A74813 - André Filipe Araújo Pereira de Sousa
A89583 - Bruno Filipe de Sousa Dias
A89597 - Luís Enes Sousa

Sistemas Operativos

Projeto - Grupo 44

Introdução

Este trabalho foi realizado no âmbito da Unidade Curricular de Sistemas Operativos. O trabalho passa pela implementação de um serviço de monitorização de execução e de comunicação entre processos. No fundo, consiste na implementação de um serviço que utiliza todas as pequenas partes de matéria dadas até ao culminar desta cadeira.

O serviço deverá permitir a um utilizador a submissão de sucessivas tarefas, cada uma delas sendo uma sequência de comandos encadeados por pipes anônimos, ou caso o utilizador pretender apenas um único comando. No entanto o ponto fulcral é mesmo a realização de uma sequência de comandos encadeados por pipes anônimos. Além de iniciar a execução das tarefas, o serviço deverá ser capaz de identificar as tarefas em execução, bem como a conclusão da sua execução. Deverá ainda terminar tarefas em execução, caso não se verifique qualquer comunicação através de pipes anônimos ao fim de um tempo especificado, e também terminar as tarefas em execução ao fim de um determinado tempo a correrem e não sejam concluídas nesse período, caso seja também neste caso especificado um tempo máximo de execução.

A interface com o utilizador (neste caso chamaremos cliente) deverá contemplar duas possibilidades: uma será através de linha de comando, indicando opções apropriadas; a outra será uma interface textual interpretada (shell), e nesse caso o comando irá aceitar instruções do utilizador através do standard input.

Este projeto foi realizado, tal como todos os exercícios feitos nas aulas, na linguagem de programação C, uma linguagem de baixo nível que foi inclusive criada para o desenvolvimento de um Sistema Operativo muito conhecido denominado Unix.

Ao longo deste trabalho fomos-nos apercebendo de alguns pormenores que não foram tão aprofundados nas aulas, e uma das partes mais importantes da realização deste Projeto foi contornar e refazer coisas que a ínicio poderiam parecer bem, mas que no fim iriam interferir com outras e arruinariam o programa. Além disso conseguimos encontrar um Bug inesperado ao qual ficamos bastante impressionados, mas que achamos que deve ser um erro de Kernel.

Arquitetura do Programa

O serviço implementado depende de duas peças fundamentais: o cliente e o servidor. O Cliente ficaria responsável por oferecer uma interface ao Utilizador que permitisse suportar as funcionalidades mínimas propostas para o programa e um Servidor sobre o qual os Utilizadores podem agir e o qual ficaria responsável por manter em memória toda a informação necessária e relevante para suportar as funcionalidades implementadas do programa.

Além destas duas peças usamos ainda uma estrutura auxiliar à qual denominamos TASK e que possui informações que ajudam e facilitam em grande quantia a implementação de algumas funcionalidades. Esta estrutura possui: id (identificador da tarefa que Utilizador pediu para executar), um pid (este pid remete ao processo responsável pela realização desta tarefa), um command (uma String, char*, onde se encontra a tarefa que o Utilizador pediu para executar) e ainda um status (este Status refere-se ao estado da tarefa - se esta está a correr, se acabou corretamente, se acabou por excesso de tempo, se acabou por inatividade entre pipes anônimos e ainda se foi interrompida pelo Utilizador).

Funcionalidades do Programa

O programa deveria possuir, se o Utilizador assim preferisse um tempo máximo definido, quer para tempo de inatividade entre pipes anônimos, quer para tempo máximo de execução de uma tarefa. Estes tempos foram definidos como variáveis globais no servidor, inicialmente a -1, que indica que não estão ainda estipulados estes tempos e as tarefas não têm assim qualquer tipo de impedimento na sua realização no que toca a tempos limite. No entanto o Utilizador tem a funcionalidade, através do Cliente de definir este tempo em segundos, e caso o fizesse estes valores seriam alterados no servidor.

As tarefas que o Utilizador pretende realizar são feitas através de um filho do servidor. Enfrentamos desta forma esta funcionalidade de modo a podermos ter várias tarefas pedidas pelo Utilizador a serem realizadas ao mesmo tempo. Assim, se o Utilizador pedir que uma tarefa seja realizada, e logo a seguir pedir uma outra, esta segunda pode, por exemplo, acabar de ser executada antes da primeira, evitando assim que as tarefas tenham uma espécie de Queue e tenham de esperar que tarefas lançadas pelo Utilizador anteriormente sejam terminadas para poderem ter a sua vez. Dentro de cada processo filho do servidor criado como referido anteriormente temos ainda mais um processo filho para cada comando da tarefa de comandos encadeados por pipes. Caso seja um único comando, e não seja assim necessário a criação de nenhum pipe, este executa normalmente. Quando temos mais do que um comando vamos ter duas situações diferentes, os filhos que remetem ao primeiro comando e aos comandos do meio (aqui a aproximação é igual, diferenciando apenas no fecho de certas extremidades dos pipes, e no redirecionamento de outras), e o filhos que remete ao último comando da tarefa.

- No caso dos filhos que remetem ao primeiro comando e aos comandos do meio, estes vão criar ainda outro processo filho que vai de facto executar o comando, depois de herdar todos os dados do pai, e o processo que o criou vai enviar um sinal de alarm a si mesmo. Tomamos esta posição porque na execução do comando e o envio de um alarm para o próprio processo surgiram alguns problemas que podem ser resolvidos desta forma. Além disso fazemos isto para estes filhos e não para o filho responsável pelo último comando porque queremos medir o tempo de inatividade dos pipes, e depois de o filho responsável pelo último comando ler do pipe, todos os pipes vão estar então utilizados e não precisaremos então de medir mais tempos de inatividade de pipes anônimos.

