Universidade do Minho

Mestrado Integrado em Engenharia Informática

Universidade do Minho

Controlo e Monitorização de Processos e Comunicação

Grupo 107

Carlos Preto (A89587)

Pedro Veloso (A89557)

Índice

Introdução

Como método de avaliação da disciplina de Sistemas Operativos, foi pedido que se elaborasse um projeto onde se implemente um serviço de monitorização de execução e de comunicação entre processos. O serviço deveria ser capaz de receber de um utilizador tarefas, cada uma delas sendo uma sequência de comandos encadeados por pipes anónimos, ou comandos simples sem qualquer tipo de encadeamento. Além de iniciar a execução das tarefas, o servidor deveria ser capaz de identificar as tarefas em execução, bem como a conclusão da sua execução. Deveria também ser capaz de terminar tarefas em execução, caso não se verifique qualquer comunicação através de pipes anónimos ao fim do tempo máximo de inatividade, que por sua vez podia ser alterado pelo utilizador. Por fim o servidor teria um tempo máximo especificado, onde qualquer execução que ultrapasse esse tempo seria abortada.

De modo a aplicar essas funcionalidades iria ser necessário perceber de conceitos como fork's, exec's e também de sinais. Conceitos esses que foram estudados ao longo das aulas de sistemas operativos e aplicados neste trabalho de uma forma conjunta.

Por fim foi-nos pedido que a interface por trás do utilizador, implementa-se duas possibilidades diferentes, ou seja, receber comandos através da linha de comados ou de apresentar uma interface textual interpretada, tipo Shell.

Para realizar este trabalho recorreu-se à linguagem C, sem qualquer utilização das funções da biblioteca de C para operações sobre ficheiros.

O que faz … como o faz

De modo a dar suporte a todas as funcionalidades pedidas (funcionalidades mínimas + funcionalidade adicional) foi desenvolvido um trabalho que permitisse criar um servidor capaz de receber vários comados via pipes com nome. Uma vez entregues os comandos por parte dos clientes, os mesmos começam a ser escritos no fifo pela ordem de entrada, preservando assim a ordem de pedidos, contudo é necessário que o servidor ao retirar os pedidos, que são retirados sempre da "cabeça" do fifo, retire apenas o equivalente ao pedido do cliente que vai atender, não retirando de nenhuma forma "bocados" de outros pedidos, começando assim a misturar os pedidos e a realizar as operações todas erradas. De forma a prevenir tal problema foi criada uma estrutura de dados (presente no argus.h) que é escrita pelo cliente no fifo e retirada pelo servidor, preservando assim sempre a autenticidade dos pedidos e nunca corrompendo os outros, pois o tamanho a retirar passa a ser fixo. Na estrutura de dados são incorporados os seguintes dados:

- Nome do fifo onde deverá ser escrita (pelo servidor) a resposta ao pedido do cliente.
- Opção a realizar (tempo-inatividade, tempo-execução, executar, listar, histórico ou terminar), passando a divisão da linha de instrução, ou seja, dividir a string que contém a opção a realizar e o comando a executar em duas strings diferentes, a ser trabalho do cliente. Optamos por ser o cliente a realizar esta divisão e a passá-la para o servidor, pois quando se utiliza o cliente direto da linha de comandos essa divisão já vem feita no argv, retirando assim trabalho desnecessário.
- O comado a realizar, ou seja, o segundo argumento da instrução (por exemplo o valor do tempo de inatividade a ser definido ou o comado do tipo p1 | p2 ... | pn a ser executado).

Uma vez realizada a leitura dos dados no fifo e guardados no formato da estrutura de dados acima explicada, o servidor deve perceber que opção, das seis opções possíveis (seis pois a opção de outupt e de ajuda não necessitam do servidor, para serem executadas), vai realizar. Este processo de seleção é na verdade muito simples pois essa opção já está guardada na estrutura de dados e basta então realizar a seleção através de *ifs* que incorporem comparações de strings.

Com esta explicação chegamos á parte onde o servidor vai realizar de facto a instrução, que varia de instrução para instrução. Se a instrução for definir o tempo máximo de inatividade ou o tempo máximo de execução, o servidor só tem de verificar se os valores passados são válidos (>0) e defini-los na variável global existente que remete a esses valores. Não é necessário preocupar-nos com o que vai acontecer nas instruções a executar, pois as mesmas são realizadas num processo filho, e por isso não são afetadas.

