Universidade do Minho

Sistemas Operativos

Trabalho Prático

Controlo e Monitorização de Processos e Comunicação

Bruno Alves 85481

Daniel Ferreira 85670

Licenciatura em Ciências da Computação

Conteúdo

1 Introdução

O objetivo deste trabalho prático consiste na implementação de um serviço de monitorização de execução e de comunicação entre processos. A interface com o utilizador deverá contemplar duas possibilidades: uma será através da linha de comando, indicando as opções apropriadas, e a outra será uma interface textual interpretada (*Shell*), e nesse caso o servidor irá aceitar instruções do utilizador através do *standard input*.

Será considerado como tarefa uma sequência de comandos encadeados por pipes anóminos. Um utilizador deverá conseguir submeter tarefas sucessivamente e para além da inicialização das mesmas, deverá ser ainda possível identificar as tarefas em execução e terminar a sua execução.

O utilizador deverá conseguir definir o tempo máximo de inactividade de comunicação num pipe anónimo e o tempo máximo de execução de uma tarefa. Ao fim do tempo máximo estabelecido caso não se verifique qualquer comunicação através de pipes anónimos ou caso o tempo de execução de tarefas ultrapasse o tempo máximo estabelecido, o servidor deve terminar as tarefas em execução.

O serviço terá que manter um histórico com as tarefas terminadas sendo que o utilizador deverá ter a opção de o consultar através de um dos comandos.

Por fim, o serviço deverá possuir uma funcionalidade de ajuda que quando utilizada deve mostrar todos os comandos possíveis e respectivos argumentos.

2 Entidades do Sistema

2.1 Servidor

O *Servidor* é responsável por ler a informação enviada por cada processo *Cliente* através de um *FIFO* para em seguida processar e executar a tarefa correspondente ao comando enviado pelo Cliente.

O *Servidor* começa por criar o *FIFO* principal, por onde vai receber as instruções provenientes dos vários processos *Cliente*. O Servidor é capaz de suportar e satisfazer vários clientes ao mesmo tempo, e como tal as instruções requeridas por uma instância não afetam a realização das restantes.

2.2 Cliente

O *Cliente* interage com o *Servidor*, solicitando a execução de instruções. Na nossa implementação do *Cliente* optamos por utilizar dois processos, um que lê as instruções e as encaminha para o Servidor, e um segundo que lê do *FIFO* para onde são enviadas as respostas obtidas pelo *Servidor*. Para tal, e como mencionado no ponto anterior, criamos um processo filho, sendo que o pai fica encarregue de receber as instruções do *stdin*, enquanto que o filho recebe as respostas e as envia para o *stdout*.

O Cliente tem disponíveis as seguintes instruções :

- ➔ Definir o tempo máximo (em segundos) de execução de uma tarefa
- ➔ Executar uma tarefa
- ➔ Listar registo histórico de tarefas terminadas
- ➔ Apresentar ajuda à utilização do serviço'«

3 Especificação

3.1 Dados

log.bin

Este ficheiro é o destino para onde são escritos todos os outputs das tarefas que o cliente quer executar.

historico.idx

Este ficheiro será para onde vão ser escritos todos os comandos executados por cada cliente.

argusd.c

É o nosso servidor.

argus.c

É o nosso cliente.

comandos.c

Contém todas as funções que são úteis para a execução das funcionalidades pedidas.

3.2 Comunicação entre as entidades do sistema

Uma das primeiras decisões com que nos deparamos foi a forma como os diversos programas deveriam comunicar durante a sua execução começando pelas interações *Cliente/Servidor*.

Tendo em conta que o sistema deve suportar a existência de um servidor e vários clientes, decidimos que a forma mais simples e eficaz de permitir ao servidor receber as instruções

dos múltiplos clientes era através de um *FIFO* principal, de nome conhecido pelo *Servido*r e *Clientes,* criado aquando a execução do servidor.

Os *Clientes* enviam instruções na forma de uma estrutura de tamanho fixo para o *pipe*. Estas instruções são depois lidas pelo *Servidor* e processadas de acordo com o comando que foi escrito previamente pelo *Cliente*..

A estrutura utilizada possui o pid do processo *Cliente* que a enviou, sendo que após a execução da instrução, o servidor envia os resultados na forma de uma outra estrutura através de um outro *FIFO* criado pelo cliente com o nome igual ao seu PID, permitindo assim que todos os clientes enviem as instruções para o *Servidor*, mas que as respetivas respostas sejam redirecionadas apenas para o *Cliente* que as enviou.

Criamos ainda um handler para *SIGINT*, de modo a que quando os programas *Cliente* e *Servidor* são terminados com *CTRL+C*, os respetivos *FIFO*'s sejam eliminados.

3.3 Estruturas utilizadas

Action

Esta estrutura é utilizada para enviar instruções de clientes para o *Servidor*. Contém o PID do *Cliente* que enviou a instrução, a opção da tarefa que o cliente quer executar(-e, -i…) e o restante da linha escrita pelo *Cliente*

```
struct action{
    pid_t pid;
    char opcao[35];
    char linha[300];
};
```

Response

Estrutura utilizada para permitir ao servidor enviar o output da execução da tarefa enviada pelo *Client*e.

```
struct response{
    char output[MAXSIZE];
};
```

4 Funcionalidades

4.1 Execução de tarefas

argus$ executar 'cut -f7 -d: /etc/passwd | uniq | wc -l'

Esta funcionalidade permite ao *Cliente* executar tarefas e corresponde à opção (-e) da linha de comandos.