- No caso do filho que remete ao último comando da tarefa, este não cria nenhum filho, porque como indicado acima este não necessita de medir nem se preocupar com o tempo de inatividade dos pipes anônimos, uma vez que estes já foram todos utilizados e não vão ser utilizados mais nenhuma vez. Este filho redireciona ainda o seu stdout para um ficheiro temporário, de forma a guardar o resultado da execução do seu comando.

Este controlo de tempo foi feito através de sinais de alarm que foram enviados para os diferentes processos e que utilizaram diferentes handlers. O processo principal de uma tarefa (pai de todos os filhos criados) utiliza um handler de SIGALRM que incrementa uma variável global que remete ao número de segundos passado na realização dessa tarefa. Se exceder o tempo de execução de uma tarefa definida pelo Utilizador, este processo vai sair com um código de saída que remete ao excesso de tempo de execução. No caso dos processos filho criados pelo processo principal que remetem a cada comando da tarefa, estes têm atribuído um tipo de handler diferente de SIGALRM que tal como no anterior incrementa uma variável global que remete ao número de segundos passado na realização desse comando em específico. Uma vez que se este processo não for realizado no tempo máximo de inatividade definido pelo utilizador, este processo não vai comunicar assim com nenhum pipe o que fará com que seja desrespeitado o tempo máximo de inatividade de um pipe, e assim este processo vai sair com um código de saída que remete ao excesso de tempo de inatividade de qualquer pipe anónimo. Posteriormente este código de saída vai ser interpretado pelo seu pai e o pai vai sair também com este código de saída remetente ao excesso de tempo de inatividade de qualquer pipe anónimo.

Além dos handlers referidos acima, existe ainda um handler para o SIGUSR1 que é utilizado sempre que uma tarefa acaba (seja também aqui por que razão for) e que vai alterar o estado de execução dessa tarefa no array que possui todas as TASK's já pedidas e vai removê-la do array de TASK's que estão a correr.

Além disso, caso a tarefa tenha terminado com sucesso, o servidor irá ler o seu resultado do ficheiro temporário e transmiti-lo para o cliente e para um ficheiro log.txt. Ao escrever no ficheiro log.txt, o servidor atualiza um ficheiro log.idx, de forma a poder acessar o resultado de uma tarefa de uma maneira mais eficiente. Este ficheiro log.idx guarda: o ID da tarefa correspondente; o offset do seu resultado no ficheiro log.txt; o tamanho em bytes do seu resultado.

A qualquer momento o Utilizador tem no entanto a possibilidade de pedir que uma tarefa, que tem de estar obrigatoriamente a ser executada (não pode já ter acabado, nesse

caso fica sem efeito), seja terminada. Assim, a tarefa vai ser interrompida a qualquer momento da sua execução, aquando pedido pelo Utilizador, impedindo o seu fim e deixando todo o progresso a meio, “deitando-o ao lixo”.

Todas as tarefas a correr estão num array de TASK’s que remetem unicamente às tarefas que estão a correr de momento. Existe ainda um array de TASK’s onde estão guardadas todas as tarefas pedidas pelo Utilizador (aqui incluem-se tanto as tasks que estão a correr, como as que já terminaram fosse porque razão fosse). O utilizador pode no entanto preferir ou ver/listar as tarefas que estão a decorrer de momento ou então apenas as que já terminaram, sendo também apresentadas as diferentes razões do seu fim.

O Utilizador pode também, para melhor orientação na utilização de serviço, pedir ajuda e receber uma lista de comandos que pode executar onde está um leque de funcionalidade que o programa é capaz de realizar.

Para finalizar, o Utilizador pode ainda consultar o resultado de uma tarefa já terminada, usando o comando output, que vai buscar a informação ao ficheiro log.txt.

Todos os erros que forem encontrados no decorrer do programa são apresentados ao Utilizador aquando o fecho do servidor.

Dificuldades encontradas

Ao longo do trabalho fomos encontrando alguns obstáculos, no entanto achamos que conseguimos ultrapassar da forma mais correta os diversos pormenores.

Uma das dificuldades foi aplicar os pequenos exercícios realizados ao longo do semestre aplicados numa escala maior onde se envolve todo o tipo de matéria dada num só. Tal como em muitos outros projetos em C, deparámo-nos também com pequenos problemas de memória. Um ponto que se não se tiver muito cuidado pode-se tornar bastante problemático, é a execução de código por parte de múltiplos processos e a hierarquia que estes transpõe.

Além disso, tivemos alguma dificuldade em sincronizar a abertura do pipe de comunicação servidor-cliente, levando a várias situações em que o cliente ficava infinitamente à espera que o servidor abrisse a sua extremidade. A solução arranjada passou por manter este pipe sempre aberto e obrigar o servidor a enviar um sinal ao cliente sempre que quisesse comunicar com ele. Com esta solução apareceu outro problema: como o pipe estaria sempre aberto, não havia maneira de comunicar um EOF (end of file). Sendo assim, decidimos criar uma macro que seria enviada pelo servidor e interpretada pelo cliente, indicando então o EOF.

Conclusão

Concluído o trabalho, achamos que foram atingidos os objetivos do mesmo e ficamos a reter alguns pontos essenciais.

Retemos com grande importância que a programação a este nível de abstração implica um grande rigor na realização e produção do código principalmente quando estamos a trabalhar com um vasto número de processos ao mesmo tempo, podendo tornar-se de facto um enorme desafio a enfrentar.

Contudo, ficamos com algumas ideias daquilo que se passas por trás daqueles comandos que executamos num terminal, e ficamos a perceber muito melhor aquilo que “está por trás das cortinas” o que se revelou, no final, muitíssimo interessante.