Caso a instrução seja pedir a lista de instruções a executar no servidor ou terminar uma instrução a executar, é necessário um pouco mais de trabalho, contudo torna-se muito simples de perceber a estratégia, se imaginarmos o servidor como uma fábrica constituída por um escritório, que realiza operações de encaminhamento de trabalhos para a respetiva zona e por um conjunto de 100 salas onde cada sala tem a capacidade de realizar um trabalho pedido, contudo as salas, apesar de poderem realizar trabalhos diferentes em simultâneo, só realizam um trabalho por vez, podendo assim a fábrica realizar até 100 trabalhos em simultâneo. Desta forma se o escritório guardar a informação de que salas estão a trabalhar, bem como o número

de telefone das salas que estão a realizar trabalhos e a informação dos trabalhos que estão a ser realizados em cada sala, torna-se muito simples para a fábrica listar todos os trabalhos a serem realizados ou parar o trabalho de uma sala. Basta então o escritório cruzar a informação das salas que estão a trabalhar com a informação dos trabalhos a decorrer em cada sala e mostrar o resultado ao cliente. Caso o objetivo for parar o trabalho de uma sala, o escritório apenas tem de ligar para a sala e dizer para parar. No caso do nosso servidor, o escritório passa a ser o servidor (processo pai) e as salas são os filhos criados pelo servidor quando é necessário executar um comando. Assim o servidor guarda em arrays os dados de que processos estão a executar bem como o pid dos filhos que estão a executar os comandos, só que neste caso o servidor não podendo ligar para os filhos, manda um sinal para o pid respetivo.

Uma outra funcionalidade, que é realizada pelo servidor é mostrar o histórico de trabalhos que já foram realizados e a forma como foram terminados. Esta funcionalidade é bem simples pois no fim de cada execução é escrito essa informação num ficheiro, o que permite a qualquer momento escrever no respetivo fifo todo o conteúdo desse ficheiro, mostrando assim o histórico.

Por fim uma última funcionalidade aplicada no servidor, é a funcionalidade de executar um comado do tipo p1 | p2 ... | pn, ou mesmo de um tipo mais simples, como por exemplo apenas p1. Voltando à analogia da fábrica, percebemos então que o escritório (no nosso caso real, o servidor) vai intercetar um pedido para executar um comando, contudo não pode ser ele a executar, pois ficaria incapacitado para receber outros pedidos enquanto não acaba-se de realizar o atual, tornado assim impossível receber pedidos para listar ou terminar algum processo em execução. Desta forma é necessário criar as salas para executar os comandos, porém enquanto cria as salas o escritório tem de guardar as informações das mesmas, como o pid da sala e o comado que se vai realizar nessa sala. Como o escritório tem um limite máximo de salas que podem estar a funcionar ao mesmo tempo (criamos esse limite para termos arrays de tamanho fixo), é necessário antes mesmo de criar as salas e mandá-las executar verificar se é possível criar mais salas. Caso seja possível, é só declarar a sala como ocupada, retirá-la do número de salas livres, e guardar as respetivas informações da sala. Assim a sala vai executar o comando, escrevendo o resultado no fifo de resposta e quando acabar, escrever o comado que executou, bem como a forma que acabou a execução no histórico, e informar o escritório (através de um sinal), que enquanto isto está a tratar outros pedidos ou bloqueado á espera de mais pedidos, que acabou, que por sua vez vai voltar a libertar o espaço da sala colocando-a como livre. Recorrendo a esta analogia explicamos o processo que ocorre no nosso servidor, durante a execução de um comando.

Explicado o funcionamento do servidor e como os pedidos chegam ao mesmo, falta referir como se comporta o cliente durante o processo de pedir/receber algo ao/do servidor, ou durante o processo de obter informações que não são dadas pelo servidor (ajuda e output). Caso seja pedido pelo cliente instruções que precisam do servidor, mas que não necessitam de resposta, o cliente cria então a estrutura de dados e escreve no fifo a mesma e acaba a etapa (diz-se etapa porque dependendo da forma como é feito o pedido da instrução, via linha de comandos ou por formato estilo Shell, há comportamentos diferentes, no primeiro caso o cliente morre no fim da etapa no segundo volta para receber outro pedido de instrução e repete este comportamento até que seja recebido o sinal SIGQUIT) , caso seja necessário uma resposta do servidor o cliente bloqueia até o fifo de resposta ser aberto para escrita por parte do servidor, e depois começa a escrever no ecrã e no ficheiro de output a resposta do servidor, acabando a etapa no fim de escrever toda a resposta do servidor e também de escrever o índice referente

ao output no ficheiro de índices. Contudo comados como ajuda e output não necessitam do servidor para serem executados, pois os dados estão em ficheiros que se encontram em "território do cliente", ou seja, basta abrir os ficheiros e mostrar as informações pedidas (no caso do output) ou é só mostrar informação que é conhecida pelo programa, no caso do comando ajuda.

