFO. O primeiro conteúdo passado no FIFO é sempre um inteiro que vai determinar qual tarefa o servidor irá executar.

Se a opção for a 0, o servidor irá ler a struct passada pelo cliente que contém as informações do programa que começou a ser executado e coloca-as no ficheiro de texto “txt.txt”.

Se a opção for a 1, o servidor irá ler a struct passada pelo cliente que contém a informação do programa que terminou, irá ao ficheiro “txt.txt” retirar a struct correspondente, de modo a atualizar o estado dos programas que estão a correr nesse instante. Cria ainda um ficheiro dentro da pasta PIDS-folder, com o conteúdo necessário para a realização dos comandos stats.

Se a opção for a 2, o servidor começa por ler do FIFO o nome do FIFO pelo qual irá retornar os resultados desta opção. É nesta opção que se irá fazer o pretendido para o stats-time. Lê os ficheiros dos diversos programas acabados e soma os tempos de execução dos mesmos. Se for apenas um programa, fornece o tempo de execução desse mesmo programa.

A opção 3 é similar à 2 só que em vez de ler os tempos de execução dos programas, lê os programas que foram executados pelos diversos pids passados como argumentos e vê se são do programa que se pretende fazer a contagem das vezes que foi executado. Neste caso, executou-se o stats-command

A opção 4 também é idêntica, só que neste caso percorremos todos os ficheiros terminados correspondentes aos pids passados como argumentos e guardamos num array todos os que são diferentes. Neste caso, executou-se o stats-uniq.

Cliente:

No cliente é feito o parse dos argumentos de modo a perceber que parte do código tem de ser executado.

No caso de ser passado um comando execute –u, o cliente irá mandar para o servidor, através do FIFO, um inteiro 0 para que este execute essa mesma opção, e uma struct com o conteúdo da inicialização do programa. Em seguida faz um fork de modo a executar o programa pretendido com os argumentos passados pelo utilizador. Por último, envia outro inteiro, neste caso o 1, para o servidor de modo que a opção 1 seja executada e manda a struct correspondente à terminação do programa.

Caso o argumento passado sege apenas status, será lido do ficheiro “txt.txt” todas as structs la contidas e será escrito o seu conteúdo no terminal (STDOUT).

Se os argumentos passados são stats-time, cria-se um FIFO cujo nome corresponde ao pid do processo atual. Envia-se um inteiro, 2, para o servidor de modo a executar o código correspondente a esta opção. Envia-se posteriormente o nome no FIFO criado para que depois o resultado do stats-time seja devolvido ao cliente bem como o numero de pids a analisar. Passa-se ainda todos os pids cuja soma dos tempos se pretende obter. Por ultimo recolhe-se os resultados obtidos no servidor e escreve-se no “standart output”.

Se os argumentos são do tipo stats-command, o processo é similar só que o inteiro que se envia para o servidor é o 3 e antes de mandar a lista de pids, manda-se o programa cuja contagem se pretende realizar.

Existe ainda o caso do stats-uniq em que o procedimento é todo idêntico ao stats-time só que o parse dos resultados altera, uma vez que recebemos vários programas.

O último caso possível é o execute –p que executa uma pipeline. Para a implementar, uma vez que o número de argumentos, começa-se a contar o número de ‘/’ para saber quantos programas temos de executar. A ideia é em todos os programas, direcionar a extremidade entrada para a saída do programa anterior. Excecionalmente, no primeiro programa apenas se direciona a extremidade de saída e no último apenas se direciona a extremidade de entrada. Em cada programa, contamos o numero de argumentos que correspondem a esse mesmo programa. Para o fazer, procura-se a próxima ’/’. Desta forma, podemos criar um array, com um número de argumentos variáveis, de forma a passar ao execv aquando da execução do programa.

Caso os argumentos passados não correspondam a nenhuma das opções, irá ser escrita uma mensagem de erro ao utilizador.

Considerações:

Escolhemos structs (StructInicio, StructFim) para enviar o conteúdo dos programas, tanto do início do programa(pid, time, programa) como da sua terminação(pid, tempo), uma vez que assim poderíamos enviar todo o conteúdo simultaneamente, fazendo apenas uma escrita.

Acabamos por guardar a informação sobre os programas a ser executados no momento em ficheiros de forma que esse conteúdo seja guardado de forma permanente.

Para ser mais fácil de perceber a tarefa que o servidor teria de executar, manda-se sempre primeiro pelo FIFO um inteiro.

Na pipeline em vez de separar os argumentos por '|' separa-se por '/' uma vez que o terminal executava a pipeline automaticamente se fosse passado com '|'.

Pontos a melhorar:

Existem algumas partes em que o código é repetido ou muito similar pelo que se poderia melhorar no que toca á reutilização de código.

Num trabalho futuro pretende-se poder utilizar as funcionalidades do stats e status não só nos programas executados isoladamente como nos programas executados numa pipeline.

Conclusão:

Este trabalho prático permitiu-nos consolidar alguns tópicos abordados nas aulas como a manipulação de ficheiros, a utilização de forks e execs, o uso de FIFOS e a utilização de dups. Enquanto nas aulas acabávamos por ter uma abordagem mais singular dos tópicos mencionados, o trabalho permitiu interligá-los, algo que é de extrema importância para que futuramente os conseguirmos aplicar em trabalhos.

Nomeadamente sobre a matéria de descritores, deu-nos uma visão mais abrangente sobre os problemas que estes podem causar caso sejam manipulados de forma errada e os cuidados que devemos de ter quando o fazemos.