Controlo e Monitorização de Processos e Comunicação

Trabalho realizado por:

João Figueiredo Martins Peixe dos Santos

Luís Filipe Cruz Sobral

Paulo Silva Sousa

A89520
João Santos

A89474
Luís Sobral

A89465
Paulo Sousa

Grupo 113
Projeto SO
2019/2020
Universidade do Minho

Índice

1 Introdução

Nesta unidade curricular fomos introduzidos a um conjunto de conteúdos que abordaram a maneira de funcionamento destes sistemas informáticos. Entre temas teóricos e "maneiras de pensar", fomos apresentados a um leque de system calls presentes nos sistemas originados pelo **UNIX**.

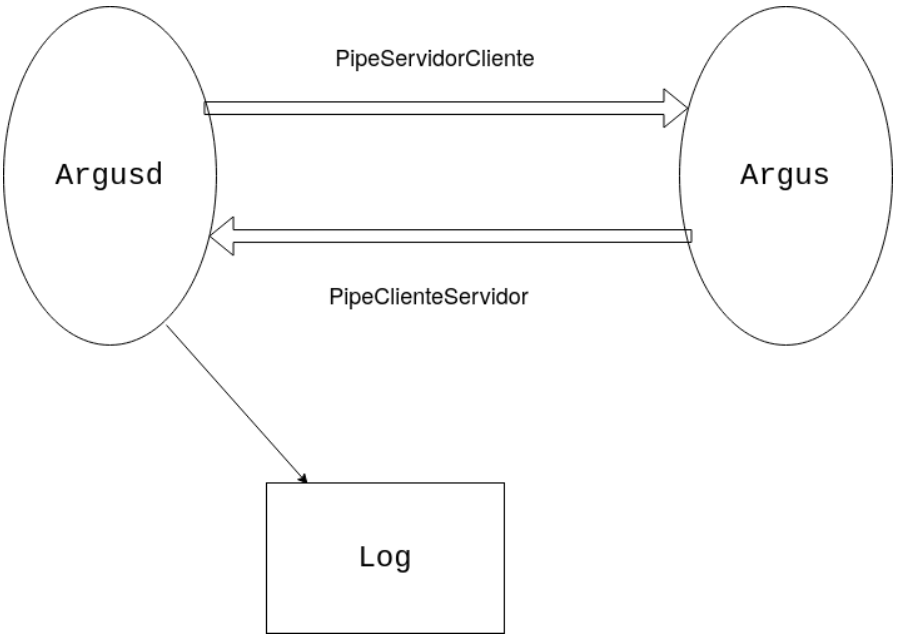
Neste contexto, foi-nos proposta a realização de um projeto que visa consolidar e avaliar a nossa aprendizagem sobre estes conceitos. O projeto consiste na implementação de um serviço de monitorização de execução e de comunicação entre processos

2 Arquitetura do Programa

A aplicação é constituída por dois programas principais, o argusd(servidor) e o argus(cliente). Estes programas comunicam entre si através de dois pipes com nome, um na direção servidor/cliente e outro na direção cliente/servidor.

O servidor argusd tem como tarefa receber os pedidos por parte dos clientes e processá-los, escrevendo o seu output(caso exista) no ficheiro log. Ainda o servidor avisa o cliente da existência de erros no input através de sinais.

Por outro lado, o cliente argus tem a possibilidade de executar comandos (tal como faria na bash), sendo estre processados pelo servidor.



Arquitetura do Programa

3 Funcionalidades Implementadas

3.1 Executar uma Tarefa

Para executar uma tarefa, primeiramente, fazemos parse do comando a executar, onde separamos os vários comandos e os seus argumentos num *char*** *comands*. De seguida, executamos, através de pipes anónimos, os comandos, tendo em atenção o tempo de execução(3.5) e o tempo de inatividade(3.6).

Se a tarefa terminar sem ultrapassar o tempo máximo de execução, sem ultrapassar o tempo máximo de inatividade e sem o cliente forçar o término desta, será dado o retorno de 0, que representa tarefa concluída.

