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, alterando estes valores 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 em simultâneo. 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 criado pelo servidor, como referido anteriormente, temos ainda mais um processo filho para cada comando da lista de comandos encadeados. De forma a permitir a comunicação de grandes quantidades de dados entre comandos e para facilitar a medição do tempo de inatividade dos pipes, criamos ainda outro processo intermediário, que lê o output do comando anterior e escreve para o comando seguinte, de forma ativa.

Caso a tarefa contenha um único comando, não sendo assim necessária a criação de pipes, este executa normalmente. Quando temos mais do que um comando vamos ter

três situações diferentes: a execução do primeiro comando, dos comandos intermediários e do último comando.

- No caso do primeiro comando, é criado um filho para executar o comando (redirecionando o stdout para um pipe) e outro filho para ler o output do comando e contar o tempo de inatividade desse pipe.
- No caso dos comandos intermediários, começamos por criar um filho para executar o comando referente. No entanto este filho terá de esperar pelo término do comando anterior. Quando o comando anterior termina, é criado outro filho para ler o output do comando e contar o tempo de inatividade (tal como na situação anterior).
- Chegando ao último comando, este é lançado através de um filho. Sendo que este comando terá de esperar que o comando anterior termine, o processo-filho principal espera pela execução de todos os comandos, antes de informar o servidor.

Resumidamente, há sempre 3 filhos ativos: o filho que está a executar o comando atual, o filho que está a controlar o tempo de inatividade do pipe associado a este comando e o filho que executará o próximo comando.

Os controlos de tempo são feitos através do uso de SIGALRM. Quanto ao tempo de execução, o processo-filho principal lança um alarm(max_execution_time) que o avisará quando *max_execution_time* segundos passarem. Caso aconteça, todos os filhos são terminados e o servidor é informado que a tarefa não concluiu. Em relação à medição do tempo de inatividade dos pipes, esta é feita por processos filhos auxiliares (descritos em cima).

Para além do SIGALRM são usados também o SIGINT e o SIGUSR1. O SIGINT é usado para informar o processo-filho principal que deve interromper a execução da tarefa, terminando todos os seus filhos. O SIGUSR1 é usado pelos processos auxiliares, para informar o processo-filho principal que foi detetada inatividade num pipe.

Em relação ao servidor, este também dá uso a sinais, nomeadamente o SIGUSR1. Este sinal é utilizado para informar do término de uma tarefa. O estado dessa tarefa é alterado no historial de tarefas do servidor (consoante o exit status do filho associado) e esta é removida do array que contém todas as tarefas em execução.

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, então, a possibilidade de solicitar o output de uma tarefa que tenha terminado com sucesso, usando o comando output. Este comando procura o ID da tarefa no ficheiro log.idx, tendo depois acesso à informação necessária para encontrar o seu output no ficheiro log.txt e lê-lo, de forma eficiente. No final, o comando envia o output para o cliente.

O Utilizador pode também 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 que estão a ser executados, são guardadas num array de TASK’s que remetem unicamente às tarefas que estão a correr de momento. Existe ainda outro array de TASK’s onde estão guardadas todas as tarefas lançadas durante a execução do servidor. Assim, o utilizador pode consultar as tarefas que estão a correr de momento no servidor, também como aquelas que já terminaram, sendo também apresentado o exi status de cada uma.

Para finalizar, o Utilizador pode, para melhor orientação na utilização do programa, pedir ajuda e receber uma lista dos comandos disponíveis no servidor.

Todos os erros que são 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.

Este problema de sincronização também surgiu na comunicação cliente-servidor, nomeadamente no envio do seu PID. Neste caso, optamos por criar um pipe provisório e exclusivamente para esta função, eliminado-o no final.

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.