Desta forma fica explicado como funciona a comunicação entre o cliente e o servidor bem como o funcionamento do servidor e do cliente relativamente aos respetivos comandos. Faltou apenas referir o facto de o servidor poder abortar comados a executar devido aos mesmos terem atingido tempos limites de execução ou de inatividade (por tempo de inatividade refere-se ao tempo que uma parte do encadeamento do comando demora a originar output, neste caso p1 | p2 ... | pn, se p1 demora-se mais do que o tempo máximo de inatividade a originar output, o comando era abortado, não chegando assim a terminar). De modo a dar apoio a esta funcionalidade é adotado um sistema de sinais, recebidos através da preparação de alarmes com os tempos respetivos.

Gostaríamos de referir também que a opção de mostrar o histórico, tem de ser realizada pelo servidor mesmo estando escrita em ficheiro, pois dependendo dos casos o ficheiro nem sempre pode estar ao alcance do cliente, pois o mesmo é escrito pelo servidor e por resultado disso está escrito "em terras do servidor".

Como testamos o programa

Ao longo do processo do processo de criação do nosso programa fomos realizando alguns testes de modo a garantir que todo estaria a funcionar no final.

Para garantir que funcionalidades como a execução de comados no servidor, realizamos testes com comados simples e com comados constituídos por pipes. Comandos como:

- cut -f7 -d: /etc/passwd | uniq | wc -l
- ls
- ls /etc | wc -l
- grep -v ^# /etc/passwd | cut -f7 -d: | uniq | wc -l
- ls -l
- date
- "abcd" (programa inexistente)
- cat historico.txt

Após garantir que a execução de comandos estava a funcionar começamos a realizar testes para perceber se o tempo máximo de execução e o tempo máximo de inatividade estavam a ser respeitados, contudo para testar tais funcionalidades foi necessário encontrar um programa que demorasse algum tempo até acabar a sua execução e um outro que ficaria sem gerar output por um certo tempo. Tendo isto em vista e de forma a obter programas com tais características foi desenvolvido dois programas auxiliares, um que nada fazia além de escrever "1111" no ecrã durante 30 segundos, e um outro que ficava 30 segundos adormecido. Com estes programas desenvolvidos conseguimos testar o tempo máximo de execução através do comando *./argus* -

e "./progTeste", e testar o tempo máximo de inatividade através do comado *./argus -e "./progTesteInat | wc -l".*

Uma vez garantida a correta execução destas funcionalidades testamos, com base nos programas auxiliares o correto funcionamento da opção de listar programas em execução, colocando os programas a executar com um tempo limite de execução grande, de modo a termos tempo de pedir num outro terminal a listagem dos programas a executar no servidor. Tendo a certeza que com programas a executar a opção estava a funcionar, ensaiámos sem nenhum programa de modo a perceber o que aconteceria nesse caso. Após uns ajustes conseguimos que mesmo sem programas a executar tivéssemos um resultado positivo, passando assim com sucesso no nosso teste.

Realizado estes testes, passamos a realizar testes às funcionalidades de terminar uma instrução em execução. Para tal, foi colocado em 2 terminais diferentes os programas auxiliares a executar, com um tempo máximo de execução elevado, uma vez mais para termos tempo de conseguir realizar a paragem forçada, e num outro terminal terminou-se os 2 programas à força. Para tal, percebemos que era mais lógico se o número da tarefa a terminar (passada como argumento) corresponde-se ao índice na lista de programas a executar, assim se for executado sequencialmente o comando com o argumento zero, todos os programas em execução irão ser parados, pois á medida que os programas vão terminado, os que ficam, ainda a executar, vão subindo na hierarquia da lista. Uma vez mais testamos parar um programa com um índice que não existisse, de modo a ver o que acontecia, e após alguns testes onde o computador se encerrava sozinho, conseguimos obter os resultados desejados.

Passando em todos os testes até ao momento, realizou-se testes à opção de output. Para isso bastou apenas executar o comando correspondente com um argumento valido e ir ao ficheiro de output verificar se estava a ser mostrado no ecrã a informação correta. Após essa fase testamos também para argumentos inválidos.