Esta funcionalidade foi testada comparando os resultados obtidos ao executar comandos na bash com os resultados obtidos quando os comandos foram utilizados por esta mesma funcionalidade.

3.2 Listar Tarefas em Execução

Para apresentar as tarefas em execução, guardamos a informação da tarefa na estrutura *Lista*(lista ligada) e escrevemos o seu conteúdo no StdOut.

De modo a testar a funcionalidade Listar (Tarefas em Execução) utilizamos um script de teste com sete comandos **"./argus -e "sleep 10""** e um **"./argus -e "sleep 3""** no meio dos anteriores imediatamente seguido de **"./argus -l"**, no fim de todos estes comando executaríamos novamente o comando **"./argus -l"**. Ora, para verificarmos se estava correta a listagem das tarefas observamos que no primeiro **"./argus -l"** aparece a tarefa com **"sleep 3"**, mas no segundo já não, mantendo todas as tarefas o seu número inicial.

3.3 Listar Histórico

Para executar esta funcionalidade criamos um *char** historico* onde, cada vez que uma tarefa terminava a sua execução, guardavamos a informação da tarefa.

Depois, ao executar "historico", escrevemos todo o conteúdo do array no SdtOut.

Esta funcionalidade foi testada analisando se qualquer tarefa após a sua conclusão aparecia no historico e se a sua "razão" de término estava também presente(e fosse a correspondente à tarefa) no histórico. Ou seja, uma tarefa ao terminar dá retorno de um inteiro, se o inteiro for 0 a tarefa foi concluída, se for 1 foi terminada pelo cliente, se for 2 terminou por exceder o tempo máximo de execução e se for 3 termina porque excedeu o tempo máximo de inatividade, este retorno tem de estar representado no historico.

3.4 Terminar uma Tarefa

Para terminar uma tarefa o cliente pode executar esta funcionalide executando o comando "terminar n", onde n é o número da tarefa que o cliente pretende terminar.

Através do n acedemos ao array de tarefas em execução para descobrir o pid desta, posteriormente, através desse mesmo pid é feito o kill desse processo e de todos os filhos que ele criou, terminando, assim, a tarefa. O retorno será 1, que representa o término de uma tarefa por pedido do utilizador.

Para testar esta funcionalidade, utilizamos comandos com uma duração de execução relativamente elevada para podermos terminar a tarefa que o executou. Por exemplo, usamos o comando **"tail -f /dev/null"** e, para verificar se a tarefa era terminada, executavamos a funcionalidade listar depois de terminar a tarefa, para confirmar que a tarefa em questão já nao se encontrava na lista. Ainda, utilizamos o comando **"ps -a"**

para confirmar, novamente, que o processo tinha sido terminado e não apenas removido da lista.

3.5 Definir o Tempo Máximo de Execução

Esta funcionalidade permite estabelecer o tempo máximo de execução de um comando.

Para isso, criamos o comando "tempo-execucao n", onde n é o novo tempo máximo de execução.

Caso o tempo de execução seja ultrapassado, o processo é morto, tal como todos os seus filhos. O retorno será 2, que representa o término de uma tarefa por ultrapassar o tempo máximo de execução.

Para testar esta funcionalidade usamos comandos com um tempo de execução bastante elevado. Por exemplo, ao escrever o comando **"tail -f /dev/null"** este será terminado após o tempo máximo definido anteriormente, podendo ser verificado no historico que a tarefa correspondente ao comando **"tail -f /dev/null"** foi terminada por tempo de execução.

3.6 Definir o Tempo Máximo de Inatividade

Esta funcionalidade permite estabelecer o tempo máximo que um pipe anónimo pode estar inativo.

Executando o comando "tempo-inactividade n", onde n é o novo limite de inatividade, o cliente pode definir um novo máximo.

Caso o tempo de inatividade seja ultrapassado, o processo é morto, tal como todos os seus filhos. O retorno será 3, que representa o término de uma tarefa por ultrapassar o tempo máximo de inatividade.