Começamos por dividir aquilo que foi escrito pelo *Cliente* em opção e linha. Por exemplo se o *Cliente* escrever "-e 'ls -la'", será guardado o "-e" na opção e o restante em linha.

Decidimos então juntar estas duas com o pid de maneira a que possamos construir uma *struct Action* que enviamos ao servidor. Este recebe o comando escrito pelo cliente, processa a sua execução e envia o output para *log.bin* e para o terminal do cliente.

4.2 Listar todos os comandos

argus$ ajuda

Esta opção permite ao cliente ver imprimido no seu terminal a lista de todos os comandos possíveis e respectivas variáveis e corresponde à opção (-h) da linha de comandos..

Neste caso o que fazemos é redirecionar o *stdout* do servidor para o *arr[1]* (sendo *arr* um *pipe*), e escrevemos lá a lista de comandos. Em seguida usamos a entrada de leitura do *pipe (arr[0])* para ler o que está no *pipe*, colocar numa *struct Respons*e e enviar para o *Cliente*.

4.3 Listar histórico de tarefas executadas

argus$ historico

Esta opção permite ao *Cliente* ver imprimido no seu terminal o histórico de tarefas terminadas e corresponde à opção (-r) da linha de comandos.

O *Servidor* vai ler do ficheiro *historico.idx* e colocar tudo o que leu numa *struct Response* que depois envia para o *Cliente* que escreveu o comando.

4.4 Alterar tempo de execução máximo de uma tarefa

argus$ tempo-execucao 20

Esta opção permite ao *Cliente* definir o tempo máximo para a execução de uma tarefa e corresponde à opção (-m n) da linha de comandos.

Definimos o tempo máximo para execução de uma tarefa como uma variável de tal modo que o *Cliente* possa definir e alterar o valor através deste comando.

Caso este tempo seja ultrapassado durante a execução de uma tarefa, será acionado um SIGALRM que decidimos tratar usando a função *catch_alarm* para fazer *kill* do processo que ultrapassou este tempo.

4.5 Teste de funcionalidades

Para se testar as funcionalidades implementadas tivemos várias abordagens.No que toca à execução de tarefas, testamos esta funcionalidade através da utilização repetida da mesma com várias tarefas diferentes, com e sem pipes, e verificávamos o output.

As funcionalidades de listar histórico e ajuda foram fáceis de testar uma vez que sabíamos o output que estas deveriam ter, logo bastava apenas executá-las e ver o output. Fomos verificando o ficheiro histórico.idx para ver se este ia guardando corretamente as tarefas.

Por fim, relativamente, ao tempo máximo de execução, primeiro executamos “-m 5” e em seguida “-e ‘sleep 100’” seguido por outro comando que produzisse output. Bastava então verificar se o output do último comando aparecia em 5 segundos ou não, para sabermos se a funcionalidade estava ou não a funcionar corretamente.

5 Conclusões

5.1 Dificuldades

Ao longo do desenvolvimento do trabalho, o grupo deparou-se com algumas decisões em termos de implementação, sendo uma das primeiras a forma de comunicação entre as várias entidades do programa, nomeadamente *Servidor/Cliente*. Decidimos utilizar somente *Pipes* para as interações, e tomando partido da utilização de estruturas de dados com tamanhos fixos, esta forma de implementação mostrou ser bastante prática e eficaz, pelo que decidimos que seria a melhor para o trabalho a ser desenvolvido.

No entanto, a maior dificuldade foi a forma como se devia tratar as tarefas em execução. Inicialmente pensamos em criar um ficheiro similar ao histórico. Após alguns problemas, surgiu a ideia de se utilizar arrays de strings de tal modo que cada string fosse um comando. Seguindo este conceito criamos um array para o histórico e outro para as tarefas em execução. Quando fosse introduzido um comando, este seria guardado em forma de string no array de tarefas em execução. Quando este terminasse, a string seria transferida desse array para o array histórico. Infelizmente o problema manteve-se e foi necessário abandonar esta abordagem e manter o formato original do histórico como ficheiro.

Este problema vinha do facto de o nosso programa pressupor que determinada tarefa após ser iniciada era “instantaneamente” finalizada ou dava erro. Ou seja, mesmo que fosse executada uma tarefa propositadamente demorada, a tarefa que fosse submetida em seguida pelo mesmo Cliente só seria executada após a anterior ter sido finalizada. Derivado deste problema fomos incapazes de implementar as restantes funcionalidades.

5.2 Apreciações Finais do Trabalho

No geral, apesar de o grupo não ter conseguido implementar todas as funcionalidades, foram consolidados os conhecimentos adquiridos ao longo do semestre, assim como utilização de novas formas de abordagem para solucionar os problemas ao longo deste trabalho prático.