Por fim realizamos, testes às restantes funcionalidades como a de ajuda ou mostrar o histórico que apesar de serem mais simples de implementação, necessitavam de alguns testes para ter a certeza de que estariam bem implementadas.

Erros encontrados

Ao longo dos testes realizados foram encontrados muitos erros, dos quais grande parte foram corrigidos, contudo surgiram alguns que mesmo identificados não conseguimos corrigir.

Um desses erros está presente no nome do fifo criado para o servidor inserir a resposta, pois, o facto do nome do fifo ser gerado de uma forma aleatória, existe sempre a possibilidade de se gerar dois nomes iguais, que de facto ocorre quando em um curto espaço de tempo se realiza duas execuções, originado assim alguns erros. Devido a este problema, não podemos realizar o *unlink* do fifo no final das execuções, pois podia acontecer terminar uma execução que iria eliminar o fifo e depois terminar uma outra que precisa do mesmo fifo para receber a resposta, levando a que o servidor não conseguisse abrir o fifo para escrita, escrevendo assim o resultado no seu standard output, e o cliente ficava bloqueado à espera que alguém abrisse o fifo para escrita .

Também percebemos que se executarmos um programa que por algum motivo dá um erro de execução (como por exemplo *segmentation fault*), o servidor não apresenta qualquer tipo de informação que remete o cliente para o sucedido.

Um erro que também encontrado, e que pensamos ter corrigido, que só ocorria às vezes, era aparecer duas vezes o mesmo output, ou aparecer duas vezes escrita a frase “max tempo de execução atingido” no fim do output (quando tal acontecia), em vez de uma vez só. Este tipo de erros só acontecia quando se utilizava um Script para testar o programa, ou quando se iniciava-se uma instrução, igual á anterior, de imediatamente a seguir á finalização da anterior.

Um outro erro encontrado, também com a utilização de Scripts, é a ocorrência do aparecimento, em certas alturas totalmente aleatórias, da frase “Paragem forçada” a seguir á frase “max tempo de execução” ou “tempo inatividade alcançado”. Apesar de não conseguirmos entender o motivo para tal acontecer, conseguimos fazer com que o filho deixasse de ficar zumbi quando isto acontecia.

Por fim, não um erro, mas uma dúvida que tivemos, foi interpretar quando o comando deveria ser abortado por atingir o máximo de tempo de inatividade. Primeiramente tínhamos percebido que um comando do tipo p1 | p2 ... | pn deveria ser abortado caso, por exemplo, p1 demorasse mais do que o tempo de inatividade a acabar a sua execução. Posteriormente, mudamos a nossa interpretação de forma a que o comando p1 | p2 ... | pn, fosse abortado caso p1 (por exemplo) não gerasse output, e desta forma não comunicasse com o pipe, por mais tempo que o tempo máximo de inatividade.

Conclusão

Após a conclusão do trabalho a equipa ficou satisfeita com o resultado, apesar de um pouco desiludida com a incapacidade de resolver os problemas acima descritos.

Ao longo do desenvolvimento do projeto sentimos que fomos aprendendo um pouco mais sobre esta nova forma de programação, com concorrência e comunicação entre diferentes programas, que a disciplina de Sistemas Operativos nos proporcionou. Contudo, não só desenvolvemos capacidades como comunicar entre programas diferentes através de pipes com nome, ou criar processos que permitem realizar mais que uma tarefa em simultâneo, podendo pôr algum deles a executar um outro programa já existente, mas também desenvolvemos o nosso "C".

Por fim falta referir, na capacidade que o nosso programa tem para evoluir, pois como as execuções já são realizados em paralelos, num filho do servidor, e cada uma delas é direcionada para um fifo (de resposta) diferente, bastando assim uma pequena alteração no programa relativo ao Cliente (colocava-se a resposta do servidor a ser lida pelo cliente num processo filho e depois o pai esperava ou não por esse filho) para poder executar comandos em *background*. Contudo tal funcionalidade não foi experimentada pois o enunciado não o pedia, por isso tal afirmação baseia-se apenas em pura teoria e como tal podem surgir complicações que não estamos a prever.

Desta forma, olhando para o resultado do programa onde conseguimos desenvolver todas as funcionalidades pedidas (funcionalidades mínimas + funcionalidade adicional) e olhando também para todo o percurso de aprendizado que realizamos, ficamos, apesar da nota que irá ser atribuída, com o sentimento de dever cumprido, pois demos o nosso melhor e colocamos o máximo de esforço possível no trabalho, o que nos permitiu evoluir de uma forma bastante positiva.