Para testar esta funcionalidade utilizamos com tempos de inatividade de pipe elevados. Por exemplo, ao executar o comando **"ls | sleep 5"**, para um tempo de inatividade de 2s, a tarefa será terminada após 2s de inatividade do pipe, podendo ser verificado no historico que a tarefa corresponde ao comando **"ls | sleep 5"** foi terminada por tempo de inatividade.

3.7 Ajuda

Para executar este comando, criamos um *char help[]* com a informação a apresentar e escrevemos o seu conteudo no StdOut.

3.8 Output

Esta funcionalidade permite ao cliente observar o output de uma determinada tarefa. O cliente ao efetuar o comando "output n", onde n é o número da tarefa pretendida, é-lhe apresentado o output da tarefa.

Esta funcionalidade segue um conjunto de operações de modo a chegar ao resultado final. Primeiro, o argusd obtém a posição no ficheiro log (ficheiro onde estão todos os output) correspondente à tarefa em questão, através da leitura do ficheiro log.idx. Ainda, através deste mesmo ficheiro, log.idx, obtém a posição da tarefa seguinte, podendo assim calcular o tamanho do output. Por fim, lê o output calculado anteriormente a partir da posição da tarefa pretendida e envia-o ao argus.

De maneira a testar esta funcionalidade executamos vários comandos como **"ls"**, **"ls -l | wc"** e **"cut -f7 -d: /etc/passwd | uniq | wc -l"**, seguidos de pedidos de output das respetivas tarefas. Ainda, utilizamos comandos com tempo de execução bastante

divergentes com o objetivo de a ordem dos outputs ser diferente da ordem do número das tarefas.

4 Sinais

De forma a poder implementar as funcionalidades decidimos recorrer à utilização de sinais:

argusd.c

- SIGCHLD -> Quando o processo filho responsável pela execução da tarefa termina, o processo responsável pelo servidor recebe este sinal. O seu handler trata de retirar a tarefa da lista de tarefas em execução e coloca-a no histórico. Além disso, este handler guarda a ordem de execução dos processos no ordIDX e escreve no ficheiro log.idx a posição em que o output da tarefa termina no ficheiro log.

executar.c

- SIGUSR1 -> Este sinal é usado para terminar uma tarefa em execução. O seu handler mata todos os filhos do processo atual e o valor de saída é 3.
- SIGUSR2 -> Este sinal é lançado quando o cliente decide terminar o programa. O seu handler envia o sinal SIGUSR1 ao processo responsável pela tarefa. O valor de saída é 1.
- SIGALARM -> Este sinal é lançado quando o tempo de alarm se esgota. O seu handler verifica se o sinal foi recebido pelo processo principal(criado pelo servidor) ou por um processo filho. Se foi o principal, então o tempo máximo de execução foi ultrapassado, é enviado o sinal SIGUSR1 para o filho responsável pela execução da tarefa e o valor de saída é 2. Se foi o filho a enviar o sinal, então o tempo máximo de inatividade foi ultrapassado e este envia o sinal SIGUSR1 para si mesmo, sendo o valor de retorno 3.
- SIGCHILD -> Este sinal serve para verificar se algum filho responsável por executar a tarefa terminou com valor de saída 3. Caso isso aconteça, o processo que o criou termina, também, com valor de saída 3.

5 Conclusão e Reflexão Crítica

Durante o desenvolvimento deste projeto, achamos um verdadeiro desafio implementar a funcionalidade de tempo de inatividade, uma vez que, inicialmente, estavamos com dificuldades em determinar qual a razão do término da tarefa, se por tempo máximo de execução, se por tempo máximo de inatividade.

No entanto, pudemos observar que a programação neste nível de abstração é particularmente exigente na gestão de erros e validação de inputs, bem como na segurança do código.

Concluindo, este trabalho ajudou-nos a consolidar os conhecimentos obtidos nesta UC ao longo do semestre, já que nos deparamos com obstáculos de elevada complexidade de raciocínio levando assim a uma maior experiência no que toca ao uso destes conhecimentos e a uma maior flexibilidade de pensamento face a problemas novos.